ДОДАТОК А

СЛАЙДИ ПРЕЗЕНТАЦІЇ

Атестаційна робота магістра

Дослідження підходів до розробки системи сегментації тіла людини на потоковому відео стосовно пози людини

Керівник:
проф. Білоус Н. В.

Виконав:
студент групи
ІПЗм-18-2
Богомаз А. С.

Визначення поз

Задача визначення пози полягає у знаходженні та ідентифікації ключових точок - суглобів. Знаходження їхніх координат дозволяє визначити їх на зображенні, а ідентифікація, яка з цих точок якому суглобу відповідає, дозволяє поєднати їх належним чином для побудови “скелета” людини.

2

Порівняння поз

Проблемою порівняння поз є те, що люди мають різні пропорції, розміри та положення у кадрі.

3

Актуальність

Система, що визначає позу на потоковому відео та може порівнювати її з еталонною може мати застосування для визначення правильності виконання вправ у спорті або медицині.

4

Постановка задачі

Для реалізації системи, що визначає правильність виконання вправ на відео з камери користувача, необхідно провести наступні дослідження:

1. Дослідити існуючі бібліотеки для визначення поз. Рекомендувати бібліотеку і метод її застосування у системі.
2. Дослідити методи порівняння визначених на зображені поз. Порівняти та рекомендувати до використання один з них.

5

Постановка задачі

Для експериментального дослідження розробити систему перевірки якості виконання вправ. На підставі її дослідження зробити висновки стосовно результатів досліджень. Надати рекомендації щодо покращення працездатності системи та її подальшого розвитку.

6

Вимоги до системи

Система повинна надавати користувачу наступний функціонал:

1. Можливість перегляду та пошуку вправ для виконання
2. Можливість перевірки правильності виконання обраної вправи
3. Можливість реєстрації та авторизації в системі
4. Можливість створення вправ

7

Аналіз існуючих бібліотек для визначення поз

Теоретичне дослідження якості визначення поз бібліотеками проводилося на підставі результатів що вони показали при тестуванні на датасеті MS COCO. Він містить понад 330 тис. зображень з яких понад 200 тисяч анотовані.

8

Аналіз існуючих бібліотек для визначення поз

В ході аналізу було знайдено три бібліотеки з відкритим початковим кодом, що надають можливість визначення пози на зображенні:

1. HighHRNet - має API для використання мовою програмування Python
2. OpenPose - має API для використання мовою програмування Python та платформою Unity
3. PoseNet - має API для використання мовами програмування Python, C++, Java, Swift, Objective C та Javascript

9

Аналіз існуючих бібліотек для визначення поз

Порівняння якості визначення поз бібліотеками

	AP	AP^{50}	AP^{75}	AP^{M}	AP^{L}
PoseNet	69.6	86.3	76.6	65.0	76.3
OpenPose	70.4	88.6	77.8	67.0	76.9
HRNet	70.0	87.8	76.1	64.8	77.3

10

Результати аналізу бібліотек для визначення поз

Аналізуючи результати порівняння точності визначення поз, можна побачити, що різниця між ними не перевершує 2%. А підтримка мов програмування JavaScript, Java та Swift є значною перевагою бібліотеки PoseNet при розробці, бо дозволяє реалізувати мобільну та веб-версію додатку без необхідності виконання обчислень на сервері, що забезпечує значно більшу швидкість при роботі.

11

Дослідження можливостей бібліотеки PoseNet

Бібліотека PoseNet має декілька налаштувань, основним серед яких є вибір архітектури нейронної мережі, що лежить в основі методу визначення поз:

1. Архітектура ResNet50 має більшу точність визначення, але меншу швидкість
2. Архітектура MobileNetV1 має меншу точність, але оптимізована для використання на мобільних пристроях

Кожна з архітектур має додаткові налаштування для оптимізації у бік якості та або у бік швидкості визначення поз.

12

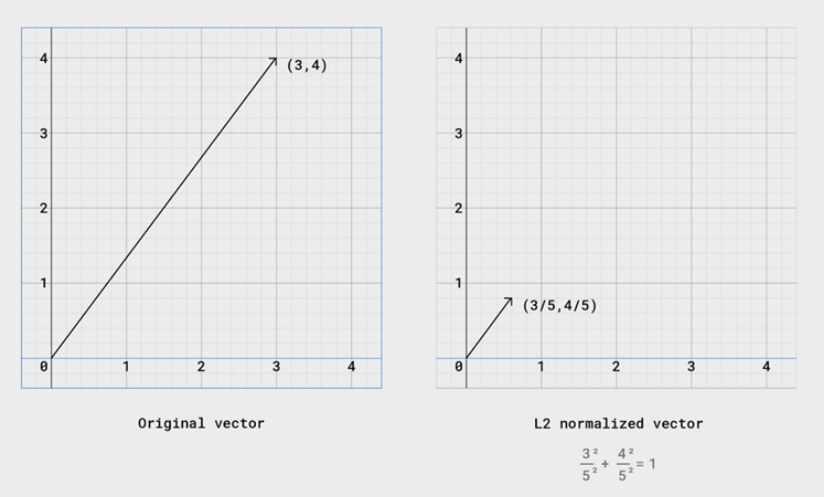
Методи порівняння поз

Методи порівняння поз базуються на побудові з даних про пози багатовимірних векторів та обчислення дистанції між ними. Для дослідження пропонуються наступні методи:

1. Метод косинусоїдної відстані - вектори будуються із значень координат ключових точок
2. Метод зваженої відстані - відстань між векторами визначається з урахуванням точності визначення кожної точки
3. Метод дистанції за обчисленими кутами - вектори будуються із значень кутів між кінцівками людини

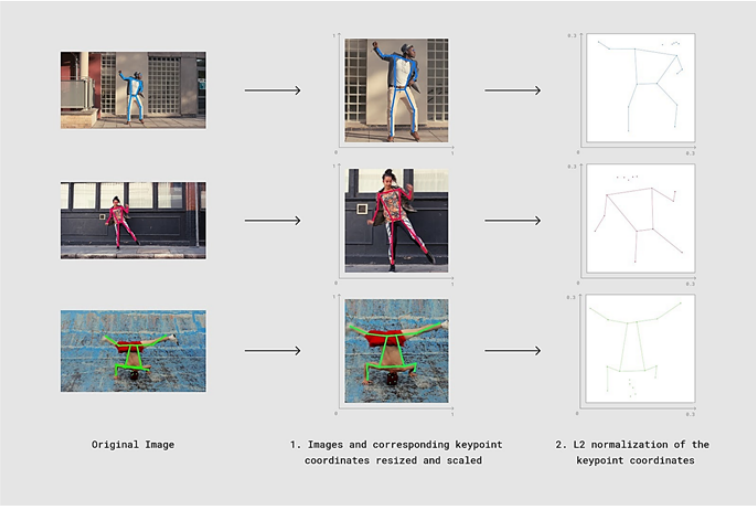
13

L2-нормалізація



14

Нормалізація зображення



15

Метод косинусоїдної відстані

$$D(Fxy, Gxy) = \sqrt{2 * (1 - cosineSimilarity(Fxy, Gxy))}$$

Fxy і *Gxy* - це два вектори, що описують пози, для порівняння після L2-нормалізації.

Fxy і *Gxy* містять лише x і y позиції для кожної з 17 ключових точок.

16

Метод зважених дистанцій

$$D(F,G) = \underbrace{\frac{1}{\sum_{k=1}^{17} Fc_k}}_{\text{Summation 1}} \times \underbrace{\sum_{k=1}^{17} Fc_k ||Fxy_k - Gxy_k||}_{\text{Summation 2}}$$

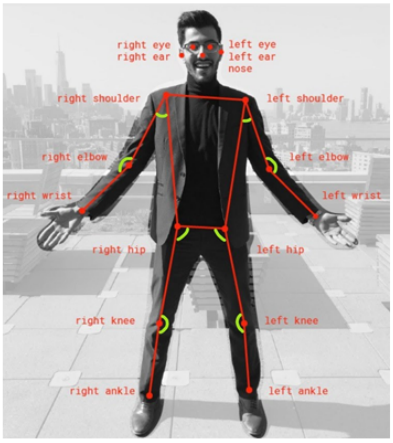
Fxy і *Gxy* - це два вектори, що описують пози, для порівняння після L2-нормалізації.

Fck - оцінка достовірності k-ї ключової точки *F*.

Fxy і *Gxy* представляють x і y позиції k-ї ключової точки для кожного вектора.

17

Метод дистанції за обчисленими кутами



Метод не потребує попередньої нормалізації координат та будується зі значень косинусів кутів, позначених на зображенні. Між отриманими векторами вимірюється косинусоїдна відстань.

18

Дослідження методів порівняння поз

В ході дослідження було зібрано датасет, що складається з 521 зображення, які об'єднуються в 43 групи поз. Кожна з груп містить зображення онакових поз з одного ракурсу. На рисунку зображено приклади зображень. Зображення *а)* та *б)* анотовані як такі, що містять однакові пози, зображення *в)* містить ту ж саму позу, але зображує її в іншому ракурсі, тому не входить до тієї ж самої групи.

а) б) в)

19

Результати дослідження методів порівняння поз

	ResNet50						MobileNetV1					
	Більша якість			Вища швидкість			Більша якість			Вища швидкість		
	CD	AD	WD	CD	AD	WD	CD	AD	WD	CD	AD	WD
$F_1^{0.05}$	48.1	50.3	60.2	47.3	46.4	43.9	0	12.2	18.1	12.4	11.3	12.9
$F_1^{0.10}$	55.6	54.8	56.3	54.2	52.1	56.1	0	23.7	27.6	21.9	19.8	22.0
$F_1^{0.15}$	55.4	56.2	58.7	59.8	60.0	60.8	32.9	34.3	35.6	26.7	27.1	30.8
$F_1^{0.20}$	58.7	57.3	65.4	61.0	60.5	62.6	30.7	35.8	36.8	28.7	33.2	37.4
$F_1^{0.25}$	60.1	57.8	64.9	62.2	62.4	64.1	33.0	34.1	33.4	30.0	33.1	37.1
$F_1^{0.30}$	58.2	56.2	59.0	60.4	60.8	59.9	33.2	32.0	31.2	31.3	32.5	35.6
$F_1^{0.35}$	55.0	55.3	52.3	51.7	53.2	51.5	30.5	30.6	28.9	31.4	29.0	34.8
$F_1^{0.40}$	49.3	50.1	47.5	44.9	49.4	45.4	29.9	30.1	26.9	29.6	28.1	32.3

20

Висновки з дослідження

Оптимальною конфігурацією за швидкістю та якістю визначення поз є конфігурація на базі архітектури ResNet50 з оптимізацією у бік швидкості.

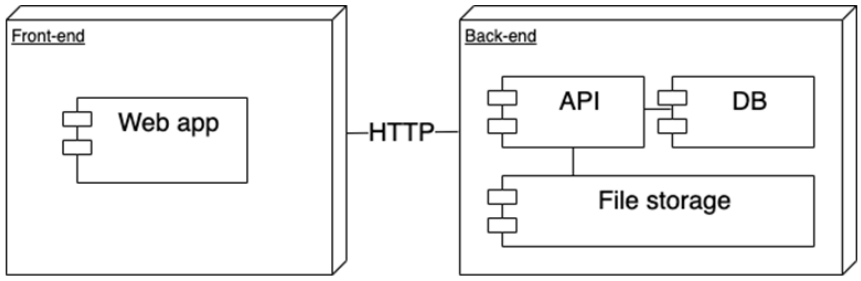
Найкращі результати порівняння поз надає метод зважених дистанцій.

Для якісного розпізнавання швидкість рухів на відео не повинна бути високою, ракурс зображень та відео повинен бути таким, щоб усі ключові точки було видно.

21

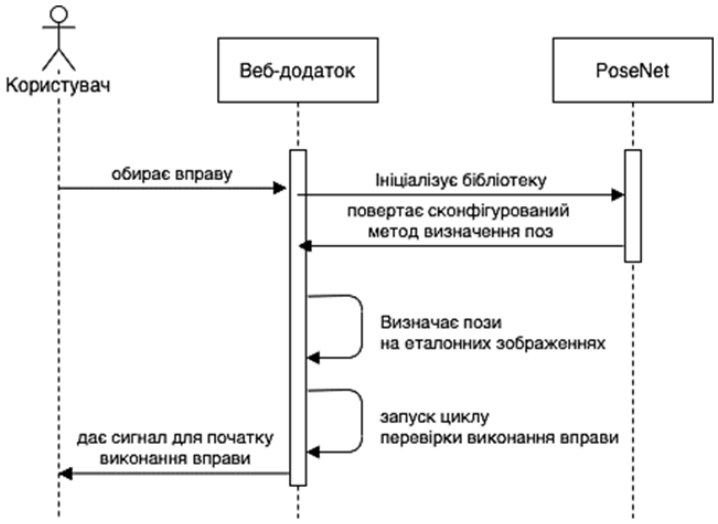
Проектування ПЗ

Діаграма розгортання



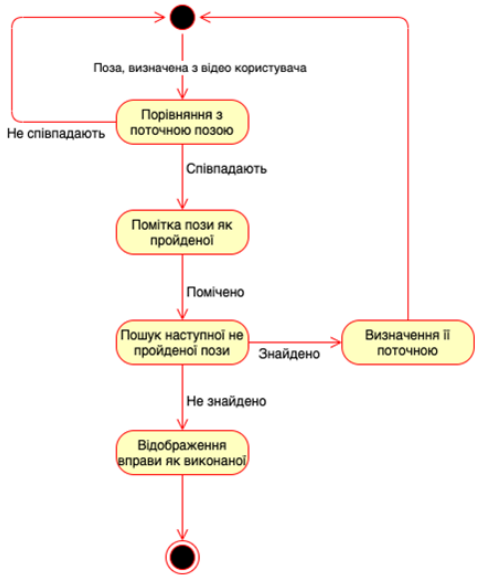
23

Діаграма послідовності для ініціалізації модулю перевірки правильності виконання вправи



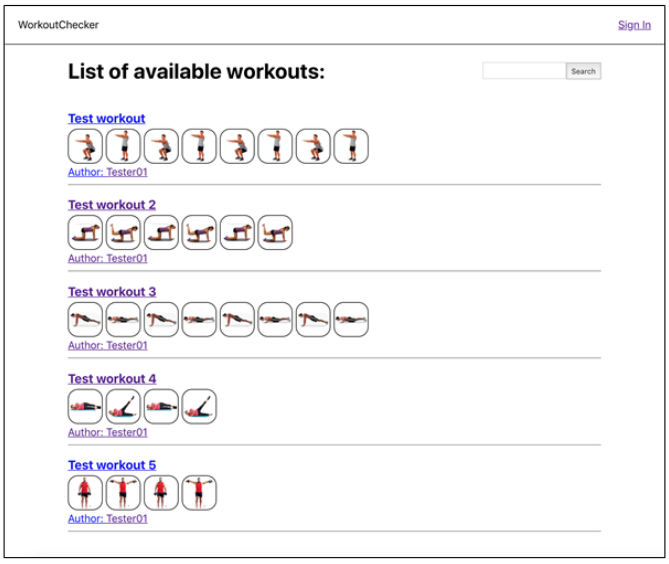
24

Діаграма активності для циклу визначення виконання вправи



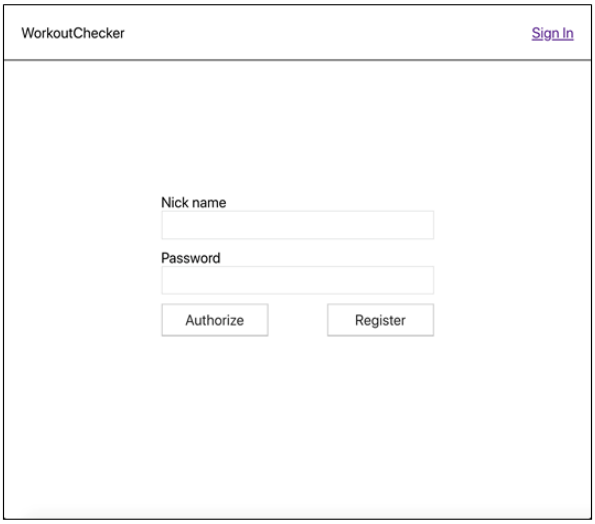
25

Інтерфейс додатку. Сторінка вибору вправи



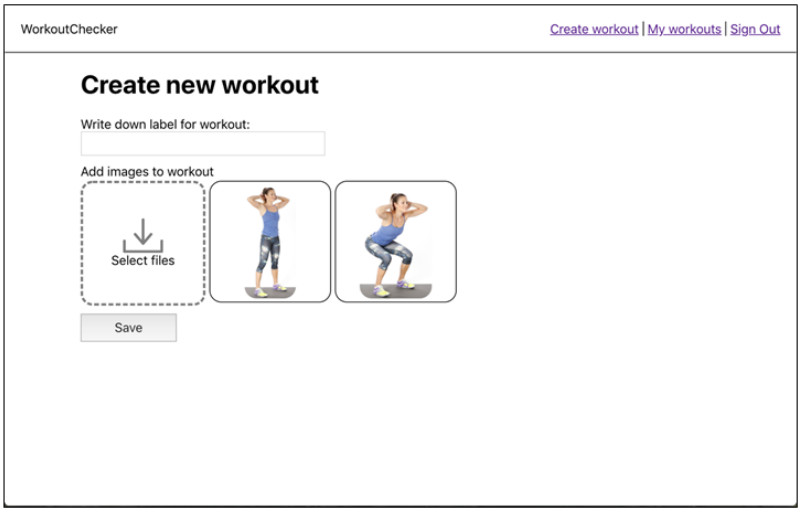
26

Інтерфейс додатку. Сторінка авторизації



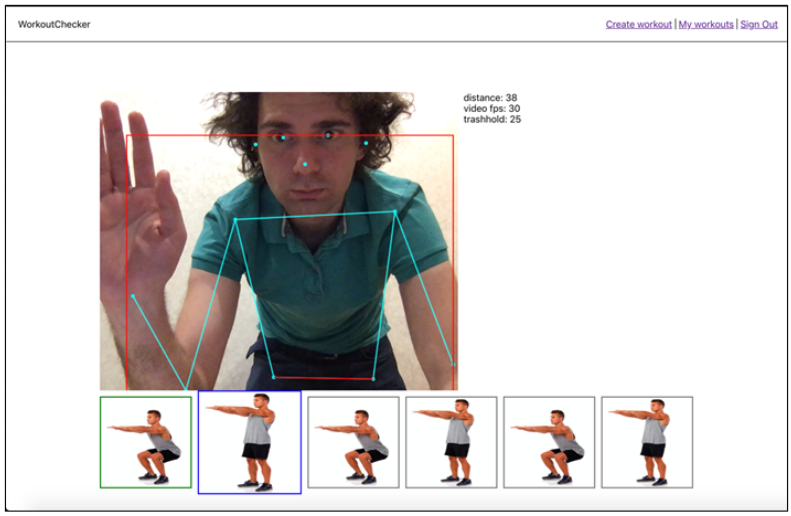
27

Інтерфейс додатку. Сторінка створення вправи



28

Інтерфейс додатку. Сторінка виконання вправи



29

Тестування

В ході тестування було визначено значення FPS для відео та для конфігурацій бібліотеки PoseNet з використанням архітектур ResNet50 та MobileNetV1.

Тестування проводилося на трьох пристроях (технічні характеристики буде наведено далі)

Були визначені основні причини помилок при визначенні пози.

30

Результати тестування

Показник FPS для відео з камери користувача на всіх пристроях: 30 кадрів на секунду.

В таблиці наведено отриману кількість кадрів, на яких було визначено позу за секунду на відповідних пристроях з відповідною конфігурацією бібліотеки.

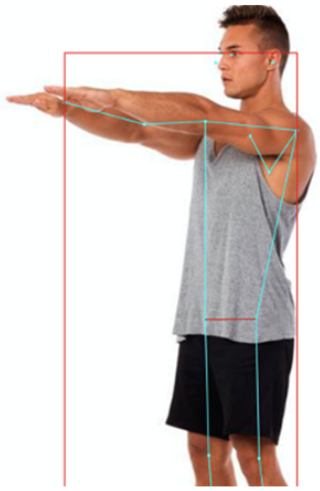
	MacBook Pro (2.9HGz Intel Core i7)	iPhone 8 (2.3 HGz Apple A11 Bionic)	Sony Xperia X (1.8 HGz Qualcomm Snapdragon 650)
ResNet50	13	8	5
MobileNetV1	24	21	17

31

Основні помилки при визначенні поз

Основною причиною помилок при визначенні поз є перетин кінцівок та “приховування” ключових точок.

Також проблеми виникають при знаходженні людей або інших об’єктів, що можуть нагадувати частини тіла людини на фоні.



32

Рекомендації щодо покращення роботи системи та подальших досліджень

Для покращення якості визначення поз на еталонних зображеннях необхідно покращити процес створення вправ, а саме визначати пози на зображенні при створенні з використанням найточнішої конфігурації для визначення. Наступним етапом може бути створення редактору для ручного виправлення помилок при визначенні пози.

В подальшому необхідно дослідити можливість та ефективність реалізації подібної системи у вигляді нативних додатків для мобільних системи та ПК, а не у вигляді веб-додатку. Браузер накладає обмеження швидкості виконання, яких немає при написанні додатків.

33

Висновки

В результаті аналізу існуючих бібліотек для визначення пози людини на зображені з відкритим кодом було виявлено що вони мають доволі близькі значення щодо якості визначення, проте бібліотека PoseNet надає значні переваги з точки зору впровадження її до системи завдяки широкій підтримці мов програмування.

Серед методів порівняння поз найкращі результати продемонстрував метод зважених дистанцій.

34

Висновки

При дослідженні можливостей бібліотеки PoseNet було визначено, що найкращі результати з точки зору швидкості та якості визначення поз дає конфігурація бібліотеки, що базується на архітектурі нейронної мережі ResNet50 з оптимізацією у бік швидкості.

Існуючі методи накладають обмеження на вправи з боку швидкості рухів. При швидких діях користувача система може не встигнути обробити зміни зображення, тому для визначення системою пози як правильної треба робити невеличкі паузи у виконанні вправи. Також точність як методів визначення так і методів порівняння надає можливість визначати пози лише з невеликою деталізацією. Наприклад, відмінність у положенні кистей будуть проігноровані системою.

35

Дякую за увагу!

ДОДАТОК Б

ЛІСТИНГ КОДУ

```
import React, { useRef, useEffect, useState, DetailedHTMLProps, VideoHTMLAttributes } from
'react';
import * as posenet from '@tensorflow-models/posenet';
import { compare } from './compare';
import { PoseState } from './models/pose-state';

import pose1Img from './content/pose 1.png';
import pose2Img from './content/pose 2.png'
import { Workout } from './components/workout';
import { PoseNet, Pose } from '@tensorflow-models/posenet';
import { PoseVideo } from './components/pose-video';
import { CurrentPose } from './CurrentPose';

const videoWidth = 600;
const videoHeight = 500;
const DISTANCE_TRASHHOLD = 25;
const MIN_CONFIDENCE = 0.3;

export function WorkoutChecker() {

  const [workout, setWorkouts] = useState<PoseState[]>(poses);
  const [appLoaded, setAppLoaded] = useState<boolean>(false);
  const [distance, setDistance] = useState<number>(0);
  const [videoPose, setVideoPose] = useState<Pose | null>(null);
  const [lastPassedTime, setLastPAssedTime] = useState<Date>(new Date());


  const videoRef = useRef<HTMLVideoElement>(null);
  const posenetRef = useRef<PoseNet | null>(null)

  useEffect(() => {
    const video = videoRef.current;
    if (!video) {
      return;
    }


    const initAsync = async () => {
      if (!navigator.mediaDevices || !navigator.mediaDevices.getUserMedia) {
        throw new Error(
          'Browser API navigator.mediaDevices.getUserMedia not available');
        }

        video.width = videoWidth;
        video.height = videoHeight;
        const stream = await navigator.mediaDevices.getUserMedia({
```

```
        'audio': false,
        'video': {
          facingMode: 'user',
          width: videoWidth,
          height: videoHeight,
        },
      });
      video.srcObject = stream;

      video.onloadeddata = () => {
        posenet.load({
          architecture: 'ResNet50',
          outputStride: 32,
          quantBytes: 1,
          inputResolution: 256,
        }).then(net => {
          posenetRef.current = net;
          console.log('loaded')
          setAppLoaded(true)
        }) ;
      };
    }

    initAsync();

}, []);

useEffect(() => {
  if (!appLoaded) {
    return;
  }
  const net = posenetRef.current;
  const video = videoRef.current;
  if (!net || !video) {
    return;
  }

  const run = async () => {

    const pose1 = await net.estimateSinglePose(video, {flipHorizontal: true});
    const img = new Image();
    const currentPose = workout.find(p => p.status === 'inprogress');
    if (!currentPose) {
      return;
    }
    img.src = currentPose.img;
    console.log(currentPose.img);
    const pose2 = await net.estimateSinglePose(img);

    const distance = Math.floor(compare(pose1, pose2, MIN_CONFIDENCE) * 100);
```

```
      if (distance < DISTANCE_TRASHHOLD && new Date().getTime() - lastPassedTime.getTime()
> 1500) {
        console.log(new Date().getTime() - lastPassedTime.getTime());
        setWorkouts(workout => {
          const current = workout.find(p => p.status === 'inprogress');
          if (!current) {
            return workout;
          }
          const next = workout.find(p => p.order === current.order + 1);
          if (!next) {
            return workout.map(p => {p.status = 'done'; return p;});
          } else {
            const others = workout.filter(p => p.order !== current.order && p.order !==
next.order);
            return [...others, {...current, status: 'done'}, {...next, status:
'inprogress'}];
          }
        });
        setLastPAssedTime(new Date());
      }

      setVideoPose(pose1);
      setDistance(distance);
    }
    run();
  }, [videoPose, lastPassedTime, appLoaded, setWorkouts, setLastPAssedTime, workout])

  return (
    <div style={{display:'flex', justifyContent: 'center', alignItems: 'center', height:
'100vh'}}>
      <div style={{width: '1000px'}}>
        <div style={{display: 'flex'}}>
          <div style={{position: 'relative', width: 600, height: 500}}>
            <video style={{ position: 'absolute', transform: 'scaleX(-1)', width:600,
height: 500, top: 0, left: 0 }} ref={videoRef} autoPlay></video>
            <div style={{position: 'absolute', top: 0, left: 0, zIndex: 5}}>
            {videoRef.current ? <PoseVideo pose={videoPose}
video={videoRef.current}></PoseVideo> : null}
            </div>
          </div>
          <div style={{marginLeft: '10px'}}>
            distance: {distance} <br />
            video fps: {(videoRef?.current?.srcObject as
MediaStream)?.getVideoTracks()[0]?.getSettings().frameRate.toFixed(0)} <br />
            trashhold: {DISTANCE_TRASHHOLD}
          </div>
        </div>
        <Workout poses={workout} />
      </div>
    </div>
  );
}
```

```
import { Pose } from '@tensorflow-models/posenet';
import { Keypoint, getBoundingBox } from "@tensorflow-models/posenet";
const similarity = require('compute-cosine-similarity');

export function compare(pose1: Pose, pose2: Pose, minConfidence = 0.1) {
    const poseVector1 = getNormalizedVector(pose1.keypoints);
    const poseVector2 = getNormalizedVector(pose2.keypoints);

    let vector1PoseXY = poseVector1.slice(0, 34);
    let vector1Confidences = pose1.keypoints.map(k => k.score)
    const vector1ConfidenceSum = vector1Confidences.reduce((sum, score) => sum + score, 0);
    let vector2PoseXY = poseVector2.slice(0, 34);
    let summation1 = 1 / vector1ConfidenceSum;
    let summation2 = 0;
    for (let i = 0; i < vector1PoseXY.length; i++) {
        let tempConf = Math.floor(i / 2);
        let tempSum =
            vector1Confidences[tempConf] * Math.abs(vector1PoseXY[i] - vector2PoseXY[i]);
        summation2 = summation2 + tempSum;
    }

    return summation1 * summation2;
}



function cosineDistanceMatching(poseVector1, poseVector2) {
    let cosineSimilarity = similarity(poseVector1, poseVector2);
    let distance = 2 * (1 - cosineSimilarity);
    return Math.sqrt(distance);
  }

export function getNormalizedVector(keypoints: Keypoint[]): number[] {
    const { minX, minY } = getBoundingBox(keypoints);
    return keypoints.map(k => {
        return [
            ...L2normalization(k.position.x - minX, k.position.y - minY),
            k.score
        ]
    }).reduce((vector: number[], [x, y]: number[]) => {
        return [...vector, x, y];
    }, [])
}

function L2normalization(x: number, y: number): [number, number] {
    const length = Math.sqrt(x * x + y * y);
    return [x / length, y / length];
}
import React from "react";

export function Auth() {
  return (
    <div style={{
```

```
      display: 'flex',
      justifyContent: 'center',
      alignItems: 'center',
      height: '80vh'
    }}>
      <div>
        <div style={{marginBottom: 10}}>
          <label>Nick name</label> <br />
          <input style={{height: 25, width: 300}} type='text' />
        </div>
        <div style={{marginBottom: 10}}>
        <label>Password</label> <br />
          <input style={{height: 25, width: 300}} type='password' />
        </div>
        <div style={{display: 'flex', justifyContent: 'space-between'}}>
          <button style={{height: 35, width: 120, fontSize: 16}}>Authorize</button>
          <button style={{height: 35, width: 120, fontSize: 16}}>Register</button>
        </div>
      </div>
    </div>
  )
}
import { Workout } from '../models/workout';
import React from 'react';
import { Link } from 'react-router-dom';

type WorkoutItemProps = {workout: Workout};

export function WorkoutItem({workout}: WorkoutItemProps) {

  const image = (source: string, index: number) => {
    return (
      <div style={{
        height: 50,
        width: 50,
        border: 'thin solid black',
        borderRadius: 15,
        marginRight: 5,
        flexShrink: 0,
        display: 'flex',
        justifyContent: 'center',
        alignItems: 'center'

      }}
      key={index}
      >
        <img
          src={source}
          alt='from workout'
          style={{
            maxWidth: '95%',
            maxHeight: '95%',
            height: 'auto',
```

```
            borderRadius: 15,
          }}
        />
      </div>
    )
  }

  return (
    <div style={{
      borderBottom: 'thin solid grey',
      // marginBottom: 15
    }}>
      <h3 style={{marginBottom: 5}}>{workout.name}</h3>
      <div style={{
        overflowX: 'scroll',
        display: 'flex',
        marginBottom: 3
      }}>
        {
          workout.poses.map(image)
        }
      </div>

      <div style={{marginBottom: 10}}>
        Author: <Link to={'/workouts'}>{workout.author}</Link>
      </div>
    </div>
  )
}
import React from "react";
import { useStore } from "../store";
import { Link } from 'react-router-dom';
import { WorkoutItem } from "./WorkoutItem";

export function WorkoutsList() {

  const wourkouts = useStore(s => s.workouts);

  return (
    <div style={{
      display: 'flex',
      justifyContent: 'center',
    }}>
      <div style={{
        width: '800px',
        overflow: 'scroll',
      }}>

        <div style={{
          display: 'flex',
          justifyContent: 'space-between',
          alignItems: 'center'
        }}>
```

```
          <h1>
            List of available workouts:
          </h1>

          <div>
            <input style={{height: 20, width: 120}} type="text" />
            <button style={{borderRadius: 0, height: '25px'}}>Search</button>
          </div>
        </div>

        <div>
          {
            wourkouts.map((w, i) => {
              return (
              <Link key={i} to={`/checker/${w.id}`}>
                <WorkoutItem workout={w} />
              </Link>
              );
            })
          }

        </div>


      </div>
    </div>
  )
}
import React, { useRef, useEffect } from "react";
import { Pose } from '@tensorflow-models/posenet';
import * as posenet from '@tensorflow-models/posenet';

const color = 'aqua';
const boundingBoxColor = 'red';
const lineWidth = 2;

export function PoseVideo({ pose, video }: { pose: Pose | null, video: HTMLVideoElement })
{

  const canvasRef = useRef<HTMLCanvasElement>(null);

  useEffect(() => {

    const canvas = canvasRef.current;
    if (!canvas) {
      return;
    }

    const ctx = canvas.getContext('2d');
    if (!ctx) {
      return;
    }
```

```
    ctx.clearRect(0, 0, video.width, video.height);
    ctx.save();
    ctx.scale(-1, 1);
    ctx.translate(-video.width, 0);
    // ctx.drawImage(video, 0, 0, video.width, video.height);
    ctx.restore();

    const minPartConfidence = 0.3;

    if(!pose) {
      console.log('!pose');
      return;
    }

    drawKeypoints(pose.keypoints, minPartConfidence, ctx);
    drawSkeleton(pose.keypoints, minPartConfidence, ctx);
    drawBoundingBox(pose.keypoints, ctx);
  })

  return (
    <canvas style={{top: 0, left: 0, zIndex: 5}} ref={canvasRef} width={600} height={500}/>
  )
}

export function drawKeypoints(keypoints, minConfidence, ctx, scale = 1) {
  for (let i = 0; i < keypoints.length; i++) {
    const keypoint = keypoints[i];

    if (keypoint.score < minConfidence) {
      continue;
    }

    const {y, x} = keypoint.position;
    drawPoint(ctx, y * scale, x * scale, 3, color);
  }
}

/**
 * Draw the bounding box of a pose. For example, for a whole person standing
 * in an image, the bounding box will begin at the nose and extend to one of
 * ankles
 */
export function drawBoundingBox(keypoints, ctx) {
  const boundingBox = posenet.getBoundingBox(keypoints);

  ctx.rect(
      boundingBox.minX, boundingBox.minY, boundingBox.maxX - boundingBox.minX,
      boundingBox.maxY - boundingBox.minY);

  ctx.strokeStyle = boundingBoxColor;
  ctx.stroke();
}
```

```
export function drawPoint(ctx, y, x, r, color) {
  ctx.beginPath();
  ctx.arc(x, y, r, 0, 2 * Math.PI);
  ctx.fillStyle = color;
  ctx.fill();
}

/**
 * Draws a pose skeleton by looking up all adjacent keypoints/joints
 */
export function drawSkeleton(keypoints, minConfidence, ctx, scale = 1) {
  const adjacentKeyPoints =
      posenet.getAdjacentKeyPoints(keypoints, minConfidence);

  adjacentKeyPoints.forEach((keypoints) => {
    drawSegment(
        toTuple(keypoints[0].position), toTuple(keypoints[1].position), color,
        scale, ctx);
  });
}


/**
 * Draws a line on a canvas, i.e. a joint
 */
export function drawSegment([ay, ax], [by, bx], color, scale, ctx) {
  ctx.beginPath();
  ctx.moveTo(ax * scale, ay * scale);
  ctx.lineTo(bx * scale, by * scale);
  ctx.lineWidth = lineWidth;
  ctx.strokeStyle = color;
  ctx.stroke();
}

function toTuple({y, x}): [any, any] {
  return [y, x];
}
import React from "react";

import pose61Img from '../content/pose 61.png'
import pose62Img from '../content/pose 62.png'

export function CreateWorkout() {

  const images = [
    pose61Img,
    pose62Img,
  ]

  const image = (source: string, index: number) => {
    return (
      <div style={{
        height: 150,
```

```
        width: 150,
        border: 'thin solid black',
        borderRadius: 15,
        marginRight: 5,
        flexShrink: 0,
        display: 'flex',
        justifyContent: 'center',
        alignItems: 'center'

      }}
      key={index}
      >
        <img
          src={source}
          alt='from workout'
          style={{
            maxWidth: '95%',
            maxHeight: '95%',
            height: 'auto',
            borderRadius: 15,
          }}
        />
      </div>
    )
  }

  return (
    <div>
      <div style={{
      display: 'flex',
      justifyContent: 'center',
    }}>
      <div style={{
        width: '800px',
        overflow: 'scroll',
      }}>
        <h1>
          Create new workout
        </h1>

        <div>
          <div style={{marginBottom: 10}}>
            <label>Write down label for workout: </label> <br />
            <input style={{height: 25, width: 300, fontSize: 14}} type='text' />
          </div>

          <div style={{marginBottom: 10}}>
            <label >Add images to workout </label> <br />
            <div style={{marginTop: 3, display: 'flex'}}>
              <div style={{
                height: 150,
                width: 150,
                border: '3px dashed grey',
```

```
                display: 'flex',
                justifyContent: 'center',
                alignItems: 'center',
                flexDirection: 'column',
                borderRadius: 15,
                marginRight: 5,
              }}>
                <svg color='grey' xmlns="http://www.w3.org/2000/svg" width="50" height="43"
viewBox="0 0 50 43"><path fill="grey" d="M48.4 26.5c-.9 0-1.7.7-1.7 1.7v11.6h-43.3v-11.6c0-
.9-.7-1.7-1.7-1.7s-1.7.7-1.7 1.7v13.2c0 .9.7 1.7 1.7 1.7h46.7c.9 0 1.7-.7 1.7-1.7v-13.2c0-
1-.7-1.7-1.7-1.7zm-24.5 6.1c.3.3.8.5 1.2.5.4 0 .9-.2 1.2-.5l10-11.6c.7-.7.7-1.7 0-2.4s-1.7-
.7-2.4 0l-7.1 8.3v-25.3c0-.9-.7-1.7-1.7-1.7s-1.7.7-1.7 1.7v25.3l-7.1-8.3c-.7-.7-1.7-.7-2.4
0s-.7 1.7 0 2.4l10 11.6z"></path></svg>
                Select files
              </div>

              {
                images.map(image)
              }
            </div>
          </div>

          <button style={{height: 35, width: 120, fontSize: 16}}>
            Save
          </button>

        </div>


      </div>
    </div>
    </div>
  )
}
```

ДОДАТОК В

СТАТТЯ ДЛЯ НАУКОВОГО ЖУРНАЛУ «РІУ»

Дослідження методів визначення та порівняння поз на потоковому відео

Білоус Н. В. – професор кафедри програмної інженерії, Харківський національний університет радіоелектроніки, Харків, Україна.

Богомаз А. С. – студент кафедри програмної інженерії, Харківський національний університет радіоелектроніки, Харків, Україна.

АНОТАЦІЯ

Актуальність. В сфері медицини та спорту актуальною є задача автоматизованого визначення правильності виконання вправ людиною. Для її вирішення пропонується реалізувати додаток, що отримує відео з камери користувача та аналізує його, порівнюючи зображення з відео з заданим набором зображень поз, що відображають правильне виконання вправи. Таким чином, кожен кадр відео порівнюється з зображенням, що відповідає наступному положенню тіла у вправі. Для зручності використання, додаток має бути доступним для використання як з комп'ютерів, так і з мобільних пристроїв. Для реалізації додатку необхідно визначити позу на відео та на зображенні, та порівняти їх зі швидкістю близькою до 40мс, для отримання частоти зміни кадрів 24 кадри на секунду. Роботу присвячено дослідженню бібліотек розпізнавання поз та методів їх порівняння та визначення які з них дозволять отримати бажані результати.

Мета. Дослідити ефективність роботи існуючих бібліотек для визначення пози людини на зображенні з точки зору швидкості та точності визначення. Дослідити методи порівняння отриманих поз з точки зору якості та швидкості. Запропонувати комбінацію бібліотеки визначення пози та методу порівняння поз для реалізації порівняння зображення з відео з позою на заданому зображенні у реальному часі.

Методи. При дослідженні методів визначення поз використовується датасет MS COCO та метрика AP, що поєднує в собі повноту та точність результатів визначення об'єктів. Для дослідження були взяті бібліотеки HRNet, MultiPoseNet, PoseNet та OpenPose. Для дослідження методів порівняння було зібрано набір зображень з 43 груп зображень на підставі зображень тренувальних вправ. Для оцінки якості порівняння було використано метрику F1. Досліджувані методи порівняння базуються на пошуку відстані між векторами, що описують пози. Метод дистанції за обчисленими кутами будує вектор зі значень кутів між кінцівками людини на зображені та розраховую косинусоїдну відстань між отриманими векторами. Методи косинусоїдної та зваженої дистанцій будують вектори з нормалізованих значень позицій суглобів на зображенні. Метод зваженої відстані також приймає до уваги точність виявлення ключової точки. Дослідження методів порівняння поз базується на результатах визначення поз бібліотекою PoseNet.

Результати. Існуючі моделі визначення поз показали схожі результати якості визначення з розбігом близько 2%. При розробці крос-платформного програмного продукту значну перевагу в швидкості має бібліотека PoseNet, що має API для роботи безпосередньо в браузері та на мобільних платформах. При дослідженні методів порівняння найбільший вплив на результати справила якість визначення пози. Серед методів порівняння найкращі результати продемонстрував метод зважених дистанцій. Швидкість визначення поз обернено пропорційна якості визначення і значно перевищує рекомендоване значення – 40мс.

Висновки. Розглянуті методи визначення поз можуть бути впроваджені для аналізу потокового відео лише при умові низької деталізації поз та швидкості рухів, в інакшому випадку швидкості та якості існуючих бібліотек недостатньо для вирішення цієї задачі. Вперше було створено порівняльну характеристику методів порівняння поз на зображенні. Результати досліджень методів порівняння поз можуть буди використані в подальших дослідженнях з використанням оптимізації для методів визначення поз.

Ключові слова: комп'ютерний зір, pose estimation, порівняння поз, PoseNet, ResNet50, MobileNetV1.

АББРЕВІАТУРИ

AP – Average precission
CD – Cosine Distance
AD – Angle Distance
WD – Weighted Distance
PSM – Pictoral Structures Model
OKS – Object Keypoint Similarity
API – Application Public Interface

ВСТУП

Однією з задач комп'ютерного зору є задача визначення тіла людини на зображенні. Існує багато методів вирішення цієї задачі, деякі базуються на специфічному обладнані (motion capture, kinect) та надають найбільшу точність, деякі дають меншу точність але не потребують додаткового обладнання та використовують меншу обчислювальну потужність. Останні мають дуже широку сферу застосування, наприклад для індивідуальних спортивних тренувань або при реабілітаційних вправах. В наведених сферах велику значимість має вирішення задачі порівняння пози людини з еталонною для виявлення правильності виконання тієї чи іншої вправи.

Задачу визначення пози людини можна визначити як задачу пошуку точок сполучення на тілі людини, також відомих як ключові точки – лікті, зап'ястя, коліна та інші. Знайшовши ці точки можна побудувати скелет людини и таким чином визначити безпосередньо позу людини.

Класичним варіантом вирішення цієї задачі є використання моделі PSM, що представляє людину у вигляді графа у якому вершинами є частини тіла, а сполученнями є суглоби.

Сучасні методи базуються на технологіях "Deep Learning" та у якості результату визначають позиції суглобів та сполучення між ними. Одним з найбільших досягнень сучасних моделей є те, що навіть якщо якогось суглобу не видно на зображення моделі можуть його передбачити з певною долею впевненості.

Для оцінки якості визначення поз було використано датасет MS COCO[1], що використовують показник OKS – схожіть ключових точок об'єкта. Він обчислюється з відстані між прогнозованими точками та розміченими точками, нормалізованими за масштабом людини. Константа масштабу та ключової точки, необхідна для вирівнювання важливості кожної

ключової точки: розташування шиї точніше, ніж розташування стегна.

$$OKS = \exp\left(-\frac{d_i^2}{2s^2k_i^2}\right), \quad (1)$$

де di – евклідова відстань між реальною ключовою точкою та передбачуваною ключовою точкою;

s – масштаб: площа обмежувального поля, поділена на загальну площу зображення;

k – константа що визначена окремо для кожної контрольної точки.

Константи для ключових точок обчислені групою дослідників з MS COCO.

Однак цей засіб порівняння поз не може бути використаний при порівнянні різних людей на різних зображеннях, бо вони матимуть різну позицію на зображенні, різні пропорції тіла та відстань до камери. Запропонований засіб порівняння повинен враховувати всі ці фактори.

Мінімальна частота зміни кадрів для якісного відображення відео – 24 кадри на секунду. Тобто час обробки кожного кадру повинен займати приблизно 40мс для повного аналізу відео-потоку. Аналіз можливий навіть у випадку, якщо процес займає більше часу, але в такому випадку буде втрачено якусь кількість кадрів, що заперечить можливість аналізу швидких рухів.

1 ПОСТАНОВКА ЗАДАЧІ

В результаті роботи необхідно дослідити існуючі алгоритми порівняння поз та запропонувати алгоритм, що буде придатним для використання при аналізі потокового відео та порівнянні його з заданим зображенням.

В результаті дослідження необхідно отримати значення часу, який займає визначення пози на зображенні.

Метод порівняння повинен враховувати, що відстань до камери та позиція в кадрі можуть відрізнятися на різних зображеннях.

2 ОГЛЯД ЛІТЕРАТУРИ

При аналізі існуючих бібліотек для визначення пози було проаналізовано інструменти, що дозволяють розпізнавання за двовимірним кольоровим зображенням та мають відкритий початковий код.

HRNet є результатом доволі простої ідеї. Більшість попередніх робіт виходили з того, що аналізували дані з високого до низького рівня деталізації. HRNet [2] підтримує представлення високої роздільної здатності протягом усього процесу. Архітектура починається з підмережі високої роздільної здатності як перший етап, і поступово додає підмережі високої та низької роздільної здатності по черзі, щоб утворювати більше слоїв та об’єднує паралельні підмережі з декількома різними роздільними здатністями. Повторні багатомасштабні синтези проводяться шляхом обміну інформацією в паралельних підмережах з різною роздільною здатністю протягом усього і протягом усього процесу.

Ця модель може бути інтегрована в програмну систему за допомоги мови програмування python з використанням бібліотеки OpenCV.

MultiPoseNet – нова архітектура для виконання визначення пози людини на зображенні методом «знизу до гори». MultiPoseNet може спільно працювати над виявленням людей, виявленням ключових точок, сегментацією особи та оцінкою пози [3]. Швидкість аналізу відео в реальному часі складає приблизно 23 кадри на секунду. Як і HRNet, MultiPoseNet має можливість інтеграції до програмного продукту за допомогою мови Python.

OpenPose має API для python та plugin для ігрового двигуна Unity. Всередині він використовує мультіпоточну оптимізацію, що дозволяє прискорити швидкість обробки зображення та відповідно знаходити більше контрольних точок на зображенні в умовах потокового відео. Згідно офіційної документації OpenPose може визначати 25 ключових точок при оцінці тіла/ніг, 2x21 ключову точку при оцінці рук та 70 точок при аналізі зображення обличчя [4].

PoseNet – базується на фреймворку TensorFlow Light та вміє розрізняти 17 ключових точок на зображені. Важливою деталлю, яку слід зазначити, є те, що дослідники підготували як ResNet, так і MobileNet модель PoseNet. Модель ResNet має більш високу точність, але має великий розмір і багато шарів, у той час як модель MobileNet розроблена для роботи на мобільних пристроях [5]. Ця модель може бути інтегрована в програмний продукт за допомогою великої кількості мов програмування, а саме в Python, C++, Java, Swift, Objective C та Javascript.

Для аналізу цих моделей було обрано датасет MS COCO, що містить близько 250 000 об’єктів людей з розміченими ключовими точками.

Методи порівняння у загальному вигляді представляють собою знаходження відстані між векторами, побудованими на підставі отриманих результатів при визначенні пози. Перший метод[6] базується на обчисленні кутів між частинами тіла. Кути вираховуються з отриманих координат суглобів, з них будується багатовимірний вектор та вираховується косинусоїдна відстань між отриманими векторами.

Наступні два методи запропоновані в якості експериментального дослідження від компанії Google [7]. Обидва методи включають нормалізацію початкової пози, побудову вектора з нормалізованих координат ключових точок та розрахунок відстані між векторами. Перший метод використовує метод косинусоїдної відстані, другий – метод зваженої відстані, яка приймає до уваги значення впевненості з якою нейронна мережа визначає положення ключової точки.

3 МАТЕРІАЛИ І МЕТОДИ

Процес нормалізації координат позицій ключових точок виглядає наступним чином. Значення координат

поставленої задачі не є значною перевагою, бо PoseNet може використовуватись на більшій кількості платформ, що значно збільшує швидкість при використанні мобільних приладів.

Таблиця 1 – Результати порівняння методів сегментації тіла людини

	AP	AP^{50}	AP^{75}	AP^{M}	AP^{L}
PoseNet	69.6	86.3	76.6	65.0	76.3
OpenPose	70.4	88.6	77.8	67.0	76.9
HRNet	70.0	87.8	76.1	64.8	77.3
MultiPoseNet	64.2	86.2	70.1	61.0	68.8

Дослідження методів порівняння поз проводилися на базі моделі PoseNet. Результати наведено в таблиці 2. Позначками CA, AD та WD позначаються результати для методів порівняння косинусоїдної дистанції, дистанції за обчисленими кутами та зваженої відстані, відповідно.

Зокрема архітектури мережі, модель має наступні налаштування. OutputStride - може мати значення 8, 16 або 32 (значення 16 та 32 підтримуються архітектурою ResNet, а повний перелік підтримуються архітектурою MobileNetV1). Він визначає вихідний крок моделі PoseNet. Чим менше значення, тим більше вихідна роздільна здатність і точніша модель, проте вона втрачає в швидкості. Multiplier - може приймати значення 1.01, 1.0, 0.75 або 0.50 (значення використовується лише архітектурою MobileNetV1). Це множник з плаваючою комою для глибини (кількість каналів) для всіх операцій згортки. Чим більше значення, тим більший розмір шарів і точніша модель за рахунок швидкості. QuantBytes - Цей аргумент керує байтами, які використовуються для квантування[12] ваги. Може приймати значення 4, 2 та 1. Вливає на якість розпізнавання та розмір моделі. Значення 4 – повнорозмірна модель, що має найвищу точність та розмір приблизно 90 мегабайт. Саме цими конфігураціями регулювалося відношення якості розпізнавання до швидкості під час експерименту.

Результати дослідження швидкості розпізнавання зібрані під час дослідження методів порівняння поз наведені у таблиці 3.

Таблиця 3 – Результати дослідження швидкості аналізу поз

	ResNet50		MobileNetV1	
	High quality	High speed	High quality	High speed
Load	55c	31c	5c	1c
Process	255 мс	105 мс	222 мс	41 мс

Результати позують, що найбільший вплив при порівнянні поз має точність їх встановлення. Серед методів їх порівняння найбільш ефективним виявився метод зважених дистанцій. Швидкість порівняння є однаковою, та зводиться до створення векторів та розрахунку значення за формулою. Обчислювальна складність цих операцій дорівнює O(n)[13], де n – кількість ключових точок. Тому швидкість роботи системи майже повністю залежить від швидкості розпізнавання пози.

Таблиця 2 – Результати дослідження методів порівняння поз

	ResNet50						MobileNetV1					
	High quality			High speed			High quality			High speed		
	CD	AD	WD	CD	AD	WD	CD	AD	WD	CD	AD	WD
$F_1^{0.05}$	48.1	50.3	60.2	47.3	46.4	43.9	0	12.2	18.1	12.4	11.3	12.9
$F_1^{0.10}$	55.6	54.8	56.3	54.2	52.1	56.1	0	23.7	27.6	21.9	19.8	22.0
$F_1^{0.15}$	55.4	56.2	58.7	59.8	60.0	60.8	32.9	34.3	35.6	26.7	27.1	30.8
$F_1^{0.20}$	58.7	57.3	65.4	61.0	60.5	62.6	30.7	35.8	36.8	28.7	33.2	37.4
$F_1^{0.25}$	60.1	57.8	64.9	62.2	62.4	64.1	33.0	34.1	33.4	30.0	33.1	37.1
$F_1^{0.30}$	58.2	56.2	59.0	60.4	60.8	59.9	33.2	32.0	31.2	31.3	32.5	35.6
$F_1^{0.35}$	55.0	55.3	52.3	51.7	53.2	51.5	30.5	30.6	28.9	31.4	29.0	34.8
$F_1^{0.40}$	49.3	50.1	47.5	44.9	49.4	45.4	29.9	30.1	26.9	29.6	28.1	32.3

6 ОБГОВОРЕННЯ

Дослідження показали, що при порівнянні поз найбільшу вагу має якість визначення поз. Проте чим більша якість розпізнавання, тим нижча швидкість роботи, а для розпізнавання потокового відео необхідна висока швидкість.

Завдяки використанню сучасних методів визначення пози існує можливість реалізації такого проекту, для поз з низькою деталізацією, тобто для таких, де допускаються значні відмінності від оригіналу.

Оптимальним варіантом з точки зору швидкості та якості визначення в ході дослідження виявилася конфігурація моделі PoseNet, що базується на архітектурі ResNet50 оптимізована для швидшого виконання. Проте її швидкості недостатньо для аналізу швидких рухів об’єкту на відео, бо швидкість аналізу значно менша за необхідні 40мс.

приводяться до таких, якими вони є в рамках прямокутника, що оточує людину. Іншими словами, обрізаємо зображення залишаючи лише зображення людини. Наступним кроком є приведення векторів до тотожного виду. Початкові зображення можуть мати різні розміри та пропорції, відповідно до пропорцій тіла людини. Виконати це можна за допомогою L2 нормалізації. В результаті отримуємо багатовимірний вектор, що описує позу людини.

Для визначення відстані між векторами скористаємося поняттям косинусоїдної схожості, що є значенням косинусу кута між векторами, що вираховується за формулою:

$$\cos(a, b) = \frac{a * b}{|a| * |b|}, \quad (2)$$

де a і b – вектори.

Використовуючи значення косинусоїдної схожості, можемо розрахувати значення відстані між векторами за формулою:

$$D(F_{xy}, G_{xy}) = \sqrt{2 * (1 - cosineSimilarity(F_{xy}, G_{xy}))}. \quad (3)$$

Метод зважених дистанцій використовує параметр confidence, що приймає значення від 0 до 1 та визначає ступінь впевненості в тому, що ключова точка знаходиться саме у передбаченому місці. Іноді ми точно знаємо, де знаходиться суглоб, наприклад, якщо ми можемо це чітко бачити; в інших випадках у нас дуже низька впевненість, наприклад, якщо суглоб відрізаний або закритий. Попередня фільтрація зображення [8] може покращити ці значення, але лише в окремих випадках. Якщо ми ігноруємо ці показники достовірності, ми втрачаємо цінні відомості про свої дані, і можемо надавати набагато велику вагу та значення тим, про які ми насправді не такі впевнені.

Для використання цієї інформації дослідники з корпорації Google Джордж Папандреу та Тайлер Чжу розробили формулу:

$$D(F, G) = \frac{1}{\sum_{k=1}^{n} F_{c_k}} * \sum_{k=1}^{n} F_{c_k} * |F_{xy_k} - G_{xy_k}|, \quad (4)$$

де G і F – два вектори поз,

n – кількість визначених контрольних точок;,

F_{ck} – значення ймовірності правильного знаходження суглоба для елемента за номером k вектору F,

F_{xy} і G_{xy} – x і y позиції k-ої ключової точки для кожного вектору.

Третій метод порівняння не потребує попередньої нормалізації координат, але вектор будується за іншим принципом. Для кожних трьох анатомічно поєднаних між собою точок вираховується косинус кута, між отриманими частинами тіла за формулою 3.1. Відмінністю від косинусоїдної схожості, що використовується у першому методі, є те, що в даному випадку отримується значення для двовимірного вектору, що описує положення двох кінцівок відносно одне одного.

4 ЕКСПЕРИМЕНТИ

Для аналізу роботи методів розпізнавання пози людини на зображенні використано датасет MS COCO та метрику AP[9]. AP без додаткових позначень позначає середній результат для досліджень зі значеннями OKS від 0.5 до 0.95 з кроком 0.05. AP^M та AP^L позначають результати відповідно для об'єктів розміром поміж 32x32 та 96x96 пікселів та більших.

В результаті цього дослідження було виявлено що значних переваг в точності визначення поз не має жодна з представлених моделей. Проте модель PoseNet має значно ширший набір сумісних платформ для виконання, у той час, коли мають API лише для Python, C++ та платформи Unity. PoseNet додатково може використовуватися у Java, Swift, Objective C та Javascript, що дозволяє використовувати її для розробки додатків як на мобільні платформи так і для веб-додатків без необхідності надсилати відео на сервери, що значно зменшує часові витрати на використання мережі і помітно прискорює процес обробки.

Дослідження методів порівняння було проведено на базі моделі PoseNet з використанням різних конфігурацій мережі. Модель має можливість використовувати мережу на основі архітектури ResNet50 та MobileNetV1. Перша має більшу точність визначення але меншу швидкість. Друга оптимізована для використання мобільними пристроями, має менший розмір та більшу швидкість, проте має значно меншу точність визначення.

Кожна з архітектур має додаткові конфігурації, що впливають на точність та швидкість[10]. При дослідженні були використані конфігурації, оптимізуючі кожну з них для як для найкращої якості, так і для найбільшої швидкості для порівняння результатів.

Аналіз виконується за наступним алгоритмом: отримуємо описи поз за допомогою нейронної мережі в різних конфігураціях, отримуємо нормалізовані вектори, що порівнюються одне з одним за допомогою обох методів порівняння, результати зберігаються та використовуються для обчислення F1-показника[11]. Цей показник обчислюється при різних порогових значеннях відстані між векторами: від 0.05 до 0.45 з кроком 0.05.

Для аналізу методів порівняння було зібрано набір зображень що складається зі 521 зображень, що об'єднуються в 43 групи поз. Переважно групи зображень складаються з однотипних фотографій спортивних тренувань.

В ході експерименту також були зібрані дані про швидкість обробки та завантаження вказаних моделей. Обчислення виконуються на процесорі 2.6 GHz Intel Core i7 та з 16GB оперативної пам'яті.

5 РЕЗУЛЬТАТИ

В результаті проведення експериментів на базі датасету COCO, отримали дані наведені в таблиці 1. Аналізуючи ці результати можна сказати, що найбільш ефективною є модель OpenPose, проте різниця з конкурентами складає близько 2%, що в контексті

Серед методів порівняння поз найкращий результат продемонстрував метод зважених дистанцій. Це пояснюється тим, що при порівнянні поз більша вага надається точкам, що були виявлені з більшою точністю і таким чином загальна поза порівнюється більш коректно, у той час як значення деталей нівелюється.

Основними недоліками такого подходу до порівняння поз є те, що результат напряму залежить від ракурсу зйомки. Таким однакові пози можуть виглядати різними і навпаки при різних положеннях камери. Для того, щоб зменшити вплив ракурсу, необхідно будувати тривимірні моделі поз і порівнювати їх. Проте цей спосіб виконується значно довше і точність побудови тривимірних моделей поз за двовимірним зображенням значно нижча.

Вирішити цю задачу можуть камери з датчиками глибини. Таке рішення використовується в технології Microsoft Kinect[14]. Перспективним рішенням, яке використовується компанією Apple у нових мобільних пристроях є датчик Lidar[15]. Розвиток цих технологій та їхнє розповсюдження може значною мірою покращити сучасні результати.

ВИСНОВКИ

В результаті дослідження були проаналізовані існуючі методи порівняння поз на потоковому відео з заданим зображенням. Аналіз точності визначення поз показав, що існуючи методи дають приблизно однакову точність і швидкість роботи системи буде залежати від оптимізації обраної моделі та методу впровадження.

Оптимізовані для мобільних платформ моделі показали задовільний результат швидкості для обробки потокового відео, проте мобільні пристрої мають нижчі технічні характеристики, ніж пристрій на якому проводилося дослідження. Також ці моделі показали дуже низькі значення точності визначення пози. Оптимальною конфігурацією, з точки зору швидкості та якості визначення, в ході дослідження виявилася конфігурація моделі PoseNet, що базується на архітектурі ResNet50 оптимізована для швидшого виконання.

Було досліджено ефективність різних методів порівняння пози і їх вплив на результати порівняння. Різниця між їх результатами складає не більше 7% в залежності від порогового значення схожості та точності визначення поз. Найбільш точним виявився метод зважених дистанцій.

Досліджені методи виявилися придатними для використання для обробки потокового відео лише за умови, що пози будуть прості з високим рівнем дозволеної неточності, а рухи на відео будуть повільні. В інакшому випадку, швидкість і точність, що були виявлені в ході дослідження будуть недостатні для коректної роботи системи.

ЛІТЕРАТУРА / ЛИТЕРАТУРА

1. Tsung-Yi Lin, Michael Maire, Serge Belongie, Lubomir Bourdev, Ross Girshick, James Hays, Pietro Perona, Deva Ramanan, C. Lawrence Zitnick, Piotr Dollár, Microsoft COCO: Common Objects in Context, Computer Vision and Pattern Recognition, 2001, arXiv:1908.10357
2. Bowen Cheng, Bin Xiao, Jingdong Wang, Honghui Shi, Thomas S. Huang, Lei Zhang, HigherHRNet: Scale-Aware Representation Learning for Bottom-Up Human Pose Estimation, Computer Vision and Pattern Recognition, 12 Mar 2020, arXiv:1908.10357
3. Muhammed Kocabas, Salih Karagoz, Emre Akbas, MultiPoseNet: Fast Multi-Person Pose Estimation using Pose Residual Network, Computer Vision and Pattern Recognition, Jul 2018, arXiv:1807.04067
4. Zhe Cao, Gines Hidalgo, Tomas Simon, Shih-En Wei, Yaser Sheikh, OpenPose: Realtime Multi-Person 2D Pose Estimation using Part Affinity Fields, Computer Vision and Pattern Recognition, *30 May 2019*, arXiv:1812.08008
5. Alex Kendall, Matthew Grimes, Roberto Cipolla, PoseNet: A Convolutional Network for Real-Time 6-DOF Camera Relocalization, *18 Feb 2016, arXiv:1505.07427*
6. Pradnya Krishnanath Borkar , Marilyn Mathew Pulinthitha , Mrs. Ashwini Pansare, 2019, Match Pose – A System for Comparing Poses, INTERNATIONAL JOURNAL OF ENGINEERING RESEARCH & TECHNOLOGY (IJERT) Volume 08, Issue 10 (October 2019)
7. Jane Friedhoff, Irene Alvarado, Move Mirror: An AI Experiment with Pose Estimation in the Browser using TensorFlow.js [Електронний ресурс] – Режим доступу: https://medium.com/tensorflow/move-mirror-an-ai-experiment-with-pose-estimation-in-the-browser-using-tensorflow-js-2f7b769f9b23
8. Білоус Н. В., Красов А. И., Власенко В. П. Видалення низькочастотної складової зображення з використанням медіанного фільтру // Журнал інженерних наук. 2016. Том 3 №2.
9. Jerome Revaud, Jon Almazan, Rafael Sampaio de Rezende, Cesar Roberto de Souza, Learning with Average Precision: Training Image Retrieval with a Listwise Loss, Computer Vision and Pattern Recognition, 2019, arXiv:1906.07589
10. Real-time Human Pose Estimation in the Browser with TensorFlow.js [Електронний ресурс] – Режим доступу: https://medium.com/tensorflow/real-time-human-pose-estimation-in-the-browser-with-tensorflow-js-7dd0bc881cd5
11. Dembczy´nski, K., Waegeman, W., Cheng, W., H¨ullermeier, E.: An exact algorithm for F-measure maximization. In: Neural Information Processing Systems (2011)
12. Zhaowei Cai, Xiaodong He, Jian Sun, and Nuno Vasconcelos. Deep learning with low precision by half-wave gaussian quantization. In Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition, pages 5918–5926, 2017. 4
13. Kalle Rutanen, O-notation in algorithm analysis, 29 Oct 2016, arXiv:1309.3210
14. Behnam Malmir, Shing I Chang, Malgorzata Rys, Dylan Darter, Quantifying Gait Changes Using Microsoft Kinect and Sample Entropy' presented at the 2018 Industrial and Systems Engineering Research Conference (ISERC) in Orlando, Florida (May 2018)
15. Mario Bijelic, Tobias Gruber, Werner Ritter, A Benchmark for Lidar Sensors in Fog: Is Detection Breaking Down?, 2018 IEEE Intelligent Vehicles Symposium (IV), DOI: 10.1109/IVS.2018.8500543