Hackathon

IA para Devs

Daniela Cruz de Malta - 353365

Gabriela Maciel Godoi - 355125

Lucas Sutelo - 353721

Sumário

1. Introdução

O presente relatório documenta o desenvolvimento de um sistema baseado em Inteligência Artificial (IA) para detectar objetos cortantes em vídeos capturados por câmeras de segurança. O objetivo é validar a viabilidade da funcionalidade dentro do software da empresa **FIAP VisionGuard**, especializada em monitoramento.

A solução consiste em um modelo de **Deep Learning** treinado para identificar objetos cortantes, como facas, tesouras e similares, em imagens e vídeos, acionando um **sistema de alerta** sempre que um objeto perigoso for detectado, podendo esse alerta ser enviado por e-mail.

O código de fonte do processo está no repositório a seguir:

https://github.com/Danizinh/Hackathon

2. Objetivos

O objetivo principal deste projeto é implementar um **modelo de detecção de objetos cortantes** para aprimorar sistemas de segurança. Para isso, foram estabelecidos os seguintes objetivos específicos:

- **Criar um dataset :** imagens de facas, tesouras e objetos cortantes em diferentes condições de iluminação e ângulos.
- **Anotar e balancear o dataset:** para garantir um modelo robusto e reduzir falsos positivos.
- **Treinar um modelo de Deep Learning:** usando a rede neural **ResNet18** para classificar imagens entre **cortante** e **não cortante**.
- **Desenvolver um sistema de detecção em tempo real:** capaz de analisar vídeos capturados por webcams ou arquivos de vídeo.
- **Implementar um sistema de alertas:** que notifique automaticamente uma central de monitoramento caso um objeto cortante seja identificado.

3. Metodologia

3.1. Coleta e Pré-processamento dos Dados

Para garantir um bom desempenho do modelo, foi utilizada uma abordagem supervisionada de aprendizado de máquina. O dataset foi composto por imagens organizadas em duas categorias:

- **Cortante (1)**: Facas, tesouras e lâminas em diferentes cenários.
- **Não Cortante (0)**: Imagens sem objetos perigosos, para evitar falsos positivos.

3.1.1. Transformações Aplicadas

As imagens foram processadas utilizando `torchvision.transforms`, aplicando:

- **Resize (224x224 pixels)** – Para manter um tamanho padronizado.
- **Conversão para Tensor** – Transformação necessária para o modelo ResNet18.
- **Normalização (`mean=[0.485, 0.456, 0.406], std=[0.229, 0.224, 0.225]`)** – Padrão para redes neurais pré-treinadas.

3.2. Modelo Utilizado

O modelo escolhido para a classificação foi a **ResNet18**, uma **rede neural convolucional (CNN)** pré-treinada no **ImageNet**. A camada final foi modificada para realizar uma **classificação binária**:

```
model.fc = torch.nn.Linear(model.fc.in_features, 2)  # Duas classes: cortante / não cortante
```

3.3. Treinamento

O modelo foi treinado usando a função de perda **CrossEntropyLoss** e otimizado com **Adam** (**lr=0.0001)**. Foram realizadas **4 épocas**, utilizando um **batch size de 8**.

Código para treinamento:

```python
def train_model(model, dataloader, epochs=4, lr=0.0001):
    criterion = torch.nn.CrossEntropyLoss()
    optimizer = torch.optim.Adam(model.parameters(), lr=lr)

    for epoch in range(epochs):
        model.train()
        total_loss = 0

        for images, labels in dataloader:
            images, labels = images.to(device), labels.to(device)
            optimizer.zero_grad()
            outputs = model(images)
            loss = criterion(outputs, labels)
            loss.backward()
            optimizer.step()
            total_loss += loss.item()

        print(f"Época {epoch+1}/{epochs}, Loss: {total_loss/len(dataloader):.4f}")

    torch.save(model.state_dict(), "modelo_cortantes.pth")
    print("Modelo salvo!")
```

```
Deseja treinar o modelo? (s/n): s
Época 1/4, Loss: 0.1527
Época 2/4, Loss: 0.0773
Época 3/4, Loss: 0.0782
Época 4/4, Loss: 0.0627
Modelo salvo!
(venv) lucas@lucas-laptop ~/Documentos/Estudos/Pós graduação - IA/Fase 5/Hackathon/project (main)$
```

Chegamos a conclusão de que o melhor número de épocas seria 4 após testarmos com 6 e até 8 épocas, e também com learning rate maiores. Observamos que a partir de 4 épocas, o modelo já estava estabilizado e tendo uma boa eficácia.

4. Implementação do Sistema

O sistema foi dividido em três módulos principais:

4.1. Detecção em Tempo Real

O script **camera_analyser.py** utiliza **OpenCV** para capturar vídeo da webcam, processa cada frame e aplica o modelo de detecção.

Fluxo:

1. Captura o vídeo com OpenCV **(cv2.VideoCapture(0)**).
2. Converte os frames para imagem PIL.

3. Usa a função **predict(image_pil)** para determinar se há um objeto cortante.
4. Se a resposta for positiva, dispara um alerta.

Trecho do código:

```python
def capture_video():
    cap = cv2.VideoCapture(0)
    if not cap.isOpened():
        print("Erro ao acessar a webcam.")
        return

    while True:
        ret, frame = cap.read()
        if not ret:
            break

        image_pil = preprocess_frame(frame)
        is_danger = predict(image_pil)

        if is_danger:
            send_alert()

        if cv2.waitKey(1) & 0xFF == ord('q'):
            break

    cap.release()
    cv2.destroyAllWindows()
```

4.2. Análise de Vídeos Gravados

O script **video_analyser.py** processa vídeos da pasta **video/**, analisando 1 frame a cada 2, para reduzir a carga de processamento. Na sequência, ele aplica a detecção no frame.

```python
video_path = 'video/video.mp4'
video_capture = cv2.VideoCapture(video_path)

while True:
    ret, frame = video_capture.read()
    if not ret:
        break

    image_pil = preprocess_frame(frame)
    is_danger = predict(image_pil)

    if is_danger:
        send_alert()

    if cv2.waitKey(1) & 0xFF == ord('q'):
        break
```

4.3. Sistema de Alertas

Se um objeto cortante for identificado, o sistema chama **send_alert()**, que pode ser adaptado para enviar notificações via **e-mail, SMS ou webhook**.

De modo a evitar que alertas fossem enviados excessivamente, criamos um timer de 10 segundos, na qual limita envios de alertas duplicados para o mesmo objeto.

```python
if is_danger:
    # Verifica se já passou 10 segundos do ultimo alerta
    if (datetime.datetime.now() - last_alert).seconds < 10:
        print(f"Alerta já enviado há {(datetime.datetime.now() - last_alert).seconds} segundos.")

    else:
        last_alert = datetime.datetime.now()
        send_alert()  # Envia alerta por e-mail
```

```python
def send_alert():
    print("Enviando alerta por e-mail...")
```