LAMPIRAN

1. Code App.py

app.py — Edited

```
import streamlit as st
from detect import *
from PIL import Image #membuka atau akses gambar
import cv2
import time
import psutil
import numpy as np
import argparse
import os
import platform
import sys
from pathlib import Path
from collections import Counter
import torch.backends.cudnn as cudnn
import torch
from numpy import random
from utils.google_utils import gsutil_getsize
from utils.metrics import fitness
from utils.torch_utils import init_torch_seeds
from models.experimental import attempt_load
from utils.datasets import LoadStreams, LoadImages
from utils.general import check_img_size, check_requirements, check_imshow, non_max_suppression,
apply_classifier, \
    scale_coords, xyxy2xywh, strip_optimizer, set_logging, increment_path
from utils.plots import plot_one_box
from utils.torch_utils import select_device, load_classifier, time_synchronized, TracedModel

#--------------------------------Web Page Designing------------------------------
hide_menu_style = """
    <style>
        MainMenu {visibility: hidden;}
         div[data-testid="stHorizontalBlock"]> div:nth-child(1)
        {
            border : 2px solid #doe0db;
            border-radius:5px;
            text-align:center;
            color:black;
            background:dodgerblue;
            font-weight:bold;
            padding: 25px;
        }
        div[data-testid="stHorizontalBlock"]> div:nth-child(2)
        {
            border : 2px solid #doe0db;
            background:dodgerblue;
            border-radius:5px;
            text-align:center;
            font-weight:bold;
            color:black;
            padding: 25px;
        }
    </style>
    """
main_title = """
            <div>
                <h1 style="color:dodgerblue;
                text-align:center; font-size:35px;
                margin-top:-95px;">
                CALCULATED POLLEN VIABILITY</h1>
            </div>
            """
sub_title = """
            <div>
                <h6 style="color:dodgerblue;
                text-align:center;
                margin-top:-40px;">
                Polen Object Detection and Counting </h6>
            </div>
            """
#---------------------------Main Function for Execution--------------------------
def main():
    st.set_page_config(page_title='Dashboard',
                       layout = 'wide',
                       initial_sidebar_state = 'auto')
    st.markdown(hide_menu_style,
                unsafe_allow_html=True)
```

```
    st.markdown(main_title,
                unsafe_allow_html=True)
    st.markdown(sub_title,
                unsafe_allow_html=True)
    inference_msg = st.empty()
    st.sidebar.title("Heart Made wkwkwkwkwk")
    activities = ["Polen", "Kecambah"] #DIUBAH TANGGAL 23/02/2023
    choice = st.sidebar.selectbox("Please Select Activity", activities)
    if choice == "Polen":
        input_source = 'Image'
        conf_thres = st.sidebar.text_input("Class confidence threshold",
                                           "0.5")
        save_output_image = st.sidebar.radio("Save output?",
                                             ('Yes', 'No'))
        if save_output_image == 'Yes':
            nosave = False
            display_labels=False
        else:
            nosave = True
            display_labels = True
        weights = "yolov7 (1).pt"
        # -------------------------- LOCAL IMAGE -------------------------
        if input_source == "Image":

            image = st.sidebar.file_uploader("Select input",
                                             type=["jpg", "jpeg","png"],
                                             accept_multiple_files=False)
            # nama_file=image.name
            if image is not None:
                our_image = Image.open(image)
                our_image.save("data/gambarnya/{}".format(image.name))
                st.image(our_image)
            if st.sidebar.button("Start Counting"):
                stframe = st.empty()
                st.markdown("""<h4 style="color:black;">
                            Memory Overall Statistics</h4>""",
                            unsafe_allow_html=True)
                kpi5, kpi6 = st.columns(2)
                with kpi5:
                    st.markdown("""<h5 style="color:black;">
                            CPU Utilization</h5>""",
                            unsafe_allow_html=True)
                    kpi5_text = st.markdown("0")
                with kpi6:
                    st.markdown("""<h5 style="color:black;">
                            Memory Usage</h5>""",
                            unsafe_allow_html=True)
                    kpi6_text = st.markdown("0")
                a = detect(weights='best.pt',
                    source="data/gambarnya/{}".format(image.name),
                    stframe=stframe,
                    #kpi5_text=kpi5_text,
                    #kpi6_text = kpi6_text,
                    conf_thres=float(conf_thres),
                    device="cpu",
                    classes=[0,1],nosave=nosave,
                    name='{}'.format(image.name)
                        )
                inference_msg.success("Inference Complete!")
                st.image(a, channels="BGR",use_column_width=True)
        torch.cuda.empty_cache()
# --------------------MAIN FUNCTION CODE-------------------------
# print(f'Done.({time.time() - t0:.3f}s)')
if __name__ == "__main__":
    try:
        main()
    except SystemExit:
        pass
```

2. Code Detect.py

detect.py

detect

detect 〉 No Selection

```python
import argparse
import time
from pathlib import Path

import cv2
import torch
import torch.backends.cudnn as cudnn
from numpy import random

from models.experimental import attempt_load
from utils.datasets import LoadStreams, LoadImages
from utils.general import check_img_size, check_requirements, check_imshow, non_max_suppression, apply_classifier, \
    scale_coords, xyxy2xywh, strip_optimizer, set_logging, increment_path
from utils.plots import plot_one_box
from utils.torch_utils import select_device, load_classifier, time_synchronized, TracedModel


def detect(source='data/gambarnya', img_size=640, conf_thres=0.25, iou_thres=0.45, device='cpu', view_img=False, save_txt=False, save_conv=False, nosave=False, classes=" ",
           agnostic_nms=True, augment=False, update=False, project='detect', name='exp', exist_ok=False, no_trace=False, weights='best.pt', stframe=None, save_img=False):
    source, weights, view_img, save_txt, imgsz, trace = str(source), 'best.pt', True, False, 640, True
    save_img = True and not source.endswith('.txt')  # save inference images
    webcam = source.isnumeric() or source.endswith('.txt') or source.lower().startswith(
        ('rtsp://', 'rtmp://', 'http://', 'https://'))

    # Directories
    save_dir = Path(increment_path(Path('detect') / 'exp', exist_ok=False))  # increment run
    (save_dir / 'labels' if save_txt else save_dir).mkdir(parents=True, exist_ok=True)  # make dir

    # Initialize
    set_logging()
    device = select_device('cpu')
    half = device.type != 'cpu'  # half precision only supported on CUDA

    # Load model
    model = attempt_load(weights, map_location=device)  # load FP32 model
    stride = int(model.stride.max())  # model stride
    imgsz = check_img_size(imgsz, s=stride)  # check img_size

    if trace:
        model = TracedModel(model, device, 640) #in

    if half:
        model.half()  # to FP16
```

```
    # Second-stage classifier
    classify = False
    if classify:
        modelc = load_classifier(name='resnet101', n=2)  # initialize
        modelc.load_state_dict(torch.load('weights/resnet101.pt',
            map_location=device)['model']).to(device).eval()

    # Set Dataloader
    vid_path, vid_writer = None, None
    if webcam:
        view_img = check_imshow()
        cudnn.benchmark = True  # set True to speed up constant image size inference
        dataset = LoadStreams(source, img_size=imgsz, stride=stride)
    else:
        dataset = LoadImages(source, img_size=imgsz, stride=stride)

    # Get names and colors
    names = model.module.names if hasattr(model, 'module') else model.names
    # colors = [[random.randint(0, 255) for _ in range(3)] for _ in names]
    colors = [[0,255,0], [0,0,255]]

    # print(colors)

    # Run inference
    if device.type != 'cpu':
        model(torch.zeros(1, 3, imgsz,
            imgsz).to(device).type_as(next(model.parameters())))  # run once
    old_img_w = old_img_h = imgsz
    old_img_b = 1

    t0 = time.time()
    for path, img, im0s, vid_cap in dataset:
        img = torch.from_numpy(img).to(device)
        img = img.half() if half else img.float()  # uint8 to fp16/32
        img /= 255.0  # 0 - 255 to 0.0 - 1.0
        if img.ndimension() == 3:
            img = img.unsqueeze(0)

        # Warmup
        if device.type != 'cpu' and (old_img_b != img.shape[0] or old_img_h !=
            img.shape[2] or old_img_w != img.shape[3]):
            old_img_b = img.shape[0]
            old_img_h = img.shape[2]
            old_img_w = img.shape[3]
            for i in range(3):
                model(img, augment=False)[0]

        # Inference
        t1 = time_synchronized()
        with torch.no_grad():   # Calculating gradients would cause a GPU memory
            leak
            pred = model(img, augment=False)[0]
```

```
        t2 = time_synchronized()

        # Apply NMS
        pred = non_max_suppression(pred, 0.25, 0.45, classes=[0, 1], agnostic=False)
        t3 = time_synchronized()

        # Apply Classifier
        if classify:
            pred = apply_classifier(pred, modelc, img, im0s)

        # Process detections
        for i, det in enumerate(pred):  # detections per image
            if webcam:  # batch_size >= 1
                p, s, im0, frame = path[i], '%g: ' % i, im0s[i].copy(), dataset.count
            else:
                p, s, im0, frame = path, '', im0s, getattr(dataset, 'frame', 0)

            p = Path(p)  # to Path
            save_path = str(save_dir / p.name)  # img.jpg
            txt_path = str(save_dir / 'labels' / p.stem) + ('' if dataset.mode == 'image' else f'_{frame}')  # img.txt
            gn = torch.tensor(im0.shape)[[1, 0, 1, 0]]  # normalization gain whwh
            if len(det):
                # Rescale boxes from img_size to im0 size
                det[:, :4] = scale_coords(img.shape[2:], det[:, :4], im0.shape).round()

                # Print results
                for c in det[:, -1].unique():
                    n = (det[:, -1] == c).sum()  # detections per class
                    s += f"{n} {names[int(c)]}{'s' * (n > 1)}, "  # add to string


                # Write results
                for *xyxy, conf, cls in reversed(det):
                    if save_txt:  # Write to file
                        xywh = (xyxy2xywh(torch.tensor(xyxy).view(1, 4)) / gn).view(-1).tolist()  # normalized xywh
                        line = (cls, *xywh, conf) if False else (cls, *xywh)  # label format
                        with open(txt_path + '.txt', 'a') as f:
                            f.write(('%g ' * len(line)).rstrip() % line + '\n')

                    if save_img or view_img:  # Add bbox to image
                        label = f'{names[int(cls)]} {conf:.2f}'
                        plot_one_box(xyxy, im0, label=label, color=colors[int(cls)], line_thickness=1)
```

```
            j = s.split(" ")
            # print(j)
            hidup = []
            mati = []

            if j[2] != "":
                mati.append(j[2])
                hidup.append(j[0])
                im0 = cv2.putText(im0, str("jumlahnya : "+ str(hidup[0]) +" yang hidup, "+ str(mati[0])+" yang mati,"+ " "), (100,100), cv2.FONT_HERSHEY_SIMPLEX, 1, (0,0,0), 2, cv2.LINE_AA)
                # im0 = cv2.putText(im0, str("viabilitasnya : "+ str(hidup) +" yang hidup, "+ str(mati)+" yang mati,"+ " "), (100,100), cv2.FONT_HERSHEY_SIMPLEX, 1, (0,0,0), 2, cv2.LINE_AA)
            else:
                hidup.append(j[0])
                im0 = cv2.putText(im0, str("jumlahnya : "+ str(hidup[0]) +" yang hidup, "), (100,100), cv2.FONT_HERSHEY_SIMPLEX, 1, (0,0,0), 2, cv2.LINE_AA)
            # Stream results
            if view_img:
                cv2.imshow(str(p), im0)
                cv2.waitKey(1)  # 1 millisecond

            # Save results (image with detections)
            if save_img:
                if dataset.mode == 'image':
                    cv2.imwrite(save_path, im0)
                    print(f" The image with the result is saved in: {save_path}")
                else:  # 'video' or 'stream'
                    if vid_path != save_path:  # new video
                        vid_path = save_path
                        if isinstance(vid_writer, cv2.VideoWriter):
                            vid_writer.release()  # release previous video writer
                        if vid_cap:  # video
                            fps = vid_cap.get(cv2.CAP_PROP_FPS)
                            w = int(vid_cap.get(cv2.CAP_PROP_FRAME_WIDTH))
                            h = int(vid_cap.get(cv2.CAP_PROP_FRAME_HEIGHT))
                        else:  # stream
                            fps, w, h = 30, im0.shape[1], im0.shape[0]
                            save_path += '.mp4'
                        vid_writer = cv2.VideoWriter(save_path, cv2.VideoWriter_fourcc(*'mp4v'), fps, (w, h))
                    vid_writer.write(im0)

    if save_txt or save_img:
        s = f"\n{len(list(save_dir.glob('labels/*.txt')))} labels saved to {save_dir / 'labels'}" if save_txt else ''
        #print(f"Results saved to {save_dir}{s}")

    print(f'Done. ({time.time() - t0:.3f}s){s}')
    return im0
```