DAFTAR PUSTAKA

Agu, E. (2014). *Digital Image Processing (CS/ECE 545) Lecture 10: Discrete Fourier Transform (DFT)*. https://web.cs.wpi.edu/~emmanuel/courses/cs545/S14/slides/lecture10.pdf

Arora, S., Maheshwari, N., & Bhatia, M. P. S. (2018). Spoofed Fingerprint Detection Based on Time Series Fingerprint Image Analysis. *2018 First International Conference on Secure Cyber Computing and Communication (ICSCCC)*, 217–222. https://doi.org/10.1109/ICSCCC.2018.8703334

Baleanu, D., Balas, V. E., & Agarwal, P. (2022). *Fractional Order Systems and Applications in Engineering*. Elsevier Science.

Garanin, D. (2007). *4-Synthesis and Analysis of Complex Waves; Fourier spectra*. https://www.lehman.edu/faculty/dgaranin/teaching-Physics_of_Sound.php

Gatto, P. A., & Awangga, R. M. (2023). *Pengelompokan Kedisiplinan Pegawai Berdasarkan Absensi Menggunakan Algoritma K-Means*. Penerbit Buku Pedia.

Geetha, T. V, & Sendhilkumar, S. (2023). *Machine Learning: Concepts, Techniques and Application*. CRC Press.

Gonzalez, R. C., & Woods, R. E. (Richard E. (2017). *Digital image processing*.

Huda, M. (2020). *Keamanan Informasi*. Nulisbuku. https://books.google.co.id/books?id=CcjZDwAAQBAJ

Morita, K. (1995). *Applied Fourier Transform*. Ohmsha.

Munawir, Fitria, L., & Hermansyah, M. (2020). Implementasi Face Recognition pada Absensi Kehadiran Mahasiswa Menggunakan Metode Haar Cascade Classifier. *Jurnal Nasional Informatika Dan Teknologi Jaringan*, *4*(2), 314–320. https://doi.org/10.30743/infotekjar.v4i2.2333

Nandy, A., & Biswas, M. (2017). *Reinforcement Learning: With Open AI, TensorFlow and Keras Using Python*. Apress. https://books.google.co.id/books?id=wTVCDwAAQBAJ

Perdana, R. N., Ardiyanto, I., & Nugroho, H. A. (2021). A Review on Face Anti-Spoofing. *IJITEE*, *5*(1), 29–35.

Putra, D. (2010). *Pengolahan Citra Digital*. ANDI OFFSET.

Rattani, A., Scheirer, W. J., & Ross, A. (2015). Open set fingerprint spoof detection across novel fabrication materials. *IEEE Transactions on Information Forensics and Security*, *10*(11), 2447–2460. https://doi.org/10.1109/TIFS.2015.2464772

Setiawan, W. (2021). *Deep Learning menggunakan Convolutional Neural Network: Teori dan Aplikasi*. Media Nusa Creative (MNC Publishing).

Setiawati, L. (2013). *Karyawan Bisa Jadi Tikus & Monster Penghisap Darah Perusahaan*. Elex Media Komputindo.

Setyadi, H. A., & Sundari. (2022). Sistem Informasi Manajemen Kehadiran dan Jam Kerja Karyawan Untuk Kelengkapan Perhitungan Gaji Karyawan. *Indonesian Journal Computer Science*, 28–33.

Smith, S. (2003). *Digital Signal Processing: A Practical Guide for Engineers and Scientists*. Elsevier Science. https://books.google.co.id/books?id=PCrcintuzAgC

Wahyuni, S. (2023). *Facial Expression Recognition Pada Desain Smarthome Terintegrasi Algoritma Genetika Untuk Pengontrolan Perangkat Listrik*.

Wong, K. K. L. (2023). *Cybernetical Intelligence: Engineering Cybernetics with Machine Intelligence*. Wiley.

LAMPIRAN

Lampiran 1 Dokumentasi Pelaksanaan Penelitian

1.1 Pengambilan Data

Lampiran 2 Kode Pemograman

2.1 Program Training Data

```
import torch
from torch import optim
from torch.nn import CrossEntropyLoss, MSELoss
from tqdm import tqdm
from tensorboardX import SummaryWriter
from src.utility import get_time
from src.model_lib.MultiFTNet import MultiFTNet
```

```
from src.data_io.dataset_loader import get_train_loader

class TrainMain:
    def __init__(self, conf):
        self.conf = conf
        self.board_loss_every = conf.board_loss_every
        self.save_every = conf.save_every
        self.step = 0
        self.start_epoch = 0
        self.train_loader = get_train_loader(self.conf)

    def train_model(self):
        self._init_model_param()
        self._train_stage()

    def _init_model_param(self):
        self.cls_criterion = CrossEntropyLoss()
        self.ft_criterion = MSELoss()
        self.model = self._define_network()
        self.optimizer = optim.SGD(self.model.module.parameters(),
                                   lr=self.conf.lr,
                                   weight_decay=5e-4,
                                   momentum=self.conf.momentum)

        self.schedule_lr = optim.lr_scheduler.MultiStepLR(
            self.optimizer, self.conf.milestones, self.conf.gamma, - 1)

        print("lr: ", self.conf.lr)
        print("epochs: ", self.conf.epochs)
        print("milestones: ", self.conf.milestones)
```

```
def _train_stage(self):
    self.model.train()
    running_loss = 0.
    running_acc = 0.
    running_loss_cls = 0.
    running_loss_ft = 0.
    is_first = True
    for e in range(self.start_epoch, self.conf.epochs):
        if is_first:
            self.writer = SummaryWriter(self.conf.log_path)
            is_first = False
        print('epoch {} started'.format(e))
        print("lr: ", self.schedule_lr.get_lr())

        for sample, ft_sample, target in tqdm(iter(self.train_loader)):
            imgs = [sample, ft_sample]
            labels = target

            loss, acc, loss_cls, loss_ft = self._train_batch_data(imgs, labels)
            running_loss_cls += loss_cls
            running_loss_ft += loss_ft
            running_loss += loss
            running_acc += acc

            self.step += 1


            loss_board = running_loss / self.board_loss_every
            self.writer.add_scalar(
                'Training/Loss', loss_board, self.step)
            acc_board = running_acc / self.board_loss_every
```

```
            self.writer.add_scalar(
                'Training/Acc', acc_board, self.step)
            lr = self.optimizer.param_groups[0]['lr']
            self.writer.add_scalar(
                'Training/Learning_rate', lr, self.step)
            loss_cls_board = running_loss_cls / self.board_loss_every
            self.writer.add_scalar(
                'Training/Loss_cls', loss_cls_board, self.step)
            loss_ft_board = running_loss_ft / self.board_loss_every
            self.writer.add_scalar(
                'Training/Loss_ft', loss_ft_board, self.step)

            running_loss = 0.
            running_acc = 0.
            running_loss_cls = 0.
            running_loss_ft = 0.

            time_stamp = get_time()
            self._save_state(time_stamp, extra=self.conf.job_name)
        self.schedule_lr.step()

    time_stamp = get_time()
    self._save_state(time_stamp, extra=self.conf.job_name)
    self.writer.close()

def _train_batch_data(self, imgs, labels):
    self.optimizer.zero_grad()
    labels = labels.to(self.conf.device)

    if imgs[0].size() == torch.Size([1, 80, 80, 3]) :
        imgs[0] = imgs[0].permute(0, 3, 1, 2).float()
```

```
        else:
            pass

        embeddings, feature_map = self.model.forward(imgs[0].to(self.conf.device))

        loss_cls = self.cls_criterion(embeddings, labels)
        loss_fea = self.ft_criterion(feature_map, imgs[1].to(self.conf.device))
        print("loss :", loss_fea)

        loss = 0.5*loss_cls + 0.5*loss_fea
        acc = self._get_accuracy(embeddings, labels)[0]
        loss.backward()
        self.optimizer.step()
        return loss.item(), acc, loss_cls.item(), loss_fea.item()

    def _define_network(self):
        param = {
            'num_classes': self.conf.num_classes,
            'img_channel': self.conf.input_channel,
            'embedding_size': self.conf.embedding_size,
            'conv6_kernel': self.conf.kernel_size}

        model = MultiFTNet(**param).to(self.conf.device)
        model = torch.nn.DataParallel(model, self.conf.devices)
        return model

    def _get_accuracy(self, output, target, topk=(1,)):
        maxk = max(topk)
        batch_size = target.size(0)
        _, pred = output.topk(maxk, 1, True, True)
        pred = pred.t()
```

```
        correct = pred.eq(target.view(1, -1).expand_as(pred))

        ret = []
        for k in topk:
            correct_k = correct[:k].view(-1).float().sum(dim=0, keepdim=True)
            ret.append(correct_k.mul_(1. / batch_size))
        return ret

    def _save_state(self, time_stamp, extra=None):
        save_path = self.conf.model_path
        torch.save(self.model.state_dict(), save_path + '/' +
                ('{}_{}_model_iter-{}.pth'.format(time_stamp, extra, self.step)))
```

2.2 Program Testing

```
import cv2
import numpy as np
import argparse
import warnings
import time
import os

from src.anti_spoof_predict import AntiSpoofPredict
from src.generate_patches import CropImage
from src.utility import parse_model_name

warnings.filterwarnings('ignore')

def detect_faces(image):
    gray = cv2.cvtColor(image, cv2.COLOR_BGR2GRAY)
    face_cascade = cv2.CascadeClassifier(cv2.data.haarcascades +
'haarcascade_frontalface_default.xml')
```

```
    faces = face_cascade.detectMultiScale(gray, scaleFactor=1.3, minNeighbors=5)
    return faces


def test_image_from_frame(image, model_test, image_cropper, model_dir):
    image_bbox = model_test.get_bbox(image)
    prediction = np.zeros((1, 3))
    test_speed = 0

    for model_name in os.listdir(model_dir):
        h_input, w_input, model_type, scale = parse_model_name(model_name)
        param = {
            "org_img": image,
            "bbox": image_bbox,
            "scale": scale,
            "out_w": w_input,
            "out_h": h_input,
            "crop": True,
        }
        if scale is None:
            param["crop"] = False
        img = image_cropper.crop(**param)
        start = time.time()
        prediction += model_test.predict(img, os.path.join(model_dir, model_name))
        test_speed += time.time() - start

    label = np.argmax(prediction)
    value = prediction[0][label] / 2
    if label == 1:
        result_text = "RealFace Score: {:.2f}".format(value)
        color = (255, 0, 0)
```

```
    else:
        result_text = "FakeFace Score: {:.2f}".format(value/2)
        color = (0, 0, 255)

    return result_text, color

def detect_video(model_dir, device_id):
    cap = cv2.VideoCapture(0)

    if not cap.isOpened():
        print("Error opening video stream or file")
        return

    model_test = AntiSpoofPredict(device_id)
    image_cropper = CropImage()



    while cap.isOpened():
        ret, frame = cap.read()
        if not ret:
            break

        faces = detect_faces(frame)
        result_text, color = test_image_from_frame(frame, model_test,
image_cropper, model_dir)

        cv2.putText(
            frame,
            result_text,
            (10, 30),
```

```
            cv2.FONT_HERSHEY_SIMPLEX, 1, color, 2)



        cv2.imshow('Face Recognition and Spoofing Detection', frame)

        if cv2.waitKey(1) & 0xFF == ord('q'):
            break

    cap.release()
    cv2.destroyAllWindows()

if __name__ == "__main__":
    desc = "test"
    parser = argparse.ArgumentParser(description=desc)
    parser.add_argument(
        "--device_id",
        type=int,
        default=0,
        help="which gpu id, [0/1/2/3]")
    parser.add_argument(
        "--model_dir",
        type=str,
        default="./resources/anti_spoof_models",
        help="model_lib used to test")

    args = parser.parse_args()

    detect_video(args.model_dir, args.device_id)
```