Отчет и инструкция по итоговой аттестации

Введение

Применение искусственного интеллекта в охране труда может помочь в автоматизации процессов контроля безопасности, обеспечении более быстрой реакции на потенциальные угрозы и снижении риска травм на производстве.

Для контроля у сотрудников использования защитного средств мы можем обучить нейронную сеть и использовать для фиксаций нарушений и отправки оповещений.

В данным момент на рынке имеется множество предложений с возможностями обнаружения нарушений с области охраны труда. Некоторые из них:

- Xeoma;
- CenterVision;
- TRASSIR Hardhat Detector;
- SecurOS Helmet Detector;
- 1С:Предприятие 8. Производственная безопасность.

Наиболее популярными библиотеками для распознавания объектов в изображения являются: OpenCV, scikit-image, PIL, PyTorch, TensorFlow.

В разработанном примере кода используется нейронная сеть на основе библиотеки PyTorch для обнаружения отсутствия у сотрудников защитных касок. Также данный алгоритм можно применять и для других защитных средств.

Код запускался в Google Colab и автоматизирован для запуска в данной среде. При запуске в других средах, возможно, потребуется установка дополнительных библиотек и изменение пути до папки с датасетами.

Этапы выполнения работы

1. **Сбор данных**

Для обучения модели были собраны из сети интернет картинки с изображением защитной каски и без него.

Датасет находится в репозитории github по ссылке:
https://github.com/Aidt87/final_project_ai_architech_course.git

Структура папок датасета:

- train – тренировочные данные;

- val – валидационные данные;

- demo - демонстрационные данные.

В каждой папке расположены папки разделенные по классификации:

- helmet – с защитной каской;

- no_helmet – без защитной каски.

2. Обучение модели

Были варианты обучения и классификации изображений с помощью библиотек TensorFlow и PyTorch с предварительно обученной моделью ResNet18, но результат классификации не удовлетворил требованиям.

Окончательным вариантом было решено использовать PyTorch с предварительно обученной моделью ResNet152.

3. Описание кода

Подключаем и импортируем библиотеки:

```
import torch
import torch.nn as nn
import torch.optim as optim
from torchvision import datasets, transforms, models
import os
from PIL import Image
from torchvision import transforms
import glob
```

Загружаем датасет из репозитория:

```
!git clone https://github.com/Aidt87/final_project_ai_architech_course.git
```

Устанавливаем путь до датасета и размер пакета. Если датасет установлен в другом месте, уставить актуальный путь для вашей системы. В виде модели используем

```
data_dir = '/content/final_project_ai_architech_course/helmet_dataset'  # Каталог с изображениями
batch_size = 50  # Размер пакета данных для обучения
```

Предварительно обрабатываем данные и подготавливаем для обучения.

```
# Предобработка данных. Подготовка к обучению.
data_transforms = {
    'train': transforms.Compose([
        transforms.RandomResizedCrop(224),  # Случайное изменение размера
и обрезка до 224x224 пикселей
        transforms.RandomHorizontalFlip(),  # Случайное горизонтальное
отражение изображения
        transforms.ToTensor(),  # Преобразование в тензор (многомерный
массив)
    ]),
    'val': transforms.Compose([
        transforms.Resize(256),  # Изменение размера до 256x256 пикселей
        transforms.CenterCrop(224),  # Обрезка до 224x224 пикселей по
центру»
        transforms.ToTensor(),  # Преобразование в тензор
    ]),
}

# Создание ImageFolder датасета для обучения и валидации с использованием
заданных трансформаций
image_datasets = {x: datasets.ImageFolder(os.path.join(data_dir, x),
data_transforms[x]) for x in ['train', 'val']}

# Создание DataLoader для загрузки данных с пакетами, перемешиванием и
указанием числа рабочих процессов
dataloaders = {x: torch.utils.data.DataLoader(image_datasets[x],
batch_size=batch_size, shuffle=True, num_workers=0) for x in ['train',
'val']}

# Определение размеров датасетов для обучения и валидации
dataset_sizes = {x: len(image_datasets[x]) for x in ['train', 'val']}

# Получение списка классов (классификация: helmet, no_helmet)
class_names = image_datasets['train'].classes
```

Подготавливаем данные для обучения. Для обучения используем PyTorch с предварительно обученной моделью ResNet152.

```
# Загрузка предварительно обученной модели ResNet152
model = models.resnet152(pretrained=True)

# Получение количества признаков в последнем полносвязном слое
num_ftrs = model.fc.in_features

# Замена последнего полносвязного слоя на слой с 2 выходами (2 класса:
helmet, no_helmet)
model.fc = nn.Linear(num_ftrs, 2)
```

```
# Определение устройства для обучения(GPU или CPU). При отсутствия к
видеокарте проводим обучеие на процессоре.
device = torch.device("cuda:0" if torch.cuda.is_available() else "cpu")

# Перемещение модели на выбранное устройство
model = model.to(device)

# Определение функции потерь (cross-entropy) и оптимизатора (SGD)
criterion = nn.CrossEntropyLoss()
optimizer = optim.SGD(model.parameters(), lr=0.001, momentum=0.9)
```

Обучаем модель на данных для тренировки.

```
# Обучение модели
# Количество эпох
num_epochs = 10

for epoch in range(num_epochs):
    # Тренируем модель
    model.train()
    running_loss = 0.0

    for inputs, labels in dataloaders['train']:
        inputs, labels = inputs.to(device), labels.to(device)
        # стохастический градиентный спуск для обновления весов модели
        optimizer.zero_grad()
        outputs = model(inputs)
        # принимает предсказанные значения модели и истинные (ожидаемые)
значения (метки) и вычисляет, насколько они различаются.
        loss = criterion(outputs, labels)
        loss.backward()
        optimizer.step()
        running_loss += loss.item()
    print(f"Epoch {epoch + 1}, Loss: {running_loss /
dataset_sizes['train']}")
```

Результат обучения:

```
Epoch 1, Loss: 0.017560668032744836
Epoch 2, Loss: 0.01670048308783564
Epoch 3, Loss: 0.01483071466972088
Epoch 4, Loss: 0.013814954922117036
Epoch 5, Loss: 0.011892781689249236
Epoch 6, Loss: 0.009974136177835793
Epoch 7, Loss: 0.008130961964870322
Epoch 8, Loss: 0.008124189900940862
Epoch 9, Loss: 0.0075129937252094
Epoch 10, Loss: 0.005643092095851898
```

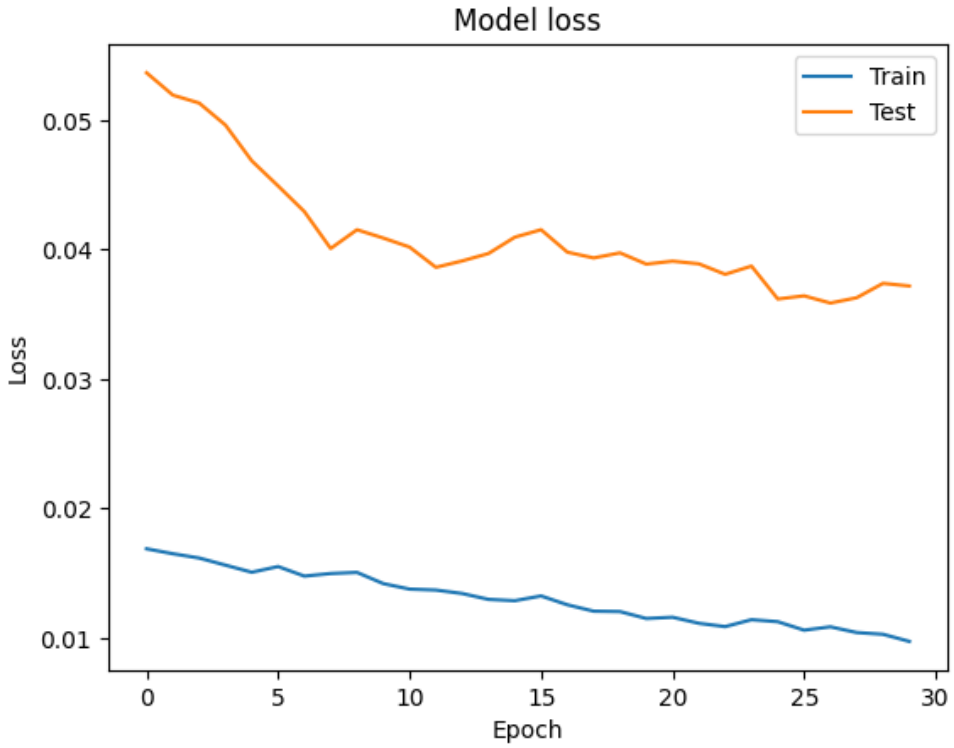
Сохраняем модель:

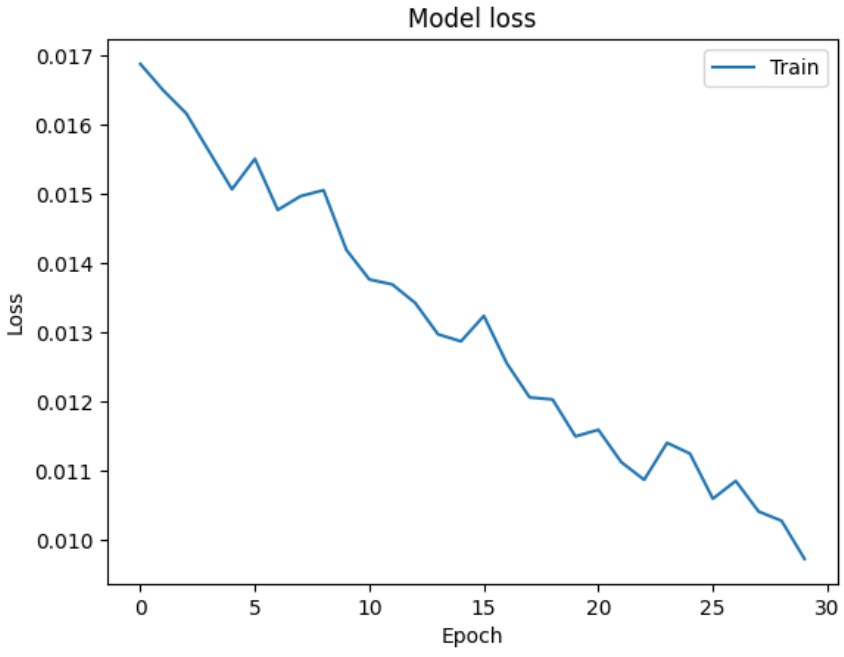
```
torch.save(model.state_dict(), 'helmet_classification_model.pth')
```

Создаем функции для классификации изображения:

```
def classify_image(image_path):

    # Загружаем изображение
    image = Image.open(image_path)
    #
    preprocess = transforms.Compose([
        transforms.Resize(256),
        transforms.CenterCrop(224),
        transforms.ToTensor(),
    ])

    # Добавление размерности батча (batch dimension)
    image = preprocess(image).unsqueeze(0)
    image = image.to(device)

    model.eval()
    with torch.no_grad():
        outputs = model(image)
        _, predicted = torch.max(outputs, 1)

    # Получаем наиболее вероятный результат
    predicted_class = class_names[predicted[0]]
    if predicted_class == 'helmet':
      predicted_class = "Есть каска"
    else:
      predicted_class = "Нет каски"

    image_paths = image_path.split('/')
    image_name = image_paths[-1]

    print(f'Для изображения {image_name} прогнозируемый тип:
{predicted_class}')

    return class_names[predicted[0]]

def classify_images(files, files_class = ''):
  all = len(files)
  correct = 0.0
  for file in files:
    # Используем функцию classify_image для классификации новых
изображений
    predicted_class = classify_image(file)
    if files_class == predicted_class:
      correct += 1

  if files_class != '':
    print(f"Accuracy: {correct/all}")
```

```
plt.plot(losses)
plt.title('Model loss')
plt.ylabel('Loss')
plt.xlabel('Epoch')
plt.legend(['Train'], loc='upper right')
plt.show()

plt.plot(losses)
plt.plot(losses_ts)
plt.title('Model loss')
plt.ylabel('Loss')
plt.xlabel('Epoch')
plt.legend(['Train', 'Test'], loc='upper right')
```





Проверяем работу модели на тестовых данных:

```
# Проверяем на тестовых файлах
validations = glob.glob(os.path.join(data_dir, 'val', 'helmet', '*.*'))
classify_images(validations, 'helmet')
print()

validations = glob.glob(os.path.join(data_dir, 'val', 'no_helmet', '*.*'))
classify_images(validations, 'no_helmet')
```

Результат проверки:

```
Для изображения helmet4.jpg прогнозируемый тип: Есть каска
Для изображения helmet1.jpg прогнозируемый тип: Есть каска
Для изображения helmet3.jpg прогнозируемый тип: Есть каска
Для изображения helmet2.jpg прогнозируемый тип: Есть каска
Accuracy: 1.0

Для изображения no_helmet1.jpg прогнозируемый тип: Нет каски
Для изображения no_helmet4.jpg прогнозируемый тип: Нет каски
Для изображения no_helmet2.jpg прогнозируемый тип: Есть каска
Для изображения no_helmet5.jpg прогнозируемый тип: Нет каски
Для изображения no_helmet3.jpg прогнозируемый тип: Нет каски
Для изображения no_helmet6.jpg прогнозируемый тип: Нет каски
Accuracy: 0.8333333333333334
```

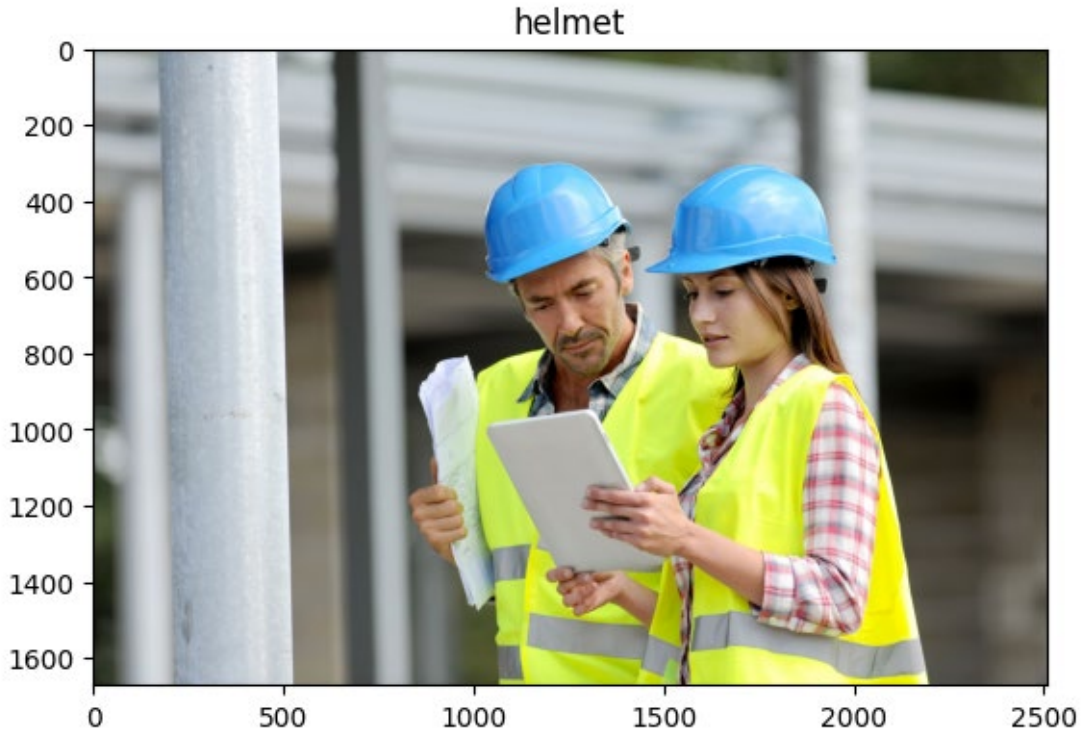
Проверяем работу модели на демонстрационных данных:

```
# Получаем список демонстрационных файлов
demonstrations = glob.glob(os.path.join(data_dir, 'demo', '*.*'))
classify_images(demonstrations)
```

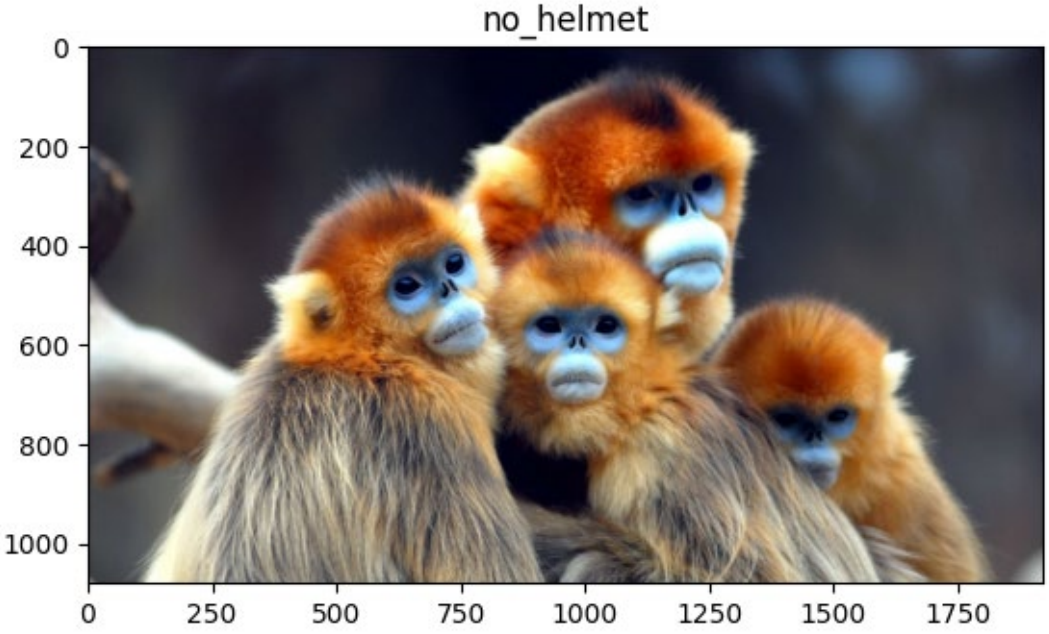
Результат проверки:

```
Для изображения no_helmet6.jpg прогнозируемый тип: Нет каски
```

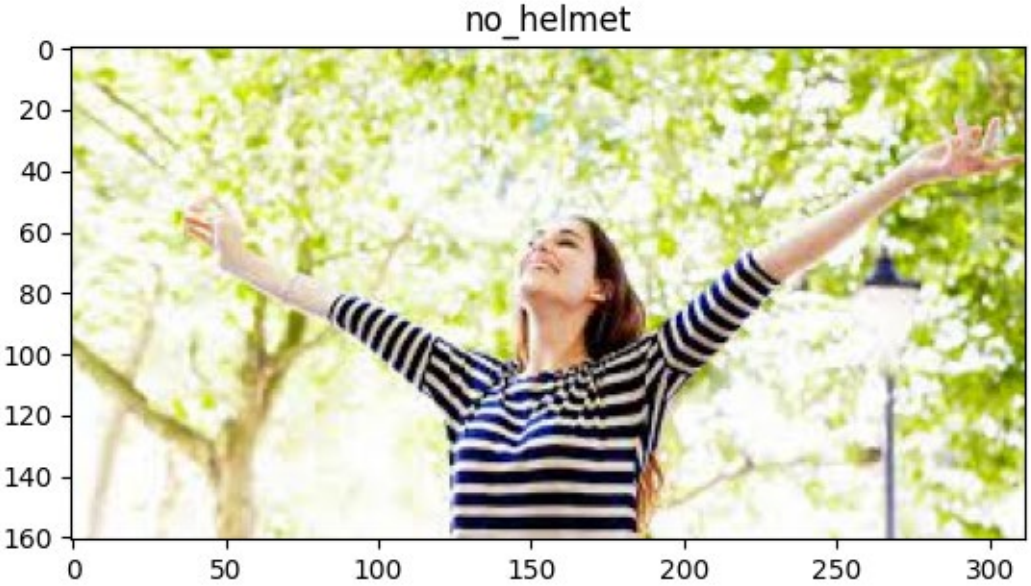

```
Для изображения helmet13.jpg прогнозируемый тип: Есть каска
```




```
Для изображения helmet5.jpg прогнозируемый тип: Есть каска
```



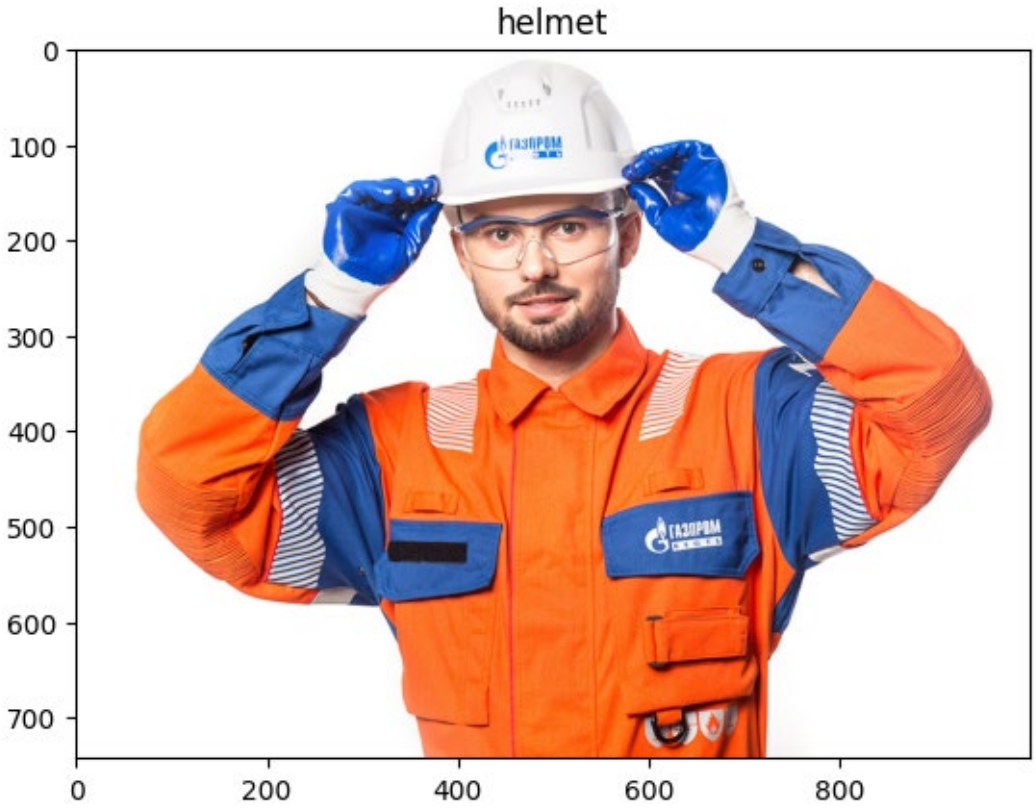
Для изображения no_helmet3.jpg прогнозируемый тип: Нет каски



Для изображения no_helmet1.jpg прогнозируемый тип: Нет каски

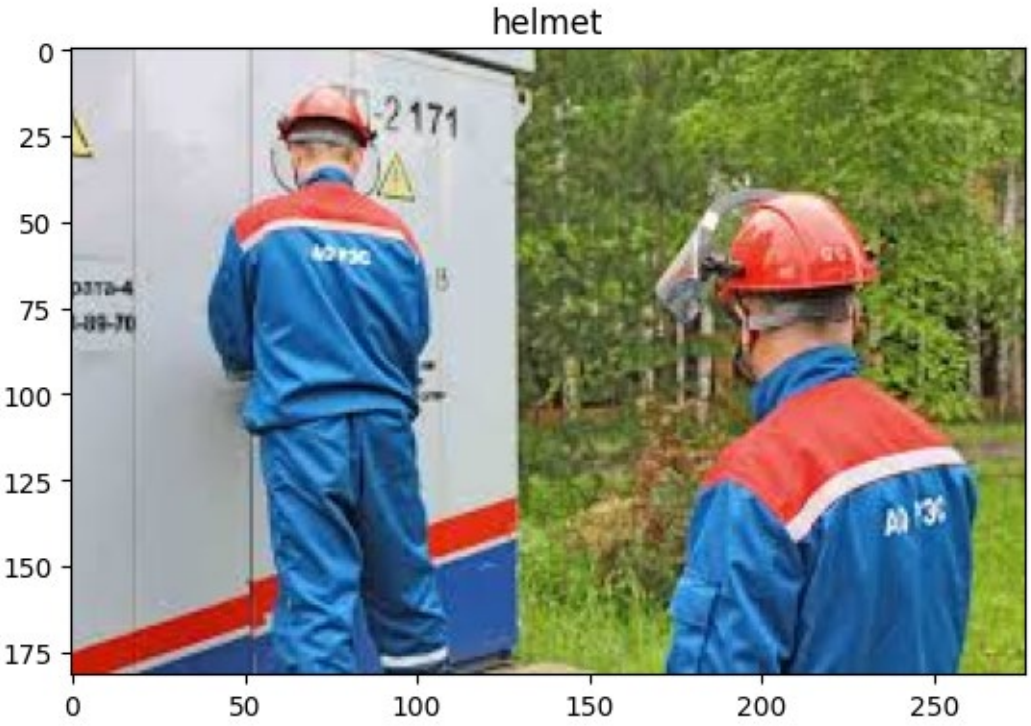
Для изображения helmet7.jpg прогнозируемый тип: Есть каска

Для изображения helmet8.jpg прогнозируемый тип: Есть каска

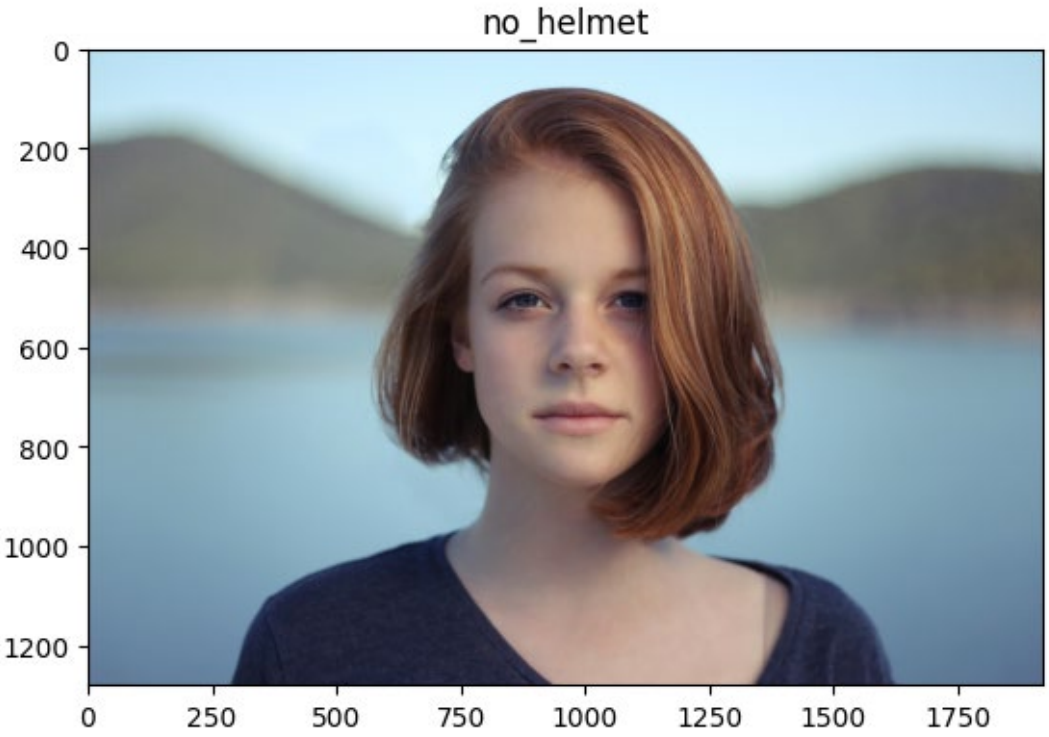


Для изображения helmet11.jpg прогнозируемый тип: Есть каска

Для изображения no_helmet13.jpg прогнозируемый тип: Нет каски

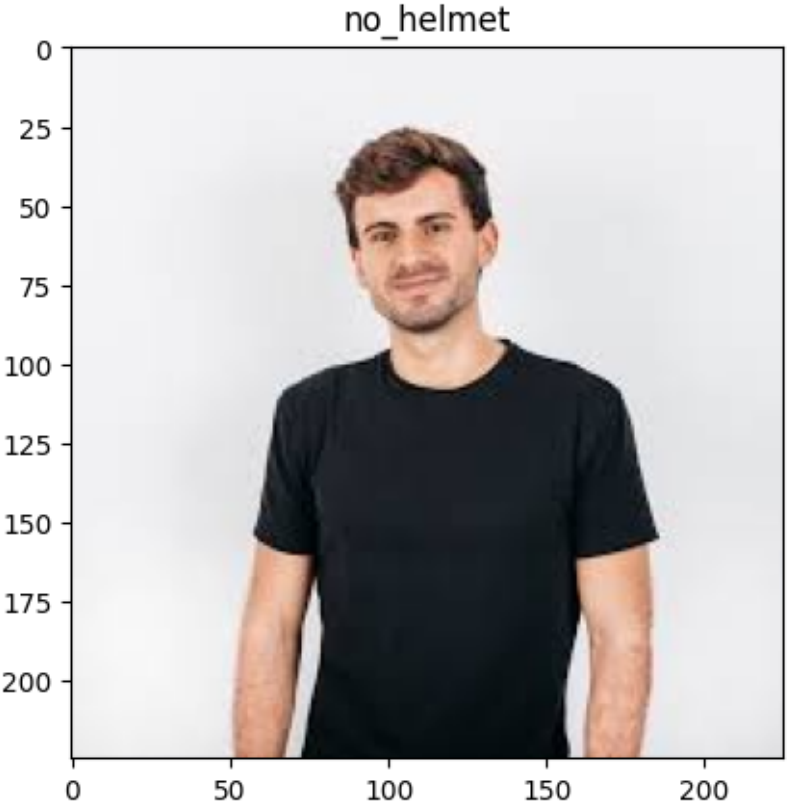


Для изображения helmet2.jpg прогнозируемый тип: Есть каска



Для изображения no_helmet10.jpg прогнозируемый тип: Нет каски

Для изображения helmet9.jpg прогнозируемый тип: Есть каска



Для изображения no_helmet2.jpg прогнозируемый тип: Нет каски

По результатам исследования данная модель позволяет с хорошей вероятностью обнаруживать наличие защитной каски у сотрудников.