Министерство образования и науки Российской Федерации

Федеральное государственное бюджетное

образовательное учреждение высшего образования

Уфимский университет науки и технологий

Отчет

по дисциплине

«Разработка мобильных приложений»

Выполнили:

студенты группы ПРО-428

Мохаммед А.А.

Вахитов Т.Ф.

Аль-Шаибани Е.Т.

Проверил:

Богданов М. Р.

Уфа 2023

**Лабораторная работа №1: Использование TensorFlow.js и обнаружение объектов на изображениях**

**Цель**

Разработка приложения, использующего модели TensorFlow.js для обнаружения объектов на изображениях, передаваемых с камеры мобильного устройства.

**Теоретические сведения**

Используются библиотеки @tensorflow/tfjs и @tensorflow-models/coco-ssd, позволяющие работать с нейросетями и моделями машинного обучения TensorFlow.js. Код позволяет обнаруживать объекты на изображениях в реальном времени с помощью камеры устройства.

**Описание функционала приложения**

Приложение предоставляет обзорную карту в реальном времени с помощью камеры устройства. Модель COCO-SSD (Common Objects in Context - Single Shot Multibox Detector) из TensorFlow.js используется для обнаружения объектов на изображениях, передаваемых с камеры. Найденные объекты выделяются на экране с помощью прямоугольников и их классов.

A computer with a bottle of water and a mouse

Description automatically generatedA bottle of water on a table

Description automatically generatedA computer and keyboard on a desk

Description automatically generated

**Ключевые особенности:**

- Использование камеры: используется компонент `TensorCamera` для получения изображений с камеры устройства.

- Обнаружение объектов: Модель COCO-SSD анализирует изображения и определяет объекты на них. Найденные объекты отмечаются на экране в реальном времени с помощью прямоугольников с указанием классов объектов.

**Заключение**

Представленный код успешно использует библиотеки TensorFlow.js для обнаружения объектов на изображениях с помощью камеры устройства. Это предоставляет базовый функционал для создания приложений, работающих с обработкой изображений в реальном времени.

**Листинг**

import \* as cocoSsd from '@tensorflow-models/coco-ssd';

import \* as tf from '@tensorflow/tfjs';

import { cameraWithTensors } from '@tensorflow/tfjs-react-native';

import { Camera } from 'expo-camera';

import React, { useEffect, useRef, useState } from 'react';

import { Dimensions, LogBox, Platform, StyleSheet, View } from 'react-native';

import Canvas from 'react-native-canvas';

const TensorCamera = cameraWithTensors(Camera);

LogBox.ignoreAllLogs(true);

const { width, height } = Dimensions.get('window');

export default function App() {

  const [model, setModel] = useState(null);

  let context = useRef(null);

  const canvas = useRef(null);

  function handleCameraStream(images) {

    const loop = async () => {

      const nextImageTensor = images.next().value;

      if (!model || !nextImageTensor) throw new Error('no model');

      model

        .detect(nextImageTensor)

        .then((predictions) => {

          drawRectangle(predictions, nextImageTensor);

        })

        .catch((err) => {

          console.log(err);

        });

      requestAnimationFrame(loop);

    };

    loop();

  }

  function drawRectangle(predictions, nextImageTensor) {

    if (!context.current || !canvas.current) {

      console.log('no context or canvas');

      return;

    }

    console.log(predictions);

    const scaleWidth = width / nextImageTensor.shape[1];

    const scaleHeight = height / nextImageTensor.shape[0];

    const flipHorizontal = Platform.OS === 'ios' ? false : true;

    context.current.clearRect(0, 0, width, height);

    for (const prediction of predictions) {

      const [x, y, boxWidth, boxHeight] = prediction.bbox;

      const boundingBoxX = flipHorizontal

        ? canvas.current.width - x \* scaleWidth - boxWidth \* scaleWidth

        : x \* scaleWidth;

      const boundingBoxY = y \* scaleHeight;

      context.current.strokeRect(

        boundingBoxX,

        boundingBoxY,

        boxWidth \* scaleWidth,

        boxHeight \* scaleHeight

      );

      context.current.fillText(

        prediction.class,

        boundingBoxX - 5,

        boundingBoxY - 5

      );

    }

  }

  const handleCanvas = async (can) => {

    if (can) {

      can.width = width;

      can.height = height;

      const ctx = can.getContext('2d');

      context.current = ctx;

      ctx.strokeStyle = 'red';

      ctx.fillStyle = 'red';

      ctx.lineWidth = 3;

      canvas.current = can;

    }

  };

  let textureDims;

  Platform.OS === 'ios'

    ? (textureDims = { height: 1920, width: 1080 })

    : (textureDims = { height: 1200, width: 1600 });

  useEffect(() => {

    (async () => {

      const { status } = await Camera.requestPermissionsAsync();

      await tf.ready();

      setModel(await cocoSsd.load());

    })();

  }, []);

  return (

    <View style={styles.container}>

      <TensorCamera

        style={styles.camera}

        type={Camera.Constants.Type.back}

        cameraTextureHeight={textureDims.height}

        cameraTextureWidth={textureDims.width}

        resizeHeight={200}

        resizeWidth={152}

        resizeDepth={3}

        onReady={handleCameraStream}

        autorender={true}

        useCustomShadersToResize={false}

      />

      <Canvas style={styles.canvas} ref={handleCanvas} />

    </View>

  );

}

const styles = StyleSheet.create({

  container: {

    flex: 1,

    backgroundColor: '#fff',

  },

  camera: {

    width: '100%',

    height: '100%',

  },

  canvas: {

    position: 'absolute',

    zIndex: 1000000,

    width: '100%',

    height: '100%',

  },

});