CalligraphyRecognizer

2023 年 8 月 23 日

[1]:

```
import pandas as pd
import random
import numpy as np
import matplotlib.pyplot as plt
from PIL import Image

import os
import pathlib
from pathlib import Path
from typing import Tuple, Dict, List

import torch
from torch.utils.data import Dataset
from torch.utils.data import DataLoader
from torch.utils.data import RandomSampler

import torchvision
from torchvision import transforms
from torchvision import datasets
from torchinfo import summary

from tqdm import tqdm
from timeit import default_timer as timer
```

0.0.1 设置好 device，以充分发挥 GPU 的计算优势，同时要兼容没有 GPU 的设备

[2]:
```
# 数据和模型都要加载到正确的设备上，否则会因不兼容而报错
device = "cuda" if torch.cuda.is_available() else "cpu"
device
```

[2]:
```
'cpu'
```

[3]:
```
# 设置数据文件夹
DATA_PATH = Path("data/")
IMAGE_PATH = DATA_PATH / "wordlib"    #
IMAGE_PATH_LIST = list(IMAGE_PATH.glob("*.gif"))

# 如果文件夹不存在，则创建一个...
if IMAGE_PATH.is_dir():
    print(f"{IMAGE_PATH} 文件夹存在，可以使用...")
else:
    print(f"{IMAGE_PATH}文件平不存在，创建中...")
    IMAGE_PATH.mkdir(parents=True, exist_ok=True)
```

```
data\wordlib 文件夹存在，可以使用…
```

0.0.2 准备数据，查找指定文件夹中包含哪些文字，并设置其 classes 和 labels

[4]:
```
# 查找指定文件夹中的 classes
def find_classes(directory: str,ext:str='gif') -> Tuple[List[str], Dict[str,␣
 ↪int],List[str]]:
    """ 根据指定文件夹下的图片文件名的第一名字形成类别 classes.

    书法图片文件命名规范为：字 _ 字体 _ 书法家 _ 文件编号.gif，如：予 _ 行书 _ 鲜于
枢 _12046.gif.

    Args:
        directory (str): target directory to load distinct words from.

    Returns:
```

```
        Tuple[List[str], Dict[str, int]]: (list_of_class_names, dict(class_name:
↪ idx...))

    Example:
        data\wordlib\予 _ 行书 _ 鲜于枢 _12046.gif 分割 _ 前面的字符是书法对应的文
字
        >>> ([" 予", " 大",...], {" 予": 203, ...})
    """
    # 1. 扫描路径下全部文件，通过文件名首字符为图片所对应的汉字这样的命名规则，得到
该路径下的全部汉字。
    image_path_list = list(pathlib.Path(directory).glob(f"*.{ext}"))
    image_classes_set = set()  # 因为相同的字有多张图，所以使用 set 集合去重
    images_classes_list=[]
    images_name_list=[]
    for  path in   image_path_list:
        image_classes_set.add(path.name.split('_')[0])
        images_name_list.append(path.name)
    classes=sorted([word for word in image_classes_set])

    # 2. 如果文件不存在或没有按要求命名，则报错
    if not classes:
        raise FileNotFoundError(f"{directory}路径下的文件可能不存在或没有按要求命
名（文件命名规则为 word_font_writer_number.gif)")

    # 3. 创建汉字列表及包含其序号的 dict
    class_to_idx=dict()
    for i,word in enumerate(classes):
        class_to_idx[word]=i

    return classes, class_to_idx, images_name_list
```

[5]:
```
## ，是模型训练的基础数据，重要，不要改动
images_classes_list,word_classes_dict,images_name_list=find_classes(IMAGE_PATH,'gif')
```

[8]:
```
print(f'文件夹{IMAGE_PATH}下有{len(images_classes_list)}个不同字的书法图片')
```

文件夹 data\wordlib 下有 483 个不同字的书法图片

[9]:
```
# 查找指定文件夹中的 writer_classes
def find_writer_classes(directory: str,ext:str='gif') -> Tuple[List[str],␣
↪Dict[str, int],List[str]]:
    """ 根据指定文件夹下的图片文件名的第一名字形成类别 classes.

    书法图片文件命名规范为：字 _ 字体 _ 书法家 _ 文件编号.gif，如：予 _ 行书 _ 鲜于
枢 _12046.gif.

    Args:
        directory (str): target directory to load distinct words from.

    Returns:
        Tuple[List[str], Dict[str, int]]: (list_of_class_names, dict(class_name:
↪ idx...))

    Example:
        data\wordlib\予 _ 行书 _ 鲜于枢 _12046.gif 最后一个分割符 _ 后面的字符是书
法对应的作者 writer
        >>> ([" 鲜于枢 ", " 王羲之 "], {" 王羲之 ": 266, ...})
    """
    # 1. 扫描路径下全部文件，通过文件名首字符为图片所对应的汉字这样的命名规则，得到
该路径下的全部汉字。
    image_path_list = list(pathlib.Path(directory).glob(f"*.{ext}"))
    image_writer_classes_set = set()  # 因为相同的字有多张图，所以使用 set 集合去
重
    images_writer_classes_list=[]
    images_writer_name_list=[]
    for  path in   image_path_list:
        image_writer_classes_set.add(path.name.split('_')[2])
        images_writer_name_list.append(path.name)
    writer_classes=sorted([word for word in image_writer_classes_set])

    # 2. 如果文件不存在或没有按要求命名，则报错
    if not writer_classes:
        raise FileNotFoundError(f"{directory}路径下的文件可能不存在或没有按要求命
名（文件命名规则为 word_font_writer_number.gif)")
```

```python
    # 3. 创建汉字列表及包含其序号的 dict
    writer_class_to_idx=dict()
    for i,word in enumerate(writer_classes):
        writer_class_to_idx[word]=i

    return writer_classes, writer_class_to_idx, images_writer_name_list
```

[10]:
```python
## ，是模型训练的基础数据，重要，不要改动
images_writer_classes_list,word_writer_classes_dict,images_writer_name_list=find_writer_classe
```

[11]:
```python
print(f'文件夹{IMAGE_PATH}下有{len(images_writer_classes_list)}个书法家的书法图
↪片')
```

```
文件夹 data\wordlib 下有 447 个书法家的书法图片
```

0.0.3 根据指定文件夹下的图片，生成文字列表，并以 Dict 保存每个文字的编号

[14]:
```python
# 以 DataFrame 形式保存字与 Label 的对应关系
df_word_label_map=pd.DataFrame.
  ↪from_dict(word_classes_dict,orient='index',columns=['label'])
df_word_label_map.reset_index(inplace=True)
df_word_label_map.columns=['word','label']
df_word_label_map.T
```

[14]:
```
        0    1    2    3    4    5    6    7    8    9   …  473  474  475  476  477  \
word    一   丁   七   万   丈   三   上   下   不   与  …    操    擞    擠    ␣
  ↪擢    擬
label   0    1    2    3    4    5    6    7    8    9   …  473  474  475  476  477

       478  479  480  481  482
word    擴    擾    攀    攘    攜
label  478  479  480  481  482

[2 rows x 483 columns]
```

```
[15]: # 以 DataFrame 形式保存字与 Label 的对应关系
df_word_writer_label_map=pd.DataFrame.
 ↪from_dict(word_writer_classes_dict,orient='index',columns=['label'])
df_word_writer_label_map.reset_index(inplace=True)
df_word_writer_label_map.columns=['word','label']
df_word_writer_label_map.T
```

```
[15]:      0   1    2    3   4    5   6    7   8    9  …  437  438  439  440  \
word    丰坊  乃贤  乔一琦  于文傅  于谦  井寂严  仲殊  任伯年  任询  伊秉绶  …  鲜
于枢  黄仲则  黄庭坚   黄慎
label    0   1    2    3   4    5   6    7   8    9  …  437  438  439  440

       441  442  443  444  445  446
word    黄潜   黄辉  黄道周   黎简  龚晴皋   龚贤
label  441  442  443  444  445  446

[2 rows x 447 columns]
```

0.0.4 定义函数 resolve_word_by_image_name，根据图片文件名找出对应的文字 (class)、标签 (Label)，并显示该文字图片

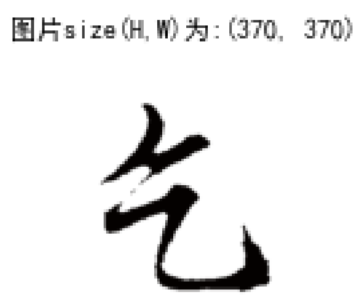
```
[16]: def␣
 ↪resolve_word_by_image_name(image_path,word_classes_dict,show=True)->(str,str,Image):
 ↪

    '''
    定义函数 resolve_word_by_image_name，根据图片文件名找出对应的文字 (class)、标
签 (Label)，并显示该文字图片
    Args:
        image_path (str): 文字图片路径和文件名.
        word_classes_dict (dict): 文字及标签的字典
        show (Boolean): 是否显示文字图片

    Returns:
        str,str: 文字 class, 文字 label
```

```
    Example:
        data\wordlib\予 _ 行书 _ 鲜于枢 _12046.gif "_" 前面的字符是书法对应的文字
        返回: " 予 ",203
    '''
    image_class = Path(str(image_path)).name.split('_')[0]
    image_label =word_classes_dict[image_class]
    print(f'图片{image_path}对应的文字是: {image_class}, 其 label 为:␣
↪{image_label}')


    with Image.open(image_path).convert('RGB') as f: #    丁 _ 草书 _ 王铎
_131029.gif data/wordlib/zxqsig.jpg
        if show:
            plt.figure(figsize=(2,2))
            plt.imshow(f)
            plt.title(f"图片 size(H,W) 为:({f.height}, {f.
↪width})",fontsize=16,fontproperties='Simhei')
            plt.axis(False)
    return image_class,image_label,f
```

[17]:

```
random_image_path = random.choice(IMAGE_PATH_LIST)
word,label,img=resolve_word_by_image_name(random_image_path,word_classes_dict,show=True)
```

图片 data\wordlib\乞 _ 行书 _ 苏轼 _11855.gif 对应的文字是: 乞, 其 label 为: 48



0.0.5 创建图片转换 Transform，将图片按某种效果进行变换

详见Pytorch 文档: ILLUSTRATION OF TRANSFORMS

[19]:
```
def␣
↪resolve_word_writer_by_image_name(image_path,word_writer_classes_dict,show=True)->(str,str,
↪

    '''
    定义函数 resolve_word_writer_by_image_name，根据图片文件名找出对应的文字作者
(class)、标签 (Label)，并显示该文字图片
    Args:
        image_path (str): 文字图片路径和文件名.
        word_writer_classes_dict (dict): 文字及标签的字典
        show (Boolean): 是否显示文字图片

    Returns:
        str,str: 作者 class, 文字作者 label

    Example:
        data\wordlib\予 _ 行书 _ 鲜于枢 _12046.gif "_" 前面的字符是书法对应的文字
        返回: " 鲜于枢",203
    '''
    image_writer_class = Path(str(image_path)).name.split('_')[2]
    image_writer_label =word_writer_classes_dict[image_writer_class]
    print(f'图片{image_path}对应的文字是: {image_writer_class}, 其 label 为:␣
↪{image_writer_label}')


    with Image.open(image_path).convert('RGB') as f: #    丁 _ 草书 _ 王铎
_131029.gif data/wordlib/zxqsig.jpg
        if show:
            plt.figure(figsize=(2,2))
            plt.imshow(f)
            plt.title(f"图片 size(H,W) 为:({f.height}, {f.
↪width})",fontsize=16,fontproperties='Simhei')
            plt.axis(False)
```

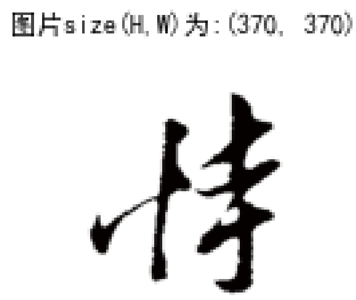
```
    return image_writer_class,image_writer_label,f
```

[23]:
```
word_writer_classes_dict['神宗']
```

[23]: 285

[29]:
```
random_image_path = random.choice(IMAGE_PATH_LIST)
word_writer,writer_label,img=resolve_word_writer_by_image_name(random_image_path,word_writer_c
```

图片 data\wordlib\恃 _ 行书 _ 姚绶 _27719.gif 对应的文字是：姚绶，其 label 为：70



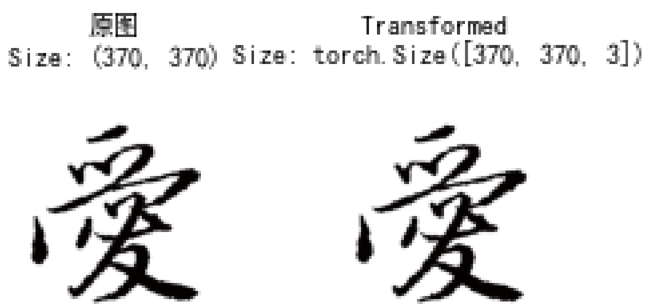
[30]:
```
# 定义 transform
# 转换效果及使用方法详见：https://pytorch.org/vision/stable/auto_examples/plot_transforms.html#sphx-glr-auto-examples-plot-transforms-py
aug_transform = transforms.Compose([
    transforms.Resize((370, 370)),
    #transforms.TrivialAugmentWide(num_magnitude_bins=31,fill=255), # how intense
    #transforms.ColorJitter(brightness=.5, hue=.3),
    transforms.RandomRotation(degrees=(-10, 10),expand=False,fill=255),
    #transforms.RandomAffine(degrees=(30, 70), translate=(0.1, 0.3), scale=(0.7, 0.9),fill=255),
    #transforms.ElasticTransform(alpha=250.0,fill=255),
    transforms.RandomPerspective(distortion_scale=0.1, p=0.2,fill=255),
    transforms.ToTensor() # use ToTensor() last to get everything between 0 & 1
])
```

[31]:
```
def␣
↪plot_one_transformed_image(image_path,transform=None,save=True,save_path='data/
↪augmented/'):

    '''
    show_transformed_image，根据图片文件名和 Transform，显示原图片和 Transformed
后的图片
    Args:
        image_path (str): 文字图片路径和文件名，如'data/wordlib/书 _ 行书 _ 王羲
之 _11946.gif'
        transform (torchvision.transforms): 效果转换器

    Returns:
        None

    '''
    with Image.open(image_path).convert('RGB') as f: #
        fig, ax = plt.subplots(figsize=(4,2))
        ax.axis(False)
        ax = fig.add_subplot(1,2,1)
        ax.imshow(f)
        ax.set_title(f"原图\nSize: {f.
↪size}",fontsize=16,fontproperties='Simhei')
        ax.axis("off")
        ax = fig.add_subplot(1,2,2)
        ax.axis(False)
        if transform is not None:
            transformed_image = transform(f).permute(1,2,0) # 如果只想看某一个
channel 的话，再接上 [:,:,0]
            if transformed_image.shape[2]==1:
                transformed_image=transformed_image.squeeze(2)
            ax.imshow(transformed_image)
            ax.set_title(f"Transformed \nSize: {transformed_image.
↪shape}",fontsize=16,fontproperties='Simhei')
            #fig.suptitle(f"{str(image_path).split('.
↪')[0]}",fontsize=16,fontproperties='Simhei')
```

```python
        if save:
            img=torchvision.transforms.ToPILImage()(transformed_image.permute(2,0,1))
            img_name=str(image_path).split('/')[-1]
            augmented_name=save_path+img_name.split('.')[0]+str(random.randint(100000,999999))+"_aug."+img_name.split('.')[-1]
            #print(augmented_name) # 输出保存的文件名
            img.save(augmented_name)
```

[32]:

```python
plot_one_transformed_image('data/wordlib/爱 _ 行书 _ 唐寅 _28699.gif',aug_transform)
```



[33]:

```python
def plot_transformed_images(image_paths, transform, n=3, seed=None,show=True,save=True,save_path='data/augmented/'):
    """Plots a series of random images from image_paths.

    Will open n image paths from image_paths, transform them
    with transform and plot them side by side.

    Args:
        image_paths (list): List of target image paths.
        transform (PyTorch Transforms): Transforms to apply to images.
        n (int, optional): Number of images to plot. Defaults to 3.
```

```
        seed (int, optional): Random seed for the random generator. Defaults to␣
↪42.
        save: save or not the transformed image file
        save_path: where to save the transformed image file
    """
    #random.seed(42)
    random_image_paths = random.sample(image_paths, k=n)
    for image_path in random_image_paths:
        try:
            with Image.open(image_path).convert('RGB') as f:
                # 转换并显示图片
                # Note: permute() 用于进行维度交换
                # (PyTorch default is [C, H, W] but Matplotlib is [H, W, C])
                transformed_image = transform(f).permute(1, 2, 0)
                if transformed_image.shape[2]==1:
                    transformed_image=transformed_image.squeeze(2)
                if save:
                    img=torchvision.transforms.ToPILImage()(transformed_image.
↪permute(2,0,1))
                    filename=image_path.name.split('.')[0]+'_'+str(random.
↪randint(100000,999999))+'_aug.'+image_path.name.split('.')[1]
                    #print(f'生成了新的增广变形文件 {filename}')
                    img.save(f'{save_path}/{filename}')

                if show:
                    fig, ax = plt.subplots(1, 2)
                    ax[0].imshow(f)
                    ax[0].set_title(f"Original \nSize: {f.size}")
                    ax[0].axis("off")

                    ax[1].imshow(transformed_image)
                    ax[1].set_title(f"Transformed \nSize: {transformed_image.
↪shape}")
                    ax[1].axis("off")
                    word_class=image_path.name.split('_')[0]
```

```
                    fig.suptitle(f"Class: {word_class}, label is :
↪{word_classes_dict[word_class]}", fontsize=16,fontproperties='Simhei')

        except:
            continue
```

0.0.6 从已有的图片中增广生成新图片并保存

[34]:
```
def␣
↪generate_augmented_images(k=2,size=4,image_path_list=None,aug_transform=None,save=True,show
↪augmented/')->None:
    """
    使用转换器随机生成增广图片并保存
    生成图片数量为：k*size

    Args:
        k=2:循环生成的次数
        size=4: 每次取样的大小
        image_path_list=IMAGE_PATH_LIST: 图片来源文件夹
        aug_transform=None: 转换器
        save=True: 是否保存到文件夹
        show=False: 是否显示生成的图片
        save_path='data/augmented/': 文件保存路径
    """

    for i in range(k):
        plot_transformed_images(image_path_list,
                                transform=aug_transform,
                                n=size,save=True,show=False,save_path='data/
↪augmented/')
```

[35]:
```
generate_augmented_images(10,10,image_path_list=IMAGE_PATH_LIST,aug_transform=aug_transform)␣
↪# 从已有的图片中增广生成 100 张图片
```

0.0.7 自定义继承自 torch.utils.data.Dataset 的数据集

[40]:
```
# 自定义继承自 torch.utils.data.Dataset 的数据集
from torch.utils.data import Dataset

# 1. torch.utils.data.Dataset 的子类
class ImageFolderWordLibDataSet(Dataset):

    # 2. 用 targ_dir 和 transform (可选) 参数初始化
    def __init__(self, targ_dir: str, transform = None, ext:str='gif'):

        # 3. 创建类属性
        # 获取文件夹下所有的图片文件全名
        self.paths = list(pathlib.Path(targ_dir).glob(f"*.{ext}")) # note: ext
为文件扩展名，可以改为 .png's 或.jpeg's
        # 设置 transforms
        self.transform = transform
        # 创建 classes 和 class_to_idx 属性
        self.classes, self.class_to_idx,_ = find_classes(targ_dir,ext)

    # 4. 定义加载图片的函数
    def load_image(self, index: int):
        "Opens an image via a path and returns it."
        image_path = self.paths[index]
        return Image.open(image_path).convert('RGB'),image_path

    # 5. 覆盖 the __len__() 方法
    def __len__(self) -> int:
        "返回样本总数"
        return len(self.paths)

    # 6. 覆盖 __getitem__() 方法 (作为 torch.utils.data.Dataset 子类必须重写该方
法)
    def __getitem__(self, index: int) -> Tuple[torch.Tensor, int]:
        "根据 index 返回一个样本的 data and label (X, y)."
        img,img_path = self.load_image(index)
```

```
        class_name  = img_path.name.split('_')[0] # 命名规则为: data_dir/
↪word_font_writer_number.gif
        class_idx = self.class_to_idx[class_name]

        # 对图片作转换
        if self.transform:
            return self.transform(img), class_idx # 返回样本 data, label (X, y)
        else:
            return img, class_idx # 返回样本 data, label (X, y)
```

[37]:

```
# 自定义继承自 torch.utils.data.Dataset 的数据集
from torch.utils.data import Dataset

# 1. torch.utils.data.Dataset 的子类
class ImageWriterWordLibDataSet(Dataset):

    # 2. 用 targ_dir 和 transform (可选) 参数初始化
    def __init__(self, targ_dir: str, transform = None, ext:str='gif'):

        # 3. 创建类属性
        # 获取文件夹下所有的图片文件全名
        self.paths = list(pathlib.Path(targ_dir).glob(f"*.{ext}")) # note: ext
为文件扩展名, 可以改为 .png's 或.jpeg's
        # 设置 transforms
        self.transform = transform
        # 创建 classes 和 class_to_idx 属性
        self.writer_classes, self.writer_class_to_idx,_ =␣
↪find_writer_classes(targ_dir,ext)

    # 4. 定义加载图片的函数
    def load_image(self, index: int):
        "Opens an image via a path and returns it."
        image_path = self.paths[index]
        return Image.open(image_path).convert('RGB'),image_path

    # 5. 覆盖 the __len__() 方法
```

```
    def __len__(self) -> int:
        "返回样本总数"
        return len(self.paths)

    # 6. 覆盖 __getitem__() 方法 (作为 torch.utils.data.Dataset 子类必须重写该方法)
    def __getitem__(self, index: int) -> Tuple[torch.Tensor, int]:
        "根据 index 返回一个样本的 data and label (X, y)."
        img,img_path = self.load_image(index)
        writer_class_name  = img_path.name.split('_')[2] # 命名规则为: data_dir/word_font_writer_number.gif
        writer_class_idx = self.writer_class_to_idx[writer_class_name]

        # 对图片作转换
        if self.transform:
            return self.transform(img), writer_class_idx # 返回样本 data, label (X, y)
        else:
            return img, writer_class_idx # 返回样本 data, label (X, y)
```

[41]:

```
# 对 train data 作转换
train_transforms = transforms.Compose([
    transforms.Resize((64, 64)),
    #transforms.RandomHorizontalFlip(p=0.5),
    transforms.ToTensor()
])

# 对 test data 只须统一 shape 并转换为 Tensor
test_transforms = transforms.Compose([
    transforms.Resize((64, 64)),
    transforms.ToTensor()
])
```

0.0.8 实例化自定义的数据集对象，并拆分为训练集和测试集

[42]:
```
# 实例化自定义的数据集对象，并拆分为训练集和测试集
data_custom = ImageFolderWordLibDataSet(targ_dir=IMAGE_PATH,
                                        transform=train_transforms,
                                        ext='gif')
train_size=int(0.9*len(data_custom))
test_size=len(data_custom)-train_size
torch.manual_seed(42)
train_dataset,test_dataset=torch.utils.data.
 ↪random_split(data_custom,[train_size,test_size])

train_dataset, test_dataset
```

[42]:
```
(<torch.utils.data.dataset.Subset at 0x182c1ceecf8>,
 <torch.utils.data.dataset.Subset at 0x182c1ceed30>)
```

[44]:
```
len(data_custom),len(train_dataset),len(test_dataset),len(train_dataset)+len(test_dataset)
```

[44]:
```
(5780, 5202, 578, 5780)
```

[43]:
```
# 实例化自定义的数据集对象，并拆分为训练集和测试集
data_writer_custom = ImageWriterWordLibDataSet(targ_dir=IMAGE_PATH,
                                        transform=train_transforms,
                                        ext='gif')
train_writer_size=int(0.9*len(data_writer_custom))
test_writer_size=len(data_writer_custom)-train_writer_size
torch.manual_seed(42)
train_writer_dataset,test_writer_dataset=torch.utils.data.
 ↪random_split(data_writer_custom,[train_writer_size,test_writer_size])

train_writer_dataset, test_writer_dataset
```

[43]:
```
(<torch.utils.data.dataset.Subset at 0x182c1ceee48>,
 <torch.utils.data.dataset.Subset at 0x182c1ceedd8>)
```

[45]:
```
len(data_writer_custom),len(train_writer_dataset),len(test_writer_dataset),len(train_writer_da
```

[45]:
```
(5780, 5202, 578, 5780)
```

0.0.9 创建随机显示图片的函数

[46]:
```
# 1. 输入参数为 dataset、文字列表
def display_random_images(dataset: torch.utils.data.dataset.Dataset,
                          classes: List[str] = None,
                          n: int = 10,
                          display_shape: bool = True,
                          seed: int = None):

    # 2. 为了好的显示效果，只允许显示 10 张
    if n > 10:
        n = 10
        display_shape = False
        print(f"为了好的显示效果，最多只允许显示 10 张图片.")

    # 3. 设置随机种子
    if seed:
        random.seed(seed)

    # 4. 获取抽样序号
    random_samples_idx = random.sample(range(len(dataset)), k=n)

    # 5. 设置 figure 大小
    plt.figure(figsize=(16, 8))

    # 6. 显示每张抽取的图片
    for i, targ_sample in enumerate(random_samples_idx):
        targ_image, targ_label = dataset[targ_sample][0],␣
↪dataset[targ_sample][1]

        # 7. 用 permute 函数调整 image 的 tensor shape 以正确显示图片:
        # tensor 的维度: [color_channels, height, width] -> 画图维度 [height,␣
↪width,color_channels]
        targ_image_adjust = targ_image.permute(1, 2, 0)
        if targ_image_adjust.shape[2]==1:
```

```
            targ_image_adjust=targ_image_adjust.squeeze(2)  # 如果图片只有 1 个通道，则需要压缩维度，去掉通道信息，否则不能正常显示
        # 将 n 幅图画在 1 行
        plt.subplot(1, n, i+1)
        plt.imshow(targ_image_adjust)
        plt.axis("off")
        if classes:
            title = f"class: {classes[targ_label]}"
            if display_shape:
                title = title + f"\nshape: {targ_image_adjust.shape}"
        plt.title(title,fontproperties='simhei')
```

[47]:
```
display_random_images(train_dataset,
                      n=18,
                      classes=images_classes_list,
                      seed=None)
```

为了好的显示效果，最多只允许显示 10 张图片.

[54]:
```
display_random_images(train_writer_dataset,
                      n=18,
                      classes=images_writer_classes_list,
                      seed=None)
```

为了好的显示效果，最多只允许显示 10 张图片.

0.0.10 用 DataLoader 来加载自定义的数据集

[55]:
```
os.cpu_count()
```

[55]: 4

[56]:
```
train_dataloader = DataLoader(dataset=train_dataset, # 使用自定义训练数据集
                              batch_size=32, # 每批次加载多少样本
                              num_workers=0, # 并行加载任务数 (越高越好,
但不高于 os.cpu_count(),0 表示任务加载)
                              shuffle=True) # 是否乱序加载?

test_dataloader = DataLoader(dataset=test_dataset, # 使用自定义测试数据集
                             batch_size=32,
                             num_workers=0,
                             shuffle=False) # 不须乱序加载

train_dataloader, test_dataloader
```

[56]:
```
(<torch.utils.data.dataloader.DataLoader at 0x182c2920748>,
 <torch.utils.data.dataloader.DataLoader at 0x182c2920630>)
```

[57]:
```
train_writer_dataloader = DataLoader(dataset=train_writer_dataset, # 使用自定义
训练数据集
                              batch_size=32, # 每批次加载多少样本
                              num_workers=0, # 并行加载任务数 (越高越好,
但不高于 os.cpu_count(),0 表示任务加载)
                              shuffle=True) # 是否乱序加载?

test_writer_dataloader = DataLoader(dataset=test_writer_dataset, # 使用自定义测
试数据集
                             batch_size=32,
                             num_workers=0,
                             shuffle=False) # 不须乱序加载

train_writer_dataloader, test_writer_dataloader
```

[57]:
```
(<torch.utils.data.dataloader.DataLoader at 0x182c2920208>,
 <torch.utils.data.dataloader.DataLoader at 0x182c2920978>)
```

[58]:
```
img, label = next(iter(train_dataloader))
# next 一次加载一批
print(f"Image shape: {img.shape} -> [batch_size, color_channels, height,␣
↪width]")
print(f"Label shape: {label.shape}")
```

```
Image shape: torch.Size([32, 3, 64, 64]) -> [batch_size, color_channels, height,
width]
Label shape: torch.Size([32])
```

[59]:
```
img_writer, label_writer = next(iter(train_writer_dataloader))
# next 一次加载一批
print(f"Image shape: {img.shape} -> [batch_size, color_channels, height,␣
↪width]")
print(f"Label shape: {label.shape}")
```

```
Image shape: torch.Size([32, 3, 64, 64]) -> [batch_size, color_channels, height,
width]
Label shape: torch.Size([32])
```

[25]:
```
train_transform = transforms.Compose([
    transforms.Resize((64, 64)),
    #transforms.TrivialAugmentWide(num_magnitude_bins=31,fill=255), # how␣
↪intense
    #transforms.ColorJitter(brightness=.5, hue=.3),
    #transforms.RandomRotation(degrees=(0, 180),expand=False,fill=255),
    #transforms.RandomAffine(degrees=(30, 70), translate=(0.1, 0.3), scale=(0.
↪7, 0.9),fill=255),
    #transforms.ElasticTransform(alpha=250.0,fill=255),
    #transforms.RandomPerspective(distortion_scale=0.5, p=0.6,fill=255),
    transforms.ToTensor() # use ToTensor() last to get everything between 0 & 1
])

# 对测试集不作增广变换
test_transforms = transforms.Compose([
```

```
    transforms.Resize((64, 64)),
    transforms.ToTensor()
])
```

0.0.11 创建 TinyVGG 模型类

[60]:
```
from torch import nn
class TinyVGG(nn.Module):
    """
    卷积神经网络的模型参考了下面的结构，该网站详细解释了该结构，并对模型参数作了很好
的可视化：
    https://poloclub.github.io/cnn-explainer/
    """
    def __init__(self, input_shape: int, hidden_units: int, output_shape: int)␣
↪-> None:
        super().__init__()
        self.conv_block_1 = nn.Sequential(
            nn.Conv2d(in_channels=input_shape,
                      out_channels=hidden_units,
                      kernel_size=3, # 卷积核大小
                      stride=1, # default
                      padding=1), # options = "valid" (no padding) or "same"␣
↪(output has same shape as input) or int for specific number
            nn.ReLU(),
            nn.Conv2d(in_channels=hidden_units,
                      out_channels=hidden_units,
                      kernel_size=3,
                      stride=1,
                      padding=1),
            nn.ReLU(),
            nn.MaxPool2d(kernel_size=2,
                         stride=2) # default stride value is same as kernel_size
        )
        self.conv_block_2 = nn.Sequential(
            nn.Conv2d(hidden_units, hidden_units, kernel_size=3, padding=1),
            nn.ReLU(),
```

```
            nn.Conv2d(hidden_units, hidden_units, kernel_size=3, padding=1),
            nn.ReLU(),
            nn.MaxPool2d(2)
        )
        self.classifier = nn.Sequential(
            nn.Flatten(),

            # 下面这一步的 in_features 设置有一定困难，如果维度计算不准，模型将报错，建议先把 self.classifier 这一层去掉，看前面结构的 output_shape 输出，
            # 再根据这个输出确定这里的 in_features
            nn.Linear(in_features=hidden_units*16*16,out_features=output_shape)
        )

    def forward(self, x: torch.Tensor):
        x = self.conv_block_1(x)
        # print(x.shape)
        x = self.conv_block_2(x)
        # print(x.shape)
        x = self.classifier(x)
        # print(x.shape)
        return x
        # return self.classifier(self.conv_block_2(self.conv_block_1(x))) # 这种用法效果相同且更高效

torch.manual_seed(42)
model_0 = TinyVGG(input_shape=3, # 颜色通道 (3 for RGB)
                  hidden_units=20,
                  output_shape=len(images_classes_list)).to(device)
model_0
```

[60]:
```
TinyVGG(
  (conv_block_1): Sequential(
    (0): Conv2d(3, 20, kernel_size=(3, 3), stride=(1, 1), padding=(1, 1))
    (1): ReLU()
    (2): Conv2d(20, 20, kernel_size=(3, 3), stride=(1, 1), padding=(1, 1))
    (3): ReLU()
```

```
    (4): MaxPool2d(kernel_size=2, stride=2, padding=0, dilation=1,
ceil_mode=False)
  )
  (conv_block_2): Sequential(
    (0): Conv2d(20, 20, kernel_size=(3, 3), stride=(1, 1), padding=(1, 1))
    (1): ReLU()
    (2): Conv2d(20, 20, kernel_size=(3, 3), stride=(1, 1), padding=(1, 1))
    (3): ReLU()
    (4): MaxPool2d(kernel_size=2, stride=2, padding=0, dilation=1,
ceil_mode=False)
  )
  (classifier): Sequential(
    (0): Flatten(start_dim=1, end_dim=-1)
    (1): Linear(in_features=5120, out_features=483, bias=True)
  )
)
```

[63]:

```
len(images_classes_list)
```

[63]: 483

[61]:

```
from torch import nn
class WriterTinyVGG(nn.Module):
    """
    卷积神经网络的模型参考了下面的结构，该网站详细解释了该结构，并对模型参数作了很好的可视化：
    https://poloclub.github.io/cnn-explainer/
    """
    def __init__(self, input_shape: int, hidden_units: int, output_shape: int)␣
↪-> None:
        super().__init__()
        self.conv_block_1 = nn.Sequential(
            nn.Conv2d(in_channels=input_shape,
                      out_channels=hidden_units,
                      kernel_size=3, # 卷积核大小
                      stride=1, # default
```

```
                    padding=1), # options = "valid" (no padding) or "same"␣
↪(output has same shape as input) or int for specific number
            nn.ReLU(),
            nn.Conv2d(in_channels=hidden_units,
                      out_channels=hidden_units,
                      kernel_size=3,
                      stride=1,
                      padding=1),
            nn.ReLU(),
            nn.MaxPool2d(kernel_size=2,
                         stride=2) # default stride value is same as kernel_size
        )
        self.conv_block_2 = nn.Sequential(
            nn.Conv2d(hidden_units, hidden_units, kernel_size=3, padding=1),
            nn.ReLU(),
            nn.Conv2d(hidden_units, hidden_units, kernel_size=3, padding=1),
            nn.ReLU(),
            nn.MaxPool2d(2)
        )
        self.classifier = nn.Sequential(
            nn.Flatten(),

            # 下面这一步的 in_features 设置有一定困难，如果维度计算不准，模型将报
错，建议先把 self.classifier 这一层去掉，看前面结构的 output_shape 输出，
            # 再根据这个输出确定这里的 in_features
            nn.Linear(in_features=hidden_units*16*16,out_features=output_shape)
        )

    def forward(self, x: torch.Tensor):
        x = self.conv_block_1(x)
        # print(x.shape)
        x = self.conv_block_2(x)
        # print(x.shape)
        x = self.classifier(x)
        # print(x.shape)
        return x
```

```
        # return self.classifier(self.conv_block_2(self.conv_block_1(x))) # 这种用法效果相同且更高效

torch.manual_seed(42)
writer_model_0 = WriterTinyVGG(input_shape=3, # 颜色通道 (3 for RGB)
                  hidden_units=20,
                  output_shape=len(images_writer_classes_list)).to(device)
writer_model_0
```

[61]:
```
WriterTinyVGG(
  (conv_block_1): Sequential(
    (0): Conv2d(3, 20, kernel_size=(3, 3), stride=(1, 1), padding=(1, 1))
    (1): ReLU()
    (2): Conv2d(20, 20, kernel_size=(3, 3), stride=(1, 1), padding=(1, 1))
    (3): ReLU()
    (4): MaxPool2d(kernel_size=2, stride=2, padding=0, dilation=1, ceil_mode=False)
  )
  (conv_block_2): Sequential(
    (0): Conv2d(20, 20, kernel_size=(3, 3), stride=(1, 1), padding=(1, 1))
    (1): ReLU()
    (2): Conv2d(20, 20, kernel_size=(3, 3), stride=(1, 1), padding=(1, 1))
    (3): ReLU()
    (4): MaxPool2d(kernel_size=2, stride=2, padding=0, dilation=1, ceil_mode=False)
  )
  (classifier): Sequential(
    (0): Flatten(start_dim=1, end_dim=-1)
    (1): Linear(in_features=5120, out_features=447, bias=True)
  )
)
```

[62]:
```
len(images_writer_classes_list)
```

[62]:
```
447
```

[64]:
```python
# 1. 从 test_dataloader 中抽取一批数据用于显示
itr=iter(test_dataloader)
img_batch, label_batch= next(itr)
```

[65]:
```python
# 1. 从 test_dataloader 中抽取一批数据用于显示
writer_itr=iter(test_writer_dataloader)
img_writer_batch, label_writer_batch= next(writer_itr)
```

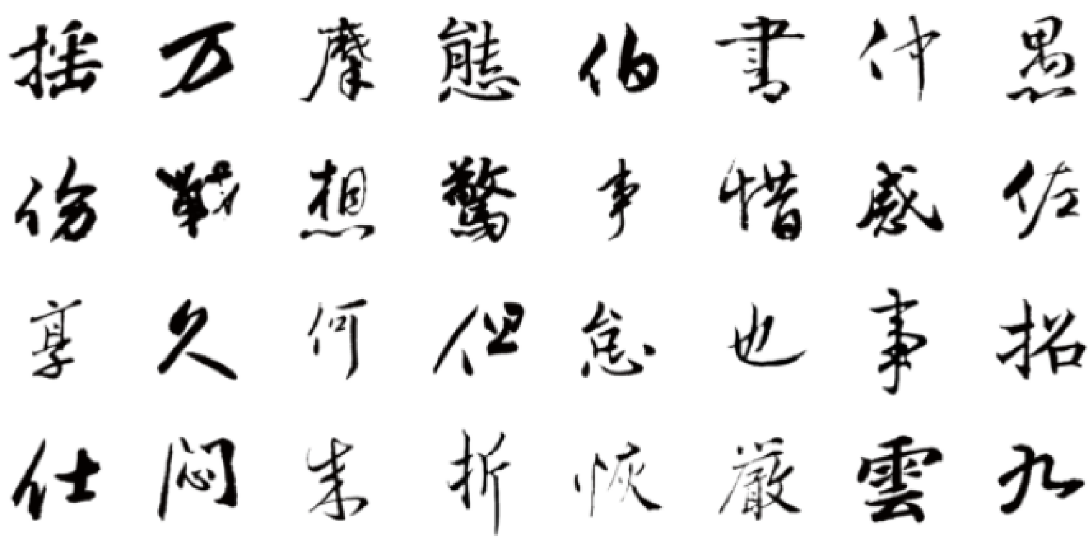
[66]:
```python
def plot_from_image_tensor(img_tensor):
    """
    把图片 tensor 显示成图片
    """
    img =img_tensor.permute(1,2,0) # 如果只想看某一个 channel 的话，再接上 [:,:,0]
    if img.shape[2]==1:
        img=img.squeeze(2)
    plt.imshow(img.cpu())  # 对于在 GPU 上的数据集，需要调用.cpu() 才能 plot
    plt.axis(False)
```

[67]:
```python
def result_compare(iterator,model):
    model.eval()
    with torch.inference_mode():
        image_batch, label_batch = next(iterator)
        image_batch=image_batch.to(device)
        pred_label=torch.argmax(model(image_batch),dim=1)
        #print(model_0(image_batch).shape,pred_label,label_batch)
        word_dict=dict()
        label_dict=dict()
        fig, ax = plt.subplots(figsize=(12,6))
        ax.axis(False)
        for i in range(len(label_batch)):
            pred_word=images_classes_list[pred_label[i]]
            word=images_classes_list[label_batch[i]]
            ax = fig.add_subplot(4,8,i+1)
            plot_from_image_tensor(image_batch[i])
            word_dict[word] = pred_word
            label_dict[label_batch[i]]=pred_label[i]
            pred_compare=pd.DataFrame.from_dict(word_dict,orient='index')
```

```
            pred_compare.reset_index(inplace=True)
            pred_compare.columns=['实际汉字','识别结果']
    return pred_compare
```

[69]:
```
result=result_compare(itr,model_0)
result.T
```

[69]:
```
       0  1  2  3  4  5  6  7  8  9  ... 21 22 23 24 25 26 27 28 29 30
实际汉字  搖  万  摩  态  伯  书  仲  愚  伤  戰  ...  也  招  仕  悶  來  折  恢 ␣
↪严  云  九
识别结果  仞  仞  仞  仞  仞  仞  仞  仞  仞  仞  ...  仞  仞  仞  仞  仞  仞  仞 ␣
↪仞  仞  仞

[2 rows x 31 columns]
```

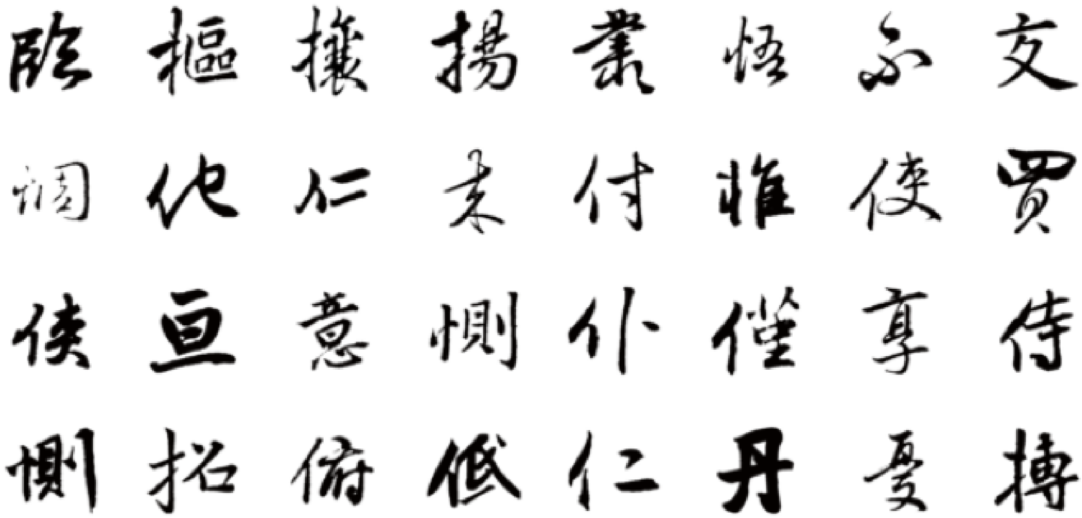


[72]:
```
def writer_result_compare(iterator,model):
    model.eval()
    with torch.inference_mode():
        image_writer_batch, label_writer_batch = next(iterator)
        image_writer_batch=image_writer_batch.to(device)
        pred_writer_label=torch.argmax(model(image_writer_batch),dim=1)
        #print(model_0(image_batch).shape,pred_label,label_batch)
```

```
        word_writer_dict=dict()
        label_writer_dict=dict()
        fig, ax = plt.subplots(figsize=(12,6))
        ax.axis(False)
        for i in range(len(label_writer_batch)):
            pred_writer_word=images_writer_classes_list[pred_writer_label[i]]
            word_writer=images_writer_classes_list[label_writer_batch[i]]
            ax = fig.add_subplot(4,8,i+1)
            plot_from_image_tensor(image_writer_batch[i])
            word_writer_dict[word_writer] = pred_writer_word
            label_writer_dict[label_writer_batch[i]]=pred_writer_label[i]
            pred_compare=pd.DataFrame.from_dict(word_writer_dict,orient='index')
            pred_compare.reset_index(inplace=True)
            pred_compare.columns=['书写人','识别结果']
    return pred_compare
```

[73]:
```
writer_result=writer_result_compare(writer_itr,writer_model_0)
writer_result.T
```

[73]:
```
        0    1    2    3    4    5    6    7    8    9    10   11   12   13  \
书写人     苏轼   李邕  欧阳询  董其昌   徐渭  赵孟頫  薛绍彭  虞世南  王守仁  范仲
淹    李倜  颜真卿  王羲之    唐寅
识别结果   黄庭坚  司马丕  司马丕  黄庭坚  司马丕  司马丕  司马丕  司马丕  司马丕 ␣
↪司马丕  司马丕  司马丕  司马丕  司马丕


       14   15   16   17   18
书写人    敬世江  程正揆   米芾  褚遂良  沈树镛
识别结果   司马丕  司马丕  司马丕  司马丕  司马丕
```



0.0.12 使用 torchinfo 来获得模型信息

[74]:
```
# torchinfo 这个包可以比较方便地显示模型结构和参数，如果 import 失败，需要安装
try:
    import torchinfo
except:
    !pip install torchinfo
    import torchinfo

from torchinfo import summary
summary(model_0, input_size=img_batch.shape) # summary 函数非常方便，只需要把一个
batch 的 shape 作为输入就能够得模型信息，不须加载真实数据
```

[74]:
```
================================================================================
==========
Layer (type:depth-idx)                   Output Shape              Param #
================================================================================
==========
TinyVGG                                  [32, 483]                 --
 Sequential: 1-1                         [32, 20, 32, 32]          --
     Conv2d: 2-1                         [32, 20, 64, 64]          560
```

```
        ReLU: 2-2                             [32, 20, 64, 64]          --
        Conv2d: 2-3                           [32, 20, 64, 64]          3,620
        ReLU: 2-4                             [32, 20, 64, 64]          --
        MaxPool2d: 2-5                        [32, 20, 32, 32]          --
    Sequential: 1-2                           [32, 20, 16, 16]          --
        Conv2d: 2-6                           [32, 20, 32, 32]          3,620
        ReLU: 2-7                             [32, 20, 32, 32]          --
        Conv2d: 2-8                           [32, 20, 32, 32]          3,620
        ReLU: 2-9                             [32, 20, 32, 32]          --
        MaxPool2d: 2-10                       [32, 20, 16, 16]          --
    Sequential: 1-3                           [32, 483]                 --
        Flatten: 2-11                         [32, 5120]                --
        Linear: 2-12                          [32, 483]                 2,473,443
================================================================================
==========
Total params: 2,484,863
Trainable params: 2,484,863
Non-trainable params: 0
Total mult-adds (M): 864.27
================================================================================
==========
Input size (MB): 1.57
Forward/backward pass size (MB): 52.55
Params size (MB): 9.94
Estimated Total Size (MB): 64.06
================================================================================
==========
```

[75]:
```
summary(writer_model_0, input_size=img_writer_batch.shape) # summary 函数非常方便，只需要把一个 batch 的 shape 作为输入就能够得模型信息，不须加载真实数据
```

[75]:
```
================================================================================
==========
Layer (type:depth-idx)                        Output Shape              Param #
================================================================================
==========
WriterTinyVGG                                 [32, 447]                 --
```

```
 Sequential: 1-1                        [32, 20, 32, 32]          --
     Conv2d: 2-1                        [32, 20, 64, 64]          560
     ReLU: 2-2                          [32, 20, 64, 64]          --
     Conv2d: 2-3                        [32, 20, 64, 64]          3,620
     ReLU: 2-4                          [32, 20, 64, 64]          --
     MaxPool2d: 2-5                     [32, 20, 32, 32]          --
 Sequential: 1-2                        [32, 20, 16, 16]          --
     Conv2d: 2-6                        [32, 20, 32, 32]          3,620
     ReLU: 2-7                          [32, 20, 32, 32]          --
     Conv2d: 2-8                        [32, 20, 32, 32]          3,620
     ReLU: 2-9                          [32, 20, 32, 32]          --
     MaxPool2d: 2-10                    [32, 20, 16, 16]          --
 Sequential: 1-3                        [32, 447]                 --
     Flatten: 2-11                      [32, 5120]                --
     Linear: 2-12                       [32, 447]                 2,289,087
================================================================================
==========
Total params: 2,300,507
Trainable params: 2,300,507
Non-trainable params: 0
Total mult-adds (M): 858.37
================================================================================
==========
Input size (MB): 1.57
Forward/backward pass size (MB): 52.54
Params size (MB): 9.20
Estimated Total Size (MB): 63.32
================================================================================
==========
```

0.0.13 创建 train_step 和 test_step 函数

主要定义了三个函数: 1. `train_step()` - 输入参数为：model, `DataLoader`，loss function 和 optimizer 2. `test_step()` - 输入参数为：model, `DataLoader`，loss function 和 optimizer 3. `train()` - 定义 train Loop，执行给定的 epochs 并返回一个结果集的 dict.

- 模型训练的标准流程：

  - 0-上 device
  - 1-model(x) 前向算结果
  - 2-loss_fn 根据结果算损失
  - 3-zero_grad 梯度全归零
  - 4-backword 反向传播算梯度
  - 5-step 更新参数

[76]:
```
def train_step(model: torch.nn.Module,
               dataloader: torch.utils.data.DataLoader,
               loss_fn: torch.nn.Module,
               optimizer: torch.optim.Optimizer):
    # model 进入训练模式
    model.train()

    # 设置 train loss and train accuracy values
    train_loss, train_acc = 0, 0

    # 对 data loader 的每个 data 批次进行训练。假如训练集有 10000 个数据，batch␣
 ↪size 为 32 的话，则有 (10000/32)=312.5 经向上取整后共 313 个批次
    # 但这不用手动计算，将 dataloader 放到 enumerate() 函数中会自动循环获取
    # 有些代码也会使用 iter(dataloader) 进行循环，区别在于 iter 不会返回批次的序号

    # 0-5 步为模型训练的标准流程：
    '''
    0-上 device
    1-前向算结果
    2-根据结果算损失
    3-zero_grad 梯度全归零
    4-backword 反向传播算梯度
    5-step 更新参数
    '''

    for batch, (X, y) in enumerate(dataloader):
        # 0. 把数据放到目标 device 上
        X, y = X.to(device), y.to(device)

        # 1. Forward pass
```

```
        y_pred = model(X)

        # 2. Calculate  and accumulate loss
        loss = loss_fn(y_pred, y)
        train_loss += loss.item()   #loss_fn 返回的是 tensor，调用.item() 转换为
numpy 的值

        # 3. Optimizer zero grad
        optimizer.zero_grad()

        # 4. Loss backward
        loss.backward()

        # 5. Optimizer step
        optimizer.step()

        # 6. Calculate and accumulate accuracy metric across all batches
        y_pred_class = torch.argmax(torch.softmax(y_pred, dim=1), dim=1)
        train_acc += (y_pred_class == y).sum().item()/len(y_pred)

    # 计算每批次 loss 和 accuracy 的平均数
    train_loss = train_loss / len(dataloader)
    train_acc = train_acc / len(dataloader)
    return train_loss, train_acc
```

[77]:
```
def test_step(model: torch.nn.Module,
              dataloader: torch.utils.data.DataLoader,
              loss_fn: torch.nn.Module):
    # 开启 test 模式，有些 dropout 的层将跳过
    model.eval()

    # 设置 test loss 和 test accuracy 为 0
    test_loss, test_acc = 0, 0

    # 不会进行梯度计算，以加快运行速度
    with torch.inference_mode():
```

```python
        # Loop through DataLoader batches
        for batch, (X, y) in enumerate(dataloader):
            # Send data to target device
            X, y = X.to(device), y.to(device)

            # 1. Forward pass
            test_pred_logits = model(X)

            # 2. Calculate and accumulate loss
            loss = loss_fn(test_pred_logits, y)
            test_loss += loss.item()

            # Calculate and accumulate accuracy
            test_pred_labels = test_pred_logits.argmax(dim=1)
            test_acc += ((test_pred_labels == y).sum().item()/
↪len(test_pred_labels))

    # 计算每个 test batch 平均损失和准确度
    test_loss = test_loss / len(dataloader)
    test_acc = test_acc / len(dataloader)
    return test_loss, test_acc
```

0.0.14 创建训练 Loop: 将 train_step() 和 test_step() 放在 train() 函数中

1. 传入参数:model, 封装了训练集和测试集的 `DataLoader`,优化器 optimizer, 损失函数 loss_fn,训练和测试的循环次数 epochs
2. 创建空的 `train_loss`, `train_acc`, `test_loss` , `test_acc` 字典
3. 对 epoches 中的每个 epoch 循环运行 train() 和 test().
4. 输出每个 epoch 的过程信息.
5. 更新每个 epoch 的 metrics 字典.
6. 返回结果

[78]:
```python
# 1. 定义 train 函数和传入参数
def train(model: torch.nn.Module,
          train_dataloader: torch.utils.data.DataLoader,
          test_dataloader: torch.utils.data.DataLoader,
```

```python
        optimizer: torch.optim.Optimizer,
        loss_fn: torch.nn.Module = nn.CrossEntropyLoss(),
        epochs: int = 5):

# 2. 创建空字典用于存储结果
results = {"train_loss": [],
    "train_acc": [],
    "test_loss": [],
    "test_acc": []
}

# 3. Training 循环
for epoch in tqdm(range(epochs)):
    train_loss, train_acc = train_step(model=model,
                                       dataloader=train_dataloader,
                                       loss_fn=loss_fn,
                                       optimizer=optimizer)
    test_loss, test_acc = test_step(model=model,
                                    dataloader=test_dataloader,
                                    loss_fn=loss_fn)

    # 4. 输出结果
    print(
        f"Epoch: {epoch+1} | "
        f"train_loss: {train_loss:.4f} | "
        f"train_acc: {train_acc:.4f} | "
        f"test_loss: {test_loss:.4f} | "
        f"test_acc: {test_acc:.4f}"
    )

    # 5. 更新结果字典
    results["train_loss"].append(train_loss)
    results["train_acc"].append(train_acc)
    results["test_loss"].append(test_loss)
    results["test_acc"].append(test_acc)
```

```
    # 6. 训练结束返回结果
    return results
```

[79]:
```
# 设置随机种子
torch.manual_seed(42)
torch.cuda.manual_seed(42)

# 设置 epochs 次数
NUM_EPOCHS = 10

# 实例化模型
model_0 = TinyVGG(input_shape=3, # number of color channels (3 for RGB)
                  hidden_units=20,
                  output_shape=len(data_custom.classes)).to(device)

# 设置损失函数和优化器
loss_fn = nn.CrossEntropyLoss()
optimizer = torch.optim.Adam(params=model_0.parameters(), lr=0.001)

# 用 timer 开始计时

start_time = timer()

# 开始训练模型 model_0
model_0_results = train(model=model_0,
                        train_dataloader=train_dataloader,
                        test_dataloader=test_dataloader,
                        optimizer=optimizer,
                        loss_fn=loss_fn,
                        epochs=NUM_EPOCHS)

# 训练结束，输出训练时长
end_time = timer()
print(f"训练时长: {end_time-start_time:.3f} seconds")
```

```
 10%|
| 1/10 [00:55<08:15, 55.07s/it]
```

```
Epoch: 1 | train_loss: 5.9084 | train_acc: 0.0363 | test_loss: 5.1641 |
test_acc: 0.1579

 20%|
| 2/10 [01:48<07:14, 54.30s/it]

Epoch: 2 | train_loss: 3.0996 | train_acc: 0.3972 | test_loss: 3.3579 |
test_acc: 0.4013

 30%|
| 3/10 [02:43<06:20, 54.31s/it]

Epoch: 3 | train_loss: 1.3065 | train_acc: 0.6790 | test_loss: 3.2859 |
test_acc: 0.4030

 40%|
| 4/10 [03:37<05:25, 54.22s/it]

Epoch: 4 | train_loss: 0.6659 | train_acc: 0.8263 | test_loss: 3.6848 |
test_acc: 0.3964

 50%|
| 5/10 [04:32<04:33, 54.73s/it]

Epoch: 5 | train_loss: 0.4472 | train_acc: 0.8882 | test_loss: 3.8848 |
test_acc: 0.4046

 60%|
| 6/10 [05:31<03:43, 56.00s/it]

Epoch: 6 | train_loss: 0.3587 | train_acc: 0.8964 | test_loss: 3.7069 |
test_acc: 0.4095

 70%|
| 7/10 [06:30<02:51, 57.14s/it]

Epoch: 7 | train_loss: 0.3176 | train_acc: 0.9005 | test_loss: 3.9286 |
test_acc: 0.4309

 80%|
| 8/10 [07:30<01:55, 57.94s/it]

Epoch: 8 | train_loss: 0.2967 | train_acc: 0.9009 | test_loss: 3.6960 |
test_acc: 0.4095
```

```
 90%|
| 9/10 [08:30<00:58, 58.68s/it]

Epoch: 9 | train_loss: 0.2894 | train_acc: 0.9008 | test_loss: 3.6407 |
test_acc: 0.4046

100%|
    | 10/10 [09:30<00:00, 57.03s/it]

Epoch: 10 | train_loss: 0.2639 | train_acc: 0.9046 | test_loss: 3.7103 |
test_acc: 0.4112
训练时长: 570.333 seconds
```

[108]:

```python
# 设置随机种子
torch.manual_seed(42)
torch.cuda.manual_seed(42)

# 设置 epochs 次数
NUM_EPOCHS =20

# 实例化模型
writer_model_0 = WriterTinyVGG(input_shape=3, # number of color channels (3 for␣
 ↪RGB)
                  hidden_units=20,
                  output_shape=len(data_writer_custom.writer_classes)).
 ↪to(device)

# 设置损失函数和优化器
loss_fn = nn.CrossEntropyLoss()
optimizer = torch.optim.Adam(params=writer_model_0.parameters(), lr=0.001)

# 用 timer 开始计时

start_time = timer()

# 开始训练模型 model_0
writer_model_0_results = train(model=writer_model_0,
```

```python
                           train_dataloader=train_writer_dataloader,
                           test_dataloader=test_writer_dataloader,
                           optimizer=optimizer,
                           loss_fn=loss_fn,
                           epochs=NUM_EPOCHS)

# 训练结束，输出训练时长
end_time = timer()
print(f"训练时长: {end_time-start_time:.3f} seconds")
```

```
  5%|
| 1/20 [00:49<15:38, 49.40s/it]

Epoch: 1 | train_loss: 4.5648 | train_acc: 0.0834 | test_loss: 4.3760 |
test_acc: 0.1069

 10%|
| 2/20 [01:37<14:38, 48.78s/it]

Epoch: 2 | train_loss: 4.3665 | train_acc: 0.0922 | test_loss: 4.2932 |
test_acc: 0.1151

 15%|
| 3/20 [02:26<13:48, 48.76s/it]

Epoch: 3 | train_loss: 4.1344 | train_acc: 0.1322 | test_loss: 4.1794 |
test_acc: 0.1480

 20%|
| 4/20 [03:15<13:01, 48.87s/it]

Epoch: 4 | train_loss: 3.4332 | train_acc: 0.2094 | test_loss: 4.3904 |
test_acc: 0.2039

 25%|
| 5/20 [04:07<12:27, 49.85s/it]

Epoch: 5 | train_loss: 2.1319 | train_acc: 0.4203 | test_loss: 5.1541 |
test_acc: 0.2336

 30%|
| 6/20 [04:58<11:43, 50.27s/it]
```

```
Epoch: 6 | train_loss: 1.3563 | train_acc: 0.6051 | test_loss: 5.9164 |
test_acc: 0.2582

 35%|
| 7/20 [05:50<11:02, 50.95s/it]

Epoch: 7 | train_loss: 0.9722 | train_acc: 0.7102 | test_loss: 6.8988 |
test_acc: 0.2434

 40%|
| 8/20 [06:45<10:27, 52.30s/it]

Epoch: 8 | train_loss: 0.7467 | train_acc: 0.7755 | test_loss: 7.9996 |
test_acc: 0.2747

 45%|
| 9/20 [07:40<09:44, 53.10s/it]

Epoch: 9 | train_loss: 0.5922 | train_acc: 0.8160 | test_loss: 9.9073 |
test_acc: 0.2763

 50%|
| 10/20 [08:36<09:00, 54.00s/it]

Epoch: 10 | train_loss: 0.4617 | train_acc: 0.8525 | test_loss: 11.1870 |
test_acc: 0.2780

 55%|
| 11/20 [09:33<08:13, 54.88s/it]

Epoch: 11 | train_loss: 0.3859 | train_acc: 0.8802 | test_loss: 12.2557 |
test_acc: 0.2681

 60%|
| 12/20 [10:32<07:29, 56.15s/it]

Epoch: 12 | train_loss: 0.3078 | train_acc: 0.9038 | test_loss: 13.3804 |
test_acc: 0.2615

 65%|
| 13/20 [11:32<06:41, 57.34s/it]

Epoch: 13 | train_loss: 0.2728 | train_acc: 0.9127 | test_loss: 15.8502 |
test_acc: 0.2911
```

```
 70%|
| 14/20 [12:34<05:52, 58.74s/it]

Epoch: 14 | train_loss: 0.2208 | train_acc: 0.9268 | test_loss: 16.4929 |
test_acc: 0.2763

 75%|
| 15/20 [13:43<05:08, 61.74s/it]

Epoch: 15 | train_loss: 0.1906 | train_acc: 0.9377 | test_loss: 17.9865 |
test_acc: 0.2714

 80%|
| 16/20 [14:47<04:09, 62.48s/it]

Epoch: 16 | train_loss: 0.1576 | train_acc: 0.9463 | test_loss: 20.7017 |
test_acc: 0.2681

 85%|
| 17/20 [15:54<03:11, 63.74s/it]

Epoch: 17 | train_loss: 0.1575 | train_acc: 0.9462 | test_loss: 20.2624 |
test_acc: 0.2681

 90%|
| 18/20 [17:00<02:09, 64.62s/it]

Epoch: 18 | train_loss: 0.1305 | train_acc: 0.9529 | test_loss: 23.3429 |
test_acc: 0.2780

 95%|
      | 19/20 [18:08<01:05, 65.52s/it]

Epoch: 19 | train_loss: 0.1285 | train_acc: 0.9578 | test_loss: 23.5883 |
test_acc: 0.2829

100%|
    | 20/20 [19:15<00:00, 57.77s/it]

Epoch: 20 | train_loss: 0.0919 | train_acc: 0.9696 | test_loss: 25.6492 |
test_acc: 0.2829
训练时长: 1155.444 seconds
```

0.0.15 查看预测结果

[109]:
```
model_0_df = pd.DataFrame(model_0_results)
model_0_df
```

[109]:

	train_loss	train_acc	test_loss	test_acc
0	5.908390	0.036299	5.164138	0.157895
1	3.099563	0.397154	3.357877	0.401316
2	1.306452	0.679022	3.285882	0.402961
3	0.665923	0.826282	3.684769	0.396382
4	0.447237	0.888165	3.884816	0.404605
5	0.358678	0.896408	3.706902	0.409539
6	0.317633	0.900541	3.928642	0.430921
7	0.296696	0.900861	3.695995	0.409539
8	0.289425	0.900754	3.640692	0.404605
9	0.263940	0.904567	3.710320	0.411184

[110]:
```
writer_model_0_df = pd.DataFrame(writer_model_0_results)
writer_model_0_df
```

[110]:

	train_loss	train_acc	test_loss	test_acc
0	4.564828	0.083397	4.375977	0.106908
1	4.366457	0.092174	4.293190	0.115132
2	4.134369	0.132243	4.179427	0.148026
3	3.433221	0.209441	4.390404	0.203947
4	2.131927	0.420309	5.154053	0.233553
5	1.356318	0.605061	5.916434	0.258224
6	0.972158	0.710187	6.898800	0.243421
7	0.746696	0.775520	7.999607	0.274671
8	0.592161	0.815972	9.907298	0.276316
9	0.461726	0.852505	11.187004	0.277961
10	0.385896	0.880240	12.255719	0.268092
11	0.307813	0.903843	13.380422	0.261513
12	0.272844	0.912662	15.850240	0.291118
13	0.220768	0.926849	16.492946	0.276316
14	0.190593	0.937734	17.986539	0.271382
15	0.157575	0.946319	20.701718	0.268092
16	0.157535	0.946213	20.262435	0.268092

```
17     0.130500    0.952923   23.342872   0.277961
18     0.128465    0.957758   23.588341   0.282895
19     0.091855    0.969559   25.649233   0.282895
```

0.0.16 绘制训练过程曲线

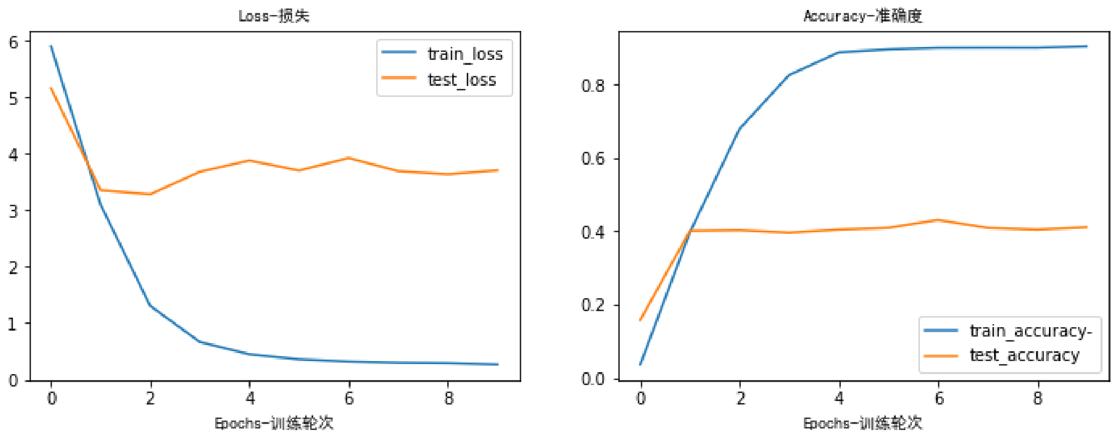
[111]:
```python
def plot_loss_curves(results: Dict[str, List[float]]):
    """ 绘制训练过程曲线.

    Args:
        results (dict): 训练过程记录 dict, 包括:
            {"train_loss": [...],
             "train_acc": [...],
             "test_loss": [...],
             "test_acc": [...]}
    """

    # 获取 Train 和 test 过程的 loss 值
    loss = results['train_loss']
    test_loss = results['test_loss']

    # 获取 train 和 test 过程的准确度 acc 值
    accuracy = results['train_acc']
    test_accuracy = results['test_acc']

    # 获取训练经历的 epoches
    epochs = range(len(results['train_loss']))

    plt.figure(figsize=(12, 4))

    # Plot loss
    plt.subplot(1, 2, 1)
    plt.plot(epochs, loss, label='train_loss')
    plt.plot(epochs, test_loss, label='test_loss')
    plt.title('Loss-损失', fontsize=16,fontproperties='Simhei')
    plt.xlabel('Epochs-训练轮次', fontsize=16,fontproperties='Simhei')
```

```
    plt.legend()

    # Plot accuracy
    plt.subplot(1, 2, 2)
    plt.plot(epochs, accuracy, label='train_accuracy-')
    plt.plot(epochs, test_accuracy, label='test_accuracy')
    plt.title('Accuracy-准确度', fontsize=16,fontproperties='Simhei')
    plt.xlabel('Epochs-训练轮次', fontsize=16,fontproperties='Simhei')
    plt.legend();
```

[112]:
```
result=result_compare(itr,model_0)
result.T
```

[112]:
```
         0  1  2  3  4  5  6  7  8  9   … 20 21 22 23 24 25 26 27 28 29
实际汉字  倉  之  扶  丙  世  憒  愛  俠  仪  上  …  振  愈  戈  恳  主  承  思 ␣
  ↪伐  耻  托
识别结果  仓  之  伏  丙  世  愤  愛  怀  修  上  …  振  忠  戈  懇  主  承  忽 ␣
  ↪戎  承  托

[2 rows x 30 columns]
```

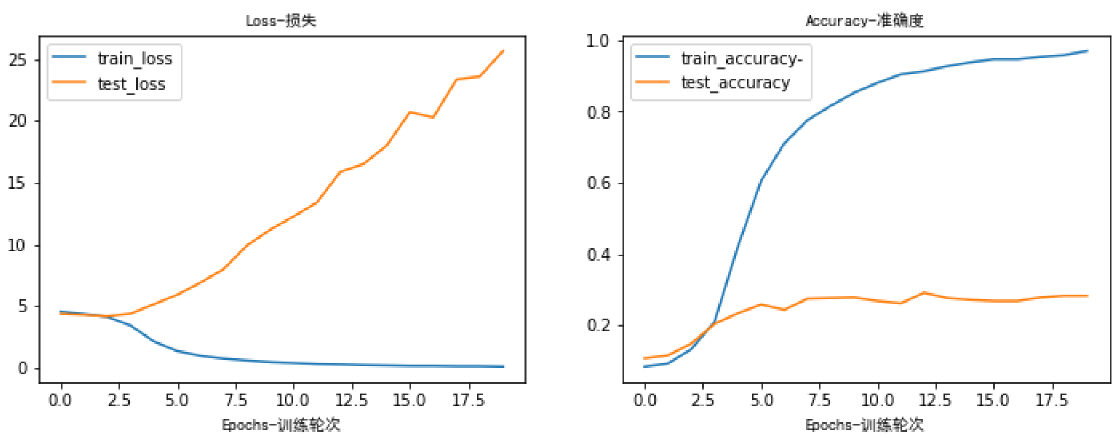
[113]:
```
plot_loss_curves(model_0_results)
```



[114]:
```python
writer_result=writer_result_compare(writer_itr,writer_model_0)
writer_result.T
```

[114]:
```
         0    1    2    3    4    5    6    7    8    9   …   11   12   13   14  \
书写人   解缙   赵构  苏轼  王铎   唐寅  柳公权  蔡襄  赵孟頫  欧阳询  敬世江  …  ␣
↪杨维桢  王羲之  弘历  陆柬之
识别结果  智永  欧阳询  汇辑  王铎  敬世江  王羲之  张照  赵孟頫   李邕  敬世江  …␣
↪  宋克  王献之  苏轼  敬世江

          15   16   17   18   19   20
书写人     近人  黄庭坚  沈粲   智果   字汇  文征明
识别结果  赵孟頫  李世民  苏轼  王羲之  董其昌  王献之

[2 rows x 21 columns]
```

[115]:
```
plot_loss_curves(writer_model_0_results)
```



0.0.17 保存和加载训练好的模型

- `torch.save` - 保存 PyTorch 模型或模型的参数 `state_dict()`.
- `torch.load` - 加载已保存的 PyTorch 对象.
- `torch.nn.Module.load_state_dict()` - 加载通过保存的 `state_dict()` 模型参数到新的 model 实例中.

[116]:
```
from pathlib import Path

# 创建用于保存模型的文件夹 (如果已存在则不操作), see: https://docs.python.org/3/
↪library/pathlib.html#pathlib.Path.mkdir
MODEL_PATH = Path("data/models")
MODEL_PATH.mkdir(parents=True, #
                 exist_ok=True # 如果路径存在也不报错
)

MODEL_NAME = "CalligraphyRegTinyVGG.pth"
MODEL_SAVE_PATH = MODEL_PATH / MODEL_NAME
MODEL_WRITER_NAME = "CalligraphyWriterRegTinyVGG.pth"
MODEL_WRITER_SAVE_PATH = MODEL_PATH / MODEL_WRITER_NAME
```

[117]:
```
# 保存模型的 state dict
print(f"Saving word regcognizer model to: {MODEL_SAVE_PATH}, word writer␣
↪recognizer model to :{MODEL_WRITER_SAVE_PATH}")

torch.save(obj=model_0.state_dict(), # 只保存 state_dict() 中可学习的参数
           f=MODEL_SAVE_PATH)
torch.save(obj=writer_model_0.state_dict(), # 只保存 state_dict() 中可学习的参数
           f=MODEL_WRITER_SAVE_PATH)
```

```
Saving word regcognizer model to: data\models\CalligraphyRegTinyVGG.pth, word
writer recognizer model to :data\models\CalligraphyWriterRegTinyVGG.pth
```

[118]:
```
# 创建一个和保存的参数具有相同结构的模型实例, 否则会报错
loaded_model_0 = TinyVGG(input_shape=3, # number of color channels (3 for RGB)
                  hidden_units=20,
                  output_shape=len(data_custom.classes))
# 加载 state_dict()
loaded_model_0.load_state_dict(torch.load(f=MODEL_SAVE_PATH,map_location=torch.
↪device(device)))

# 将模型发送到相应的 device
loaded_model_0 = loaded_model_0.to(device)
```

[119]:
```
# 创建一个和保存的参数具有相同结构的模型实例，否则会报错
loaded_writer_model_0 = WriterTinyVGG(input_shape=3, # number of color channels␣
 ↪(3 for RGB)
                    hidden_units=20,
                    output_shape=len(data_writer_custom.writer_classes))
# 加载 state_dict()
loaded_writer_model_0.load_state_dict(torch.
 ↪load(f=MODEL_WRITER_SAVE_PATH,map_location=torch.device(device)))

# 将模型发送到相应的 device
loaded_writer_model_0 = loaded_writer_model_0.to(device)
```

[120]:
```
print(type(loaded_model_0.state_dict()))  # 查看 state_dict 所返回的类型，是一个
“顺序字典 OrderedDict”

for param_tensor in loaded_model_0.state_dict(): # 字典的遍历默认是遍历 key，所以
param_tensor 实际上是键值
    print(param_tensor,'\t',loaded_model_0.state_dict()[param_tensor].shape)
```

```
<class 'collections.OrderedDict'>
conv_block_1.0.weight    torch.Size([20, 3, 3, 3])
conv_block_1.0.bias      torch.Size([20])
conv_block_1.2.weight    torch.Size([20, 20, 3, 3])
conv_block_1.2.bias      torch.Size([20])
conv_block_2.0.weight    torch.Size([20, 20, 3, 3])
conv_block_2.0.bias      torch.Size([20])
conv_block_2.2.weight    torch.Size([20, 20, 3, 3])
conv_block_2.2.bias      torch.Size([20])
classifier.1.weight      torch.Size([483, 5120])
classifier.1.bias        torch.Size([483])
```

[121]:
```
print(type(loaded_writer_model_0.state_dict()))  # 查看 state_dict 所返回的类型，
是一个“顺序字典 OrderedDict”

for param_tensor in loaded_writer_model_0.state_dict(): # 字典的遍历默认是遍历␣
 ↪key，所以 param_tensor 实际上是键值
```

```
    print(param_tensor,'\t',loaded_writer_model_0.state_dict()[param_tensor].
↪shape)
```

```
<class 'collections.OrderedDict'>
conv_block_1.0.weight    torch.Size([20, 3, 3, 3])
conv_block_1.0.bias      torch.Size([20])
conv_block_1.2.weight    torch.Size([20, 20, 3, 3])
conv_block_1.2.bias      torch.Size([20])
conv_block_2.0.weight    torch.Size([20, 20, 3, 3])
conv_block_2.0.bias      torch.Size([20])
conv_block_2.2.weight    torch.Size([20, 20, 3, 3])
conv_block_2.2.bias      torch.Size([20])
classifier.1.weight      torch.Size([447, 5120])
classifier.1.bias        torch.Size([447])
```

[122]:
```
for i,param in enumerate(loaded_model_0.parameters()):
    print(i,param.shape)
```

```
0 torch.Size([20, 3, 3, 3])
1 torch.Size([20])
2 torch.Size([20, 20, 3, 3])
3 torch.Size([20])
4 torch.Size([20, 20, 3, 3])
5 torch.Size([20])
6 torch.Size([20, 20, 3, 3])
7 torch.Size([20])
8 torch.Size([483, 5120])
9 torch.Size([483])
```

0.0.18 使用预训练模型作预测

[123]:
```
result=result_compare(itr,loaded_model_0)
result.T
```

[123]:
```
        0  1  2  3  4  5  6  7  8  9  … 20 21 22 23 24 25 26 27 28 29
实际汉字  戀  东  惬  乞  不  披  严  亢  併  惟  …  为  挂  揚  慘  挟  愧  扃 ␣
↪掛  两  依
```

```
识别结果  戀  东  悈  慈  不  披  严  丸  並  惟  …  思  伴  拈  慘  挾  愧  扃
↪伴  两  依

[2 rows x 30 columns]
```

```
[124]: writer_result=writer_result_compare(writer_itr,loaded_writer_model_0)
       writer_result.T
```

```
[124]:      0    1    2   3    4    5   6     7   8    9  …  11   12   13   14  \
书写人   王羲之  欧阳询   饶介  林逋  敬世江  康里子山  姚绶  王知敬  苏轼  赵孟頫
↪ …  颜真卿   汇辑  陆柬之  文征明
识别结果  赵孟頫  王羲之  王羲之  王铎  王羲之   唐寅  姚绶  王羲之  苏轼  敬世江
↪ …   赵构  赵孟頫  陆柬之  赵孟頫

      15   16   17   18  19  20
书写人   张羽  何绍基  李邕  康有为  米芾  解缙
识别结果  张羽  何绍基  李邕   唐寅  米芾  唐寅

[2 rows x 21 columns]
```

[125]:
```
def get_image_by_file_name(image_path,show=True)->(str,str,Image):

    '''
    定义函数 get_image_by_file_name，根据图片文件名返回图片内容，并显示该图片
    Args:
        image_path (str): 文字图片路径和文件名.
        show (Boolean): 是否显示文字图片

    Returns:
        img:图片内容

    Example:
        data\wordlib\予 _ 行书 _ 鲜于枢 _12046.gif "_" 前面的字符是书法对应的文字

    '''
    print(image_path)
    img=Image.open(image_path).convert('RGB')  #   丁 _ 草书 _ 王铎 _131029.
↪gif data/wordlib/zxqsig.jpg
    if show:
            plt.figure(figsize=(2, 2))
            plt.imshow(img)
```

```
            plt.title(f"图片 size(H,W) 为:({img.height}, {img.
↪width})",fontsize=16,fontproperties='Simhei')
            plt.axis(False)
    return img
```

[126]:
```
def predict_by_image_name(image_path,model):
    model.eval()
    with torch.inference_mode():
        query_image=get_image_by_file_name(image_path,show=True)
        img=test_transforms(query_image).unsqueeze(0).to(device)
        pred_label=torch.argmax(model(img),dim=1)
        print(f'\n图片文字预测为:\"{images_classes_list[pred_label]}\", 其 Label
为{pred_label.item()}')
    return images_classes_list[pred_label], pred_label.item()
```

[127]:
```
def predict_writer_by_image_name(image_path,model):
    model.eval()
    with torch.inference_mode():
        query_image=get_image_by_file_name(image_path,show=True)
        img=test_transforms(query_image).unsqueeze(0).to(device)
        pred_label=torch.argmax(model(img),dim=1)
        print(f'\n图片文字预测为:\"{images_writer_classes_list[pred_label]}\", 其
Label 为{pred_label.item()}')
    return images_writer_classes_list[pred_label], pred_label.item()
```

[128]:
```
predict_writer_by_image_name(r'data\wordlib\擬 _ 行书 _ 苏轼 _31314.
↪gif',loaded_writer_model_0)
```

```
data\wordlib\擬 _ 行书 _ 苏轼 _31314.gif

图片文字预测为:" 苏轼", 其 Label 为 307
```

[128]:
```
('苏轼', 307)
```

0.0.19 更多参考

- PyTorch `Dataset` and `DataLoader`datasets and dataloaders tutorial notebook.
- PyTorch `torchvision.transforms`documentation.
- Demos of `transforms` in action in the illustrations of transforms tutorial.
- PyTorch `torchvision.datasets` documentation.

[ ]:

```

```