توضیحات پیاده سازی مقاله درس بینایی ماشین

عنوان مقاله:

AI-enabled radiologist in the loop: novel AI-based framework to augment radiologist performance for COVID-19 chest CT medical image annotation and classification from pneumonia

عنوان مقاله به فارسی:

رادیولوژیست مجهز به هوش مصنوعی در حلقه: چارچوب جدید مبتنی بر هوش مصنوعی برای تقویت عملکرد رادیولوژیست برای حاشیه نویسی تصویر پزشکی سی تی سینه COVID-۱۹ و طبقه بندی از ذات الریه

نگارنده:

محمد اکبری

بهار 1403

فصل1:

1-1:دیتاست ها و داده های مقاله:

دیتاست های این مقاله به طورعمومی در دسترس نبود.

چرا که از پردازش چندین تصویر پزشکی یا همان تصاویر مدیکال برای جمع آوری دیتاست استفاده شد.

دیتاست مورد استفاده در این مقاله دیتاستی است که برای مقاله ای مشابه مقاله مورد انتخاب است.

2-1کد نویسی قسمت اول:

#!/usr/bin/env python3

from termcolor import cprint, colored as c

def inc(d, label):

if label in d:

d[label] += 1

else:

d[label] = 1

def precision\_recall(output, target):

assert len(output) == len(target), "output len: {} != target len: {}".format(len(output), len(target))

labels = set(target)

TP = {}

TP\_plus\_FN = {}

TP\_plus\_FP = {}

for i in range(len(output)):

inc(TP\_plus\_FN, target[i])

inc(TP\_plus\_FP, output[i])

if target[i] == output[i]:

inc(TP, output[i])

for label in labels:

if label not in TP\_plus\_FN:

TP\_plus\_FN[label] = 0

if label not in TP\_plus\_FP:

TP\_plus\_FP[label] = 0

precision = {label: 0. if TP\_plus\_FP[label] ==0 else ((TP[label] if label in TP else 0) / float(TP\_plus\_FP[label])) for label in labels}

recall = {label: 0. if TP\_plus\_FN[label] ==0 else ((TP[label] if label in TP else 0) / float(TP\_plus\_FN[label])) for label in labels}

return precision, recall, TP, TP\_plus\_FN, TP\_plus\_FP

def F\_score(p, r):

f\_scores = {

label: None if p[label] == 0 and r[label] == 0 else (0 if p[label] == 0 or r[label] == 0 else 2 / (1 / p[label] + 1 / r[label]))

for label in p

}

return f\_scores

def print\_f\_score(output, target):

"""returns:

p<recision>,

r<ecall>,

f<-score>,

{"TP", "p", "TP\_plus\_FP"} """

p, r, TP, TP\_plus\_FN, TP\_plus\_FP = precision\_recall(output, target)

f = F\_score(p, r)

# cprint("Label: " + c((" " + str(10))[-5:], 'red') +

# "\tPrec: " + c(" {:.1f}".format(0.335448 \* 100)[-5:], 'green') + '%' +

# " ({:d}/{:d})".format(1025, 1254).ljust(14) +

# "Recall: " + c(" {:.1f}".format(0.964 \* 100)[-5:], 'green') + "%" +

# " ({:d}/{:d})".format(15, 154).ljust(14) +

# "F-Score: " + (c(" {:.1f}".format(0.5 \* 100)[-5:], "green") + "%")

# )

for label in f.keys():

cprint("Label: " + c((" " + str(label))[-5:], 'red') +

"\tPrec: " + c(" {:.1f}".format(p[label] \* 100)[-5:], 'green') + '%' +

" ({:d}/{:d})".format((TP[label] if label in TP else 0), TP\_plus\_FP[label]).ljust(14) +

"Recall: " + c(" {:.1f}".format((r[label] if label in r else 0) \* 100)[-5:], 'green') + "%" +

" ({:d}/{:d})".format((TP[label] if label in TP else 0), TP\_plus\_FN[label]).ljust(14) +

"F-Score: " + (" N/A" if f[label] is None else (c(" {:.1f}".format(f[label] \* 100)[-5:], "green") + "%"))

)

# return p, r, f, \_

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

import torch

import torch.autograd as autograd

output = [1,1,1,1,1,2,0,2,2,2,2]

output = torch.LongTensor(output)

# target = [0,0,2,1,2,2,1,2,1,2,0]

target = [1,3,2,3,3,3,3,3,0,3,3]

target = torch.LongTensor(target)

output = autograd.Variable(output)

target = autograd.Variable(target)

print('output')

print(output.data.numpy().tolist())

print('target')

print(target.data.numpy().tolist())

precision, recall, TP, TP\_plus\_FN, TP\_plus\_FP = precision\_recall(output.data.numpy().tolist(), target.data.numpy().tolist())

print('precision')

print(precision)

print('recall')

print(recall)

print('TP')

print(TP)

print('TP\_plus\_FN')

print(TP\_plus\_FN)

print('TP\_plus\_FP')

print(TP\_plus\_FP)

# print(dic)

f\_scores = F\_score(precision, recall)

print('f\_scores')

print(f\_scores)

# print(f\_scores.keys())

print('\r')

print\_f\_score(output.data.numpy().tolist(), target.data.numpy().tolist())

3-1کد نویسی قسمت دوم:

import numpy as np

import pandas as pd

import os

import shutil

import matplotlib.pyplot as plt

import torch.nn as nn

import torch.nn.functional as F

import torch.optim as optim

import torch

import torchvision

from torchvision import datasets, transforms

from torch.utils.data import Dataset, DataLoader

from PIL import Image

from sklearn.preprocessing import MultiLabelBinarizer

import warnings

warnings.filterwarnings('ignore')

import torchvision.models as models

from sklearn.metrics import accuracy\_score, classification\_report, confusion\_matrix,roc\_auc\_score

import pickle

from metric import print\_f\_score

data\_transforms = {

'train': transforms.Compose([

transforms.Resize((224,224)),

transforms.RandomAffine(0, shear=0.2, scale=(0.8,1.2)),

transforms.RandomHorizontalFlip(),

transforms.ToTensor(),

transforms.Normalize([0.485, 0.456, 0.406], [0.229, 0.224, 0.225])

]),

'val': transforms.Compose([

transforms.Resize((224,224)),

transforms.ToTensor(),

transforms.Normalize([0.485, 0.456, 0.406], [0.229, 0.224, 0.225])

]),

}

def Load\_Image\_Information(path):

image\_Root\_Dir = './image\_Merge/'

iamge\_Dir = os.path.join(image\_Root\_Dir, path)

return Image.open(iamge\_Dir).convert('RGB')

class my\_Data\_Set(nn.Module):

def \_\_init\_\_(self, meta\_filepath, transform=None, target\_transform=None, loader=None):

super(my\_Data\_Set, self).\_\_init\_\_()

metainfo = pd.read\_csv(meta\_filepath)

images = []

symptom = []

for i in range(metainfo.shape[0]):

images.append(metainfo.iloc[i, 0])

symptom.append(metainfo.iloc[i, 1:].values.tolist())

mlb = MultiLabelBinarizer(classes=['CVD19', 'GGO', 'Csld', 'CrPa', 'Aibr', 'InSep'])

labels = np.array(mlb.fit\_transform(symptom), dtype=np.float64)

self.images = images

self.labels = labels

self.transform = transform

self.target\_transform = target\_transform

self.loader = loader

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

imageName = self.images[item]

label = self.labels[item]

image = self.loader(imageName)

if self.transform is not None:

image = self.transform(image)

label = torch.FloatTensor(label)

return image, label

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

return len(self.images)

class densenet121\_COVID(nn.Module):

def \_\_init\_\_(self):

super(densenet121\_COVID,self).\_\_init\_\_()

net = models.densenet121(pretrained=True)

num\_input = net.classifier.in\_features

net.classifier = nn.Linear(num\_input, 6)

self.densenet121\_out = net

def forward(self,x):

x=self.densenet121\_out(x)

# print(x.shape)

return F.sigmoid(x)

class loss\_fun(nn.Module):

def \_\_init\_\_(self):

super(loss\_fun, self).\_\_init\_\_()

def forward(self, output, target):

loss = torch.zeros((target.shape[0], target.shape[1])).to(device)

loss1 = self.\_\_loss\_nosym(output, target).to(device)

loss2 = self.\_\_loss\_sym(output, target).to(device)

for i in range(target.shape[0]):

w1 = torch.prod((target[i][1:] > 0).any()).to(device)

w2 = (1 - w1).to(device)

loss[i] = w2 \* loss1[i] + w1 \* loss2[i]

loss[:,0] = 5 \* loss[:,0]

return torch.mean(loss),torch.mean(loss1)

def \_\_loss\_nosym(self, output, target):

loss = torch.zeros((target.shape[0], target.shape[1]))

loss[:, 0] = criterion(output, target)[:, 0]

return loss

def \_\_loss\_sym(self, output, target):

loss = criterion(output, target)

return loss

def test(model, model\_path,test\_loader):

checkpoint = torch.load(model\_path)

model.load\_state\_dict(checkpoint['state\_dict'])

model.eval()

test\_loss = 0

correct = 0

predicates\_all, target\_all = [], []

with torch.no\_grad():

for data, target in test\_loader:

data, target = data.to(device), target.to(device)

output = model(data)

#print(output)

\_,covid\_loss=loss\_fun(output, target)

test\_loss += (covid\_loss\*data.shape[0]).item()

pred = output.ge(0.5).float()[:,0]

correct += pred.eq(target[:,0].data.view\_as(pred)).sum()

predicates\_all += pred.cpu().numpy().tolist()

target\_all += target[:,0].data.cpu().numpy().tolist()

test\_loss /= len(test\_loader.dataset)

correct=correct.item()

test\_acc=100. \* correct / len(test\_loader.dataset)

print('\nTest--> Avg. loss: {:.6f} acc: {:.3f}% '.format(test\_loss,test\_acc))

return predicates\_all, target\_all

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

LR = 0.00005

test\_dataset = my\_Data\_Set('./data\_split/test\_meta.csv', transform=data\_transforms['val'],

loader=Load\_Image\_Information)

if len(test\_dataset) < 4:

val\_test\_batchsize = 1

else:

val\_test\_batchsize = 4

test\_loader = DataLoader(test\_dataset, val\_test\_batchsize, shuffle=False)

cnn = densenet121\_COVID()

print(cnn)

device = torch.device("cuda:14" if torch.cuda.is\_available() else "cpu")

cnn.to(device)

optimizer = optim.Adam(cnn.parameters(), lr=LR, weight\_decay=0.0005)

criterion = nn.BCELoss(size\_average=False, reduce=False)

loss\_fun = loss\_fun()

model\_path = './best\_model/COVID19\_densenet121\_94\_97.path.tar'

predicates\_all, target\_all = test(cnn, model\_path, test\_loader)

print(predicates\_all)

print(target\_all)

print\_f\_score(predicates\_all, target\_all)

print(accuracy\_score(target\_all, predicates\_all))

print(classification\_report(target\_all, predicates\_all))

print("AUC",roc\_auc\_score(target\_all, predicates\_all))

print(confusion\_matrix(target\_all, predicates\_all))

4-1کد نویسی قسمت سوم آموزش مدل:

import numpy as np

import pandas as pd

import os

import shutil

import matplotlib.pyplot as plt

import torch.nn as nn

import torch.nn.functional as F

import torch.optim as optim

import torch

import torchvision

from torchvision import datasets, transforms

from torch.utils.data import Dataset, DataLoader

from PIL import Image

from sklearn.preprocessing import MultiLabelBinarizer

import warnings

warnings.filterwarnings('ignore')

import torchvision.models as models

from sklearn.metrics import accuracy\_score, classification\_report, confusion\_matrix,roc\_auc\_score

import pickle

from metric import print\_f\_score

data\_transforms = {

'train': transforms.Compose([

transforms.Resize((224,224)),

transforms.RandomAffine(0, shear=0.2, scale=(0.8,1.2)),

transforms.RandomHorizontalFlip(),

transforms.ToTensor(),

transforms.Normalize([0.485, 0.456, 0.406], [0.229, 0.224, 0.225])

]),

'val': transforms.Compose([

transforms.Resize((224,224)),

transforms.ToTensor(),

transforms.Normalize([0.485, 0.456, 0.406], [0.229, 0.224, 0.225])

]),

}

def Load\_Image\_Information(path):

image\_Root\_Dir = './image\_Merge/'

iamge\_Dir = os.path.join(image\_Root\_Dir, path)

return Image.open(iamge\_Dir).convert('RGB')

class my\_Data\_Set(nn.Module):

def \_\_init\_\_(self, meta\_filepath, transform=None, target\_transform=None, loader=None):

super(my\_Data\_Set, self).\_\_init\_\_()

metainfo = pd.read\_csv(meta\_filepath)

images = []

symptom = []

for i in range(metainfo.shape[0]):

images.append(metainfo.iloc[i, 0])

symptom.append(metainfo.iloc[i, 1:].values.tolist())

mlb = MultiLabelBinarizer(classes=['CVD19', 'GGO', 'Csld', 'CrPa', 'Aibr', 'InSep'])

labels = np.array(mlb.fit\_transform(symptom), dtype=np.float64)

self.images = images

self.labels = labels

self.transform = transform

self.target\_transform = target\_transform

self.loader = loader

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

imageName = self.images[item]

label = self.labels[item]

image = self.loader(imageName)

if self.transform is not None:

image = self.transform(image)

label = torch.FloatTensor(label)

return image, label

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

return len(self.images)

class densenet169\_COVID(nn.Module):

def \_\_init\_\_(self):

super(densenet169\_COVID,self).\_\_init\_\_()

net = models.densenet169(pretrained=True)

num\_input = net.classifier.in\_features

net.classifier = nn.Linear(num\_input, 6)

self.densenet169\_out = net

def forward(self,x):

x=self.densenet169\_out(x)

# print(x.shape)

return F.sigmoid(x)

def save\_checkpoint(model, state, filename):

# model\_is\_cuda = next(model.parameters()).is\_cuda

# model = model.module if model\_is\_cuda else model

state['state\_dict'] = model.state\_dict()

torch.save(state,filename)

class loss\_fun(nn.Module):

def \_\_init\_\_(self):

super(loss\_fun, self).\_\_init\_\_()

def forward(self, output, target):

loss = torch.zeros((target.shape[0], target.shape[1])).to(device)

loss1 = self.\_\_loss\_nosym(output, target).to(device)

loss2 = self.\_\_loss\_sym(output, target).to(device)

for i in range(target.shape[0]):

w1 = torch.prod((target[i][1:] > 0).any()).to(device)

w2 = (1 - w1).to(device)

loss[i] = w2 \* loss1[i] + w1 \* loss2[i]

loss[:,0] = 5 \* loss[:,0]

return torch.mean(loss),torch.mean(loss1)

def \_\_loss\_nosym(self, output, target):

loss = torch.zeros((target.shape[0], target.shape[1]))

loss[:, 0] = criterion(output, target)[:, 0]

return loss

def \_\_loss\_sym(self, output, target):

loss = criterion(output, target)

return loss

def train(model,n\_epochs,train\_loader,val\_loader):

## per epoch

train\_losses = []

train\_acces=[]

val\_losses=[]

val\_acces=[]

best\_acc=0

min\_loss=10

for epoch in range(1,n\_epochs+1):

##per batch average len(train\_loader.dataset)/batch

train\_allloss=[]

val\_allloss=[]

## batch

train\_correct=0

val\_correct=0

#######################################################

model.train()

for batch\_idx, (data, target) in enumerate(train\_loader):

data, target = data.to(device), target.to(device)

output = model(data)

bp\_loss ,covid\_loss= loss\_fun(output, target)

optimizer.zero\_grad()

bp\_loss.backward()

optimizer.step()

train\_allloss.append(covid\_loss.item())

pred = output.ge(0.5).float()[:,0]

train\_correct += pred.eq(target[:,0].data.view\_as(pred)).sum()##batch

train\_loss=np.average(train\_allloss)

train\_losses.append(train\_loss)

train\_correct=train\_correct.item()

train\_acc=100.\*train\_correct/len(train\_loader.dataset)

train\_acces.append(float(train\_acc))

########################################################

model.eval()

for data,target in val\_loader:

data, target = data.to(device), target.to(device)

output=model(data)

\_,covid\_loss=loss\_fun(output, target)

val\_allloss.append(covid\_loss.item())

pred = output.ge(0.5).float()[:,0]

val\_correct += pred.eq(target[:,0].data.view\_as(pred)).sum()##batch

val\_loss=np.average(val\_allloss)

val\_losses.append(val\_loss)

val\_correct=val\_correct.item()

val\_acc=100.\*val\_correct/len(val\_loader.dataset)

val\_acces.append(float(val\_acc))

#######################################################

print('\ntrain--> Epoch[{}] loss: {:.6f} acc: {:.3f}% '.format(epoch,train\_loss,train\_acc))

print('Evaluation--> Epoch[{}] loss: {:.6f} acc: {:.3f}% '.format(epoch,val\_loss,val\_acc))

# if val\_acc>best\_acc:

# model\_path='./model/COVID19\_multbest.path.tar'

# print("\n=> found better validated model, saving to %s" % model\_path)

# save\_checkpoint(model,{'epoch':epoch,

# 'optimizer':optimizer.state\_dict(),

# 'best\_acc':best\_acc},

# model\_path)

# best\_acc=val\_acc

if val\_loss < min\_loss:

model\_path='./model/COVID19\_multbest.path.tar'

print("\n=> found better validated model, saving to %s" % model\_path)

save\_checkpoint(model,{'epoch':epoch,

'optimizer':optimizer.state\_dict(),

'best\_acc':best\_acc},

model\_path)

min\_loss=val\_loss

return train\_losses,train\_acces,val\_losses,val\_acces

def test(model, model\_path,test\_loader):

checkpoint = torch.load(model\_path)

model.load\_state\_dict(checkpoint['state\_dict'])

model.eval()

test\_loss = 0

correct = 0

predicates\_all, target\_all = [], []

with torch.no\_grad():

for data, target in test\_loader:

data, target = data.to(device), target.to(device)

output = model(data)

\_,covid\_loss=loss\_fun(output, target)

test\_loss += (covid\_loss\*data.shape[0]).item()

pred = output.ge(0.5).float()[:,0]

correct += pred.eq(target[:,0].data.view\_as(pred)).sum()

predicates\_all += pred.cpu().numpy().tolist()

target\_all += target[:,0].data.cpu().numpy().tolist()

test\_loss /= len(test\_loader.dataset)

correct=correct.item()

test\_acc=100. \* correct / len(test\_loader.dataset)

print('\nTest--> Avg. loss: {:.6f} acc: {:.3f}% '.format(test\_loss,test\_acc))

return predicates\_all, target\_all

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

LR = 0.00005

n\_epochs = 15

train\_batch\_size = 32

val\_test\_batchsize = 4

print('LR', LR)

print('n\_epochs', n\_epochs)

print('train\_batch\_size', train\_batch\_size)

print('val\_test\_batchsize', val\_test\_batchsize)

train\_dataset = my\_Data\_Set('./data\_split/train\_meta.csv', transform=data\_transforms['train'],

loader=Load\_Image\_Information)

val\_dataset = my\_Data\_Set('./data\_split/val\_meta.csv', transform=data\_transforms['val'],

loader=Load\_Image\_Information)

test\_dataset = my\_Data\_Set('./data\_split/test\_meta.csv', transform=data\_transforms['val'],

loader=Load\_Image\_Information)

train\_loader = DataLoader(train\_dataset, train\_batch\_size, shuffle=True)

val\_loader = DataLoader(val\_dataset, val\_test\_batchsize, shuffle=True)

test\_loader = DataLoader(test\_dataset, val\_test\_batchsize, shuffle=False)

cnn = densenet169\_COVID()

print(cnn)

device = torch.device("cuda:0" if torch.cuda.is\_available() else "cpu")

cnn.to(device)

optimizer = optim.Adam(cnn.parameters(), lr=LR, weight\_decay=0.0005)

criterion = nn.BCELoss(size\_average=False, reduce=False)

loss\_fun = loss\_fun()

train\_losses, train\_acces, val\_losses, val\_acces = train(cnn, n\_epochs, train\_loader, val\_loader)

train\_losses, train\_acces, val\_losses, val\_acces = np.array(train\_losses), np.array(train\_acces), np.array(

val\_losses), np.array(val\_acces)

result = np.vstack((train\_losses, train\_acces, val\_losses, val\_acces))

######save

def save\_result(result):

filename = 'resluts\_train\_val.pkl'

with open(filename, 'wb') as fo:

pickle.dump(result, fo)

save\_result(result)

plt.plot(train\_acces)

plt.plot(val\_acces)

plt.title('model accuracy')

plt.ylabel('accuracy')

plt.xlabel('epoch')

plt.legend(['train', 'val'], loc='lower right')

plt.savefig('./mult\_acc.png', format='png', dpi=80)

plt.show()

plt.plot(train\_losses)

plt.plot(val\_losses)

plt.title('model loss')

plt.ylabel('loss')

plt.xlabel('epoch')

plt.legend(['train', 'val'], loc='upper right')

plt.savefig('./mult\_loss.png', format='png', dpi=80)

plt.show()

model\_path = './model/COVID19\_multbest.path.tar'

predicates\_all, target\_all = test(cnn, model\_path, test\_loader)

print\_f\_score(predicates\_all, target\_all)

print(accuracy\_score(target\_all, predicates\_all))

print(classification\_report(target\_all, predicates\_all))

print("AUC", roc\_auc\_score(target\_all, predicates\_all))

print(confusion\_matrix(target\_all, predicates\_all))