train.py | oct-11-train.py

1 Addition, 21 Deletions, 9 Changes

A train.py

```
import shutil
import os
import time
from datetime import datetime
import random
import argparse
import numpy as np
- import torchviz¬
from tqdm import tqdm

import torch
import torch.nn as nn
import torch.optim as optim
import torch.nn.functional as F
- from sklearn.metrics import roc_curve¬
from tensorboardX import SummaryWriter

from dataloader import MRDataset
from models.mrnet import MRNet

from sklearn import metrics
import csv
import utils as ut

- # torch.autograd.set_detect_anomaly(True)¬

def train_model(model, train_loader, epoch, num_epochs, optimizer, writer, current_lr, device, log_every=100):
    """
    Procedure to train a model on the training set
    """
    model.train()

    model = model.to(device)

    y_preds = []
    y_trues = []
    losses = []

    for i, (image, label, weight) in enumerate(train_loader):

        image = image.to(device)
        label = label.to(device)
        weight = weight.to(device)

        prediction = model(image.float())

        loss = F.binary_cross_entropy_with_logits(prediction, label, weight=weight)
¬
        optimizer.zero_grad()
        loss.backward()
        optimizer.step()

        loss_value = loss.item()
        losses.append(loss_value)

```

B oct-11-train.py

```
import shutil
import os
import time
from datetime import datetime
import random
import argparse
import numpy as np
from tqdm import tqdm

import torch
import torch.nn as nn
import torch.optim as optim
import torch.nn.functional as F
from tensorboardX import SummaryWriter

from dataloader import MRDataset
from models.mrnet import MRNet

from sklearn import metrics
import csv
import utils as ut


def train_model(model, train_loader, epoch, num_epochs, optimizer, writer, current_lr, device, log_every=100):
    """
    Procedure to train a model on the training set
    """
    model.train()

    model = model.to(device)

    y_preds = []
    y_trues = []
    losses = []

    for i, (image, label, weight) in enumerate(train_loader):

        image = image.to(device)
        label = label.to(device)
        weight = weight.to(device)

        prediction = model(image.float())

        loss = F.binary_cross_entropy_with_logits(prediction, label, weight=weight)
        ¬
        optimizer.zero_grad()
        loss.backward()
        optimizer.step()

        loss_value = loss.item()
        losses.append(loss_value)

```

A train.py

```
        probas = torch.sigmoid(prediction)

        y_trues.append(int(label[0]))
        y_preds.append(probas[0].item())

        try:
            auc = metrics.roc_auc_score(y_trues, y_preds)
-           accuracy = metrics.accuracy_score(y_trues, (np.array(y_preds) > 0.5).astype(int))
-           sensitivity = metrics.recall_score(y_trues, (np.array(y_preds) > 0.5).astype(int))
-           specificity = metrics.recall_score(1 - np.array(y_trues), 1 - (np.array(y_preds) > 0.5).astype(int))
        except:
            auc = 0.5
-           accuracy = 0.5
-           sensitivity = 0.5
-           specificity = 0.5

        writer.add_scalar('Train/Loss', loss_value,
                                      epoch * len(train_loader) + i)
        writer.add_scalar('Train/AUC', auc, epoch * len(train_loader) + i)

        if (i % log_every == 0) & (i > 0):
-           print(
                '''[Epoch: {0} / {1} |Single batch number : {2} / {3} ]| avg train loss {4} | train auc : {5} | lr : {6}'''.
                format(
                    epoch + 1,
                    num_epochs,
                    i,
                    len(train_loader),
                    np.round(np.mean(losses), 4),
                    np.round(auc, 4),
                    current_lr
                )
            )

    writer.add_scalar('Train/AUC_epoch', auc, epoch)

    train_loss_epoch = np.round(np.mean(losses), 4)
    train_auc_epoch = np.round(auc, 4)
    train_accuracy_epoch = np.round(accuracy, 4)
    train_sensitivity_epoch = np.round(sensitivity, 4)
-   train_specificity_epoch = np.round(specificity, 4)


-   return train_loss_epoch, train_auc_epoch, train_accuracy_epoch, train_sensitivity_epoch, train_specificity_epoch
-
```

B oct-11-train.py

```
        probas = torch.sigmoid(prediction)

        y_trues.append(int(label[0]))
        y_preds.append(probas[0].item())

        try:
            auc = metrics.roc_auc_score(y_trues, y_preds)
        except:
            auc = 0.5

        writer.add_scalar('Train/Loss', loss_value,
                                      epoch * len(train_loader) + i)
        writer.add_scalar('Train/AUC', auc, epoch * len(train_loader) + i)

        if (i % log_every == 0) & (i > 0):
            print('''[Epoch: {0} / {1} | Single batch number : {2} / {3} ]| avg train loss {4} | train auc : {5} | lr : {6}'''.
                  format(
                      epoch + 1,
                      num_epochs,
                      i,
                      len(train_loader),
                      np.round(np.mean(losses), 4),
                      np.round(auc, 4),
                      current_lr
                  )
                  )

    writer.add_scalar('Train/AUC_epoch', auc, epoch)

    train_loss_epoch = np.round(np.mean(losses), 4)
    train_auc_epoch = np.round(auc, 4)

    return train_loss_epoch, train_auc_epoch


```

A train.py

```
- ¬
def evaluate_model(model, val_loader, epoch, num_epochs, writer, current_lr, device, log_every=20, return_predictions=False):¬
    """
    Procedure to evaluate a model on the validation set
    """
    model.eval()

    y_trues = []
    y_preds = []
    y_class_preds = []
-    adv_y_trues = []¬
-    adv_y_preds = []¬
-    adv_y_class_preds = []¬
    losses = []
-    adv_losses = []¬
-    percent = args.advtrain_percent¬
- ¬
-    # running adv validation¬
-    for i, (image, label, weight) in enumerate(val_loader):¬
-        stop_on = int(1130 * percent)¬
-        if i > stop_on:¬
-            break¬
-        image = image.to(device)¬
-        label = label.to(device)¬
-        weight = weight.to(device)¬
-        epsilon = args.epsilon¬
-        adv_image = fgsm_attack(model, F.binary_cross_entropy_with_logits, image, label, weight, epsilon, device="mps")¬
-        adv_prediction = model.forward(adv_image.float())¬
-        adv_loss = F.binary_cross_entropy_with_logits(adv_prediction, label, weight=weight)¬
-        adv_loss_value = adv_loss.item()¬
-        adv_losses.append(adv_loss_value)¬
-        adv_probas = torch.sigmoid(adv_prediction)¬
-        adv_y_trues.append(int(label[0]))¬
-        adv_y_preds.append(adv_probas[0].item())¬
-        adv_y_class_preds.append((adv_probas[0] > 0.5).float().item())¬
- ¬
-        try:¬
-            adv_auc = metrics.roc_auc_score(adv_y_trues, adv_y_preds)¬
-        except:¬
-            adv_auc = 0.5¬
- ¬
-    writer.add_scalar('Adv Val/Loss', adv_loss_value, epoch * len(val_loader) + i)¬
-    writer.add_scalar('Adv Val/AUC', adv_auc, epoch * len(val_loader) + i)¬
-    writer.add_scalar('Adv Val/AUC_epoch', adv_auc, epoch)¬
-    print('Adv Val/AUC_epoch', adv_auc, epoch)¬
-    val_adv_loss_epoch = np.round(np.mean(adv_losses), 4)¬
```

B oct-11-train.py

```
def evaluate_model(model, val_loader, epoch, num_epochs, writer, current_lr, device, log_every=20):¬
    """
    Procedure to evaluate a model on the validation set
    """
    model.eval()

    y_trues = []
    y_preds = []
    y_class_preds = []
    losses = []
```

A train.py

```
-    val_adv_auc_epoch = np.round(adv_auc, 4)¬
-    try:¬
-        print("adv_y_trues: ", adv_y_trues, "adv_y_class_preds: ", adv_y_class_preds)¬
-        val_adv_accuracy, val_adv_sensitivity, val_adv_specificity = ut.accuracy_sensitivity_specificity(adv_y_trues, adv_y_class_preds)¬
-        val_adv_accuracy = np.round(val_adv_accuracy, 4)¬
-        val_adv_sensitivity = np.round(val_adv_sensitivity, 4)¬
-        val_adv_specificity = np.round(val_adv_specificity, 4)¬
-    except:¬
-        val_adv_accuracy = 0.5¬
-        val_adv_sensitivity = 0.5¬
-        val_adv_specificity = 0.5¬

     for i, (image, label, weight) in enumerate(val_loader):

         image = image.to(device)
         label = label.to(device)
         weight = weight.to(device)

         prediction = model.forward(image.float())

         loss = F.binary_cross_entropy_with_logits(prediction, label, weight=weight)

         loss_value = loss.item()
         losses.append(loss_value)

         probas = torch.sigmoid(prediction)

         y_trues.append(int(label[0]))
         y_preds.append(probas[0].item())
         y_class_preds.append((probas[0] > 0.5).float().item())

         try:
             auc = metrics.roc_auc_score(y_trues, y_preds)
         except:
             auc = 0.5

         writer.add_scalar('Val/Loss', loss_value, epoch * len(val_loader) + i)
         writer.add_scalar('Val/AUC', auc, epoch * len(val_loader) + i)

         if (i % log_every == 0) & (i > 0):
-            print(¬
                 '''[Epoch: {0} / {1} |Single batch number : {2} / {3} ] | avg val loss {4} | val auc : {5} | lr : {6}'''.¬
                 format(¬
                     epoch + 1,¬
                     num_epochs,¬
                     i,¬
                     len(val_loader),¬
                     np.round(np.mean(losses), 4),¬
```

B oct-11-train.py

```
    for i, (image, label, weight) in enumerate(val_loader):

        image = image.to(device)
        label = label.to(device)
        weight = weight.to(device)

        prediction = model.forward(image.float())

        loss = F.binary_cross_entropy_with_logits(prediction, label, weight=weight)

        loss_value = loss.item()
        losses.append(loss_value)

        probas = torch.sigmoid(prediction)

        y_trues.append(int(label[0]))
        y_preds.append(probas[0].item())
        y_class_preds.append((probas[0] > 0.5).float().item())

        try:
            auc = metrics.roc_auc_score(y_trues, y_preds)
        except:
            auc = 0.5

        writer.add_scalar('Val/Loss', loss_value, epoch * len(val_loader) + i)
        writer.add_scalar('Val/AUC', auc, epoch * len(val_loader) + i)

        if (i % log_every == 0) & (i > 0):
            print('''[Epoch: {0} / {1} | Single batch number : {2} / {3} ] | avg val loss {4} | val auc : {5} | lr : {6}'''.¬
                  format(¬
                      epoch + 1,¬
                      num_epochs,¬
                      i,¬
                      len(val_loader),¬
                      np.round(np.mean(losses), 4),¬
```

A train.py

```
                    np.round(auc, 4),¬
                    current_lr¬
                )¬
            )¬

      writer.add_scalar('Val/AUC_epoch', auc, epoch)
-     print('Val AUC: {}'.format(auc))¬

      val_loss_epoch = np.round(np.mean(losses), 4)
      val_auc_epoch = np.round(auc, 4)

      val_accuracy, val_sensitivity, val_specificity = ut.accuracy_sensitivity_specificity(y_trues, y_class_preds)
      val_accuracy = np.round(val_accuracy, 4)
      val_sensitivity = np.round(val_sensitivity, 4)
      val_specificity = np.round(val_specificity, 4)
-     if return_predictions:¬
-         return val_loss_epoch, val_auc_epoch, val_accuracy, val_sensitivity, val_specificity, y_preds, y_trues, val_adv_loss_epoch, val_adv_auc_epoch, val_adv_accuracy, val_adv_sensitivity, val_adv_specificity, adv_y_preds, adv_y_trues¬
-     else:¬
-         return val_loss_epoch, val_auc_epoch, val_accuracy, val_sensitivity, val_specificity¬
-     '''¬
-     if return_predictions:¬
-         return val_loss_epoch, val_auc_epoch, val_accuracy, val_sensitivity, val_specificity, y_preds, y_trues¬
-     else:¬
-         return val_loss_epoch, val_auc_epoch, val_accuracy, val_sensitivity, val_specificity¬
-     '''¬


  def get_lr(optimizer):
      for param_group in optimizer.param_groups:
          return param_group['lr']
- ¬
- # CHECK BUG¬
- def fgsm_attack(model, loss, image, label, weight, eps, device):¬
-     image = image.to(device)¬
-     label = label.to(device)¬
-     weight = weight.to(device)¬
-     image.requires_grad = True¬
- ¬
-     outputs = model(image.float())¬
- ¬
-     # model.zero_grad()¬
-     adversarial_loss = loss(outputs, label, weight=weight).to(device)¬
```

B oct-11-train.py

```
                      np.round(auc, 4),¬
                      current_lr¬
                  )¬
                  )¬

      writer.add_scalar('Val/AUC_epoch', auc, epoch)

      val_loss_epoch = np.round(np.mean(losses), 4)
      val_auc_epoch = np.round(auc, 4)

      val_accuracy, val_sensitivity, val_specificity = ut.accuracy_sensitivity_specificity(y_trues, y_class_preds)
      val_accuracy = np.round(val_accuracy, 4)
      val_sensitivity = np.round(val_sensitivity, 4)
      val_specificity = np.round(val_specificity, 4)

+     return val_loss_epoch, val_auc_epoch, val_accuracy, val_sensitivity, val_specificity¬

  def get_lr(optimizer):
      for param_group in optimizer.param_groups:
          return param_group['lr']
```

```
-    adversarial_loss.backward()
-
-    attack_image = image + eps * image.grad.sign()
-    attack_image = torch.clamp(attack_image, 0, 1).detach()
-    return attack_image
-
-
-def train_model_adv(model, epsilon, train_loader, epoch, num_epochs, optimizer, writer, current_lr, device, log_every,
-                    retrain_percentage):
-    """
-    Procedure to train a model on the training set by adversarial training
-    Method: FGSM
-    """
-    model.train()
-
-    model = model.to(device)
-
-    y_preds = []
-    y_trues = []
-    losses = []
-
-    train_times = int(retrain_percentage * len(train_loader))
-
-    for i, (image, label, weight) in enumerate(train_loader):
-
-        if i == train_times:
-            break
-
-        image = image.to(device)
-        label = label.to(device)
-        weight = weight.to(device)
-
-        # adversarial perturbation
-        if epsilon > 0:
-            adv_image = fgsm_attack(model, F.binary_cross_entropy_with_logits, image, label, weight, epsilon, device)
-        else:
-            adv_image = image
-
-        # adversarial training with perturbed image
-        prediction = model(adv_image.float())
-
-        adv_loss = F.binary_cross_entropy_with_logits(prediction, label, weight=weight)

-        optimizer.zero_grad()
-        adv_loss.backward()
-        optimizer.step()
-
-        loss_value = adv_loss.item()
-        losses.append(loss_value)
-
-        probas = torch.sigmoid(prediction)
-
-        y_trues.append(int(label[0]))
-        y_preds.append(probas[0].item())
```

```


        # torchviz.make_dot(prediction.mean(), params=dict(model.named_parameters())).render("prediction", format="png")

        try:
            auc = metrics.roc_auc_score(y_trues, y_preds)
            accuracy = metrics.accuracy_score(y_trues, (np.array(y_preds) > 0.5).astype(int))
            sensitivity = metrics.recall_score(y_trues, (np.array(y_preds) > 0.5).astype(int))
            specificity = metrics.recall_score(1 - np.array(y_trues), 1 - (np.array(y_preds) > 0.5).astype(int))
        except:
            auc = 0.5
            accuracy = 0.5
            sensitivity = 0.5
            specificity = 0.5

        writer.add_scalar('Train/Loss', loss_value, epoch * train_times + i)
        writer.add_scalar('Train/AUC', auc, epoch * train_times + i)

        if (i % log_every == 0) & (i > 0):
            print(
                '''[Epoch: {0} / {1} |Single batch number : {2} / {3} ] | avg train loss {4} | train auc : {5} | lr : {6} | eps: {7}'''.
                format(
                    epoch + 1,
                    num_epochs,
                    i,
                    train_times,
                    np.round(np.mean(losses), 4),
                    np.round(auc, 4),
                    current_lr,
                    epsilon
                )
            )

    print("================== adv train ==================")
    print("train times: ", train_times)
    print("y_trues: ", y_trues, "y_preds: ", y_preds)

    writer.add_scalar('Train/AUC_epoch', auc, epoch)

    train_loss_epoch = np.round(np.mean(losses), 4)
    train_auc_epoch = np.round(auc, 4)
    train_accuracy_epoch = np.round(accuracy, 4)
    train_sensitivity_epoch = np.round(sensitivity, 4)
    train_specificity_epoch =
```

A train.py

```
np.round(specificity, 4)¬
−  ¬
−      return train_loss_epoch, train_auc_epoch, train_accuracy_epoch, train_sensitivity_epoch, train_specificity_epoch¬
−  ¬
−  ¬
def run(args):
    random.seed(args.seed)
    torch.manual_seed(args.seed)
    np.random.seed(args.seed)
    torch.cuda.manual_seed_all(args.seed)

    # create dirs to store experiment checkpoints, logs, and results
    exp_dir_name = args.experiment
    exp_dir = os.path.join('experiments', exp_dir_name)
    if not os.path.exists(exp_dir):
        os.makedirs(exp_dir)
        os.makedirs(os.path.join(exp_dir, 'models'))
        os.makedirs(os.path.join(exp_dir, 'logs'))
        os.makedirs(os.path.join(exp_dir, 'results'))

    log_root_folder = exp_dir + "/logs/{0}/{1}/".format(args.task, args.plane)
    if args.flush_history == 1:
        objects = os.listdir(log_root_folder)
        for f in objects:
            if os.path.isdir(log_root_folder + f):
                shutil.rmtree(log_root_folder + f)

    now = datetime.now()
    logdir = log_root_folder + now.strftime("%Y%m%d-%H%M%S") + "/"
    os.makedirs(logdir)

    writer = SummaryWriter(logdir)

    # create training and validation set
    train_dataset = MRDataset(args.data_path, args.task, args.plane, train=True)
    train_loader = torch.utils.data.DataLoader(train_dataset, batch_size=1, shuffle=True, num_workers=4,¬
−
    drop_last=False)¬

    validation_dataset = MRDataset(args.data_path, args.task, args.plane, train=False)
    validation_loader = torch.utils.data.DataLoader(validation_dataset, batch_size=1, shuffle=-True, num_workers=2,¬
−
    drop_last=False)¬

    if torch.cuda.is_available():
```

B oct-11-train.py

```
def run(args):
    random.seed(args.seed)
    torch.manual_seed(args.seed)
    np.random.seed(args.seed)
    torch.cuda.manual_seed_all(args.seed)

    # create dirs to store experiment checkpoints, logs, and results
    exp_dir_name = args.experiment
    exp_dir = os.path.join('experiments', exp_dir_name)
    if not os.path.exists(exp_dir):
        os.makedirs(exp_dir)
        os.makedirs(os.path.join(exp_dir, 'models'))
        os.makedirs(os.path.join(exp_dir, 'logs'))
        os.makedirs(os.path.join(exp_dir, 'results'))

    log_root_folder = exp_dir + "/logs/{0}/{1}/".format(args.task, args.plane)
    if args.flush_history == 1:
        objects = os.listdir(log_root_folder)
        for f in objects:
            if os.path.isdir(log_root_folder + f):
                shutil.rmtree(log_root_folder + f)

    now = datetime.now()
    logdir = log_root_folder + now.strftime("%Y%m%d-%H%M%S") + "/"
    os.makedirs(logdir)

    writer = SummaryWriter(logdir)

    # create training and validation set
    train_dataset = MRDataset(args.data_path, args.task, args.plane, train=True)
    train_loader = torch.utils.data.DataLoader(train_dataset, batch_size=1, shuffle=True, num_workers=4, drop_last=False)¬

    validation_dataset = MRDataset(args.data_path, args.task, args.plane, train=False)
    validation_loader = torch.utils.data.DataLoader(validation_dataset, batch_size=1, shuffle=-True, num_workers=2, drop_last=False)¬

    if torch.cuda.is_available():
```

A train.py

```
        device = torch.device('cuda')
    else:
        device = torch.device('mps')¬

    # create the model
    mrnet = MRNet()
    mrnet = mrnet.to(device)

    if args.advtrain == 1:¬
-        weights_name = f'./experiments/baseline/models/model_{args.prefix_name}_{args.task}_{args.plane}.pth'¬
-        print("[INFO] Loading weights:", weights_name)¬

-        model = torch.load(weights_name)¬
-        model = model.to(device)¬
-        # load weights¬
-        model.eval()¬
-¬
-    if args.advtrain == 1:¬
-        optimizer = optim.Adam(model.parameters(), lr=args.lr, weight_decay=0.01)¬
-    else:¬
-        optimizer = optim.Adam(mrnet.parameters(), lr=args.lr, weight_decay=0.01)¬
-¬
    if args.lr_scheduler == "plateau":
        scheduler = torch.optim.lr_scheduler.ReduceLROnPlateau(
            optimizer, patience=5, factor=.3, threshold=1e-4, verbose=True)
    elif args.lr_scheduler == "step":
        scheduler = torch.optim.lr_scheduler.StepLR(
            optimizer, step_size=3, gamma=args.gamma)

    best_val_loss = float('inf')
    best_val_auc = float(0)
    best_val_accuracy = float(0)
    best_val_sensitivity = float(0)
    best_val_specificity = float(0)

    num_epochs = args.epochs
    iteration_change_loss = 0
    patience = args.patience
    log_every = args.log_every

    t_start_training = time.time()
-    all_preds = []¬
-    all_labels = []¬

    # train and test loop
    for epoch in range(num_epochs):
        current_lr = get_lr(optimizer)
-¬
-        t_start = time.time()¬
-¬
-        # train¬
-        if args.advtrain == 1:¬
-            train_loss, train_auc,
```

B oct-11-train.py

```
        device = torch.device('cuda')
    else:
        device = torch.device('cpu')¬

    # create the model
    mrnet = MRNet()
    mrnet = mrnet.to(device)

    optimizer = optim.Adam(mrnet.parameters(), lr=args.lr, weight_decay=0.01)¬

    if args.lr_scheduler == "plateau":
        scheduler = torch.optim.lr_scheduler.ReduceLROnPlateau(
            optimizer, patience=5, factor=.3, threshold=1e-4, verbose=True)
    elif args.lr_scheduler == "step":
        scheduler = torch.optim.lr_scheduler.StepLR(
            optimizer, step_size=3, gamma=args.gamma)

    best_val_loss = float('inf')
    best_val_auc = float(0)
    best_val_accuracy = float(0)
    best_val_sensitivity = float(0)
    best_val_specificity = float(0)

    num_epochs = args.epochs
    iteration_change_loss = 0
    patience = args.patience
    log_every = args.log_every

    t_start_training = time.time()

    # train and test loop
    for epoch in range(num_epochs):
        current_lr = get_lr(optimizer)
```

```python
train_accuracy, train_sensitivity, train_specificity = train_model_adv(model, args.epsilon, train_loader, epoch, num_epochs, optimizer,
                                                                                            writer, current_lr, device, log_every, args.advtrain_percent)
            val_loss, val_auc, val_accuracy, val_sensitivity, val_specificity, val_preds, val_labels, val_adv_loss_epoch, val_adv_auc_epoch, val_adv_accuracy, val_adv_sensitivity, val_adv_specificity, adv_y_preds, adv_y_trues = evaluate_model(model, validation_loader, epoch, num_epochs, writer, current_lr, device, return_predictions=True)
        else:
            train_loss, train_auc, train_accuracy, train_sensitivity, train_specificity = train_model(mrnet, train_loader, epoch, num_epochs, optimizer, writer, current_lr,
                                                                                            device, log_every)
            val_loss, val_auc, val_accuracy, val_sensitivity, val_specificity, val_preds, val_labels = evaluate_model(mrnet, validation_loader, epoch, num_epochs, writer, current_lr, device, return_predictions=True)
            # calculate samples [find error]


        all_preds.extend(val_preds)
        all_labels.extend(val_labels)

        # all_adv_preds.extend(adv_y_preds)
        # all_adv_labels.extend(adv_y_trues)

        if args.lr_scheduler == 'plateau':
            scheduler.step(val_loss)
        elif args.lr_scheduler == 'step':
            scheduler.step()

        t_end = time.time()
        delta = t_end - t_start

        learning_curve_csv = f'learning_curve_{args.prefix_name}_{args.task}_{args.plane}.csv'

        filename = os.path.join(exp_dir, 'results', learning_curve_csv)
        # Check if file exists
        if os.path.exists(filename):
            mode = 'a'
        else:
            mode = 'w'

        # Open file and append or write to it
        with open(filename, mode) as res_file:
            fa = csv.writer(res_file, delimiter=',', quotechar='|', quoting=csv.QUOTE_MINIMAL)
            if mode == 'w':
                # write headers if the file
```

```
is newly created
                fa.writerow(['epoch', 'train_loss', 'train_auc', 'train_accuracy', 'train_sensitivity', 'train_specificity', 'val_loss', 'val_auc', 'val_accuracy', 'val_sensitivity', 'val_specificity', 'val_adv_loss', 'val_adv_auc', 'val_adv_accuracy', 'val_adv_sensitivity', 'val_adv_specificity'])
            fa.writerow([epoch, train_loss, train_auc, train_accuracy, train_sensitivity, train_specificity, val_loss, val_auc, val_accuracy, val_sensitivity, val_specificity, val_adv_loss_epoch, val_adv_auc_epoch, val_adv_accuracy, val_adv_sensitivity, val_adv_specificity])


        print("train loss : {0} | train auc {1} | val loss {2} | val auc {3} | elapsed time {4} s".format(
            train_loss, train_auc, val_loss, val_auc, delta))

        iteration_change_loss += 1
        print('-' * 30)

        if val_auc > best_val_auc:
            best_val_auc = val_auc
            best_val_accuracy = val_accuracy
            best_val_sensitivity = val_sensitivity
            best_val_specificity = val_specificity
            if bool(args.save_model):
                file_name = f'model_{args.prefix_name}_{args.task}_{args.plane}.pth'
                for f in os.listdir(exp_dir + '/models/'):
                    if (args.task in f) and (args.plane in f) and (args.prefix_name in f):
                        os.remove(exp_dir + f'/models/{f}')
                if args.advtrain == 1:
                    torch.save(model, exp_dir + f'/models/{file_name}')
                else:
                    torch.save(mrnet, exp_dir + f'/models/{file_name}')

        if val_loss < best_val_loss:
            best_val_loss = val_loss
            iteration_change_loss = 0

        if iteration_change_loss == patience:
            print('Early stopping after {0} iterations without the decrease of the val loss'.
                  format(iteration_change_loss))
            break

    # save results to csv file
    with open(os.path.join(exp_dir,
'results', f'model_{args.prefix_name}_{args.task}_{args.plane}-results.csv'),
-              'w') as res_file:
-        fw = csv.writer(res_file, delimiter=',', quotechar='|', quoting=csv.QUOTE_MINIMAL)
-        fw.writerow(['LOSS', 'AUC-best', 'Accuracy-best', 'Sensitivity-best', 'Specifity-best'])
-        fw.writerow([best_val_loss, best_val_auc, best_val_accuracy, best_val_sensitivity, best_val_specificity])
-        res_file.close()
-
-
-    # draw ROC curve for best model on validation set
-    fpr = []
-    tpr = []
-    # all_labels are validation ground truth labels
-    # all_preds are validation predictions from the model
-    fpr, tpr, thresholds = roc_curve(all_labels, all_preds)
-    print(fpr, tpr)
-    filename = "roc_curve_" + args.prefix_name + "_" + args.task + "_" + args.plane
-    with open(os.path.join(exp_dir, 'results', filename), 'w') as f:
-        writer = csv.writer(f)
-        writer.writerow(['FPR', 'TPR', 'Threshold'])
-        for i in range(len(fpr)):
-            writer.writerow([fpr[i], tpr[i], thresholds[i]])
-
-    t_end_training = time.time()
-    print(f'training took {t_end_training - t_start_training} s')
-
-
- def parse_arguments():
-    parser = argparse.ArgumentParser()
-    parser.add_argument('-t', '--task', type=str, required=True,
-                        choices=['abnormal', 'acl', 'meniscus'])
-    parser.add_argument('-p', '--plane', type=str, required=True,
-                        choices=['sagittal', 'coronal', 'axial'])
-    parser.add_argument('--data-path', type=str)
-    parser.add_argument('--prefix_name', type=str, required=True)
-    parser.add_argument('--experiment', type=str, required=True)
-    parser.add_argument('--augment', type=int, choices=[0, 1], default=1)
-    parser.add_argument('--lr_scheduler', type=str,
-                        default='plateau', choices=['plateau', 'step'])
-    parser.add_argument('--seed', type=int,
default=42)
-    parser.add_argument('--gamma', type=float, default=0.5)
-    parser.add_argument('--epochs', type=int, default=50)
-    parser.add_argument('--lr', type=float, default=1e-6)
-    parser.add_argument('--flush_history', type=int, choices=[0, 1], default=0)
-    parser.add_argument('--save_model', type=int, choices=[0, 1], default=1)
-    parser.add_argument('--patience', type=int, default=50)
-    parser.add_argument('--log_every', type=int, default=100)
-
-    # Adversarial training arguments
-    parser.add_argument('--advtrain', type=int, choices=[0, 1], default=0)
-    parser.add_argument('--advtrain_percent', type=float)
-    parser.add_argument('--epsilon', type=float)
-
-    args = parser.parse_args()
-    return args
-
-
- if __name__ == "__main__":
-    args = parse_arguments()
-    run(args)
```