ДОДАТОК А

Програмна реалізація

preprocessing.py

```
import os
import json
import gc
import cv2
import torch
import numpy as np
import pandas as pd
from facenet_pytorch import MTCNN


device = torch.device('cuda:0' if torch.cuda.is_available() else 'cpu')
facenet = MTCNN(image_size=200, margin=20, keep_all=True, factor=0.5, device=device, post_process=False, select_largest=False).eval()


def get_videos_list_with_metadata(folder):
    files = os.listdir(folder)
    videos_names = [file for file in files if file[-3:] == 'mp4']
    metadata = pd.read_json(folder + 'metadata.json', orient='index')
    assert len(metadata) == len(videos_names)
    return videos_names, metadata


def extract_store_faces(filename, folder, destination_folder, debug=False):
    max_faces_counter = dict()
    found_faces = 0
    total_frames = 0
    try:
        cap = cv2.VideoCapture(folder + filename)
        ret = True
        while cap.isOpened() and ret:
            try:
                ret, frame = cap.read()
                total_frames += 1
                if ret and frame is not None:
                    frame = cv2.cvtColor(frame, cv2.COLOR_BGR2RGB)
                    face = facenet(frame,
                            save_path=destination_folder + filename[:-4] + '_' +
                                      "{:04d}".format(total_frames) + '.jpg')
                    if face is not None:
                        cnt = len(face)
                        found_faces += cnt
                        if cnt in max_faces_counter:
                            max_faces_counter[cnt] += 1
                        else:
                            max_faces_counter[cnt] = 1
            except Exception as e:
                print('Error: ', e)
                print('Frame_number: ', total_frames, '; file: ', filename)
            if debug and total_frames > 10:
                break
        gc.collect()
    except Exception as e:
        print('Error: ', e)
        print('; file: ', filename)
```

```
            finally:
                # get max_faces
                mf = np.array([[k, v] for k, v in max_faces_counter.items()])
                max_faces = int(mf[mf[:, 1].argmax(), 0])
                # prepare and save metadata
                metadata    =    {'max_faces':    max_faces,    'found_faces':
found_faces, 'total_frames': total_frames}
                with open(destination_folder + filename[:-4] + '.json', 'w')
as fp:
                    json.dump(metadata, fp)
                cap.release()


        def    process_folder(folder,    destination_folder,    start_index=0,
debug=False):
            files, metadata = get_videos_list_with_metadata(folder)
            i = 0
            for file in files:
                try:
                    if start_index > i:
                        continue
                    i += 1
                    extract_store_faces(file,   folder,   destination_folder,
debug=debug)
                    if debug and i > 5:
                        break
                except Exception as e:
                    print("\n\n----------------------------------------\nVideo
#{}, {}: {}".format( i, folder+file, e))


        if __name__ == '__main__':
            dfolder = 'X:/DFDC_DATA/train/'
            path_to_data_folders = "X:/DFDC_DATA/"
            fmt = "dfdc_train_part_{}/"
            # process_folder(path_to_data_folders + fmt.format(1), dfolder)
            # folder1 = 'X:/DFDC_DATA/dfdc_train_part_2/'
            # fn1 = 'aejroilouc.mp4'
            # extract_faces(fn1, folder1, dfolder, debug=True)
            # for foldnum in range(2):
            #         process_folder(path_to_data_folders+fmt.format(foldnum),
dfolder, 845)
```

model.py

```
        import os
        import tqdm
        import random
        import cv2
        import torch
        import numpy as np
        import pandas as pd
        import matplotlib.pyplot as plt
        from dfdc.preprocessing import get_videos_list_with_metadata

        random.seed(0)
        np.random.seed(0)
        torch.manual_seed(0)
        torch.backends.cudnn.deterministic = True


        def show_progress(train_loss_hist, valid_loss_hist, store=False):
            plt.close()
            plt.title('Train/validation loss history')
```

```
          plt.plot(train_loss_hist, label='train loss')
          plt.plot(valid_loss_hist, label='validation loss')
          plt.ylabel('Loss')
          plt.xlabel('Epoch')
          plt.legend()
          if store:
              plt.savefig('iter_{}.png'.format(len(valid_loss_hist)),
dpi=300, bbox_inches='tight')
          plt.show(block=False)


      class FakeDataset:
          def __init__(self, meta_folder, train_folder):
              self.train_folder = train_folder
              tmp = get_videos_list_with_metadata(meta_folder)
              self.raw_metadata = tmp[1]
              self._process_metadata()
              self._prepare_dataset()
              self._split_dataset()

          def _process_metadata(self):
              self.metadata                                                =
self.raw_metadata.reset_index().rename(columns={'index': 'filename'})
              self.metadata['filename']                                    =
self.metadata['filename'].map(lambda x: x[:-4], na_action='ignore')
              self.metadata['original']                                    =
self.metadata['original'].map(lambda x: x[:-4], na_action='ignore')
              self.metadata['label']  =  self.metadata['label'].map({'FAKE':
0., 'REAL': 1.})
              self.metadata = self.metadata.drop(columns=['split'])
              self.metadata.loc[self.metadata['original'].isna(),
'original'] = \
                  self.metadata.loc[self.metadata['original'].isna(),
'filename']
              self.metadata    =    self.metadata.sort_values(['original'],
kind='mergesort')

          def _prepare_dataset(self):
              files_in_train_folder = os.listdir(self.train_folder)
              needed_files   =   [[file,   file.split('_')]   for   file   in
files_in_train_folder
                                  if len(file.split('_')) == 2]
              filenames_from_meta = set(self.metadata['filename'])
              needed_files   =   [[file[0],   file[1][0]]   for   file   in
needed_files if file[1][0] in filenames_from_meta]
              self.X_filenames         =         pd.DataFrame(needed_files,
columns=['filename', 'filename_woe'])
              self.dataset      =      self.X_filenames.merge(self.metadata,
left_on='filename_woe', right_on='filename',
                                                     how='inner',
validate='m:1', suffixes=('', '_meta'))
              self.dataset     =     self.dataset[['filename',     'label',
'original']]\
                  .sort_values(['original'],              kind='mergesort',
ignore_index=True)

          def read_file(self, filename):
              try:
                  frame = cv2.imread(self.train_folder + filename)
                  frame = cv2.cvtColor(frame, cv2.COLOR_BGR2RGB)
                  return np.transpose(frame, (2, 0, 1))/256.
              except Exception as e:
                  print('{}: {}'.format(filename, e))
                  return None
```

```
        def _split_dataset(self):
            size = len(self.dataset)
            train_part = int(0.8*size)
            treshhold = self.dataset.iloc[train_part]['original']
            treshhold_index  =  self.dataset[self.dataset['original']  ==
treshhold].index[-1] + 1
            self.train = self.dataset.iloc[:treshhold_index]
            self.valid = self.dataset.iloc[treshhold_index:]
            self.train = self.train.reset_index(drop=True)
            self.valid = self.valid.reset_index(drop=True)

            self.valid['filename_w'] = self.valid['filename'].str[:-9]
            groups  =  self.valid.groupby('filename_w',  as_index=False,
sort=False).groups
            ids = []
            for k, v in groups.items():
                #       if (dataset.valid[dataset.metadata['filename'] ==
k] == 1.:
                idx_ = np.random.permutation(len(v))[:2]
                #     else:
                ids.append(v[idx_].values)
            ids = np.stack(ids, axis=1).flatten()
            self.test = self.valid.iloc[ids]


    class DeepFakeDetector(torch.nn.Module):
        def __init__(self):
            super(DeepFakeDetector, self).__init__()
            activation_function = torch.nn.ReLU()
            pooling_layer = torch.nn.MaxPool2d(kernel_size=2, stride=2)

            self.conv1_1 = torch.nn.Conv2d(
                in_channels=3, out_channels=8, kernel_size=3, padding=1)
            self.conv1_2 = torch.nn.Conv2d(
                in_channels=8, out_channels=16, kernel_size=3, padding=1)
            self.act1 = activation_function
            self.bn1 = torch.nn.BatchNorm2d(num_features=16)
            self.pool1 = pooling_layer

            self.conv2_1 = torch.nn.Conv2d(
                in_channels=16,      out_channels=32,      kernel_size=3,
padding=1)
            self.conv2_2 = torch.nn.Conv2d(
                in_channels=32,      out_channels=32,      kernel_size=3,
padding=1)
            self.act2 = activation_function
            self.bn2 = torch.nn.BatchNorm2d(num_features=32)
            self.pool2 = pooling_layer
            self.conv3 = torch.nn.Conv2d(
                in_channels=32,      out_channels=32,      kernel_size=3,
padding=1)
            self.act3 = activation_function
            self.pool3 = torch.nn.AdaptiveAvgPool2d(1)

            self.fc1 = torch.nn.Linear(32, 64)
            self.act4 = torch.nn.Tanh()

            self.fc2 = torch.nn.Linear(64, 1)

        def forward(self, x):

            x = self.conv1_2(self.conv1_1(x))  # 16 x 200 x 200
            x = self.act1(x)
```

```
            x = self.bn1(x)
            x = self.pool1(x)  # 16 x 100 x 100

            x = self.conv2_2(self.conv2_1(x))  # 32 x 100 x 100
            x = self.act2(x)
            x = self.bn2(x)
            x = self.pool2(x)  # 32 x 50 x 50

            x = self.conv3(x)
            x = self.act3(x)
            x = self.pool3(x)
            x = x.view(x.size(0), x.size(1) * x.size(2) * x.size(3))

            x = self.fc1(x)
            x = self.act4(x)
            x = self.fc2(x)

            return x


    def train(net, dataset, epoch_count=3):
        device = torch.device('cuda:0' if torch.cuda.is_available() else
'cpu')
        net = net.to(device)
        loss = torch.nn.BCEWithLogitsLoss()
        optimizer = torch.optim.Adam(net.parameters(), lr=1.0e-3)

        batch_size = 64
        test_batch_size = 64

        accumulated_loss = 0
        train_loss_history = []
        test_accuracy_history = []
        test_loss_history = []
        bp = 0
        examples = 0

        X_train = dataset.train['filename']
        y_train = dataset.train['label'].values

        X_test = dataset.test['filename']
        y_test = dataset.test['label'].values

        X_test  =  torch.stack([torch.tensor(dataset.read_file(filename))
for filename in X_test])
        y_test = torch.tensor([y_test]).transpose(1, 0)

        for epoch in range(epoch_count):
            order = np.random.permutation(len(X_train))
            for   start_index   in   tqdm.tqdm(range(0,   len(X_train),
batch_size)):
                optimizer.zero_grad()
                net.train()

                batch_indexes    =    order[start_index:start_index    +
batch_size]

                X_batch = torch.stack(
                    [torch.tensor(dataset.read_file(filename))        for
filename in X_train[batch_indexes]]).to(device,

dtype=torch.float)
                y_batch                                                =
torch.tensor([y_train[batch_indexes]]).transpose(1,           0).to(device,
```

```
dtype=torch.float)

                preds = net.forward(X_batch)

                loss_value = loss(preds, y_batch)
                loss_value.backward()
                optimizer.step()

                examples += len(y_batch)
                accumulated_loss                                    +=
loss_value.data.cpu().item()*len(y_batch)

                del X_batch, y_batch, preds, loss_value
                torch.cuda.empty_cache()

                bp += 1
                if bp % 500 == 10:
                    train_loss_history.append(accumulated_loss          /
examples)
                    accuracy = 0
                    accumulated_loss = 0
                    examples = 0
                    for  start_index2  in  tqdm.tqdm(range(0,  len(X_test),
test_batch_size)):
                        net.eval()

                        X_batch  =  X_test[start_index2:  start_index2  +
test_batch_size].to(device, dtype=torch.float)
                        y_batch  =  y_test[start_index2:  start_index2  +
test_batch_size].to(device, dtype=torch.float)

                        test_preds = net.forward(X_batch)

                        loss_value = loss(test_preds, y_batch)

                        examples += len(y_batch)
                        accumulated_loss  +=  loss_value.data.cpu().item()
* len(y_batch)
                        accuracy                                    +=
(torch.round(torch.nn.Sigmoid()(test_preds))                        ==
y_batch).float().sum().data.cpu().item()

                        del X_batch, y_batch, test_preds, loss_value
                        torch.cuda.empty_cache()

                    test_loss_history.append(accumulated_loss/examples)
                    test_accuracy_history.append(accuracy/examples)
                    accumulated_loss = 0
                    torch.save(net.state_dict(),
'model_iter_{}'.format(len(test_accuracy_history)))
                    show_progress(train_loss_history,   test_loss_history,
True)

                    print('epoch:    {},    accuracy:    {}'.format(bp/500,
accuracy/examples))
          print('---------------')
          return test_accuracy_history, test_loss_history


      if __name__ == '__main__':
          dataset      =      FakeDataset('X:/DFDC_DATA/dfdc_train_part_0/',
'X:/DFDC_DATA/train_data/train/')
          dfnet = DeepFakeDetector()
          train(dfnet, dataset)
```

ДОДАТОК Б

Відомость аттестаційнной роботи магістра

Позначення	Найменування	Дод. відомості
	Текстові документи	
1.	Пояснювальна записка	66с.
	Інші документи	
2.	Презентаційні матеріали	__ плакатів

					Розробка системи викриття підробок з використанням штучних нейронних мереж
.	.	Прізвище та ініціали.	Підп.	Дата	
Розробив		Голдобін А.О.			Шифр групи: СШІм-18-2 | Код напр./спец.: 122
Перевірив		Терзіян В.Я.			
Н.контр.					ХНУРЕ кафедра ШІ
Затв.		Філатов В.О.			