CMT-Co: Contrastive Learning with Character Movement Task for Handwritten Text Recognition

Xiaoyi Zhang[1][0000−0002−5345−2110], Jiapeng Wang[1][0000−0002−2060−3488], Lianwen Jin[1,2⋆][0000−0002−5456−0957], Yujin Ren[1][0000−0001−9647−6713], and Yang Xue[1][0000−0002−1947−4957]

[1] South China University of Technology, Guangzhou, China
[2] SCUT-Zhuhai Institute of Modern Industrial Innovation, Zhuhai, China
{xy_zhang08,scutjpwang}@foxmail.com,
{lianwen.jin, yujinren98}@gmail.com, yxue@scut.edu.cn

A APPENDIX

In this appendix, we first give the pseudo-code and more examples of Text-Aug in Section A.1. Then, the detailed architecture of ResNet29 which is the backbone of CMT-Co is shown in Section A.2. Finally, we present the results of the ablation experiments for serialized instances in contrastive learning in Section A.3.

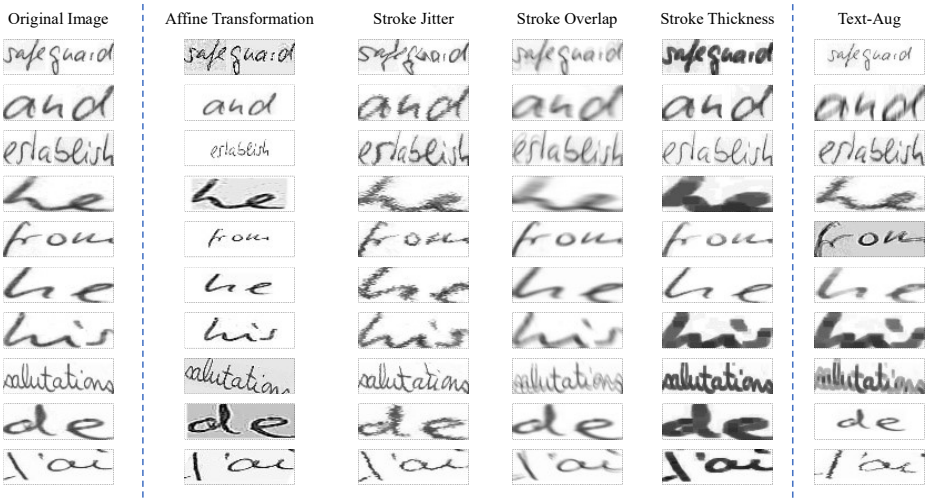

Fig. 1. Examples of various aspects in Text-Aug.

⋆ Corresponding author.

A.1 Text-Aug

Text-Aug is a data augmentation strategy suitable for handwritten text. It includes four aspects: affine transformation, stroke jitter, stroke overlap, and stroke thickness. More examples of Text-Aug are shown in Figure 1. The detailed pseudo-code is shown below. Here we adopt the `imgaug`, `numpy`, and `cv2` packages to implement it.

```
from imgaug import augmenters as iaa
import numpy as np
import cv2

def img_erode(images,random_state,parents,hooks):
    kernel = np.ones((4,4),np.uint8)
    f=lambda img:cv2.erode(img,kernel,iterations=2)
    images=[f(img)[:,:,np.newaxis] for img in images]
    return images
def img_dilate(images,random_state,parents,hooks):
    kernel = np.ones((2,2),np.uint8)
    f=lambda img:cv2.dilate(img,kernel,iterations=1)
    images=[f(img)[:,:,np.newaxis] for img in images]
    return images
def img_opening(images,random_state,parents,hooks):
    kernel = np.ones((5,5),np.uint8)
    f=lambda img:cv2.morphologyEx(img,cv2.MORPH_OPEN,kernel)
    images=[f(img)[:,:,np.newaxis] for img in images]
    return images
def img_closing(images,random_state,parents,hooks):
    kernel = np.ones((4,4),np.uint8)
    f=lambda img:cv2.morphologyEx(img,cv2.MORPH_CLOSE,kernel)
    images=[f(img)[:,:,np.newaxis] for img in images]
    return images
text_aug=iaa.Sequential([
    iaa.SomeOf((2,4),[
        #1. Affine Transformation
        iaa.Affine(scale=(0.5,1.05),cval=255),
        iaa.Affine(translate_percent={"x": (-0.03,0.03),"y"
    :(-0.07,0.07)},cval=255),
        iaa.OneOf([
            iaa.Affine(rotate=(-3,3),cval=255),
            iaa.Affine(shear={"x": (-30,30),"y":(-10,10)},
    cval=255),
        ]),
        iaa.Sharpen(alpha=(0.3, 0.7), lightness=(0.6, 1.4)),

        #2. Stroke Jitter
        iaa.PiecewiseAffine(scale=(0.01, 0.05), mode='edge',
    nb_rows=(2,4), nb_cols=(4,8)),
        iaa.imgcorruptlike.ElasticTransform(severity=(1,2)),
        #3. Stroke Overlap
```

```
        iaa.BlendAlpha((0.0,0.5),iaa.Affine(rotate=(-8, 8),
    mode='edge'),per_channel=False),

        iaa.OneOf([
            #4. Stroke Thickness
            iaa.Lambda(img_erode),
            iaa.Lambda(img_dilate),
            iaa.SigmoidContrast(gain=(2, 8), cutoff=(0.4,
    0.6)),
            iaa.MedianBlur(k=(3, 7)),
            iaa.Lambda(img_opening),
            iaa.Lambda(img_closing),
            #3. Stroke Overlap
            iaa.MotionBlur(k=9),
            iaa.GaussianBlur(sigma=(0.0, 2.0)),
        ]),
    ],random_order=True),
    iaa.Resize({"height":32, "width":100}),
])
```

Table 1. Architecture of ResNet29. ks, c, s, and p represent the size of the convolution kernel, the dimension of the convolution kernel, stride, and padding, respectively. The output size is $height \times width$.

Layer	Output Size	Configuration		
		ks, c	s	p
Input	32×100	-	-	-
Conv1	32×100	$3 \times 3, 32$	(1,1)	(1,1)
Conv2	32×100	$3 \times 3, 64$	(1,1)	(1,1)
Maxpool1	16×50	2×2	(2,2)	(0,0)
Block1	16×50	$\begin{bmatrix} 3 \times 3, 128 \\ 3 \times 3, 128 \end{bmatrix} \times 1$	(1,1)	(1,1)
Conv3	16×50	$3 \times 3, 128$	(1,1)	(1,1)
Maxpool2	8×25	2×2	(2,2)	(0,0)
Block2	8×25	$\begin{bmatrix} 3 \times 3, 256 \\ 3 \times 3, 256 \end{bmatrix} \times 2$	(1,1)	(1,1)
Conv4	8×25	$3 \times 3, 256$	(1,1)	(1,1)
Maxpool3	4×26	2×2	(2,1)	(0,1)
Block3	4×26	$\begin{bmatrix} 3 \times 3, 512 \\ 3 \times 3, 512 \end{bmatrix} \times 5$	(1,1)	(1,1)
Conv5	4×26	$3 \times 3, 512$	(1,1)	(1,1)
Block4	4×26	$\begin{bmatrix} 3 \times 3, 512 \\ 3 \times 3, 512 \end{bmatrix} \times 3$	(1,1)	(1,1)
Conv6	2×27	$2 \times 2, 512$	(2,1)	(0,1)
Conv7	1×26	$2 \times 2, 512$	(1,1)	(0,0)

A.2 ResNet29

In CMT-Co, both the momentum encoder and the encoder are ResNet29. The detailed architecture of ResNet29 is shown in Table 1.

A.3 Ablation Experiment of Serialization Instance

SeqCLR [1] performs contrastive learning after mapping a sequential feature map to instances. There are three types of instance-mapping functions: all-to-instance, window-to-instance, and frame-to-instance. The all-to-instance method treats all frames of a sequential feature map as a single instance for contrastive learning. The window-to-instance approach treats fixed-window frames of a sequential feature map as a single instance for contrastive learning. The frame-to-instance approach treats each frame as a single instance for contrastive learning. Our method CMT-Co uses the whole word image as a single instance in contrastive learning, which is equivalent to the all-to-instance method in SeqCLR.

To verify the effect of different serialization instance methods in CMT-Co, we conduct some ablation experiments according to the instance-mapping method of SeqCLR, as shown in Table 2. Note that the frame-to-instance method has the same settings as in SeqCLR, which eventually transforms the sequential feature map into T instances ($T = 5$). As can be seen from Table 2, in the framework of CMT-Co, taking the whole word image as a single instance works best. Because this allows the model to learn the semantic information of the entire word, and then combine it with the character feature learning of CMT to get better results.

Table 2. The ablation for serialized instances in contrastive learning.

Method	All-to-instance	Window-to-instance	Frame-to-instance
Accuracy	**81.3**	80.9	81.1

References

1. Aberdam, A., Litman, R., Tsiper, S., Anschel, O., Slossberg, R., Mazor, S., Manmatha, R., Perona, P.: Sequence-to-sequence contrastive learning for text recognition. In: CVPR. pp. 15302–15312 (2021)