توضیحات پیاده سازی مقاله درس بینایی ماشین

عنوان مقاله:

Gastrointestinal abnormality detection and classification using empirical wavelet transform and deep convolutional neural network from endoscopic images

نگارنده:

شقایق حبیب زاده

بهار 1403

فصل1:

1-1:دیتاست ها و داده های مقاله:

دیتاست های این مقاله به طورعمومی در دستری بود به حجم 58گیگ.

متاسفانه امکان دانلود دیتاست را نداشتم پس از داده های یکسان و رفرنس های دیگر برای پیاده سازی استفاده کردم به همین دلیل من از چندین دیتا ست متفاوت برای بررسی و پیاده سازی این مقاله استفاده کردم.

1-2:رفرنس ها یا مراجع

با توجه به رفرنس های مقاله دیتاهایی که استفاده شده ترکیبی از چند مقاله است.

بنابراین دیتاست مورد استفاده به شکل یکسان مورد استفاده نیست.

1-3:روش جست و جوی دیتاست ها در گیت هاب:

1\_ابتدا کلمات متناسب با مقاله و عنوان مقاله در گیت هاب سرچ شد.

2\_در مرحله دوم کلمات کلیدی مقاله در گیت هاب سرچ شد.

قسمت اول:

import torch

import numpy as np

import cv2

from torch.utils.data import Dataset

from pathlib import Path

data\_path = Path('data')

class AngyodysplasiaDataset(Dataset):

def \_\_init\_\_(self, img\_paths: list, to\_augment=False, transform=None, mode='train', limit=None):

self.img\_paths = img\_paths

self.to\_augment = to\_augment

self.transform = transform

self.mode = mode

self.limit = limit

def \_\_len\_\_(self):

if self.limit is None:

return len(self.img\_paths)

else:

return self.limit

def \_\_getitem\_\_(self, idx):

if self.limit is None:

img\_file\_name = self.img\_paths[idx]

else:

img\_file\_name = np.random.choice(self.img\_paths)

img = load\_image(img\_file\_name)

if self.mode == 'train':

mask = load\_mask(img\_file\_name)

img, mask = self.transform(img, mask)

return to\_float\_tensor(img), torch.from\_numpy(np.expand\_dims(mask, 0)).float()

else:

mask = np.zeros(img.shape[:2])

img, mask = self.transform(img, mask)

return to\_float\_tensor(img), str(img\_file\_name)

def to\_float\_tensor(img):

return torch.from\_numpy(np.moveaxis(img, -1, 0)).float()

def load\_image(path):

img = cv2.imread(str(path))

return cv2.cvtColor(img, cv2.COLOR\_BGR2RGB)

def load\_mask(path):

mask = cv2.imread(str(path).replace('images', 'masks').replace(r'.jpg', r'\_a.jpg'), 0)

return (mask > 0).astype(np.uint8)

قسمت دوم

from pathlib import Path

import argparse

import cv2

import numpy as np

from tqdm import tqdm

def general\_dice(y\_true, y\_pred):

if y\_true.sum() == 0:

if y\_pred.sum() == 0:

return 1

else:

return 0

return dice(y\_true == 1, y\_pred == 1)

def general\_jaccard(y\_true, y\_pred):

if y\_true.sum() == 0:

if y\_pred.sum() == 0:

return 1

else:

return 0

return jaccard(y\_true == 1, y\_pred == 1)

def jaccard(y\_true, y\_pred):

intersection = (y\_true \* y\_pred).sum()

union = y\_true.sum() + y\_pred.sum() - intersection

return (intersection + 1e-15) / (union + 1e-15)

def dice(y\_true, y\_pred):

return (2 \* (y\_true \* y\_pred).sum() + 1e-15) / (y\_true.sum() + y\_pred.sum() + 1e-15)

if \_\_name\_\_ == '\_\_main\_\_':

parser = argparse.ArgumentParser()

arg = parser.add\_argument

arg('--train\_path', type=str, default='data/train/angyodysplasia/masks', help='path where train images with ground truth are located')

arg('--target\_path', type=str, default='predictions/UNet', help='path with predictions')

args = parser.parse\_args()

result\_dice = []

result\_jaccard = []

for file\_name in tqdm(list(Path(args.train\_path).glob('\*'))):

y\_true = (cv2.imread(str(file\_name), 0) > 255 \* 0.5).astype(np.uint8)

pred\_file\_name = Path(args.target\_path) / (file\_name.stem.replace('\_a', '') + '.png')

y\_pred = (cv2.imread(str(pred\_file\_name), 0) > 255 \* 0.5).astype(np.uint8)

result\_dice += [dice(y\_true, y\_pred)]

result\_jaccard += [jaccard(y\_true, y\_pred)]

print('Dice = ', np.mean(result\_dice), np.std(result\_dice))

print('Jaccard = ', np.mean(result\_jaccard), np.std(result\_jaccard))

قسمت سوم

"""

Script generates predictions, splitting original images into tiles, and assembling prediction back together

"""

import argparse

from prepare\_train\_val import get\_split

from dataset import AngyodysplasiaDataset

import cv2

from models import UNet, UNet11, UNet16, AlbuNet34

import torch

from pathlib import Path

from tqdm import tqdm

import numpy as np

import utils

# import prepare\_data

from torch.utils.data import DataLoader

from torch.nn import functional as F

from transforms import (ImageOnly,

Normalize,

CenterCrop,

DualCompose)

img\_transform = DualCompose([

CenterCrop(512),

ImageOnly(Normalize())

])

def get\_model(model\_path, model\_type):

"""

:param model\_path:

:param model\_type: 'UNet', 'UNet11', 'UNet16', 'AlbuNet34'

:return:

"""

num\_classes = 1

if model\_type == 'UNet11':

model = UNet11(num\_classes=num\_classes)

elif model\_type == 'UNet16':

model = UNet16(num\_classes=num\_classes)

elif model\_type == 'AlbuNet34':

model = AlbuNet34(num\_classes=num\_classes)

elif model\_type == 'UNet':

model = UNet(num\_classes=num\_classes)

else:

model = UNet(num\_classes=num\_classes)

state = torch.load(str(model\_path))

state = {key.replace('module.', ''): value for key, value in state['model'].items()}

model.load\_state\_dict(state)

if torch.cuda.is\_available():

return model.cuda()

model.eval()

return model

def predict(model, from\_file\_names, batch\_size: int, to\_path):

loader = DataLoader(

dataset=AngyodysplasiaDataset(from\_file\_names, transform=img\_transform, mode='predict'),

shuffle=False,

batch\_size=batch\_size,

num\_workers=args.workers,

pin\_memory=torch.cuda.is\_available()

)

for batch\_num, (inputs, paths) in enumerate(tqdm(loader, desc='Predict')):

inputs = utils.variable(inputs, volatile=True)

outputs = model(inputs)

for i, image\_name in enumerate(paths):

mask = (F.sigmoid(outputs[i, 0]).data.cpu().numpy() \* 255).astype(np.uint8)

h, w = mask.shape

full\_mask = np.zeros((576, 576))

full\_mask[32:32 + h, 32:32 + w] = mask

(to\_path / args.model\_type).mkdir(exist\_ok=True, parents=True)

cv2.imwrite(str(to\_path / args.model\_type / (Path(paths[i]).stem + '.png')), full\_mask)

if \_\_name\_\_ == '\_\_main\_\_':

parser = argparse.ArgumentParser()

arg = parser.add\_argument

arg('--model\_path', type=str, default='data/models/UNet', help='path to model folder')

arg('--model\_type', type=str, default='UNet', help='network architecture',

choices=['UNet', 'UNet11', 'UNet16', 'AlbuNet34'])

arg('--batch-size', type=int, default=4)

arg('--fold', type=int, default=-1, choices=[0, 1, 2, 3, 4, -1], help='-1: all folds')

arg('--workers', type=int, default=12)

args = parser.parse\_args()

if args.fold == -1:

for fold in [0, 1, 2, 3, 4]:

\_, file\_names = get\_split(fold)

print(len(file\_names))

model = get\_model(str(Path(args.model\_path).joinpath('model\_{fold}.pt'.format(fold=fold))),

model\_type=args.model\_type)

print('num file\_names = {}'.format(len(file\_names)))

output\_path = Path(args.model\_path)

output\_path.mkdir(exist\_ok=True, parents=True)

predict(model, file\_names, args.batch\_size, output\_path)

else:

\_, file\_names = get\_split(args.fold)

model = get\_model(str(Path(args.model\_path).joinpath('model\_{fold}.pt'.format(fold=args.fold))),

model\_type=args.model\_type)

print('num file\_names = {}'.format(len(file\_names)))

output\_path = Path(args.model\_path)

output\_path.mkdir(exist\_ok=True, parents=True)

predict(model, file\_names, args.batch\_size, output\_path)

آموزش قسمت اول

import argparse

import json

from pathlib import Path

from validation import validation\_binary

import torch

from torch import nn

from torch.optim import Adam

from torch.utils.data import DataLoader

import torch.backends.cudnn as cudnn

import torch.backends.cudnn

from models import UNet, UNet11, UNet16, LinkNet34, AlbuNet34

from loss import LossBinary

from dataset import AngyodysplasiaDataset

import utils

from prepare\_train\_val import get\_split

from transforms import (DualCompose,

ImageOnly,

Normalize,

HorizontalFlip,

Rotate,

CenterCrop,

RandomHueSaturationValue,

VerticalFlip)

def main():

parser = argparse.ArgumentParser()

arg = parser.add\_argument

arg('--jaccard-weight', default=0.3, type=float)

arg('--device-ids', type=str, default='0', help='For example 0,1 to run on two GPUs')

arg('--fold', type=int, help='fold', default=0)

arg('--root', default='runs/debug', help='checkpoint root')

arg('--batch-size', type=int, default=1)

arg('--limit', type=int, default=10000, help='number of images in epoch')

arg('--n-epochs', type=int, default=100)

arg('--lr', type=float, default=0.0001)

arg('--workers', type=int, default=12)

arg('--model', type=str, default='UNet', choices=['UNet', 'UNet11', 'LinkNet34', 'UNet16', 'AlbuNet34'])

args = parser.parse\_args()

root = Path(args.root)

root.mkdir(exist\_ok=True, parents=True)

num\_classes = 1

if args.model == 'UNet':

model = UNet(num\_classes=num\_classes)

elif args.model == 'UNet11':

model = UNet11(num\_classes=num\_classes, pretrained=True)

elif args.model == 'UNet16':

model = UNet16(num\_classes=num\_classes, pretrained=True)

elif args.model == 'LinkNet34':

model = LinkNet34(num\_classes=num\_classes, pretrained=True)

elif args.model == 'AlbuNet':

model = AlbuNet34(num\_classes=num\_classes, pretrained=True)

else:

model = UNet(num\_classes=num\_classes, input\_channels=3)

if torch.cuda.is\_available():

if args.device\_ids:

device\_ids = list(map(int, args.device\_ids.split(',')))

else:

device\_ids = None

model = nn.DataParallel(model, device\_ids=device\_ids).cuda()

loss = LossBinary(jaccard\_weight=args.jaccard\_weight)

cudnn.benchmark = True

def make\_loader(file\_names, shuffle=False, transform=None, limit=None):

return DataLoader(

dataset=AngyodysplasiaDataset(file\_names, transform=transform, limit=limit),

shuffle=shuffle,

num\_workers=args.workers,

batch\_size=args.batch\_size,

pin\_memory=torch.cuda.is\_available()

)

train\_file\_names, val\_file\_names = get\_split(args.fold)

print('num train = {}, num\_val = {}'.format(len(train\_file\_names), len(val\_file\_names)))

train\_transform = DualCompose([

CenterCrop(512),

HorizontalFlip(),

VerticalFlip(),

Rotate(),

ImageOnly(RandomHueSaturationValue()),

ImageOnly(Normalize())

])

val\_transform = DualCompose([

CenterCrop(512),

ImageOnly(Normalize())

])

train\_loader = make\_loader(train\_file\_names, shuffle=True, transform=train\_transform, limit=args.limit)

valid\_loader = make\_loader(val\_file\_names, transform=val\_transform)

root.joinpath('params.json').write\_text(

json.dumps(vars(args), indent=True, sort\_keys=True))

utils.train(

init\_optimizer=lambda lr: Adam(model.parameters(), lr=lr),

args=args,

model=model,

criterion=loss,

train\_loader=train\_loader,

valid\_loader=valid\_loader,

validation=validation\_binary,

fold=args.fold

)

if \_\_name\_\_ == '\_\_main\_\_':

main()

آموزش قسمت دوم

#!/usr/bin/env bash

export CUDA\_DEVICE\_ORDER=PCI\_BUS\_ID

export CUDA\_VISIBLE\_DEVICES=0,1,2,3

for i in 0 1 2 3 4

do

python train.py --device-ids 0,1,2,3 --limit 10000 --batch-size 16 --n-epochs 10 --fold $i --model UNet16

python train.py --device-ids 0,1,2,3 --limit 10000 --batch-size 16 --n-epochs 15 --fold $i --lr 0.00001 --model UNe16

done