In [ ]:

```
# This Python 3 environment comes with many helpful analytics libraries installe
# It is defined by the kaggle/python Docker image: https://github.com/kaggle/doc
# For example, here's several helpful packages to load

import numpy as np # linear algebra
import pandas as pd # data processing, CSV file I/O (e.g. pd.read_csv)

# Input data files are available in the read-only "../input/" directory
# For example, running this (by clicking run or pressing Shift+Enter) will list

import os
# for dirname, _, filenames in os.walk('/kaggle/input'):
#     for filename in filenames:
#         print(os.path.join(dirname, filename))

# You can write up to 20GB to the current directory (/kaggle/working/) that gets
# You can also write temporary files to /kaggle/temp/, but they won't be saved o
```

In [ ]:

```python
import os
import sys

import cv2
import numpy as np
import pandas as pd
from PIL import Image
from matplotlib import pyplot as plt
import seaborn as sns
import time
import random
import shutil

from easydict import EasyDict
from tqdm import tqdm

import scipy as sp
from sklearn.model_selection import StratifiedKFold, GroupKFold, KFold  # 交叉
from tqdm.auto import tqdm

import torch
import torch.nn as nn
import torch.nn.functional as F
from torch.optim import Adam, SGD, AdamW

import torchvision.models as models
from torch.utils.data import DataLoader, Dataset
from torch.optim.lr_scheduler import CosineAnnealingWarmRestarts, CosineAnneal
import albumentations as A
from albumentations.pytorch import ToTensorV2
import timm

#import loss_func
from torch.cuda.amp import autocast, GradScaler
import warnings

warnings.filterwarnings('ignore')
```

In [ ]:

```python
CFG = EasyDict({
    "model_name":"resnet50",
    "num_class": 10,
    "image_size":(32,32),
    "pretrained":True,
    "epochs":5,
    "batch_size":64,
    "num_workers":2,
    "device":torch.device('cuda'),
    "size_h": 32,
    "size_w": 32,
    "lr":3e-4,
    "weight_decay":1e-6,

})
OUTPUT_DIR = './'
```

In [ ]:

```python
train = pd.read_csv("/kaggle/input/boolart-image-classification/train.csv")
train
```

In [ ]:

```python
# ====================================================
# Dataset
# ====================================================
class TrainDataset(Dataset):
    def __init__(self,df,transform=None):
        self.df = df
        self.file_names = df['id'].values # 获取图片文件名
        self.labels = df['target'].values # 获取训练集图片target值
        self.transform = transform

    def __len__(self):  # len(train_dataset) 调用
        return len(self.df)

    # 读取图片
    def __getitem__(self,idx): # 这里的idx如何读取呢？---通过 [num] 正常传入序号
        self.file_path = f'/kaggle/input/boolart-image-classification/train_imag
        image = np.array(Image.open(self.file_path).convert("RGB"))


        if self.transform:
            image = self.transform(image=image)['image']
        else:
            image = cv2.resize(image,(CFG.size_h,CFG.size_w)) # 和原码不一样
#             image = image[np.newaxis,:,:] # 添加一个新的轴
            image = torch.from_numpy(image).float() #  ndarray -> pytorch

        label = torch.tensor(self.labels[idx]).long() # tensor

        return image/255, label
```

In [ ]:

```python
def get_transform(*,data):
    if data == 'train':
        return A.Compose([

            A.Resize(CFG.size_w, CFG.size_h),
            A.HorizontalFlip(p=0.5), # 水平翻转
            A.VerticalFlip(p=0.5),   # 垂直翻转
#             A.RandomBrightnessContrast(p=0.2),
            ToTensorV2()  # 把数据转化为Pytorch格式
        ])
    elif data == 'valid':
        return A.Compose([
            A.Resize(CFG.size_w, CFG.size_h),
            ToTensorV2()  # 把数据转化为Pytorch格式
        ])
```

0.1 数据集定义

In [ ]:

```
full_train_ds = TrainDataset(train)
train_ds = TrainDataset(train[:28440],transform=get_transform(data='train'))
valid_ds   = TrainDataset(train[28440:],transform=get_transform(data='valid'))

train_loader = DataLoader(train_ds,batch_size=CFG.batch_size,pin_memory=True,dr
valid_loader = DataLoader(valid_ds,batch_size=CFG.batch_size*2,pin_memory=True,
```

In [ ]:

```
def show_images(imgs,num_rows,num_cols,titles=None,scale=1.5):
    figsize = (num_cols*scale,num_rows*scale)

    # 创建一个包含 num_rows行，num_cols列 的子图， figsize是显示绘图窗口的大小
    _,axes = plt.subplots(num_rows,num_cols,figsize=figsize)  # axes 轴
    axes = axes.flatten()

    for i, (ax,img) in enumerate(zip(axes,imgs)): # ax—一张图的轴 img—一张图的数
        if torch.is_tensor(img):
            # 图片张量
            img = img.permute(1,2,0).numpy()*255
            ax.imshow(img.astype(np.uint8))
        else:
            # PIL图片--这个数据集
            ax.imshow(img) # 把img画在ax底图上
        ax.axes.get_xaxis().set_visible(False) # set_visible(False) 隐藏坐标轴
        ax.axes.get_yaxis().set_visible(False)
        ax.set_title(y[i].item()) # 迭代y 在一个batch_size中
    return axes

X, y = next(iter(train_loader))  # X 为一个batch_size的图片的array， y为label
show_images(X, 8, 8, y) # 显示一个batch_size,且返回值为axes的值，也就是下面这些图
```

1 Model

In [ ]:

```
class CustomModel(nn.Module):
    def __init__(self, cfg, pretrained=False):
        super().__init__()
        self.cfg = cfg
        self.model = timm.create_model(self.cfg.model_name, pretrained=pretraine
        #print(self.model)

        if 'efficientnet' in self.cfg.model_name:
            self.n_features = self.model.classifier.in_features
            self.model.global_pool = nn.Identity()
            self.model.classifier = nn.Identity()

        elif 'resnet' in self.cfg.model_name:
            self.n_features = self.model.fc.in_features
            self.model.global_pool = nn.Identity()
            self.model.fc = nn.Identity()

        self.pooling = nn.AdaptiveAvgPool2d(1)
        self.classifier = nn.Sequential(
                            #nn.Conv2d(self.n_features, self.n_features // 8, 1)
                            #nn.LeakyReLU(),
                            #nn.BatchNorm2d(self.n_features // 8),
                            nn.Conv2d(self.n_features, 44, 1),
                            #nn.Sigmoid()
                        )

    def forward(self, x):
        bs = x.size(0) # 返回x的batch_size
        features = self.model(x)
        pool_feature = self.pooling(features)
        output = self.classifier(pool_feature).view(bs, -1)
        return output
```

1.1 定义训练和验证流程

In [ ]:

```
# ====================================================
# train, valid
# ====================================================
def train_fn(model, optimizer, train_loader, criterion, device):

    model.to(device)
    model.train()
    train_loss = []

    for step,  (images,  labels) in enumerate(train_loader):
        images = images.to(device)
        labels = labels.to(device)


        y_preds = model(images)
        loss = criterion(y_preds, labels)

        optimizer.zero_grad() # 清零梯度
        loss.backward() # 计算梯度

        optimizer.step() # 优化器更新

        train_loss.append(loss.item())

    return np.mean(train_loss)

def valid_fn(model, valid_loader, criterion, device):
    model.to(device)
    model.eval()
    eval_loss = []

    for step,  (images,  labels) in enumerate(valid_loader):

        images = images.to(device)
        labels = labels.to(device)
        output = model(images)

        loss = criterion(output, labels.long())
        eval_loss.append(loss.item())

    return np.mean(eval_loss)

```

In [ ]:

```
criterion = nn.CrossEntropyLoss()
model = CustomModel(CFG, pretrained=True)
optimizer = Adam(model.parameters(),  lr=CFG.lr)
```

In [ ]:

```
OUTPUT_DIR = ’,/’
if not os.path.exists(OUTPUT_DIR):
    os.makedirs(OUTPUT_DIR)
```

In [ ]:

```
for epoch in range(CFG.epochs):
    train_loss = train_fn(model,optimizer,train_loader,criterion,CFG.device)
    val_loss   = valid_fn(model,valid_loader,criterion,CFG.device)
    print(f"Epoch: {epoch+1},train loss: {train_loss:.4f},val loss: {val_loss:.4
```

训练文件保存

In [ ]:

```
torch.save({'model': model.state_dict()},OUTPUT_DIR + f'{CFG.model_name}_best_sc
```

2 加载测试数据

In [ ]:

```
test = '../input/boolart-image-classification/test_image/'
test_data = pd.read_csv("/kaggle/input/boolart-image-classification/sample_submi
```

In [ ]:

```
test_data
```

In [ ]:

```
class TestDataset(Dataset):
    def __init__(self,df,transform=None):
        self.df = df['id'].values
        self.transform=transform

    def __len__(self):
        return len(self.df)

    def __getitem__(self,idx):
        self.file_path = test + f"{self.df[idx]}.jpg"
        image = np.array(Image.open(self.file_path).convert("RGB"))

        if self.transform:
            image = self.transform(image=image)['image']
        else:
            image = image[np.newaxis,:,:]
            image = torch.from_numpy(image).float()

        return image/255,self.df[idx]

```

2.0.1 test_loader加载

In [ ]:

```
test_dataset  = TestDataset(test_data,transform=get_transform(data='valid'))
test_loader   = DataLoader(test_dataset,batch_size=CFG.batch_size,shuffle=False,
                           num_workers=CFG.num_workers)
```

3 推理

In [ ]:

```
def predict(model,models_path,test_loader,device):

    tk0 = tqdm(enumerate(test_loader), total=len(test_loader))
    pre = []
    image_id = []
    for i, (images,img_ids) in tk0:
        image_id += list(img_ids.numpy())
        images = images.to(device)

        for model_path in models_path:
            model.load_state_dict(torch.load(model_path)['model'])
            model.eval()
            with torch.no_grad():
#                 y_pred = F.softmax(model(images)).to('cpu').numpy()
                    y_pred = F.softmax(model(images),1)
            y_preds = y_pred.to('cpu').numpy()
        predictions = F.softmax(torch.from_numpy(y_preds),dim=1)
        _,predict_y = torch.max(predictions,dim=1)
        predict_y = np.array(predict_y).tolist()
        pre += predict_y

#     for step, batch in enumerate(test_loader):
#         output = model(batch["image"].to(device))
#         prediction = torch.argmax(output['prediction'],1)
#             predictions.append(prediction.cpu()).numpy() # 预测数据

#         predictions = np.concatenate(predictions,axis=0)
    return pre,image_id
```

In [ ]:

```
# model_path = ['./tf_efficientnet_b2_fold0_best_score.pth']

models_path = [OUTPUT_DIR + f'{CFG.model_name}_best_score.pth']
predictions,img_id = predict(model,models_path, test_loader, CFG.device)
```

4 Submission

In [ ]:

```
df = pd.DataFrame({
    "id":img_id,
    "predict":predictions
})
df.to_csv("./submission.csv",index=False)
df
```

In [ ]:

```

```