Rapor

A - Atik Tespit Sistemi

Atik tespit sistemi atiklarin mobil uygulama ile fotograf alinarak hangi geri donusum kutusuna atilmasi gerektigi hakkinda bilgiler veren, atilacak nesnenin ozelliklerini gosteren sistemdir. Bu sistemin tasarlanmasinda kullanilacak olan teknolojiler asagidaki gibidir:

Mobil uygulama

- Mobil uygulama yapiminda **React Native** kullanilacaktir
- Mobil uygulama basit bir arayuz ile kullanicinin sonuca en hizli yol ile ulasmasini saglamayi hedeflemektedir. Kullanici ana ekrandaki buton ile hemen fotograf alabilecek ve bilgileri gorebilecek.
- Web destekli calisacak

Web uygulamasi

- Web uygulamasinin yapiminda Asp .Net kullanilacak
- Nesne tanima **Python OpenCv** kutuphanesi ile yapilacak
- Python ile Rest Api olusturularak baglanti saglanacak
- Veri tabani islemleri **MsSql** ile saglanacak
- Kullanicinin cektigi fotograflar makine ogrenmesine katkida bulunarak dogruluk oranini artiracak

Oncelik olarak Web uygulamasi yapilacaktir. Mobil uzerinden de kullanilabilecek sekilde yani responsive olacak sitede javascript ile kamera acma olaylari gerceklestirilip fotograf alinacak ve fotograf veritabanina eklenecek devaminda python`da veri eslestirme islemleri yapilip geri donus saglayacak ve bilgileri kullaniciya gosterecektir. Nesne tanima islemleri icin elimizde bulunan veriler bircok acidan cekilmis ve benzeri nesnelerin fotograflari bulunmali ki makine ogrenmesi yapila bilsin. Veritabaninin buyumesi ve makine ogrenmesine cekilen fotograflarin katkida bulunmasi icin, kullanicinin aldigi fotograflar da veritabaninda tutulacak. ImageNet uzerinde buldugumuz fotograflar ile nesne tanima sisteminin olusturulmasi planlanmaktadir. EPA (ABD Cevre Koruma Ajansi) sayesinde nesnelerin vermesi gereken bilgiler Turkceye cevrilerek hazirlanmistir.

B - Gant Semasi

C - Bugune kadar yapilanlar

1. EPA verileri tek tek Turkceye cevrildi ve atiklarin nereye atilmasi gerektigindeki bilgiler kesinlestirildi. Atiklar hakkinda kisa istatistik bilgileri de buna dahil.
   a. https://atik-tespit-sistemi.netlify.app/plugins/atik-verileri.pdf
2. Web sitesinin Template`i hazirlandi
   a. https://atik-tespit-sistemi.netlify.app/index.html
3. Javascript ile Web uzerinde kamera acma ve fotograf alma islemleri gerceklestirildi.
4. En yakin geri donusum kutusunu bildirecek olan harita sisteminin cok basit tasarimi hazirlandi.
5. MsSql`de veritabani olusturuldu.
6. ImageNet uzerinden fotograflarin alinmasi icin gerekenler yapildi
7. Python OpenCv ogreniminde aldigimiz egitim ile basit bir proje yaptik ve egitime devam etmekteyiz.

Kamera acma ve fotograf alma modal`inin javascript ile yazilmis hali:

```
let camera_button = document.querySelector("#camera-btn");
let video = document.querySelector("#video");
let click_button = document.querySelector("#click-photo");
let canvas = document.querySelector("#canvas");
camera_button.addEventListener('click', async function() {
    let stream = await navigator.mediaDevices.getUserMedia({ video: true, audio: false });
  video.srcObject = stream;
});
click_button.addEventListener('click', function() {
    canvas.getContext('2d').drawImage(video, 0, 0, canvas.width, canvas.height);
    let image_data_url = canvas.toDataURL('image/jpeg');
    // data url of the image
    console.log(image_data_url);
});
var photo_modal = document.getElementById("photoModal");
var photo_btn = document.getElementById("camera-btn");
var photo_span = document.getElementsByClassName("close")[0];
photo_btn.onclick = function() {
  photo_modal.style.display = "block";
}
photo_span.onclick = function() {
  photo_modal.style.display = "none";
}

var pick_modal = document.getElementById("pickModal");
var pick_btn = document.getElementById('pick-btn');
var pick_span = document.getElementsByClassName('close')[1];
pick_btn.onclick = function () {
  pick_modal.style.display = "block";
}
pick_span.onclick = function () {
  pick_modal.style.display = "none";
}


window.onclick = function(event) {
  if (event.target == photo_modal || event.target == pick_modal) {
    photo_modal.style.display = "none";
    pick_modal.style.display = "none";
  }
  }
```

Python ile yaptigimiz deneme projesinin kodlari:

```
import cv2
backsub = cv2.createBackgroundSubtractorMOG2()
kamera = cv2.VideoCapture(0)
kamera.set(cv2.CAP_PROP_BUFFERSIZE, 1024)
genislik = 704
yukseklik = 288
konumSayac = 0
girenSayisi = 0
simdiki_veri = []
gecmis_veri = []
while True:
    ret, frame = kamera.read()
    if kamera:
        fgmask = backsub.apply(frame, None, 0.018)
        cv2.line(frame, (0, yukseklik), (genislik, yukseklik), (0, 255, 0), 2)
        contours, hierarchy = cv2.findContours(
            fgmask, cv2.RETR_EXTERNAL, cv2.CHAIN_APPROX_NONE)
        for contour in contours:
            (x, y, w, h) = cv2.boundingRect(contour)
            if w > 65 and h > 65:
                simdiki_veri.append([x, y])
                cv2.rectangle(frame, (x, y), (x+w, y+h), (0, 0, 255), 2)
        yer_listesi = []
        try:
            for i in range(len(simdiki_veri)):
                mini = 10000
                for k in range(len(gecmis_veri)):
                    diff_x = simdiki_veri[i][0]-gecmis_veri[k][0]
                    diff_y = simdiki_veri[i][1]-gecmis_veri[k][1]
                    distance = (diff_x*diff_x)+(diff_y * diff_y)
                    if(distance < mini):
                        mini = distance
                        konumSayac = k
                yer_listesi.append(konumSayac)
        except IndexError:
            continue
        try:
            for i in yer_listesi:
                for k in range(1, len(yer_listesi)):
                    if i == yer_listesi[k]:
                        yer_listesi.pop(k)
                        yer_listesi.insert(k, "pass")
        except IndexError:
            continue
        for i in range(len(simdiki_veri)):
            try:
                if yer_listesi[i] == "pass":
                    pass
                else:
                    y_previous = gecmis_veri[yer_listesi[i]][1]
                    if(simdiki_veri[i][1] < yukseklik and y_previous > yukseklik):
                        girenSayisi = girenSayisi+1

            except IndexError:
                continue
        gecmis_veri = simdiki_veri
        simdiki_veri = []

        cv2.putText(frame, "say: "+str(girenSayisi), (0, 20),
                    cv2.FONT_HERSHEY_SIMPLEX, 0.6, (0, 255, 0), 2)
        cv2.imshow("Python OpenCV", frame)

        key = cv2.waitKey(60)
        if key == ord('q'):
            kamera.release()
            cv2.destroyAllWindows()
            break
```

Node js ile python calistirma denemesi:

```
const express = require('express')
const {spawn} = require('child_process');
const app = express()
const port = 3000
app.get('/', (req, res) => {

 var dataToSend;
 // spawn new child process to call the python script
 const python = spawn('python', ['deneme.py']);
 // collect data from script
 python.stdout.on('data', function (data) {
  console.log('Pipe data from python script ...');
  dataToSend = data.toString();
 });
 // in close event we are sure that stream from child process is closed
 python.on('close', (code) => {
 console.log(`child process close all stdio with code ${code}`);
 // send data to browser
 res.send(dataToSend)
 });

})
app.listen(port, () => console.log(`Example app listening on port
${port}!`))
```

Sinifta yaptigimiz flask ile api calismasi

```
from flask import Flask, jsonify
# vsdl olarak yap. parametreyi url uzerinden yonlendirme security acik olusabilir
app = Flask(__name__)

wastes = [{ 'id':"1 ",
            'atik-foto':"FOTOGRAF",
            'atik-isim':"Icecek kartonu",
            'atik-detay':"EPA, aseptik ve katlanir karton, kagit torba, ambalaj kagidi ve diger kagit ve karton ambalaj uretiminin 201
        },{
            'id':"2",
            'atik-foto':"fotograf",
            'atik-isim':"Plastik sise",
            'atik-detay':" Gerekirse yiyecek ve icecek kaplarini hafifce yikayin.Geri donusum plastik posetsiz, temiz, bos ve kuru olm
        }]


@app.route('/')
```

```
def index():
    return "Welcome tt api"

@app.route("/wastes",methods=['GET'])
def get():
    return jsonify({'Wastes':wastes })

@app.route("/wastes/<int:id>",methods=["GET"])
def get_waste(id):
    return jsonify({'waste':wastes[id]})

@app.route("/wastes",methods=['POST'])
def create():
    waste = { 'id':"2",
            'atik-foto':"fotograf",
            'atik-isim':"Plastik sise",
            'atik-detay':" Gerekirse yiyecek ve icecek kaplarini hafifce yikayin.Geri donusum plastik posetsiz, temiz, bos ve kuru olm
            }
    wastes.append(waste)
    return jsonify({'Created': waste})

@app.route("/wastes/<int:id>",methods = ['PUT'])
def waste_update(id):
    wastes[id]['atik-detay'] = "XYZ"
    return jsonify({'waste':wastes[id]})

@app.route("/wastes/<int:id>",methods = ['DELETE'])
def delete(id):
    wastes.remove(wastes[id])
    return jsonify({'result':True})

if __name__ == "__main__":
    app.run(debug=True)
```

ImageNet`den verilerin filtrelenerek alinmasi icin gereken kodlar

```
#!/usr/bin/env python3
import os
import numpy as np
import requests
import argparse
import json
import time
import logging
import csv

from multiprocessing import Pool, Process, Value, Lock

from requests.exceptions import ConnectionError, ReadTimeout, TooManyRedirects, MissingSchema, InvalidURL

parser = argparse.ArgumentParser(description='ImageNet image scraper')
parser.add_argument('-scrape_only_flickr', default=True, type=lambda x: (str(x).lower() == 'true'))
parser.add_argument('-number_of_classes', default = 10, type=int)
parser.add_argument('-images_per_class', default = 10, type=int)
parser.add_argument('-data_root', default='' , type=str)
parser.add_argument('-use_class_list', default=False,type=lambda x: (str(x).lower() == 'true'))
parser.add_argument('-class_list', default=[], nargs='*')
parser.add_argument('-debug', default=False,type=lambda x: (str(x).lower() == 'true'))

parser.add_argument('-multiprocessing_workers', default = 8, type=int)

args, args_other = parser.parse_known_args()

if args.debug:
    logging.basicConfig(filename='imagenet_scarper.log', level=logging.DEBUG)

if len(args.data_root) == 0:
    logging.error("-data_root is required to run downloader!")
    exit()

if not os.path.isdir(args.data_root):
    logging.error(f'folder {args.data_root} does not exist! please provide existing folder in -data_root arg!')
    exit()


IMAGENET_API_WNID_TO_URLS = lambda wnid: f'http://www.image-net.org/api/imagenet.synset.geturls?wnid={wnid}'

current_folder = os.path.dirname(os.path.realpath(__file__))

class_info_json_filename = 'imagenet_class_info.json'
class_info_json_filepath = os.path.join(current_folder, class_info_json_filename)
```

```
class_info_dict = dict()

with open(class_info_json_filepath) as class_info_json_f:
    class_info_dict = json.load(class_info_json_f)

classes_to_scrape = []

if args.use_class_list == True:
   for item in args.class_list:
       classes_to_scrape.append(item)
       if item not in class_info_dict:
           logging.error(f'Class {item} not found in ImageNete')
           exit()

elif args.use_class_list == False:
    potential_class_pool = []
    for key, val in class_info_dict.items():

        if args.scrape_only_flickr:
            if int(val['flickr_img_url_count']) * 0.9 > args.images_per_class:
                potential_class_pool.append(key)
        else:
            if int(val['img_url_count']) * 0.8 > args.images_per_class:
                potential_class_pool.append(key)

    if (len(potential_class_pool) < args.number_of_classes):
        logging.error(f"With {args.images_per_class} images per class there are {len(potential_class_pool)} to choose from.")
        logging.error(f"Decrease number of classes or decrease images per class.")
        exit()

    picked_classes_idxes = np.random.choice(len(potential_class_pool), args.number_of_classes, replace = False)

    for idx in picked_classes_idxes:
        classes_to_scrape.append(potential_class_pool[idx])


print("Picked the following clases:")
print([ class_info_dict[class_wnid]['class_name'] for class_wnid in classes_to_scrape ])

imagenet_images_folder = os.path.join(args.data_root, 'imagenet_images')
if not os.path.isdir(imagenet_images_folder):
    os.mkdir(imagenet_images_folder)


scraping_stats = dict(
    all=dict(
        tried=0,
        success=0,
        time_spent=0,
    ),
    is_flickr=dict(
        tried=0,
        success=0,
        time_spent=0,
    ),
    not_flickr=dict(
        tried=0,
        success=0,
        time_spent=0,
    )
)

def add_debug_csv_row(row):
    with open('stats.csv', "a") as csv_f:
        csv_writer = csv.writer(csv_f, delimiter=",")
        csv_writer.writerow(row)

class MultiStats():
    def __init__(self):

        self.lock = Lock()

        self.stats = dict(
            all=dict(
                tried=Value('d', 0),
                success=Value('d',0),
                time_spent=Value('d',0),
            ),
            is_flickr=dict(
                tried=Value('d', 0),
                success=Value('d',0),
                time_spent=Value('d',0),
            ),
            not_flickr=dict(
                tried=Value('d', 0),
                success=Value('d', 0),
                time_spent=Value('d', 0),
            )
        )
    def inc(self, cls, stat, val):
        with self.lock:
            self.stats[cls][stat].value += val

    def get(self, cls, stat):
        with self.lock:
            ret = self.stats[cls][stat].value
        return ret

multi_stats = MultiStats()


if args.debug:
    row = [
        "all_tried",
        "all_success",
        "all_time_spent",
        "is_flickr_tried",
        "is_flickr_success",
        "is_flickr_time_spent",
        "not_flickr_tried",
        "not_flickr_success",
        "not_flickr_time_spent"
    ]
    add_debug_csv_row(row)

def add_stats_to_debug_csv():
    row = [
        multi_stats.get('all', 'tried'),
        multi_stats.get('all', 'success'),
        multi_stats.get('all', 'time_spent'),
        multi_stats.get('is_flickr', 'tried'),
        multi_stats.get('is_flickr', 'success'),
        multi_stats.get('is_flickr', 'time_spent'),
        multi_stats.get('not_flickr', 'tried'),
        multi_stats.get('not_flickr', 'success'),
        multi_stats.get('not_flickr', 'time_spent'),
    ]
    add_debug_csv_row(row)

def print_stats(cls, print_func):

    actual_all_time_spent = time.time() - scraping_t_start.value
    processes_all_time_spent = multi_stats.get('all', 'time_spent')

    if processes_all_time_spent == 0:
        actual_processes_ratio = 1.0
    else:
        actual_processes_ratio = actual_all_time_spent / processes_all_time_spent

    #print(f"actual all time: {actual_all_time_spent} proc all time {processes_all_time_spent}")

    print_func(f'STATS For class {cls}:')
    print_func(f' tried {multi_stats.get(cls, "tried")} urls with'
               f' {multi_stats.get(cls, "success")} successes')

    if multi_stats.get(cls, "tried") > 0:
        print_func(f'{100.0 * multi_stats.get(cls, "success")/multi_stats.get(cls, "tried")}% success rate for {cls} urls ')
    if multi_stats.get(cls, "success") > 0:
        print_func(f'{multi_stats.get(cls,"time_spent") * actual_processes_ratio / multi_stats.get(cls,"success")} seconds spent per {



lock = Lock()
url_tries = Value('d', 0)
scraping_t_start = Value('d', time.time())
class_folder = ''
class_images = Value('d', 0)

def get_image(img_url):

    #print(f'Processing {img_url}')

    #time.sleep(3)

    if len(img_url) <= 1:
        return


    cls_imgs = 0
    with lock:
        cls_imgs = class_images.value

    if cls_imgs >= args.images_per_class:
        return
```

```
logging.debug(img_url)

cls = ''

if 'flickr' in img_url:
    cls = 'is_flickr'
else:
    cls = 'not_flickr'
    if args.scrape_only_flickr:
        return

t_start = time.time()

def finish(status):
    t_spent = time.time() - t_start
    multi_stats.inc(cls, 'time_spent', t_spent)
    multi_stats.inc('all', 'time_spent', t_spent)

    multi_stats.inc(cls,'tried', 1)
    multi_stats.inc('all', 'tried', 1)

    if status == 'success':
        multi_stats.inc(cls,'success', 1)
        multi_stats.inc('all', 'success', 1)

    elif status == 'failure':
        pass
    else:
        logging.error(f'No such status {status}!!')
        exit()
    return


with lock:
    url_tries.value += 1
    if url_tries.value % 250 == 0:
        print(f'\nScraping stats:')
        print_stats('is_flickr', print)
        print_stats('not_flickr', print)
        print_stats('all', print)
        if args.debug:
            add_stats_to_debug_csv()

try:
    img_resp = requests.get(img_url, timeout = 1)
except ConnectionError:
    logging.debug(f"Connection Error for url {img_url}")
    return finish('failure')
except ReadTimeout:
    logging.debug(f"Read Timeout for url {img_url}")
    return finish('failure')
except TooManyRedirects:
    logging.debug(f"Too many redirects {img_url}")
    return finish('failure')
except MissingSchema:
    return finish('failure')
except InvalidURL:
    return finish('failure')

if not 'content-type' in img_resp.headers:
    return finish('failure')

if not 'image' in img_resp.headers['content-type']:
    logging.debug("Not an image")
    return finish('failure')

if (len(img_resp.content) < 1000):
    return finish('failure')

logging.debug(img_resp.headers['content-type'])
logging.debug(f'image size {len(img_resp.content)}')

img_name = img_url.split('/')[-1]
img_name = img_name.split("?")[0]

if (len(img_name) <= 1):
    return finish('failure')

img_file_path = os.path.join(class_folder, img_name)
logging.debug(f'Saving image in {img_file_path}')

with open(img_file_path, 'wb') as img_f:
    img_f.write(img_resp.content)

    with lock:
        class_images.value += 1
        logging.debug(f'Scraping stats')
        print_stats('is_flickr', logging.debug)
        print_stats('not_flickr', logging.debug)
        print_stats('all', logging.debug)

        return finish('success')


for class_wnid in classes_to_scrape:

    class_name = class_info_dict[class_wnid]["class_name"]
    print(f'Scraping images for class \"{class_name}\"')
    url_urls = IMAGENET_API_WNID_TO_URLS(class_wnid)

    time.sleep(0.05)
    resp = requests.get(url_urls)

    class_folder = os.path.join(imagenet_images_folder, class_name)
    if not os.path.exists(class_folder):
        os.mkdir(class_folder)

    class_images.value = 0

    urls = [url.decode('utf-8') for url in resp.content.splitlines()]

    #for url in  urls:
    #    get_image(url)

    print(f"Multiprocessing workers: {args.multiprocessing_workers}")
    with Pool(processes=args.multiprocessing_workers) as p:
        p.map(get_image,urls)
```

ImageNet`den alacagimiz birkac ornek csv:

n02876657,bottle,1228,737
n02877266,bottle,1384,1375

n02946921,can,1203,625

Bu kodlar kullanilarak 160gb veri icinden sadece bu verileri indirebilecegiz

Verileri arastirirken kullandigimiz kaynaklar

- https://www.epa.gov/recycle
- http://www.cevresehirkutuphanesi.com/assets/files/slider_pdf/7NiB09QPiroJ.pdf
- https://tr.wikipedia.org/wiki/Geri_dönüşüm
- https://www.bso-oberursel.de/de/downloads-formulare/abfall/bso-grosswohnanl-tuerkisch.pdf?cid=bkk
- https://www.yesilist.com/karton-bardak-kagit-pecete-pizza-kutusu-geri-donusum-kutularina-atilmamasi-gereken-12-sey/
- https://www.yesilist.com/geri-donusume-giris-hangi-cop-hangi-kutuya-atilmali/

D - Projede arastirdigimiz ve yapamadiklarimiz

Python kodunu Asp ile veya Javascript ile nasil calistiracagimizi bulamamistik ve Api ile yapilmasi gerektigini soylediniz. Bunu nasil yapacagimizi arastirmaya devam ediyoruz

E - istenilen ilerlemeyi yapamama sebebimiz

Yapilmasi gereken projenin basit olmadigini dusunuyoruz ve hic bilmedigimiz bir alanda calisma yaptigimiz icin (egitimlerini aliyoruz) gerekenden daha fazla zamana ihtiyacimiz var.

F - Final dönemine kadar bitirebilecekmiyiz?

Insallah