基于MindSpore的深度学习模型在多媒体领域的实践

作者：彭博

目录

contents

MindSpore

背景

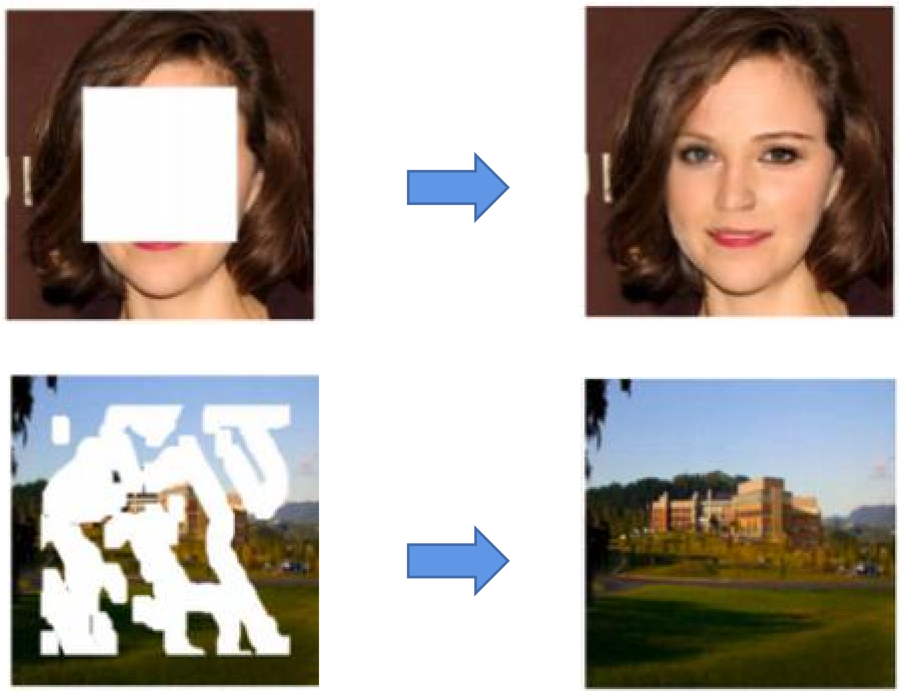
学院与华为有教学合作，基于华为云ModelArts平台进行实践

使用华为海思昇腾AtlasDK开发者套件进行开发

MindSpore可以最佳匹配昇腾处理器，最大程度地发挥硬件能力，帮助开发者缩短训练时间，提升推理性能

应用背景

图像修复是指对图像受损区域进行重建，使其内容在视觉上逼真，语义上一致的过程，在实际应用中，如图片编辑、干扰物移除、受损部位修复等方面都有很好的表现。



相关方法

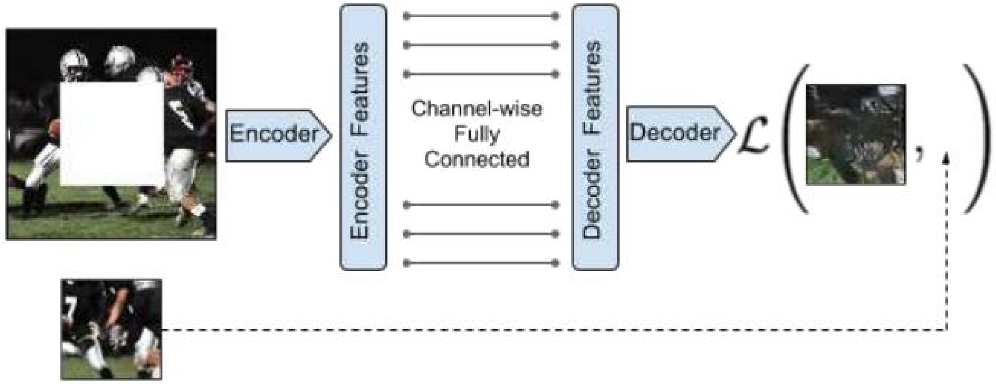


Figure 2: Context Encoder. The context image is passed through the encoder to obtain features which are connected to the decoder using channel-wise fully-connected layer as described in Section 3.1. The decoder then produces the missing regions in the image.

Pathak D, Krahenbuhl P, Donahue J, et al. Context encoders: Feature learning by inpainting[C]//Proceedings of the IEEE conference on computer vision and pattern recognition. 2016: 2536-2544.

相关方法

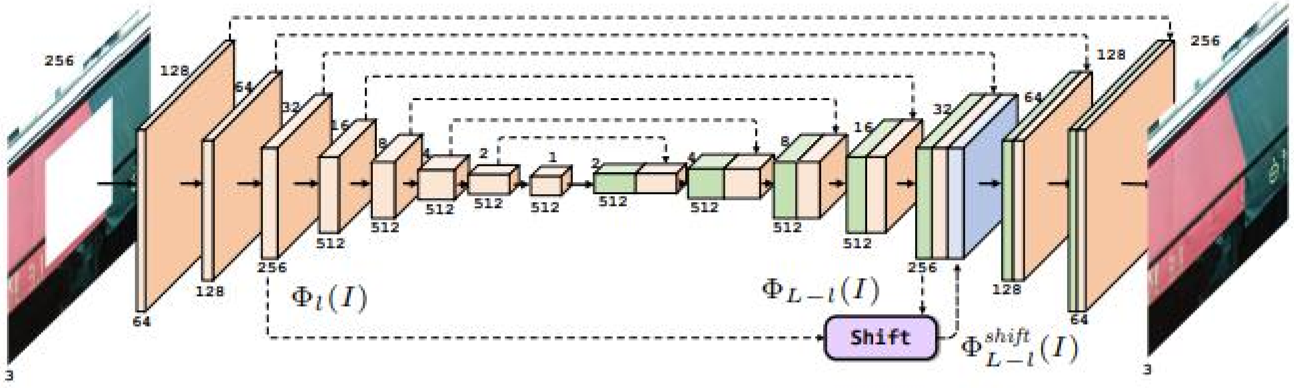


Fig. 2. The architecture of our model. We add the shift-connection layer at the resolution of 32×32.

然而，当缺失区域变大时，前景和背景像素之间的相关性减弱，导致语义模糊结果。

Yan Z, Li X, Li M, et al. Shift-net: Image inpainting via deep feature rearrangement[C]//Proceedings of the European conference on computer vision (ECCV). 2018: 1-17.

相关方法

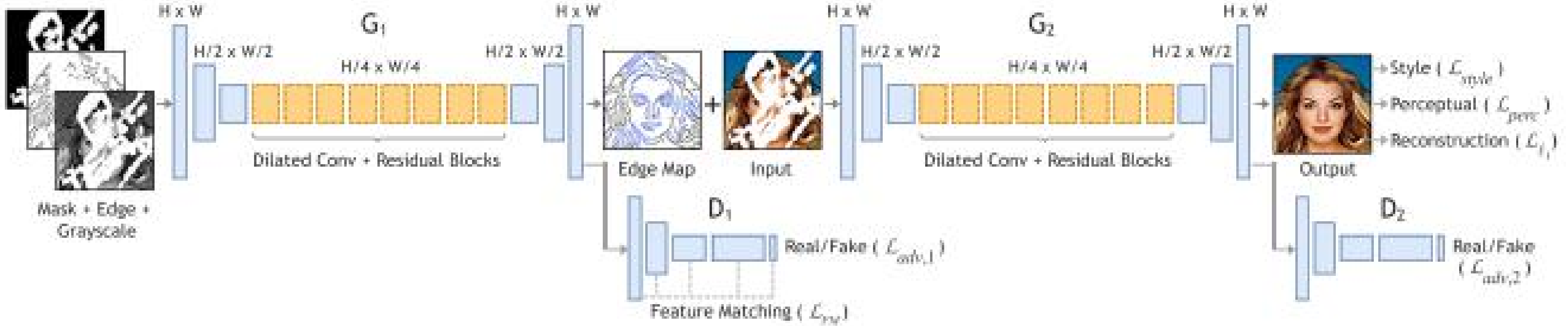


Figure 2: Summary of our proposed method. Incomplete grayscale image and edge map, and mask are the inputs of G_1 to predict the full edge map. Predicted edge map and incomplete color image are passed to G_2 to perform the inpainting task.

Nazeri K, Ng E, Joseph T, et al. Edgeconnect: Generative image inpainting with adversarial edge learning[J]. arXiv preprint arXiv:1901.00212, 2019.

相关方法

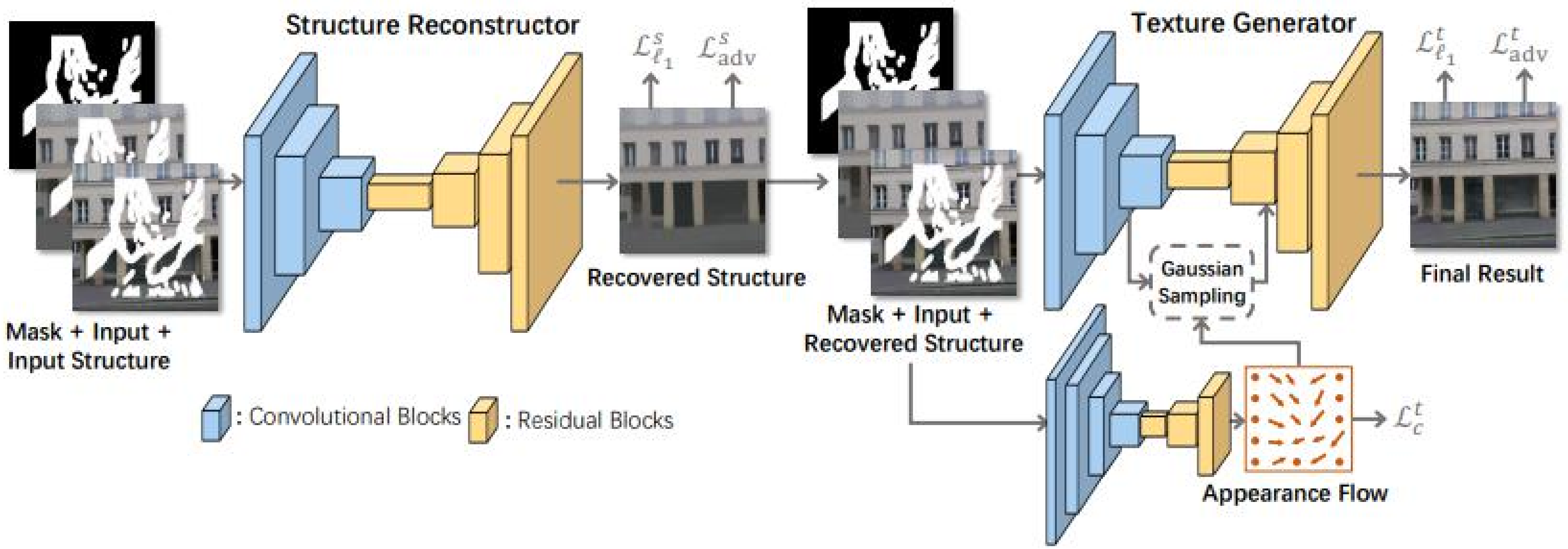


Figure 2. Overview of our StructureFlow. Our model first generates global structures (*i.e.* edge-preserved smooth images) using structure reconstructor. Then texture generator is used to yield high-frequency details and output the final results. We add the appearance flow to our texture generator to sample features from existing regions.

Ren Y, Yu X, Zhang R, et al. Structureflow: Image inpainting via structure-aware appearance flow[C]//Proceedings of the IEEE/CVF International Conference on Computer Vision. 2019: 181-190.

相关方法

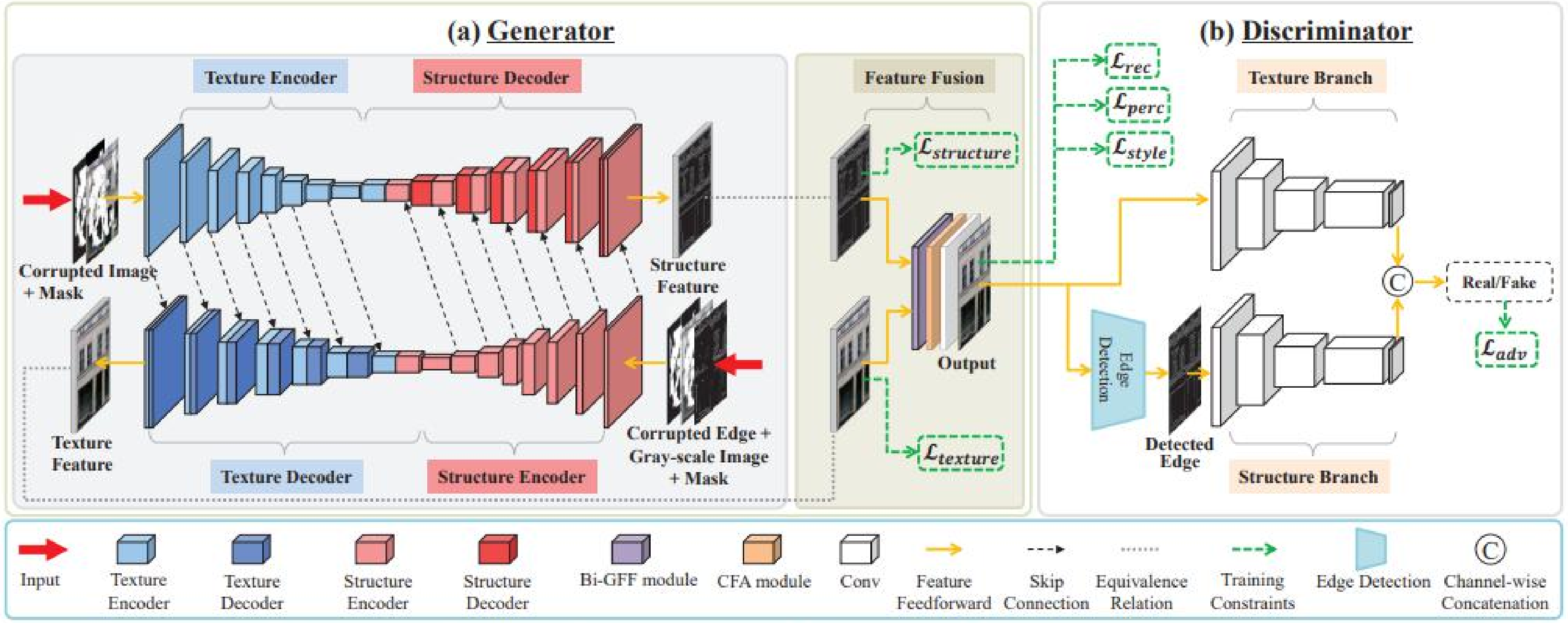


Figure 2: Overview of the proposed method (best viewed in color). **Generator**: Image inpainting is cast into two subtasks, *i.e.*, *structure-constrained texture synthesis* (left, blue) and *texture-guided structure reconstruction* (right, red), and the two parallel-coupled streams borrow encoded deep features from each other. The Bi-directional Gated Feature Fusion (Bi-GFF) module and Contextual Feature Aggregation (CFA) module are stacked at the end of the generator to further refine the results. **Discriminator**: The image branch estimates the generated texture, while the edge branch guides structure reconstruction.

Guo X, Yang H, Huang D. Image inpainting via conditional texture and structure dual generation[C]//Proceedings of the IEEE/CVF International Conference on Computer Vision. 2021: 14134-14143.

基于MindSpore深度图像修复技术实践

相关方法

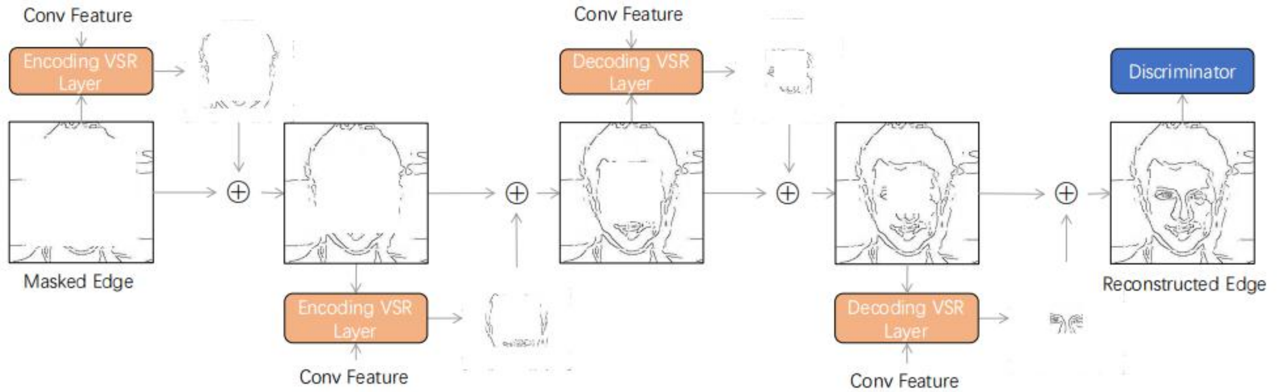
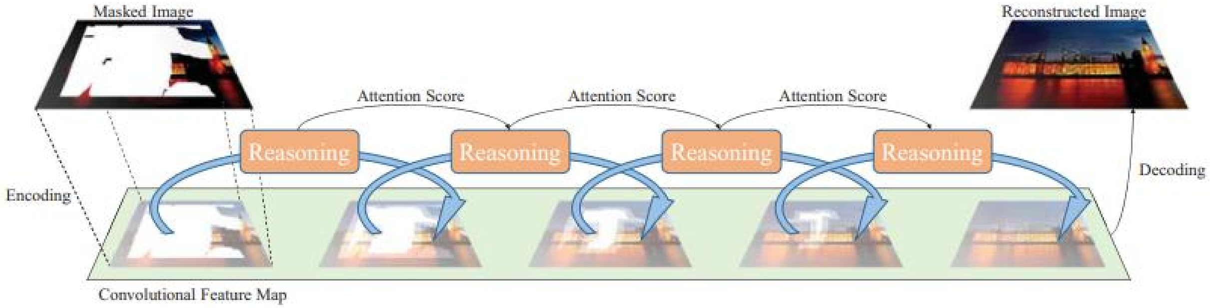


Figure 1: Progressive Reconstruction of Visual Structure. A small part of the new structure is produced in each VSR layer. At the beginning, the known information is limited and so the encoding layers only estimate the outer parts of the missing structure. As the information accumulates during the feeding forward procedure, the decoding layers can have the capability to restore the missing inner parts. The generated parts are collected and sent to discriminator simultaneously.

Progressive Reconstruction of Visual Structure for Image Inpainting

相关方法



The overview of our proposed inpainting scheme. The masked image is first mapped into the convolutional feature space and processed by a shared Feature Reasoning module recurrently. After the feature map is fully recovered, the generated feature maps are merged together (Omitted in this figure) and the merged feature is translated back to a RGB image.

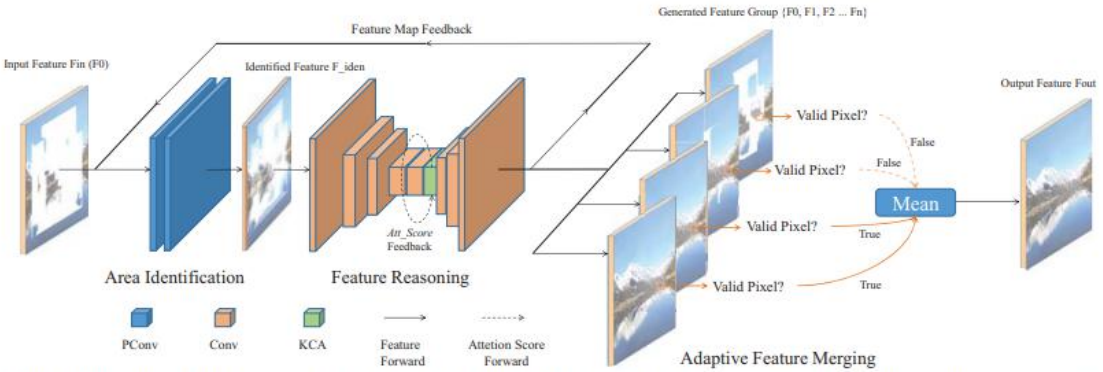
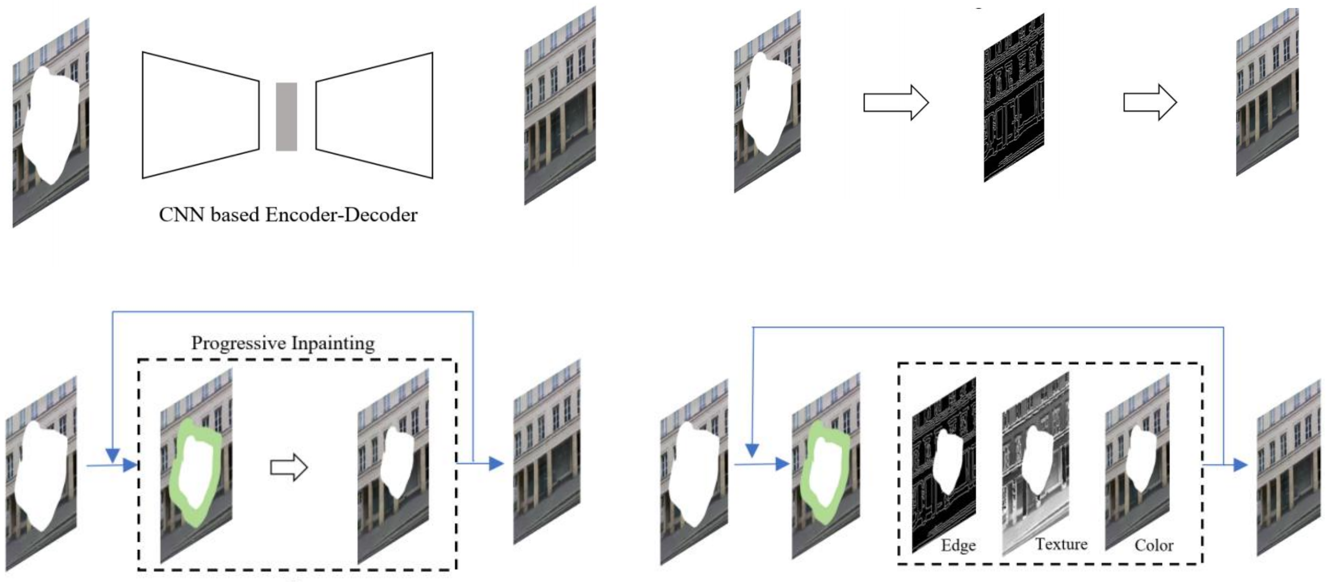


Figure 2. Illustration of the Recurrent Feature Reasoning module. The area identification process and the feature reasoning process are performed continuously. After several times of reasoning, the feature maps are merged in an adaptive fashion and an output feature map of a fixed channel numbers are generated. The module is Plug-In-and-Play and can be placed in any layer of an existing network.

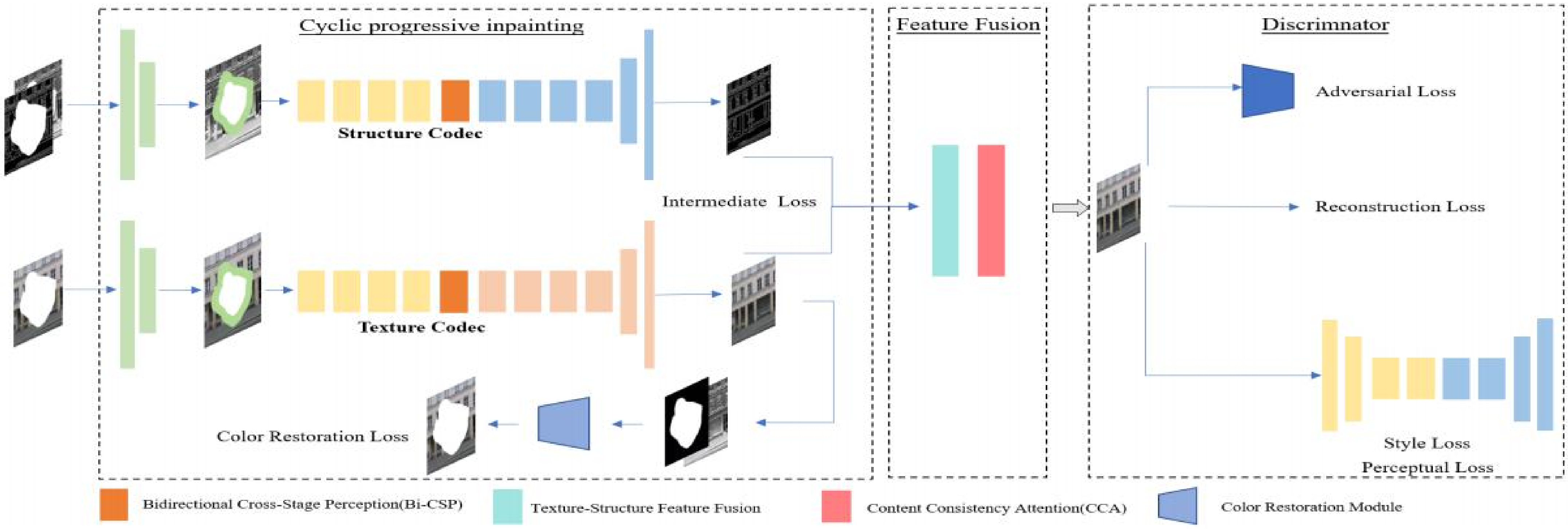
Li J, Wang N, Zhang L, et al. Recurrent feature reasoning for image inpainting[C]//Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition. 2020: 7760-7768.

基于MindSpore深度图像修复技术实践

相关方法



基于MindSpore深度图像修复技术实践



实现

Partial Convolution

普通卷积作用在图片的损坏区域时，大多数的计算都被浪费了，因为损坏区域的像素值为0。
同时，卷积核在做计算时不能区分损坏和未损坏的区域，对两部分的信息差并不敏感。

每一层PConv的运算可以用以下公式来表达:

$$x' = \begin{cases} \mathbf{W}^T(\mathbf{X} \odot \mathbf{M}) \frac{\mathrm{sum}(\mathbf{1})}{\mathrm{sum}(\mathbf{M})} + b, & \text{if } \mathrm{sum}(\mathbf{M}) > 0 \\ 0, & \text{otherwise} \end{cases}$$

$$m' = \begin{cases} 1, & \text{if } \mathrm{sum}(\mathbf{M}) > 0 \\ 0, & \text{otherwise} \end{cases}$$

Guilin Liu, Fitsum A. Reda, Kevin J. Shih, Ting-Chun Wang, Andrew Tao, Bryan Catanzaro; Proceedings of the European Conference on Computer Vision (ECCV), 2018, pp. 85-100

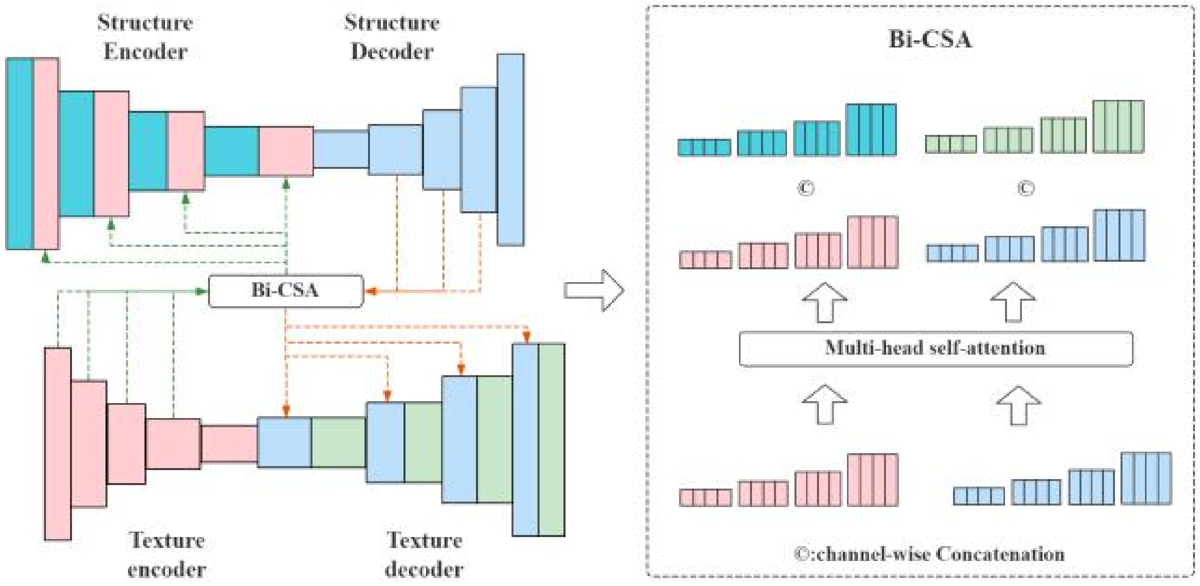
```python
from mindspore.common.initializer import Initializer, One
from mindspore import nn, ops, Tensor, context
from mindspore import dtype as mstype

class PartialConv2d(nn.Conv2d):
    def __init__(self, *args, **kwargs):
        super(PartialConv2d, self).__init__(*args, **kwargs)

        self.weight_maskUpdater = Tensor(
            shape=(self.out_channels, self.in_channels, self.kernel_size[0], self.kernel_size[1]), dtype=mstype.float32,
            init=One())
        self.slide_winsize = self.weight_maskUpdater.shape[1] * self.weight_maskUpdater.shape[2] * \
                             self.weight_maskUpdater.shape[3]
        self.last_size = (None, None)
        self.update_mask = None
        self.mask_ratio = None

    def construct(self, input, mask=None):
        if mask is not None or self.last_size != (input.data.shape[2], input.data.shape[3]):
            self.last_size = (input.data.shape[2], input.data.shape[3])
            if self.weight_maskUpdater.type() != input.type():
                self.weight_maskUpdater = self.weight_maskUpdater.to(input)
            if mask is None:
                mask = Tensor(shape=(input.data.shape[0], input.data.shape[1], input.data.shape[2],
                                     input.data.shape[3]), dtype=mstype.float32,
                              init=One()).to_tensor(input)
            self.update_mask = nn.Conv2d(mask, self.weight_maskUpdater, bias=None, stride=self.stride,
                                         padding=self.padding, dilation=self.dilation, groups=1)
            self.mask_ratio = self.slide_winsize / (self.update_mask + 1e-8)
            self.mask_ratio = ops.mul(self.mask_ratio, self.update_mask)

        if self.update_mask.type() != input.type() or self.mask_ratio.type() != input.type():
            self.update_mask.to_tensor(input)
            self.mask_ratio.to_tensor(input)
        raw_out = super(PartialConv2d, self).construct(ops.mul(input, mask) if mask is not None else input)

        output = ops.mul(raw_out, self.mask_ratio)
        if self.return_mask:
            return output, self.update_mask
        else:
            return output
```

实现

```python
class Generator(nn.cell):
    def __init__(self, image_in_channels=3, edge_in_channels=2, out_channels=3, init_weights=True):
        super(Generator, self).__init__()
        self.freeze_ec_bn = False
        # -----------------------
        # texture encoder-decoder
        # -----------------------
        self.ec_texture_1 = PConvBNActiv(image_in_channels, 64, bn=False, sample='down-7')
        self.ec_texture_2 = PConvBNActiv(64, 128, sample='down-5')
        self.ec_texture_3 = PConvBNActiv(128, 256, sample='down-5')
        self.ec_texture_4 = PConvBNActiv(256, 512, sample='down-3')
        self.ec_texture_5 = PConvBNActiv(512, 512, sample='down-3')
        self.ec_texture_6 = PConvBNActiv(512, 512, sample='down-3')
        self.ec_texture_7 = PConvBNActiv(512, 512, sample='down-3')

        self.dc_texture_7 = PConvBNActiv(512 + 512, 512, activ='leaky')
        self.dc_texture_6 = PConvBNActiv(512 + 512, 512, activ='leaky')
        self.dc_texture_5 = PConvBNActiv(512 + 512, 512, activ='leaky')
        self.dc_texture_4 = PConvBNActiv(512 + 256, 256, activ='leaky')
        self.dc_texture_3 = PConvBNActiv(256 + 128, 128, activ='leaky')
        self.dc_texture_2 = PConvBNActiv(128 + 64, 64, activ='leaky')
        self.dc_texture_1 = PConvBNActiv(64 + out_channels, 64, activ='leaky')

        # -------------------------
        # structure encoder-decoder
        # -------------------------
        self.ec_structure_1 = PConvBNActiv(edge_in_channels, 64, bn=False, sample='down-7')
        self.ec_structure_2 = PConvBNActiv(64, 128, sample='down-5')
        self.ec_structure_3 = PConvBNActiv(128, 256, sample='down-5')
        self.ec_structure_4 = PConvBNActiv(256, 512, sample='down-3')
        self.ec_structure_5 = PConvBNActiv(512, 512, sample='down-3')
        self.ec_structure_6 = PConvBNActiv(512, 512, sample='down-3')
        self.ec_structure_7 = PConvBNActiv(512, 512, sample='down-3')

        self.dc_structure_7 = PConvBNActiv(512 + 512, 512, activ='leaky')
        self.dc_structure_6 = PConvBNActiv(512 + 512, 512, activ='leaky')
        self.dc_structure_5 = PConvBNActiv(512 + 512, 512, activ='leaky')
        self.dc_structure_4 = PConvBNActiv(512 + 256, 256, activ='leaky')
        self.dc_structure_3 = PConvBNActiv(256 + 128, 128, activ='leaky')
        self.dc_structure_2 = PConvBNActiv(128 + 64, 64, activ='leaky')
        self.dc_structure_1 = PConvBNActiv(64 + 2, 64, activ='leaky')
```

```python
def construct(self, input_image, input_edge, mask):
    ec_textures = {}
    ec_structures = {}
    input_texture_mask = ops.Concat((mask, mask, mask), axis=1)
    ec_textures['ec_t_0'], ec_textures['ec_t_masks_0'] = input_image, input_texture_mask
    ec_textures['ec_t_1'], ec_textures['ec_t_masks_1'] = self.ec_texture_1(ec_textures['ec_t_0'], ec_textures['ec_t_masks_0'])
    ec_textures['ec_t_2'], ec_textures['ec_t_masks_2'] = self.ec_texture_2(ec_textures['ec_t_1'], ec_textures['ec_t_masks_1'])
    ec_textures['ec_t_3'], ec_textures['ec_t_masks_3'] = self.ec_texture_3(ec_textures['ec_t_2'], ec_textures['ec_t_masks_2'])
    ec_textures['ec_t_4'], ec_textures['ec_t_masks_4'] = self.ec_texture_4(ec_textures['ec_t_3'], ec_textures['ec_t_masks_3'])
    ec_textures['ec_t_5'], ec_textures['ec_t_masks_5'] = self.ec_texture_5(ec_textures['ec_t_4'], ec_textures['ec_t_masks_4'])
    ec_textures['ec_t_6'], ec_textures['ec_t_masks_6'] = self.ec_texture_6(ec_textures['ec_t_5'], ec_textures['ec_t_masks_5'])
    ec_textures['ec_t_7'], ec_textures['ec_t_masks_7'] = self.ec_texture_7(ec_textures['ec_t_6'], ec_textures['ec_t_masks_6'])

    input_structure_mask = ops.Concat((mask, mask), axis=1)
    ec_structures['ec_s_0'], ec_structures['ec_s_masks_0'] = input_edge, input_structure_mask
    ec_structures['ec_s_1'], ec_structures['ec_s_masks_1'] = self.ec_structure_1(ec_structures['ec_s_0'], ec_structures['ec_s_masks_0'])
    ec_structures['ec_s_2'], ec_structures['ec_s_masks_2'] = self.ec_structure_2(ec_structures['ec_s_1'], ec_structures['ec_s_masks_1'])
    ec_structures['ec_s_3'], ec_structures['ec_s_masks_3'] = self.ec_structure_3(ec_structures['ec_s_2'], ec_structures['ec_s_masks_2'])
    ec_structures['ec_s_4'], ec_structures['ec_s_masks_4'] = self.ec_structure_4(ec_structures['ec_s_3'], ec_structures['ec_s_masks_3'])
    ec_structures['ec_s_5'], ec_structures['ec_s_masks_5'] = self.ec_structure_5(ec_structures['ec_s_4'], ec_structures['ec_s_masks_4'])
    ec_structures['ec_s_6'], ec_structures['ec_s_masks_6'] = self.ec_structure_6(ec_structures['ec_s_5'], ec_structures['ec_s_masks_5'])
    ec_structures['ec_s_7'], ec_structures['ec_s_masks_7'] = self.ec_structure_7(ec_structures['ec_s_6'], ec_structures['ec_s_masks_6'])

    dc_texture, dc_tecture_mask = ec_structures['ec_s_7'], ec_structures['ec_s_masks_7']
    for _ in range(7, 0, -1):
        ec_texture_skip = 'ec_t_{:d}'.format(_)
        ec_texture_masks_skip = 'ec_t_masks_{:d}'.format(_)
        dc_conv = 'dc_texture_{:d}'.format(_)
        dc_texture = ops.Concat(dc_texture, ec_textures[ec_texture_skip],axis=1)
        dc_tecture_mask = ops.Concat((dc_tecture_mask, ec_textures[ec_texture_masks_skip]), axis=1)
        dc_texture, dc_tecture_mask = getattr(self, dc_conv)(dc_texture, dc_tecture_mask)

    dc_structure, dc_structure_masks = ec_textures['ec_t_7'], ec_textures['ec_t_masks_7']
    for _ in range(7, 0, -1):

        ec_structure_skip = 'ec_s_{:d}'.format(_)
        ec_structure_masks_skip = 'ec_s_masks_{:d}'.format(_)
        dc_conv = 'dc_structure_{:d}'.format(_)

        dc_structure = ops.Concat((dc_structure, ec_structures[ec_structure_skip]), dim=1)
        dc_structure_masks = ops.Concat((dc_structure_masks, ec_structures[ec_structure_masks_skip]), dim=1)

        dc_structure, dc_structure_masks = getattr(self, dc_conv)(dc_structure, dc_structure_masks)

    # -------------------
    # Projection Function
    # -------------------
    projected_image = self.texture_feature_projection(dc_texture)
    projected_edge = self.structure_feature_projection(dc_structure)
```

基于MindSpore深度图像修复技术实践



$$\hat{T}_i = \sigma(T_i) = \varphi(\alpha(\frac{W_{Q_i}^{\top} T_i^{\top} \cdot H_t W_K}{\sqrt{C_{\sum}}})) \cdot W_V^{\top} H^T,$$

$$\hat{S}_i = \sigma(S_i) = \varphi(\alpha(\frac{W_{Q_i}^{\top} S_i^{\top} \cdot H_s W_K}{\sqrt{C_{\sum}}})) \cdot W_V^{\top} H^T,$$

$$S_{dec}^i = Concat\left[\hat{T}_i; \hat{S}_i\right],$$

$$T_{dec}^i = Concat\left[\hat{S}_i; \hat{T}_i\right].$$

$$T = \left\{T^i \in \mathbb{R}^{\frac{W \times H}{i^2} \times C_i}, i = 1, \ldots, 4\right\}$$

$$S = \left\{S^i \in \mathbb{R}^{\frac{W \times H}{i^2} \times C_i}, i = 1, \ldots, 4\right\}$$

Dataset		Paris Street View			DTD			Places2		
Mask Ratio		0%-30%	30%-50%	50%-70%	0%-30%	30%-50%	50%-70%	0%-30%	30%-50%	50%-70%
SSIM	Patch Match	0.885	0.766	0.497	0.937	0.865	0.713	0.776	0.606	0.467
	Pconv	0.928	0.805	0.623	0.960	0.871	0.736	0.870	0.767	0.552
	Edge Connect	0.945	0.806	0.641	0.956	0.876	0.738	0.896	0.761	0.553
	RFR	0.939	0.810	0.653	0.945	0.862	0.741	0.915	0.782	0.555
	CTSDG	0.945	0.813	0.652	0.968	0.886	0.742	0.917	0.786	0.559
	Ours	**0.947**	**0.817**	**0.657**	**0.971**	**0.892**	**0.745**	**0.922**	**0.790**	**0.565**
PSNR	Patch Match	29.29	24.46	19.43	27.05	23.93	21.88	25.28	19.88	16.76
	Pconv	30.52	25.79	21.42	28.43	25.66	23.56	26.76	21.79	18.41
	Edge Connect	31.03	25.71	21.58	27.75	25.82	22.57	26.33	21.45	18.37
	RFR	31.22	25.80	21.67	27.51	25.71	22.88	27.45	22.55	18.84
	CTSDG	31.39	25.83	21.65	28.54	25.97	22.90	27.49	22.63	18.96
	Ours	**31.47**	**26.12**	**21.85**	**28.95**	**26.47**	**23.22**	**27.73**	**22.82**	**19.29**

$$\mathcal{L}_{rec} = \|I_{out} - I_{gt}\|_1$$

$$\mathcal{L}_{adv} = \min_G \max_D \mathbb{E}_{I_{gt},S_{gt}}[log D(I_{gt}, S_{gt})] + \mathbb{E}_{I_{out},S_{out}} log[1 - D(I_{out}, S_{out})]$$

$$\mathcal{L}_{col} = \left\|(\hat{I}_{col} - I_{gt}) \odot M\right\|_1$$

$$\mathcal{L}_{final} = \lambda_{inter}\mathcal{L}_{inter} + \lambda_{rec}\mathcal{L}_{rec} + \lambda_{perc}\mathcal{L}_{perc} + \lambda_{style}\mathcal{L}_{style} + \lambda_{adv}\mathcal{L}_{adv} + \lambda_{col}\mathcal{L}_{col}$$

MindSpore

一、获取安装命令

查看所有版本和接口变更 >

版本	1.7.0	1.6.2	Nightly		
硬件平台	Ascend 910	Ascend 310	GPU CUDA 10.1	GPU CUDA 11.1	CPU
操作系统	Linux-aarch64	Linux-x86_64	Windows-x64	MacOS-aarch64	MacOS-x86_64
编程语言	Python 3.7	Python 3.8	Python 3.9		
安装方式	Pip	Conda	Source	Docker	Binary

安装命令

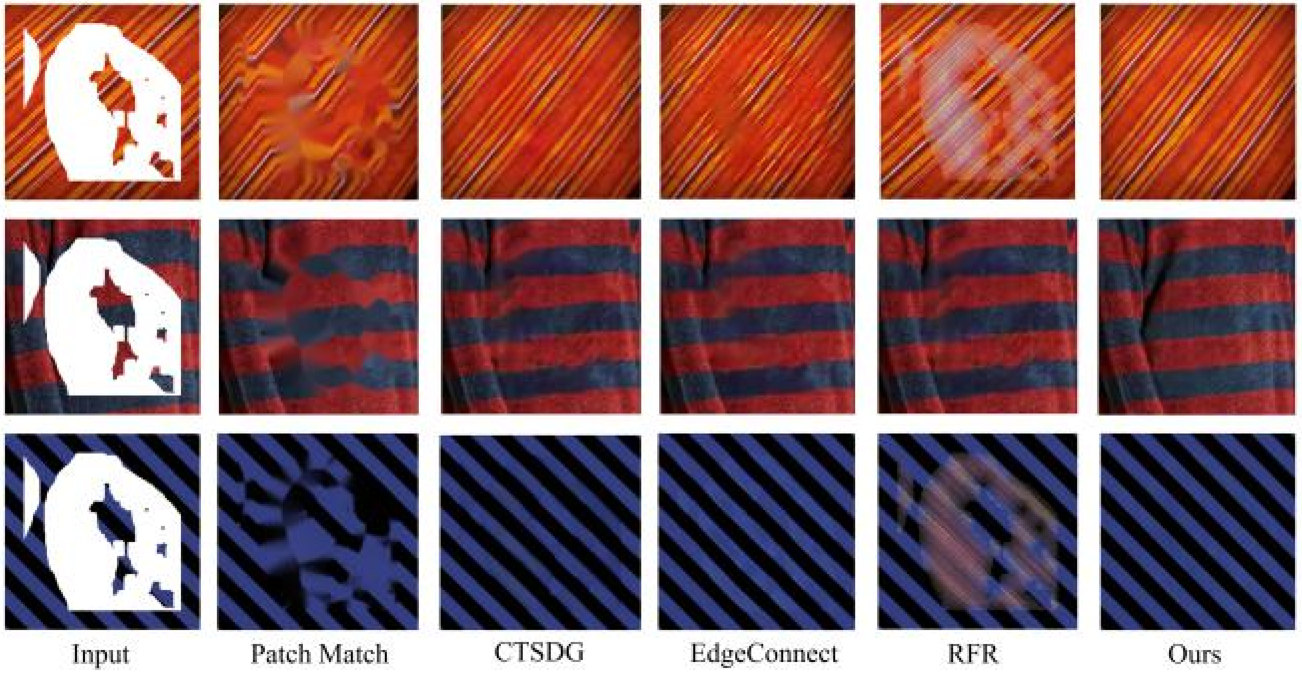
```
conda install mindspore-gpu=1.7.0 cudatoolkit=11.1 -c mindspore -c conda-forge
# 注意参考下方安装指南，添加运行所需的环境变量配置
```

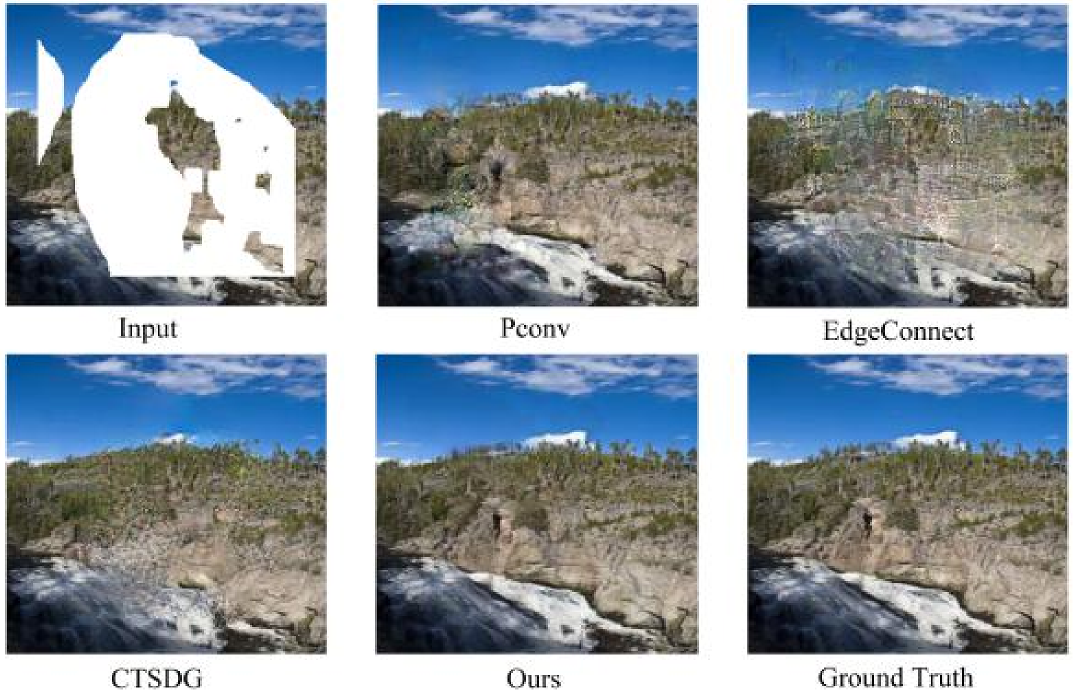
在使用自动安装脚本之前，需要确保系统正确安装了NVIDIA GPU驱动。CUDA 10.1要求最低显卡驱动版本为418.39；CUDA 11.1要求最低显卡驱动版本为450.80.02。执行以下指令检查驱动版本。

基于MindSpore深度图像修复技术实践

Input　Pconv　CTSDG　EdgeConnect　RFR　Ours

基于MindSpore深度图像修复技术实践



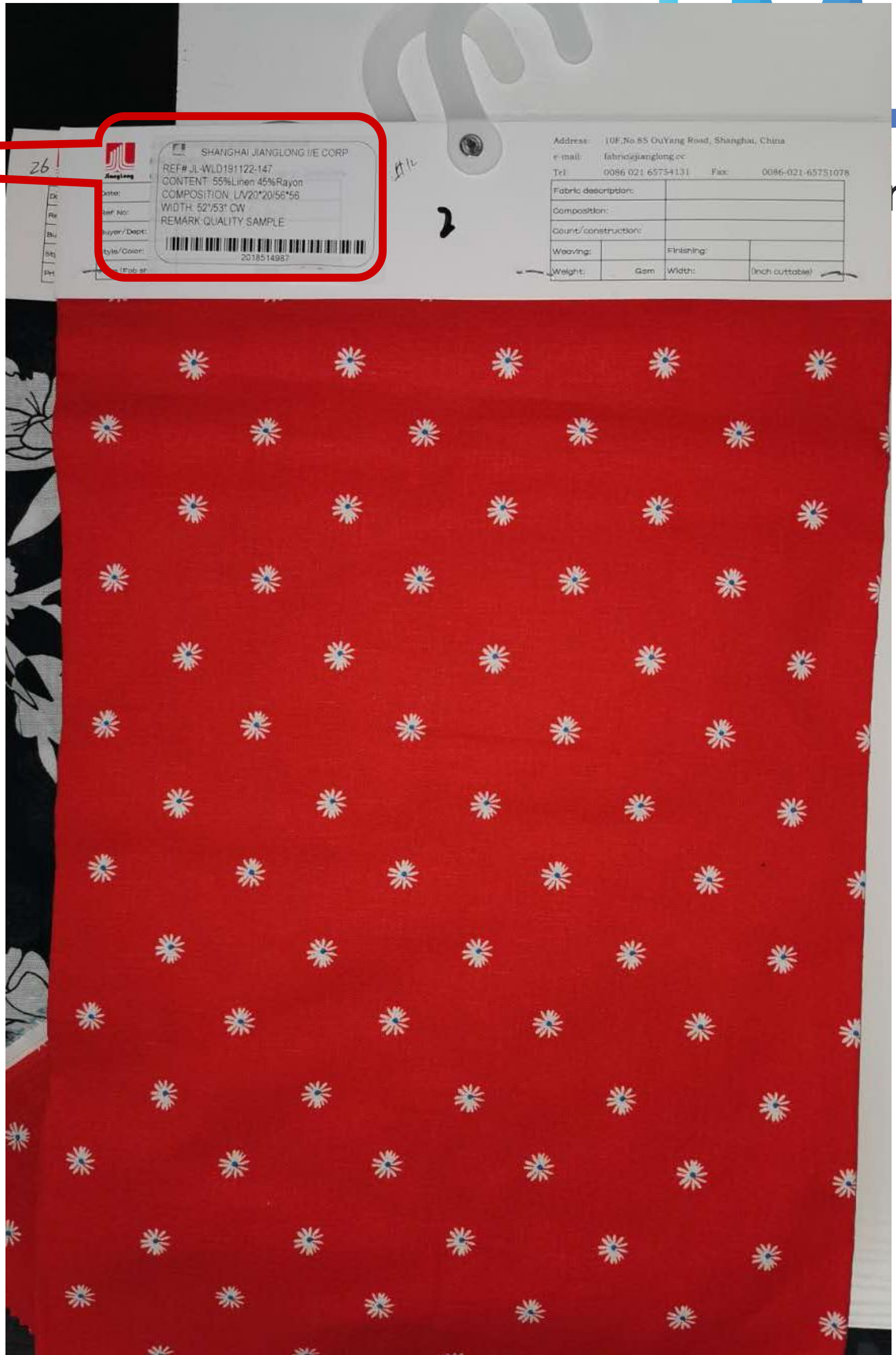


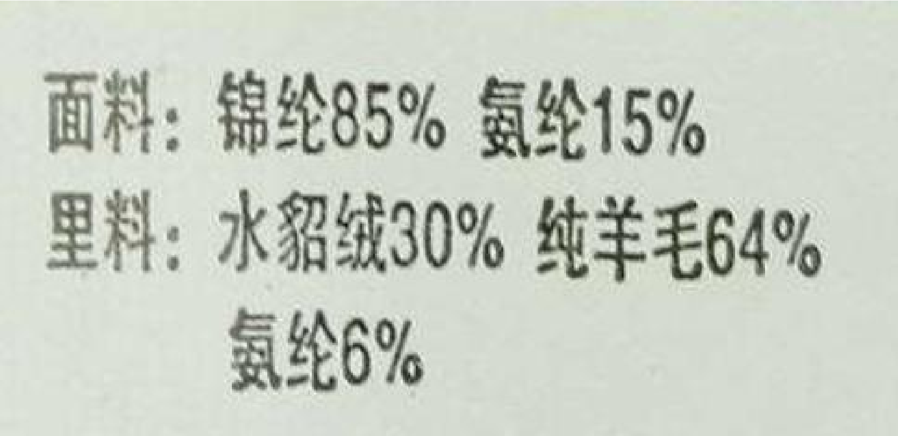
基于MindSpore深度图像修复技术实践

Input　　w/o CSP&CCA　　w/o Colorization　　Ours

	w/o CPI	w/o Bi-CSP	w/o CCA	PMPN(ours)
SSIM	0.866	0.870	0.885	0.892
PSNR	25.98	26.02	26.19	26.47

基于MindSpore的多模态技术在纺织工业应用





基于MindSpore的多模态技术在纺织工业应用

MindSpore

检测效率低下
检测3-5个工作日
棉麻成分不容易分辨

存在环境污染
化学溶解法
破坏样本
对环境存在污染

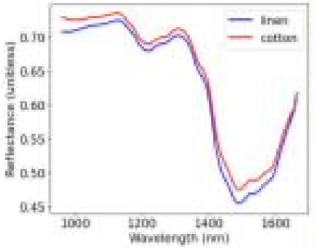
高成本
时间成本
费用成本

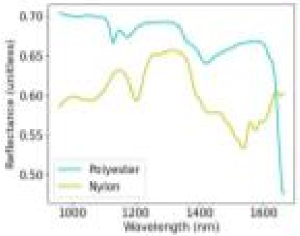
信息化集成水平低
纸质简单记录疵点
难以统一管理
质量问题难以追溯

基于MindSpore的多模态技术在纺织工业应用

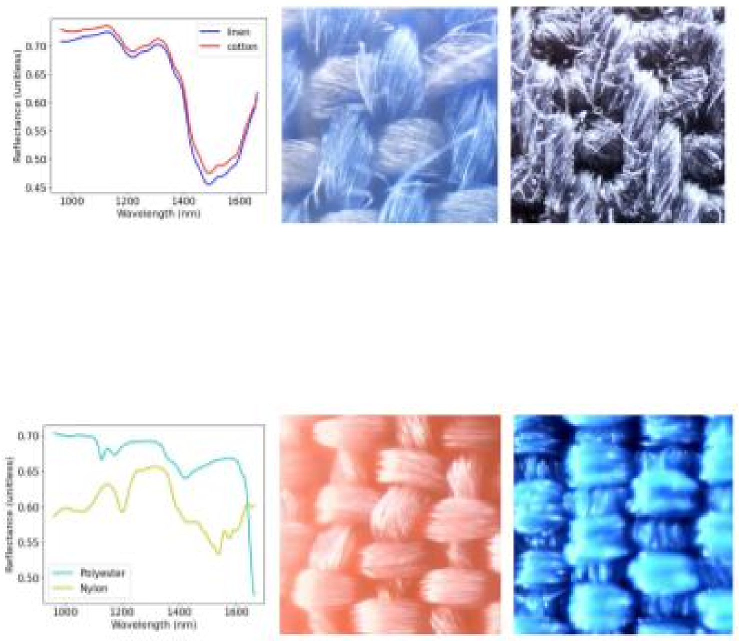
MindSpore

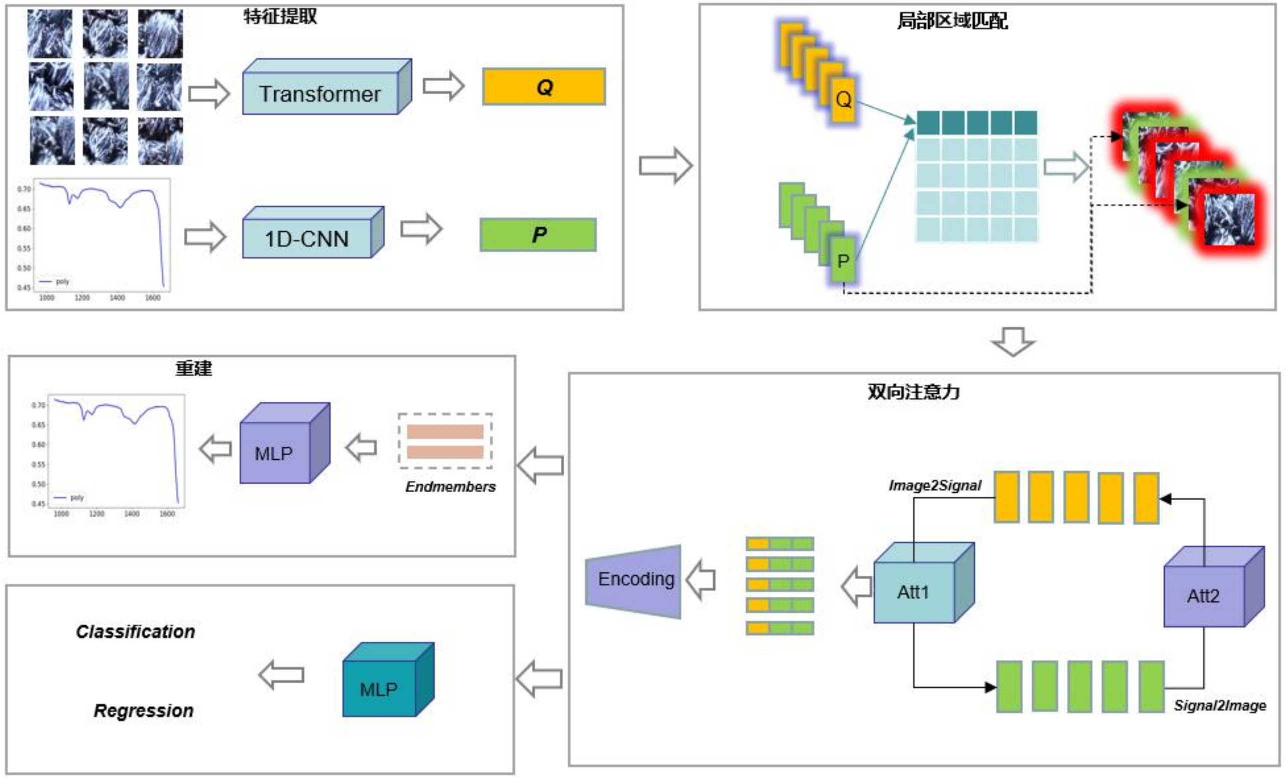
Cotton Linen Poly Nylon Viscose





基于MindSpore的多模态技术在纺织工业应用





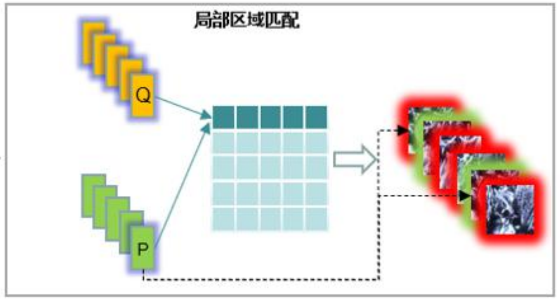
基于MindSpore的多模态技术在纺织工业应用

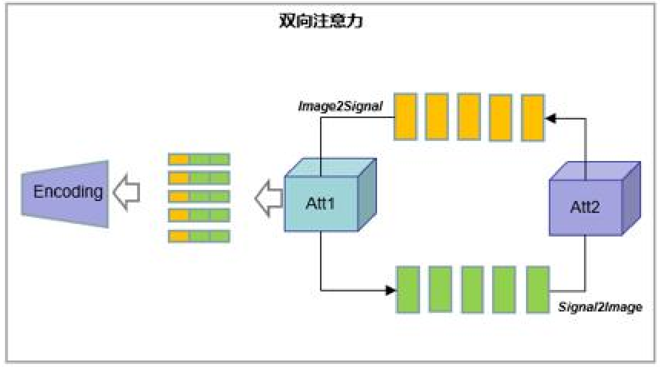
```python
from mindspore import nn
class Attention(nn.Cell):
    def __init__(self,
                 dim: int,
                 num_heads: int = 8,
                 keep_prob: float = 1.0,
                 attention_keep_prob: float = 1.0):
        super(Attention, self).__init__()
        self.num_heads = num_heads
        head_dim = dim // num_heads
        self.scale = Tensor(head_dim ** -0.5)
        self.qkv = nn.Dense(dim, dim * 3)
        self.attn_drop = nn.Dropout(attention_keep_prob)
        self.out = nn.Dense(dim, dim)
        self.out_drop = nn.Dropout(keep_prob)
        self.mul = ops.Mul()
        self.reshape = ops.Reshape()
        self.transpose = ops.Transpose()
        self.unstack = ops.Unstack(axis=0)
        self.attn_matmul_v = ops.BatchMatMul()
        self.q_matmul_k = ops.BatchMatMul(transpose_b=True)
        self.softmax = nn.Softmax(axis=-1)

    def construct(self, x):
        """Attention construct."""
        b, n, c = x.shape
        qkv = self.qkv(x)
        qkv = self.reshape(qkv, (b, n, 3, self.num_heads, c // self.num_heads))
        qkv = self.transpose(qkv, (2, 0, 3, 1, 4))
        q, k, v = self.unstack(qkv)
        attn = self.q_matmul_k(q, k)
        attn = self.mul(attn, self.scale)
        attn = self.softmax(attn)
        attn = self.attn_drop(attn)
        out = self.attn_matmul_v(attn, v)
        out = self.transpose(out, (0, 2, 1, 3))
        out = self.reshape(out, (b, n, c))
        out = self.out(out)
        out = self.out_drop(out)
        return out
```

```python
class TransformerEncoder(nn.Cell):
    def __init__(self,
                 dim: int,
                 num_layers: int,
                 num_heads: int,
                 mlp_dim: int,
                 keep_prob: float = 1.,
                 attention_keep_prob: float = 1.0,
                 activation: nn.Cell = nn.GELU,
                 norm: nn.Cell = nn.LayerNorm):
        super(TransformerEncoder, self).__init__()
        layers = []
        for _ in range(num_layers):
            normalization1 = norm((dim,))
            normalization2 = norm((dim,))
            attention = Attention(dim=dim,
                                  num_heads=num_heads,
                                  keep_prob=keep_prob,
                                  attention_keep_prob=attention_keep_prob)
            feedforward = FeedForward(in_features=dim,
                                      hidden_features=mlp_dim,
                                      activation=activation,
                                      keep_prob=keep_prob)
            layers.append(
                nn.SequentialCell([
                    nn.SequentialCell([normalization1, attention]),
                    nn.SequentialCell([normalization2, feedforward])
                ])
            )
        self.layers = nn.SequentialCell(layers)

    def construct(self, x):
        """Transformer construct."""
        return self.layers(x)
```

```python
def construct(self, x):
    x = self.patch_embedding(x)
    cls_tokens = self.tile(self.cls_token, (x.shape[0], 1, 1))
    x = self.concat((cls_tokens, x))
    x += self.pos_embedding
    x = self.pos_dropout(x)
    x = self.transformer(x)
    x = self.norm(x)
    return x
```

基于MindSpore的多模态技术在纺织工业应用

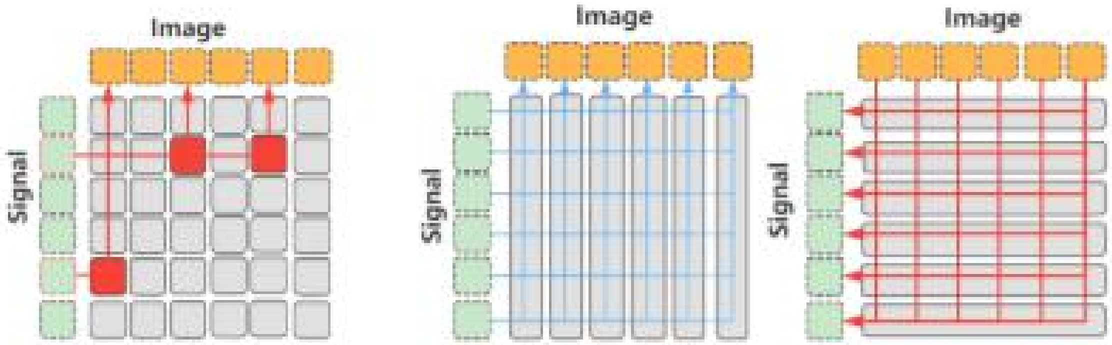




$$Q = f_{trans}(C) = W_2 \Phi_{ReLU}(W_1(C)),$$

$$SIM_{i,j} = \left\langle \frac{P_i}{||P_i||}, \frac{Q_j}{||Q_j||} \right\rangle.$$

$$Q_{rel} = [Q_1; Q_2; ...; Q_n] = F_{rel_n}(SIM, P_i)$$



(a) Candidate region generation (b) Image-Signal Correlation Attention

$$H = [P_i; P_i \odot Q_j], \qquad U = [Q_j; P_i \odot Q_j],$$

$$S_{i,j} = w_s H \in \mathbb{R}, \qquad M_{i,j} = w_m U \in \mathbb{R},$$

$$a_t = softmax(S_{t:}), \qquad b_t = softmax(M_{t:}).$$

$$\widehat{Q}_{:t} = \sum_i a_{ti} Q_{:i}, \qquad \widehat{P}_{:t} = \sum_j b_{tj} P_{:j}.$$

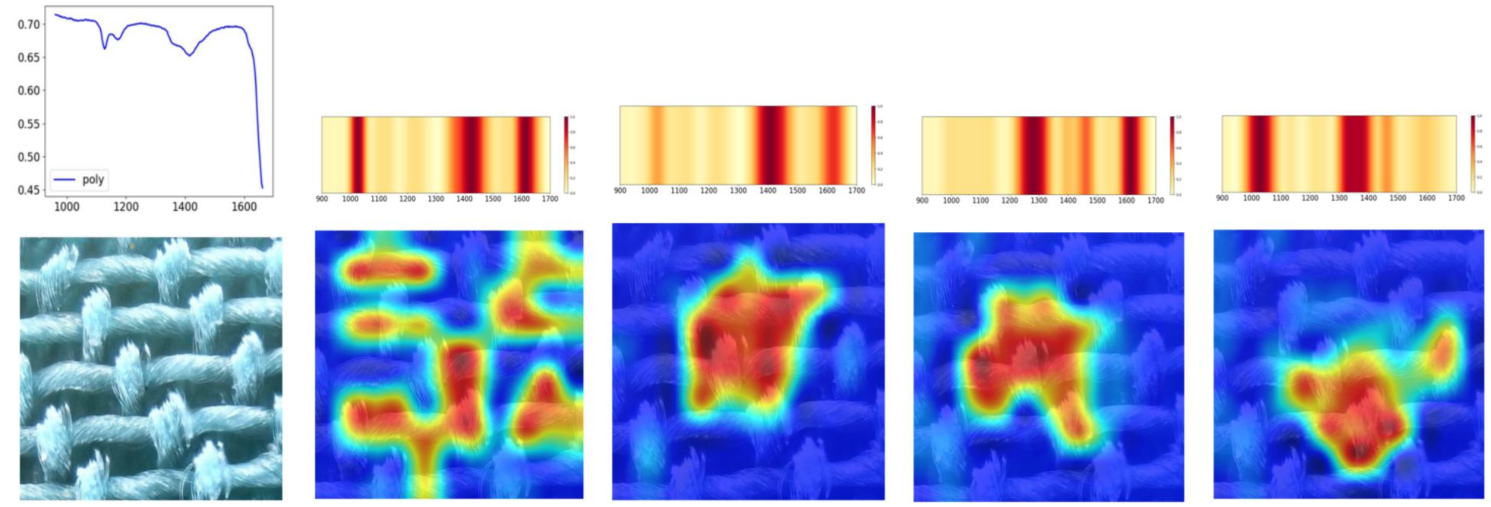
基于MindSpore的多模态技术在纺织工业应用

Modalities	Model	Classification (acc %)	Regression for principal composition (MAE)				
			Cotton	Linen	Nylon	Poly	Viscose
NIR signals	SVM	79.5	0.134	0.112	0.085	0.107	0.095
	LSTM	85.3	0.082	0.106	0.079	0.104	0.097
	CSI-Net	90.5	0.073	0.093	0.08	0.089	0.091
	MCNN	88.4	0.077	0.097	0.084	0.087	0.096
	DBLSTM-WS	87.1	0.074	0.09	0.081	0.085	0.086
	SSC-Net	88	0.08	0.101	0.093	0.088	0.091
Microscopic images	ViT	78.4	0.124	0.127	0.117	0.124	0.115
	CU-Net	78.4	0.122	0.121	0.115	0.118	0.109
	DeepTEN	83.5	0.105	0.114	0.112	0.109	0.12
	DEP	84.1	0.103	0.112	0.112	0.103	0.101
bimodal	M-Transformer	91.3	0.077	0.091	0.072	0.071	0.08
	BIDAF	89.8	0.081	0.093	0.075	0.101	0.086
	MBT	90.5	0.081	0.089	0.074	0.073	0.079
	SL-AE	92.0	0.086	0.087	0.076	0.069	0.073
	ISiC-Net (ours)	**95.2**	**0.056**	**0.064**	**0.067**	**0.055**	**0.063**

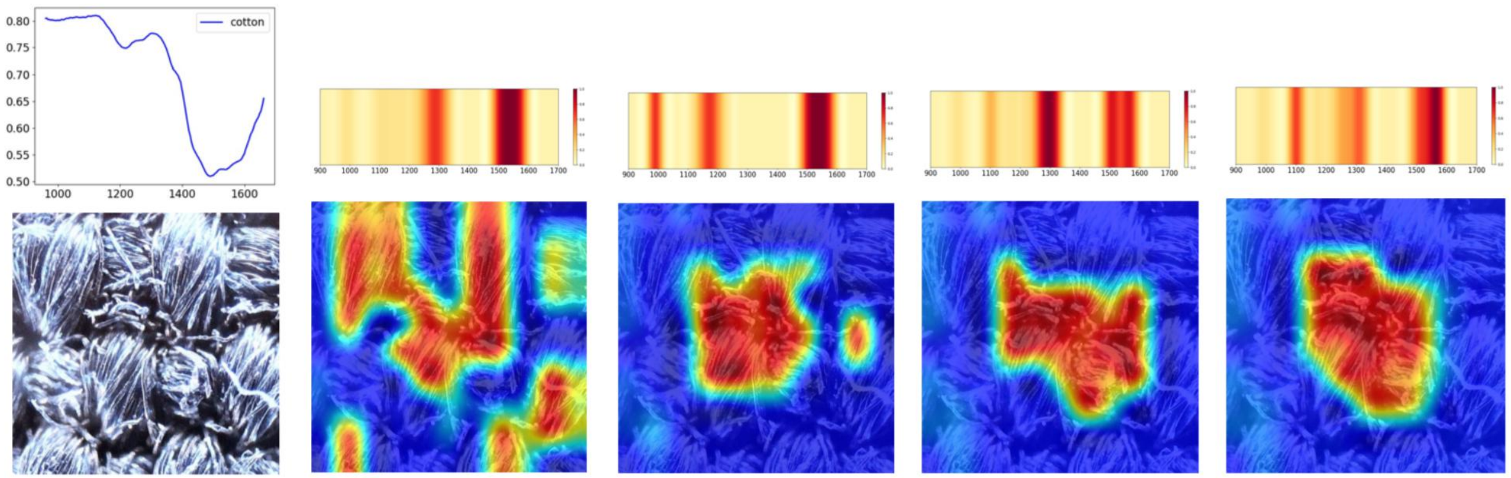
Method	Linen/Cotton	Cotton/Poly	Poly/Nylon	Nylon/Viscose
LSSVM	0.056	0.053	0.061	0.072
Random Forest	0.051	0.059	0.063	0.074
LR	0.049	0.053	0.06	0.076
PLS	0.047	0.05	0.057	0.068
CSI-Net	0.049	0.046	0.053	0.058
MCNN	0.044	0.042	0.051	0.056
M-Transformer	0.034	0.032	0.039	0.044
BIDAF	0.039	0.038	0.042	0.042
MBT	0.035	0.029	0.035	0.053
SL-AE	0.032	0.031	0.035	0.047
ISiC-Net (ours)	**0.027**	**0.022**	**0.032**	**0.036**

基于MindSpore的多模态技术在纺织工业应用

MindSpore



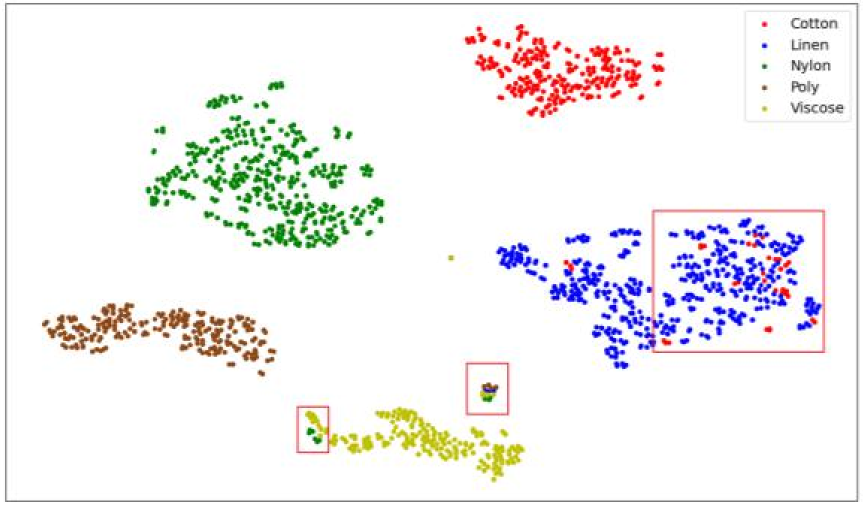
(a) Image (b) Ours (c) M-Trans (d) Bi-DAF (e) SL-AE

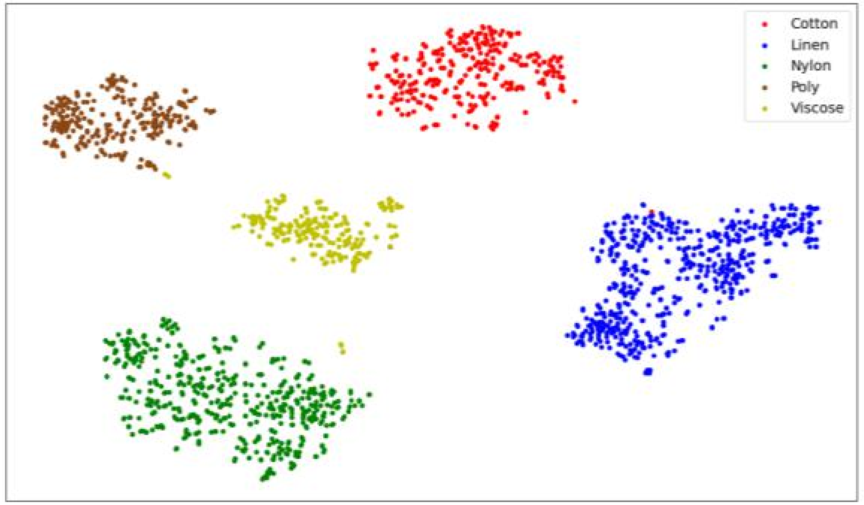


(a) Image (b) Ours (c) M-Trans (d) Bi-DAF (e) SL-AE

基于MindSpore的多模态技术在纺织工业应用

Method	Linen/Cotton	Cotton/Poly	Poly/Nylon	Nylon/Viscose	Classification accuracy(%)
ISiC-Net(Signal)	0.048	0.046	0.053	0.059	88.9
ISiC-Net(Signal+Image)	0.043	0.045	0.055	0.056	86.1
ISiC-Net(Signal+Image+RG)	0.034	0.032	0.041	0.045	90.3
ISiC-Net(Signal+Image+RG+ISiCA)	0.029	0.027	0.034	0.038	94.3
ISiC-Net(Signal+Image+RG+ISiCA+REC)	0.027	0.022	0.032	0.036	95.2





THANK YOU