Bem vindo a documentação do projeto, aqui terá mais alguns detalhes do projeto que não constam no Readme.

Caso já queira executar direto no colab, basta acessar: https://colab.research.google.com/drive/153mcmwWoSCypodpletesmG2X0Iyg0Qv0?usp=sharing

São dependências deste projeto:

- Ultralytics (https://docs.ultralytics.com/)
- Sendgrid (https://github.com/sendgrid/sendgrid-python/)
- OpenCV (https://github.com/opencv/opencv/tree/4.6.0)
- ffmpeg, python-dotenv e tqdm

Executando a aplicação

Antes de começar, verifique se você atendeu ao seguintes requisitos:

- Python na versão 3.11.
- Colocou os 2 arquivos de vídeos em uma pasta chamada vídeos, o nome deles devem ser respectivamente video_1.mp4 e video_2.mp4; Caso opte por outro caminho ou nome de arquivo, será necessário atualizar main.py;
- Instalou as dependências do projeto: `pip install -r requirements.txt`

Após isso `Basta executar o arquivo main.py`

Como configurar a aplicação

- Em constants.py você pode encontrar várias constantes que serão utilizadas por todo o projeto.
- Crie um arquivo .env para configurar as variáveis de ambiente, elas são utilizadas para o disparo de email.

Funcionamento geral

O projeto em si é composto basicamente de três fases:

1. Dividir imagens e rótulos em treino, teste e avaliação (opcional usando a função contida em útil ou manualmente);
2. Treinar o modelo a partir de um existente;
3. Processar vídeo.

O ponto de partida é o arquivo main.py.

Constantes (constants.py)

A maior parte das constantes do projeto estão centralizadas aqui, no arquivo você encontrará comentários sobre o papel de algumas delas.

Treinamento (train_model.py)

Contém uma única função, que justamente carrega o modelo base (constante **BASE_MODEL**), e realiza o treinamento segundo o arquivo de configuração indicado em **DATASET_CONFIG_FILE**, após o treinamento ser concluído o modelo treinado é salvo com o nome conforme **FINE_TUNED_MODEL**.

Processando vídeo (video_reader.py)

Em video_reader.py você encontrará duas funções para processar vídeo e 3 relacionada ao que fazer com cada frame.

```
def execute_video(model: YOLO):
```

Inicializa processamento do vídeo em tempo real vindo de uma câmera, em cada frame é chamado a função **detect_in_frame.** Para encerrar a captura basta tocar na tecla q.

```
def process_video(video_path: str, model: YOLO, must_record=False,
output_file="processado.mp4"):
```

Diferente a anterior, essa vai carregar um arquivo de vídeo ao invés de ser em tempo real. A cada frame também será executado a função **detect_in_frame** e no final, caso **must_record** seja **True** ele salvará o **output_file**.

```
def detect_in_frame(model: YOLO, frame):
```

Pega os resultados vindo do processamento do modelo para cada frame, e caso o *confidence* esteja dentro do limite definido na constante **CONFIDENCE** e a label esteja na **WARNING_LIST**, então é chamado às funções **handle_warning** e **draw_rectangle**;

`def handle_warning(label, confidence, frame):`

Dispara um alerta de segurança com a imagem anexada respeitando um intervalo mínimo entre os alertas (constante **MINIMUM_INTERVAL_EMAIL_IN_SECONDS**).

`def draw_rectangle(frame, x1, y1, x2, y2, label, confidence):`

Desenhar um retângulo com o label sobre a área passada como parâmetro.

Envio de alerta

O projeto ao detectar um objeto cortante ele enviará um email com a imagem anexada, isso é possível através da função **send_email** disponibilizada em **email_service.py**.

`def send_email(subject, body, attachment_content=None, attachment_name=None, attachment_type='image/jpg'):` Função que recebe título, conteúdo e imagem para anexo (opcional), para enviar para o destino informado em **.env**; é necessário que as seguintes variáveis estejam definidas:

- SENDGRID_API_KEY
- FROM_EMAIL
- TO_EMAIL

Para o email ser enviado é necessario que a constante **DISABLE_EMAIL_SERVICE** esteja com valor **False.**

Funções utilitárias (util.py)

Dentro de util.py você encontrará a função **split_img_label** e dentro dela a função **move_files** com a seguinte assinatura e responsabilidade:

`def split_img_label(images_path, labels_path, train_ratio=0.7, val_ratio=0.2, test_ratio=0.1, seed=42):` Função para distribuir as imagens e labels para treino, validação e teste.

`def move_files(file_list, split):` Função para mover os arquivos para um destino específico, além disso move txt que tenha o mesmo nome da imagem.