Министерство науки и высшего образования Российской Федерации
Федеральное государственное бюджетное
образовательное учреждение высшего образования
«Комсомольский-на-Амуре государственный университет»

Проект «робо-пес “Finder”»

Руководитель СКБ Г. В. Москалец

Подпись/дата

Ответственный исполнитель А. М. Черников

Подпись/дата

2022

Карточка проекта

Название	«робо-пес “Finder”»
Тип проекта	Инициативный (инициативный, в рамках учебного курса, учебная работа, другое)
Исполнители	Ответственный исполнитель А. М. Черников
Срок реализации	31.06.2022

Использованные программные средства

Наименование	Версия
Windows	10
Python	3.10
Unity	
PyTorch	1.11.0+cu113
numpy	1.22.3
pandas	1.4.2
opencv-python	4.2.2.64
torchvision	0.12.0+cu113
mediapipe	0.8.10
tqdm	4.64.0
scikit-learn	1.0.2
PyYAML	6.0

Министерство науки и высшего образования Российской Федерации
Федеральное государственное бюджетное
образовательное учреждение высшего образования
«Комсомольский-на-Амуре государственный университет»

ЗАДАНИЕ
на разработку

Название проекта: робо-пес “Finder”

Назначение: поиск объектов

Область использования: не ограничена

Функциональное описание устройства: система распознает жест и ищет соответствующий этому жесту объект. Далее стреляет в объект из пушки.

Техническое описание устройства: система принимает данные с видеокамеры для классификации жеста. Данные подаются в нейронную сеть, которая определяет, что это за жест. Далее система принимает данные с видеокамеры и дальномера и обрабатывает их при помощи нейронной сети для выполнения команды, зашифрованной в жесте.

Требования: ОС: Windows, Linux или Mac OS; Видеокарта: Nvidia; Свободное место на диске: 1гб. и более.

Перечень графического материала: в ходе работы над проектом используются графики, изображения, 3D модели

План работ:

Наименование работ	Срок
Разработка архитектуры программного обеспечения	1 сентября – 15 сентября
Поиск датасета и предобработка данных	16 сентября – 30 сентября
Создание класса dataset.	1 октября – 31 октября
Разработка архитекруты нейронной сети	1 ноября – 20 ноября
Разработка модуля train.py, отладка обучения	21 ноября – 8 декабря
Обучение нейронной сети	9 декабря – 1 января
Тестирование нейронной сети	2 января – 31 января
Написание inference	1 февраля – 28 февраля
Установка Unity, создание модели робота в Unity	1 марта – 31 марта
Задание механизма управления роботом(валидация через клавиатуру)	1 апреля – 14 апреля
Объединение кодов inference и нейронной сети и Unity	15 апреля – 15 мая
Тестирование	16 мая - 31 мая

Руководитель СКБ Г. В. Москалец

Подпись/дата

Содержание

1 Общие положения

Настоящий документ представляет собой руководство администратора программного обеспечения «робо-пес “Finder”» далее ПО. Руководство определяет порядок установки, настройки и администрирования ПО. Перед установкой и эксплуатацией системы рекомендуется внимательно ознакомиться с настоящим руководством. Документ подготовлен в соответствии с РД 50-34.698-90 - в части структуры и содержания документов, и в соответствии с ГОСТ 34.201-89 - в части наименования и обозначения документов.

1.1 Наименование программы

Наименование программного продукта: робо-пес “Finder”.

1.2 Наименование документов, на основании которых ведется проектирование системы

Создание ПО осуществляется на основании требований и положений следующих документов: задание.

1.3 Перечень организаций, участвующих в разработке системы

Разработчики: студент Комсомольского-на-Амуре государственного технического университета А.М. Черников.

Заказчик: Комсомольский-на-Амуре государственный университет, студенческое конструкторское бюро «Интеллектуальные технологии».

1.4 Сведения об использованных при проектировании нормативно технических документах

При проектировании использованы следующие нормативно-технические документы:

- ГОСТ 19.101-77 – виды программ и программных документов.
- ГОСТ 19.106-78 Требования к программным документам, выполненным печатным способом.
- ГОСТ 19.201-78 Техническое задание, требования к содержанию и оформлению;
- ГОСТ 19.301-79 Программа и методика испытаний. Требования к содержанию и оформлению.
- ГОСТ 2.004-88. Единая система конструкторской документации. Общие требования к выполнению конструкторских технологических документов на печатающих и графических устройствах вывода ЭВМ.

2 Назначение и принцип действия

2.1 Назначение изделия

Программное обеспечение для регрессионного моделирования

2.2 Области использования изделия

Программное обеспечение может применяться для уборки мусора, работы на складе и на опасных объектах.

2.3 Принцип действия

Система распознает жест и выполняет соответствующую этому жесту команду.

3. Описание программного обеспечения

3.1 Описание работы программного обеспечения

Результат работы интеллектуальной системы – выполнение команды, соответствующей показанному жесту.

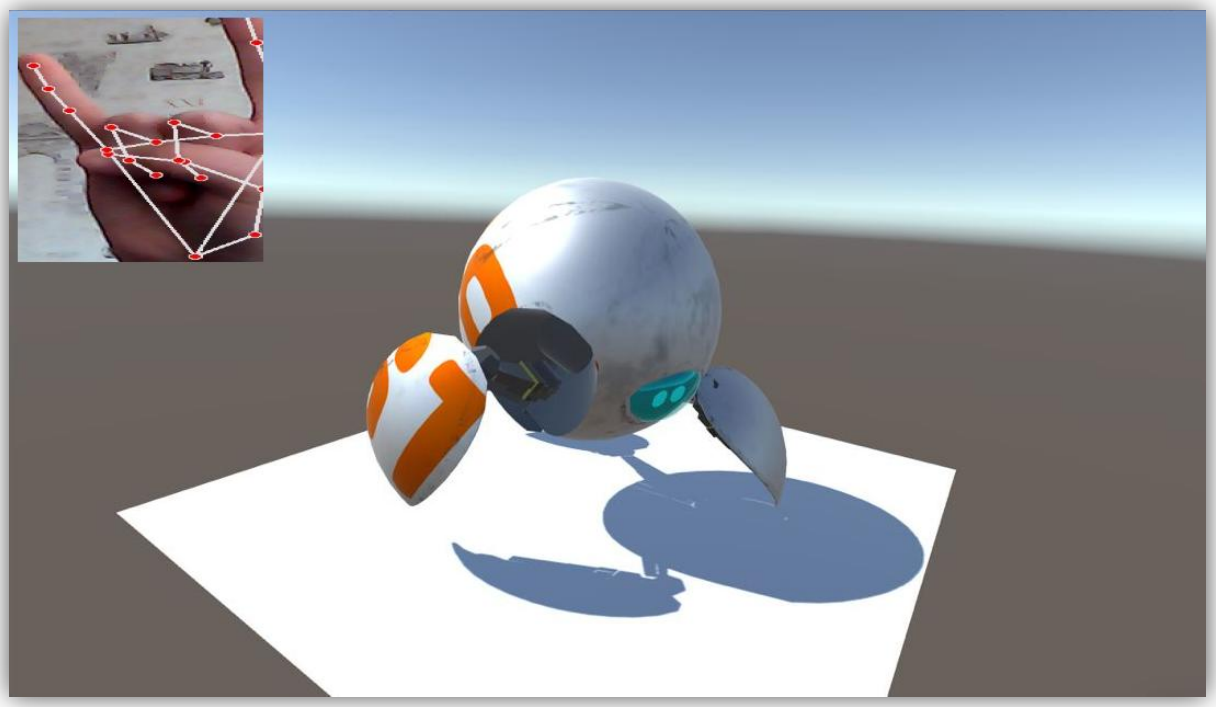

Рисунок 1 – ходьба

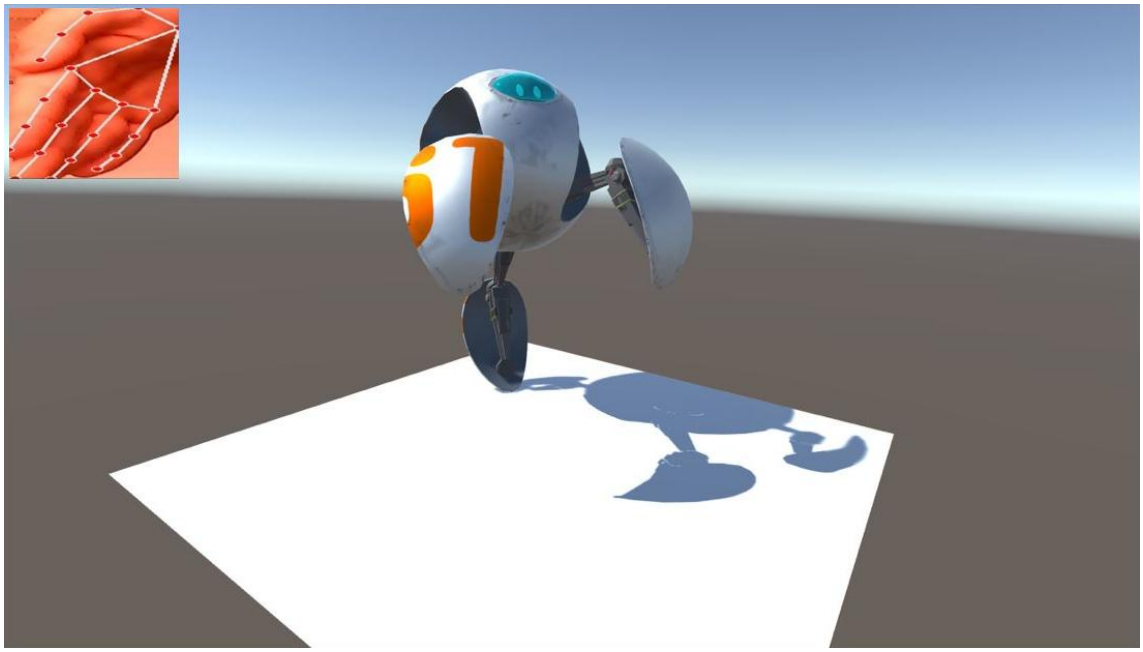

Рисунок 2 – прыжок

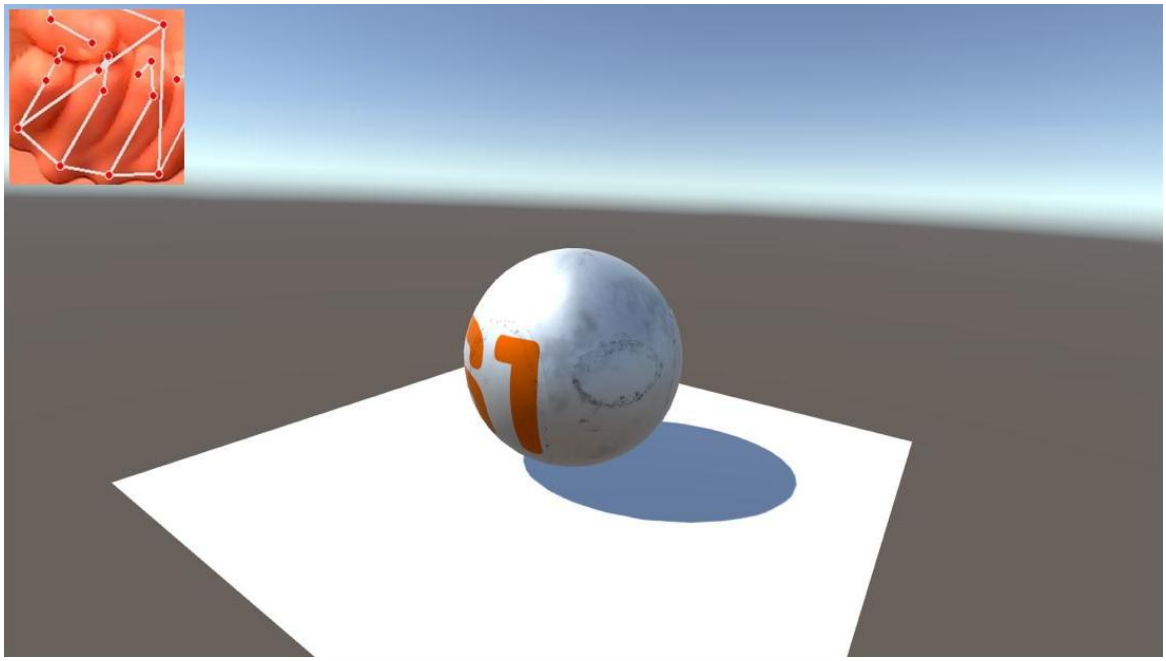

Рисунок 3 - свернуться

3.2 Входные данные

Картинка с камеры или подобного устройства

4. Текст программы

Листинг 1 – dataset.py

```
import torch
from torch.utils.data import Dataset
import numpy as np
import os
import pandas as pd
import cv2
from torchvision import transforms


class GestureDatasetDots(Dataset):

    def __init__(self, x_skeleton, y_gesture):
        super(GestureDatasetDots, self).__init__()
        self.gesture = {"rock": 0, "paper": 1, "scissors": 2, "goat": 3,
"dislike": 4, "like": 5}
        self.x_skeleton = x_skeleton
        self.y_gesture = y_gesture

    def __len__(self):
        return len(self.y_gesture)

    def __getitem__(self, index):
        x_skeleton = self.x_skeleton[index]
        y_gesture = self.y_gesture[index]

        return torch.tensor(x_skeleton, dtype=torch.float32), \
               torch.tensor(self.gesture[y_gesture], dtype=torch.long)


class GestureDatasetPics(Dataset):

    def __init__(self, csv_path):
        super(GestureDatasetPics, self).__init__()
        self.gesture = {"rock": 0, "paper": 1, "scissors": 2, "goat": 3,
"dislike": 4, "like": 5}
        self.gesture1 = {0: "rock", 1: "paper", 2: "scissors", 3: "goat", 4:
"dislike", 5: "like"}
        self.df = pd.read_csv(csv_path)
        self.df = self.df[self.df["class"] != 5]
        self.df = self.df.reset_index()
        self.x_hands_path = self.df["Path_img"]
        print(self.x_hands_path)
        self.x_dot_hands_path = self.df["Path_dots"]
        print(self.x_dot_hands_path)
        self.target = list(map(int, list(self.df["class"])))
        # remove ToPILImage------------------------------------------------------
--!!!!!!!!!!!!
        self.train_transform = transforms.Compose([transforms.ToPILImage(),

transforms.RandomHorizontalFlip(),

transforms.RandomRotation(degrees=(-180, 180)),

transforms.ColorJitter(brightness=0.4, contrast=0.4, saturation=0.4),
                                                transforms.ToTensor()])
```

```
        self.test_transform = transforms.Compose([transforms.ToPILImage(),
transforms.ToTensor()])

    def __len__(self):
        return len(self.target)

    def __getitem__(self, index):
        x_hand = cv2.imread(self.x_hands_path[index])
        x_dot_hand = cv2.imread(self.x_dot_hands_path[index])
        target = self.target[index]

        return self.train_transform(x_hand), \
               self.train_transform(x_dot_hand), \
               torch.tensor(target, dtype=torch.long)


if __name__ == '__main__':
    test_dataset = GestureDatasetPics("/Users/illusivesheep/Repositories/ультра
датасет/dots/train_dots.csv")
    hand, dots, label = test_dataset.__getitem__(1)
    print(hand)
    print(hand.size())
```

Листинг 1 – model.py

```
import torch
from torch import nn
import numpy as np


class GestureModel(torch.nn.Module):

    def __init__(self, n_hidden_neurons):
        super(GestureModel, self).__init__()

        self.d = nn.Dropout(p=0.5)
        self.activation = torch.nn.ReLU()
        self.fc1 = torch.nn.Linear(21 * 3, n_hidden_neurons)
        self.bn1 = nn.BatchNorm1d(n_hidden_neurons)
        self.fc2 = torch.nn.Linear(n_hidden_neurons, 6)
        self.bn2 = nn.BatchNorm1d(6)

    def forward(self, x):
        x = x.view(x.shape[0], -1)
        x = self.d(self.activation(self.bn1(self.fc1(x))))
        x = self.activation(self.bn2(self.fc2(x)))
        return x


def test_model():
    model = GestureModel(100)

    print(model)

    x_test =
torch.tensor(np.load("/Users/illusivesheep/Repositories/data/test_coords.npy")[1
], dtype=torch.float32)
    x_test = torch.unsqueeze(x_test, 0)
```

```
    model.eval()
    with torch.no_grad():
        print(model(x_test))


class HandFuzingModel(torch.nn.Module):
    def __init__(self, n_hidden_neurons, num_hand_features,
num_hand_dots_features):
        super(HandFuzingModel, self).__init__()
        self.n_hidden_neurons = n_hidden_neurons
        self.num_hand_features = num_hand_features
        self.num_hand_dots_features = num_hand_dots_features

        self.fc_hand = torch.nn.Linear(512, 5)
        self.fc_dot_hand = torch.nn.Linear(512, 5)
        #можно torch no grad или изменить фузинг

        self.bn = torch.nn.BatchNorm1d(5)
        self.dp = torch.nn.Dropout(0.5)
        self.activation = torch.nn.ReLU()
        self.fc = torch.nn.Linear(2*n_hidden_neurons, 5)

    def forward(self, x_hand, x_dots_hand):
        hand_feautures = x_hand.view(-1, self.num_hand_features)
        hand_dots_feautures = x_dots_hand.view(-1, self.num_hand_dots_features)
        # hand_dots_feautures = hand_dots_feautures.to(torch.float)

        hand_feautures = self.activation(self.fc_hand(hand_feautures))
        hand_dots_feautures =
self.activation(self.fc_dot_hand(hand_dots_feautures))

        fuze = torch.cat((hand_feautures, hand_dots_feautures), 1)#.view(-1,
128)

        fuze_out = self.activation(self.bn(self.fc(fuze)))
        # fuze out = self.dp(fuze out)

        return fuze_out


if __name__ == '__main__':
    test_model()
```

Листинг 3 – main.py

```
import os
import torch
from PreProcess import csv_gen
from train import train, train_cnn
from classic_learning import learning

from config.config_classes import StartConfig
from utils import get_config_data


def start_mode():
    start_config =
StartConfig(**get_config_data(os.path.join(os.environ['CFG_PATH'],
os.environ['START_CFG_NAME'])))
    device = torch.device(start_config.gpu)
    # start_modes = {"preprocessing": csv_gen(start_config.data_path),
    #                "train": train_cnn(start_config, device),
    #                "classic": learning(start_config)}

    # path_dataset = start_config.data_path
    # path_images = os.path.join(path_dataset, "images_train_test")
    # path = os.path.join(path_images, "train")
    # classes_folders = [folder for folder in os.listdir(path) if "." not in
folder and "crop" not in folder]
    # path_folder = os.path.join(path, classes_folders[0])
    # image_names = [image for image in os.listdir(path_folder) if ".jpg" in
image]
    # print(classes_folders)
    # print(len(image_names))

    # _ = start_modes[start_config.mode]

    # csv_gen(start_config.data_path)

    train_cnn(start_config, device)


if __name__ == '__main__':
    start_mode()
```

Листинг 4 – prepare_models.py

```
import os
import torchvision.models as models
import torch


def prepare_models(model_hand_name="resnet18", model_dot_hand_name="resnet18",
model_path="./"):
    model_hand = models.__dict__[model_hand_name](pretrained=True)
    model_dot_hand = models.__dict__[model_dot_hand_name](pretrained=True)

    model_hand.cpu()
    model_dot_hand.cpu()

    torch.save(model_hand, os.path.join(model_path,
f"hand_model{model_hand_name}.pth"))
    torch.save(model_dot_hand, os.path.join(model_path,
f"hand_model{model_dot_hand_name}.pth"))

    return model_hand, model_dot_hand
```

Листинг 5 – PreProcess.py

```
import cv2
import os
import numpy as np
import mediapipe as mp
import pandas as pd


def csv_gen(path_dataset):
    df = pd.DataFrame()

    modes = ["train", "test", "val"]
    gesture = {"rock": 0, "paper": 1, "scissors": 2, "goat": 3, "dislike": 4,
"like": 5}

    path_dots = os.path.join(path_dataset, "dots")
    path_images = os.path.join(path_dataset, "images_train_test")

    try:
        os.mkdir(os.path.join(path_dataset, "dots",))
    except FileExistsError:
        pass
    try:
        os.mkdir(os.path.join(path_dataset, "crop",))
    except FileExistsError:
        pass

    for mode in modes:
        path = os.path.join(path_images, mode)
        classes_folders = [folder for folder in os.listdir(path) if "." not in
folder and "crop" not in folder]

        try:
            os.mkdir(os.path.join(path_dataset, "dots", mode))
        except FileExistsError:
            pass
```

```
        try:
            os.mkdir(os.path.join(path_dataset, "crop", mode))
        except FileExistsError:
            pass

        for folder in classes_folders:

            path_folder = os.path.join(path, folder)
            image_names = [image for image in os.listdir(path_folder) if ".jpg"
in image]

            try:
                os.mkdir(os.path.join(path_dataset, "dots", mode, folder))
            except FileExistsError:
                pass
            try:
                os.mkdir(os.path.join(path_dataset, "crop", mode, folder))
            except FileExistsError:
                pass

            for image_name in image_names:
                print(path_folder + "/" + image_name)
                image = cv2.imread(os.path.join(path_folder, image_name))
                mp_hands = mp.solutions.hands
                with mp_hands.Hands(
                        static_image_mode=True,
                        max_num_hands=1,
                        min_detection_confidence=0.7) as hands:
                    results = hands.process(cv2.cvtColor(image,
cv2.COLOR_BGR2RGB))

                    if results.multi_hand_landmarks:
                        for hand_landmarks in results.multi_hand_landmarks:

                            coef_x = 0.5 - hand_landmarks.landmark[9].x
                            coef_y = 0.5 - hand_landmarks.landmark[9].y
                            blank_image = np.zeros((224, 224, 3), np.uint8)

                            X_arr = list()
                            Y_arr = list()

                            for i, point in enumerate(hand_landmarks.landmark):
                                X = int((coef_x + point.x) * 224)
                                Y = int((coef_y + point.y) * 224)

                                cv2.circle(blank_image, (X, Y), 1, (255, 255,
255), 3)

                                cv2.imwrite(os.path.join(path_dots, folder,
image_name),
                                            blank_image)

                                X_arr.append(point.x)
                                Y_arr.append(point.y)

                df = df.append({'Image': image_name,
                                'Path_dots': os.path.join(path_dataset, "dots",
mode, folder, image_name),
                                'Path_img': os.path.join(path_dataset, "crop",
```

```
mode, folder, image_name),
                                # 'landmark_id': i,
                                # 'X': X,
                                # 'Y': Y,
                                # 'Z': point.z,
                                'class': gesture[folder]}, ignore_index=True)

                cv2.imwrite(os.path.join(path_dataset, "dots", mode, folder,
image_name), blank_image)
                image_crop = cv2.imread(os.path.join(path_folder, image_name))
                height, width, channels = image_crop.shape
                image_crop = image_crop[int(min(Y_arr) * height):int(max(Y_arr)
* height),
                                        int(min(X_arr) * width):int(max(X_arr) *
width)]
                image_crop = cv2.resize(image_crop, dsize=(224, 224))
                cv2.imwrite(os.path.join(path_dataset, "crop", mode, folder,
image_name), image_crop)

        # df[["class", "landmark_id", "X", "Y"]] = df[["class", "landmark_id",
"X", "Y"]].astype(int)
        df.to_csv(os.path.join(path_dataset, f"{mode}_dots.csv"))
```

Листинг 6 – train.py

```
import os

import tqdm
import random
import numpy as np

from torch.utils.tensorboard import SummaryWriter

import torch
import torch.optim as optim
from torch.optim.lr_scheduler import StepLR
from torch.utils.data import DataLoader
from torch import nn

from model import GestureModel, HandFuzingModel
from dataset import GestureDatasetDots, GestureDatasetPics

from sklearn.metrics import accuracy_score, f1_score, precision_score, \
    recall_score, classification_report, confusion_matrix
from prepare_models import prepare_models


def train(config_data):
    model = GestureModel(100)

    writer_train = SummaryWriter("tensor_board_graphs/train")
    writer_test = SummaryWriter("tensor_board_graphs/test")

    x_train = np.load(os.path.join(config_data.data_path, "npy",
f"train_coords.npy"))
    y_train = np.load(os.path.join(config_data.data_path, "npy",
f"train_labels.npy"))
```

```
    x_val = np.load(os.path.join(config_data.data_path, "npy",
f"val_coords.npy"))
    y_val = np.load(os.path.join(config_data.data_path, "npy",
f"val_labels.npy"))

    train_dataset = GestureDatasetDots(x_train, y_train)
    train_dataloader = DataLoader(train_dataset,
batch_size=config_data.batch_size)

    val_dataset = GestureDatasetDots(x_val, y_val)
    val_dataloader = DataLoader(val_dataset, batch_size=config_data.batch_size)

    random.seed(0)
    np.random.seed(0)
    torch.manual_seed(0)
    torch.cuda.manual_seed(0)

    opt = optim.Adam([{'params': list(model.parameters()), 'lr':
config_data.learning_rate}],
                     weight_decay=config_data.weight_decay)
    scheduler = StepLR(opt, step_size=10, gamma=0.1)
    loss = nn.CrossEntropyLoss()

    for e in range(config_data.epochs):
        model.train()
        running_loss = 0.0
        precision = 0
        for x, y in iter(tqdm.tqdm(train_dataloader)):
            opt.zero_grad()
            prediction = model(x)

            # print(y.size())
            # print(prediction.detach().numpy())
            # for cur_x in prediction.detach().numpy():
            #     max = 0
            #     for gesture in cur x:
            #         if max

            precision += precision_score(y.view(-1).cpu(),
torch.max(prediction.data.cpu(), 1)[1], average='micro')
            # print(precision)
            loss_batch = loss(prediction, y) / len(y)
            loss_batch.backward()
            opt.step()
            running_loss += loss_batch.item()

        scheduler.step()

        running_loss = running_loss / len(train_dataloader)
        precision = precision / len(train_dataloader)

        writer_train.add_scalar('Loss_train', running_loss, e)
        writer_train.add_scalar('Precision_train', precision, e)

        # print('epoch = %d train_loss = %.4f' % (e, running_loss))
        # print('precision = %.4f' % (precision))

        model.eval()
        running_val_loss = 0.0
```

```
        precision = 0

        for x, y in iter(tqdm.tqdm(val_dataloader)):
            with torch.no_grad():
                prediction = model(x)
                precision += precision_score(y.view(-1).cpu(),
torch.max(prediction.data.cpu(), 1)[1], average='micro')
                loss_val_batch = loss(prediction, y) / len(y)
                running_val_loss += loss_val_batch.item()

        precision = precision / len(train_dataloader)
        running_val_loss = running_val_loss / len(val_dataloader)
        writer_test.add_scalar('Precision_test', precision, e)
        writer_test.add_scalar('Loss_test', running_val_loss, e)
        # print('epoch = %d val_loss = %.4f' % (e, running_val_loss))


def train_cnn(config_data, device):
    try:
        os.mkdir(os.path.join(config_data.model_path))
    except FileExistsError:
        pass

    usePretrainedFisungModel = False

    if usePretrainedFisungModel:
        model_hand = torch.load('models/model_hand_120.pth')
        model_dot_hand = torch.load('models/model_dot_hand_120.pth')
        model_fuzing_hand = torch.load('models/model_fuzing_hand_120.pth')
    else:
        model hand, model dot hand = prepare models("resnet18", "resnet18")
        model_hand = nn.Sequential(*(list(model_hand.children())[:-1]))
        model_dot_hand = nn.Sequential(*(list(model_dot_hand.children())[:-1]))
        model_fuzing_hand = HandFuzingModel(5, 512, 512)

    for param in model hand.parameters():
        param.requires_grad = True
    for param in model_dot_hand.parameters():
        param.requires_grad = True
    for param in model_fuzing_hand.parameters():
        param.requires grad = True

    writer_train = SummaryWriter("tensor_board_graphs/train")
    writer_test = SummaryWriter("tensor_board_graphs/test")

    loss = torch.nn.CrossEntropyLoss()
    loss.to(device)

    opt = optim.Adam([{'params': (list(model_hand.parameters()) +
                                  list(model_dot_hand.parameters())),
                       'lr': config_data.learning_rate},
                      {'params': list(model_fuzing_hand.parameters()),
                       'lr': config_data.learning_rate *
config_data.learning_rate_fusing_coefficient},
                      ], weight_decay=config_data.weight_decay)

    # opt = torch.optim.Adamax(params=iter(list(model_hand.parameters()) +
list(model_dot_hand.parameters())),
    #                          lr=config_data.learning_rate,
```

```
    #                           betas=(0.9, 0.999),
    #                           eps=1e-08,
    #                           weight_decay=config_data.weight_decay)

    scheduler = StepLR(opt, step_size=10, gamma=0.1)
    # scheduler = torch.optim.lr_scheduler.CosineAnnealingLR(opt,
    #                                                        5,
    #                                                        eta_min=0,
    #                                                        last_epoch=-1)

    model_fuzing_hand = model_fuzing_hand.to(device)
    model_dot_hand = model_dot_hand.to(device)
    model_hand = model_hand.to(device)

    train_dataset = GestureDatasetPics(os.path.join(config_data.data_path,
"train_dots.csv"))
    val_dataset = GestureDatasetPics(os.path.join(config_data.data_path,
"test_dots.csv"))

    train_dataloader = DataLoader(train_dataset,
batch_size=config_data.batch_size)
    val_dataloader = DataLoader(val_dataset, batch_size=config_data.batch_size)

    for e in range(0, config_data.epochs + 0):
        print("epoch = ", e)

        model_hand.train()
        model_dot_hand.train()
        model_fuzing_hand.train()

        running loss = 0.0
        precision = 0
        f1 = 0
        classification = 0
        accuracy = 0

        for x_hand, x_dot, y in iter(tqdm.tqdm(train_dataloader)):
            x_hand = x_hand.to(device)
            x_dot = x_dot.to(device)
            y = y.to(device)

            opt.zero_grad()
            hand_prediction = model_hand(x_hand)
            dot_prediction = model_dot_hand(x_dot)
            prediction = model_fuzing_hand(hand_prediction, dot_prediction)

            prediction_for_metrics = torch.max(prediction.data.cpu(), 1)[1]
            target_for_metrics = y.view(-1).cpu()

            precision += precision_score(target_for_metrics,
prediction_for_metrics,
                                         average='micro')
            f1 += f1_score(target_for_metrics, prediction_for_metrics,
                           average='micro',
                           zero_division='warn')
            # classification += classification_report(y,
torch.max(prediction.data, 1)[1],
            #                                         labels=[0, 1, 2, 3, 4, 5])
            accuracy += accuracy_score(target_for_metrics,
```

```
prediction_for_metrics, )
            # print(prediction_for_metrics)
            # print(prediction)
            # print(prediction.size())
            # print(y.size())
            loss_batch = loss(prediction, y) / len(y)
            loss_batch.backward()
            opt.step()
            running_loss += loss_batch.item()

        scheduler.step()

        running_loss = running_loss / len(train_dataloader)
        precision = precision / len(train_dataloader)
        f1 = f1 / len(train_dataloader)
        # classification = classification / len(train_dataloader)
        accuracy = accuracy / len(train_dataloader)

        writer_train.add_scalar('Loss_train', running_loss, e)
        writer_train.add_scalar('Precision_train', precision, e)
        writer_train.add_scalar('f1_train', f1, e)
        # writer_train.add_scalar('classification_train', classification, e)
        writer_train.add_scalar('accuracy_train', accuracy, e)

        # print('epoch = %d train_loss = %.4f' % (e, running_loss))
        # print('precision = %.4f' % (precision))

        # Save models
        if e % 5 == 0:
            torch.save(model_hand, os.path.join(config_data.model_path,
'model_hand_{}.pth'.format(e)))
            torch.save(model_dot_hand, os.path.join(config_data.model_path,
'model_dot_hand_{}.pth'.format(e)))
            torch.save(model_fuzing_hand, os.path.join(config_data.model_path,
'model_fuzing_hand_{}.pth'.format(e)))

        model_fuzing_hand.eval()
        running_val_loss = 0.0
        precision = 0
        f1 = 0
        classification = 0
        accuracy = 0

        for x_hand, x_dot, y in iter(tqdm.tqdm(val_dataloader)):
            x_hand = x_hand.to(device)
            x_dot = x_dot.to(device)
            y = y.to(device)

            with torch.no_grad():
                hand_prediction = model_hand(x_hand)
                dot_prediction = model_dot_hand(x_dot)
                prediction = model_fuzing_hand(hand_prediction, dot_prediction)

                prediction_for_metrics = torch.max(prediction.data.cpu(), 1)[1]
                target_for_metrics = y.view(-1).cpu()

                precision += precision_score(target_for_metrics,
prediction_for_metrics,
                                             average='micro')
```

```
                f1 += f1_score(target_for_metrics, prediction_for_metrics,
                               average='micro',
                               zero_division='warn')
                # classification += classification_report(y,
torch.max(prediction.data, 1)[1],
                #                                         labels=[0, 1, 2, 3, 4,
5])
                accuracy += accuracy_score(target_for_metrics,
prediction_for_metrics)

                loss_val_batch = loss(prediction, y) / len(y)
                running_val_loss += loss_val_batch.item()

        precision = precision / len(train_dataloader)
        f1 = f1 / len(train_dataloader)
        # classification = classification / len(train_dataloader)
        accuracy = accuracy / len(train_dataloader)
        running_val_loss = running_val_loss / len(val_dataloader)

        writer_test.add_scalar('Loss_test', running_loss, e)
        writer_test.add_scalar('Precision_test', precision, e)
        writer_test.add_scalar('f1_test', f1, e)
        # writer_test.add_scalar('classification_test', classification, e)
        writer_test.add_scalar('accuracy_test', accuracy, e)

        # print('epoch = %d val_loss = %.4f' % (e, running_val_loss))
```

Листинг 7 – train_test_divide.py

```
import os
import cv2

data_path = "/Users/illusivesheep/Repositories/ультра датасет"

classes_folders = ["бумага", "камень", "лайк", "дизлайк", "коза", "ножницы"]
modes = ["train", "test"]

for cur_class in classes_folders:

    path = os.path.join(data_path, cur_class)
    index = 0
    for image_name in [x for x in os.listdir(path) if x.find(".png") >= 0]:
        image_path = os.path.join(path, image_name)
        index += 1
        if index > 10:
            index = 0
        image = cv2.imread(image_path)
        print(image_path)
        if index == 2 or index == 5 or index == 9:
            cv2.imwrite(os.path.join(data_path, modes[1], cur_class,
image_name), image)
        else:
            cv2.imwrite(os.path.join(data_path, modes[0], cur_class,
image_name), image)
```

Листинг 8 – config_classes.py

```
from dataclasses import dataclass


@dataclass
class StartConfig:
    gpu: int
    mode: list
    data_path: str
    log_path: str
    model_path: str
    learning_rate: float
    learning_rate_fusing_coefficient: float
    weight decay: float
    loss_type: str
    epochs: int
    batch_size: int
    pretrained_models: bool
```

Листинг 9 - start_config.yaml

```
gpu: 0
mode: [preprocessing] # preprocessing, classic, train', test',inference
data_path: D:\Datasets\Gestures
log_path: log
model_path: models
learning_rate: 0.00001
learning_rate_fusing_coefficient: 10
weight_decay: 0.0005
loss_type: MSE #L2, Smooth L1, MSE
epochs: 200
batch_size: 65
pretrained_models: False
```

Листинг 10 – utils.py

```
import yaml


def get_config_data(path_to_config: str) -> dict:
    with open(path_to_config, 'r') as config_file:  # encoding = utf-8
        return yaml.load(config_file, Loader=yaml.FullLoader)
```