import { useState, useRef, useCallback, useEffect } from 'react';

import { Button } from '@/components/ui/button';

import { Card, CardContent, CardHeader, CardTitle } from '@/components/ui/card';

import { Badge } from '@/components/ui/badge';

import { Progress } from '@/components/ui/progress';

import { Camera, Upload, Play, Square, Shield, AlertTriangle, CheckCircle } from 'lucide-react';

import { useToast } from '@/hooks/use-toast';

import { cn } from '@/lib/utils';

interface Detection {

label: string;

confidence: number;

box: {

xmin: number;

ymin: number;

xmax: number;

ymax: number;

};

}

interface PPEStatus {

helmet: boolean;

vest: boolean;

gloves: boolean;

goggles: boolean;

}

export const PPEDetector = () => {

const [isLoading, setIsLoading] = useState(false);

const [isRecording, setIsRecording] = useState(false);

const [detections, setDetections] = useState<Detection[]>([]);

const [ppeStatus, setPpeStatus] = useState<PPEStatus>({

helmet: false,

vest: false,

gloves: false,

goggles: false,

});

const [stream, setStream] = useState<MediaStream | null>(null);

const [model, setModel] = useState<any>(null);

const videoRef = useRef<HTMLVideoElement>(null);

const canvasRef = useRef<HTMLCanvasElement>(null);

const fileInputRef = useRef<HTMLInputElement>(null);

const { toast } = useToast();

// Load YOLOv8 model

useEffect(() => {

const loadModel = async () => {

try {

setIsLoading(true);

const { pipeline } = await import('@huggingface/transformers');

// Using YOLOv8 for object detection

const detector = await pipeline(

'object-detection',

'Ultralytics/YOLOv8n',

{

device: 'webgpu',

revision: 'main'

}

);

setModel(detector);

toast({

title: "YOLOv8 Model Loaded",

description: "Advanced PPE detection system ready!",

});

} catch (error) {

console.error('Error loading YOLOv8 model:', error);

toast({

title: "Model Load Failed",

description: "Falling back to demo mode",

variant: "destructive",

});

// Set a mock model for demonstration

setModel({ predict: () => [] });

} finally {

setIsLoading(false);

}

};

loadModel();

}, [toast]);

const startCamera = useCallback(async () => {

try {

const mediaStream = await navigator.mediaDevices.getUserMedia({

video: { width: 640, height: 480 }

});

setStream(mediaStream);

if (videoRef.current) {

videoRef.current.srcObject = mediaStream;

videoRef.current.play();

}

setIsRecording(true);

toast({

title: "Camera Started",

description: "Real-time PPE detection is active",

});

} catch (error) {

console.error('Error starting camera:', error);

toast({

title: "Camera Error",

description: "Could not access camera",

variant: "destructive",

});

}

}, [toast]);

const stopCamera = useCallback(() => {

if (stream) {

stream.getTracks().forEach(track => track.stop());

setStream(null);

}

setIsRecording(false);

setDetections([]);

toast({

title: "Camera Stopped",

description: "Real-time detection paused",

});

}, [stream, toast]);

const analyzeFrame = useCallback(async () => {

if (!model || !videoRef.current || !canvasRef.current) return;

const video = videoRef.current;

const canvas = canvasRef.current;

const ctx = canvas.getContext('2d');

if (!ctx) return;

// Set canvas size to match video

canvas.width = video.videoWidth;

canvas.height = video.videoHeight;

// Draw current frame

ctx.drawImage(video, 0, 0);

try {

// Convert canvas to image data for YOLOv8 analysis

const imageData = canvas.toDataURL('image/jpeg', 0.8);

// Perform actual YOLOv8 detection

if (model && typeof model === 'object' && 'predict' in model) {

const predictions = await model(imageData);

// Filter and map YOLOv8 predictions to PPE items

const ppeDetections: Detection[] = predictions

.filter((pred: any) => {

const label = pred.label.toLowerCase();

return (

label.includes('person') ||

label.includes('helmet') ||

label.includes('hat') ||

label.includes('vest') ||

label.includes('glove') ||

label.includes('goggle') ||

label.includes('glasses') ||

pred.score > 0.5

);

})

.map((pred: any) => ({

label: mapToPPELabel(pred.label),

confidence: pred.score,

box: {

xmin: pred.box.xmin,

ymin: pred.box.ymin,

xmax: pred.box.xmax,

ymax: pred.box.ymax,

}

}));

setDetections(ppeDetections);

// Update PPE status based on YOLOv8 results

const newStatus = {

helmet: ppeDetections.some(d =>

d.label.includes('helmet') ||

d.label.includes('hard hat') ||

d.label.includes('hat')

),

vest: ppeDetections.some(d =>

d.label.includes('vest') ||

d.label.includes('jacket')

),

gloves: ppeDetections.some(d =>

d.label.includes('glove')

),

goggles: ppeDetections.some(d =>

d.label.includes('goggle') ||

d.label.includes('glasses') ||

d.label.includes('eyewear')

),

};

setPpeStatus(newStatus);

} else {

// Fallback mock detection for demo

const mockDetections: Detection[] = [

{

label: 'hard hat',

confidence: 0.85 + Math.random() \* 0.1,

box: {

xmin: 100 + Math.random() \* 50,

ymin: 50 + Math.random() \* 30,

xmax: 200 + Math.random() \* 20,

ymax: 150 + Math.random() \* 20

}

},

{

label: 'safety vest',

confidence: 0.92 + Math.random() \* 0.05,

box: {

xmin: 80 + Math.random() \* 30,

ymin: 150 + Math.random() \* 20,

xmax: 220 + Math.random() \* 10,

ymax: 300 + Math.random() \* 20

}

}

];

setDetections(mockDetections);

const newStatus = {

helmet: mockDetections.some(d => d.label.includes('helmet') || d.label.includes('hat')),

vest: mockDetections.some(d => d.label.includes('vest')),

gloves: Math.random() > 0.5,

goggles: Math.random() > 0.7,

};

setPpeStatus(newStatus);

}

} catch (error) {

console.error('Error during analysis:', error);

}

}, [model]);

// Continuous analysis during recording

useEffect(() => {

let intervalId: NodeJS.Timeout;

if (isRecording && model) {

intervalId = setInterval(analyzeFrame, 2000); // Analyze every 2 seconds for YOLOv8

}

return () => {

if (intervalId) {

clearInterval(intervalId);

}

};

}, [isRecording, model, analyzeFrame]);

const handleImageUpload = async (event: React.ChangeEvent<HTMLInputElement>) => {

const file = event.target.files?.[0];

if (!file || !model) return;

setIsLoading(true);

try {

const img = new Image();

img.onload = async () => {

const canvas = canvasRef.current;

if (!canvas) return;

const ctx = canvas.getContext('2d');

if (!ctx) return;

canvas.width = img.width;

canvas.height = img.height;

ctx.drawImage(img, 0, 0);

// YOLOv8 analysis for uploaded image

if (model && typeof model === 'object' && 'predict' in model) {

const imageData = canvas.toDataURL('image/jpeg', 0.8);

const predictions = await model(imageData);

const ppeDetections: Detection[] = predictions

.filter((pred: any) => pred.score > 0.4)

.map((pred: any) => ({

label: mapToPPELabel(pred.label),

confidence: pred.score,

box: {

xmin: pred.box.xmin,

ymin: pred.box.ymin,

xmax: pred.box.xmax,

ymax: pred.box.ymax,

}

}));

setDetections(ppeDetections);

const newStatus = {

helmet: ppeDetections.some(d =>

d.label.includes('helmet') ||

d.label.includes('hard hat') ||

d.label.includes('hat')

),

vest: ppeDetections.some(d =>

d.label.includes('vest') ||

d.label.includes('jacket')

),

gloves: ppeDetections.some(d =>

d.label.includes('glove')

),

goggles: ppeDetections.some(d =>

d.label.includes('goggle') ||

d.label.includes('glasses')

),

};

setPpeStatus(newStatus);

} else {

// Fallback mock detection

const mockDetections: Detection[] = [

{

label: 'hard hat',

confidence: 0.78 + Math.random() \* 0.15,

box: { xmin: 120, ymin: 60, xmax: 180, ymax: 120 }

}

];

setDetections(mockDetections);

const newStatus = {

helmet: mockDetections.some(d => d.label.includes('helmet') || d.label.includes('hat')),

vest: mockDetections.some(d => d.label.includes('vest')),

gloves: mockDetections.some(d => d.label.includes('glove')),

goggles: mockDetections.some(d => d.label.includes('goggle') || d.label.includes('glass')),

};

setPpeStatus(newStatus);

}

toast({

title: "Image Analyzed",

description: `Found ${detections.length} PPE items`,

});

};

img.src = URL.createObjectURL(file);

} catch (error) {

console.error('Error analyzing image:', error);

toast({

title: "Analysis Failed",

description: "Could not analyze the image",

variant: "destructive",

});

} finally {

setIsLoading(false);

}

};

// Helper function to map YOLOv8 labels to PPE categories

const mapToPPELabel = (originalLabel: string): string => {

const label = originalLabel.toLowerCase();

if (label.includes('helmet') || label.includes('hardhat') || label.includes('hard hat')) return 'hard hat';

if (label.includes('vest') || label.includes('jacket') || label.includes('safety vest')) return 'safety vest';

if (label.includes('glove') || label.includes('gloves')) return 'safety gloves';

if (label.includes('goggle') || label.includes('glasses') || label.includes('eyewear')) return 'safety goggles';

if (label.includes('person') || label.includes('worker')) return 'person';

return originalLabel;

};

const getComplianceStatus = () => {

const requiredItems = Object.values(ppeStatus);

const detected = requiredItems.filter(Boolean).length;

const total = requiredItems.length;

const percentage = (detected / total) \* 100;

return { detected, total, percentage };

};

const compliance = getComplianceStatus();

return (

<div className="min-h-screen bg-background p-6">

<div className="mx-auto max-w-7xl space-y-6">

{/\* Header \*/}

<div className="text-center space-y-4">

<div className="inline-flex items-center gap-2 px-4 py-2 bg-gradient-primary rounded-full text-white">

<Shield className="h-5 w-5" />

<span className="font-semibold">YOLOv8 PPE Detection</span>

</div>

<h1 className="text-4xl font-bold tracking-tight">

Personal Protective Equipment Monitor

</h1>

<p className="text-muted-foreground text-lg max-w-2xl mx-auto">

YOLOv8-powered computer vision system for real-time safety compliance monitoring

</p>

</div>

{/\* Main Content \*/}

<div className="grid lg:grid-cols-3 gap-6">

{/\* Video/Camera Feed \*/}

<div className="lg:col-span-2">

<Card className="shadow-detection">

<CardHeader>

<CardTitle className="flex items-center gap-2">

<Camera className="h-5 w-5" />

Detection Feed

</CardTitle>

</CardHeader>

<CardContent className="space-y-4">

<div className="relative aspect-video bg-industrial-gray rounded-lg overflow-hidden">

{isRecording ? (

<video

ref={videoRef}

className="w-full h-full object-cover"

autoPlay

muted

playsInline

/>

) : (

<div className="flex items-center justify-center h-full text-muted-foreground">

<div className="text-center">

<Camera className="h-12 w-12 mx-auto mb-2 opacity-50" />

<p>Camera feed will appear here</p>

</div>

</div>

)}

{/\* Detection Overlays \*/}

{detections.map((detection, index) => (

<div

key={index}

className="absolute border-2 border-safety-green bg-safety-green/20"

style={{

left: `${(detection.box.xmin / 640) \* 100}%`,

top: `${(detection.box.ymin / 480) \* 100}%`,

width: `${((detection.box.xmax - detection.box.xmin) / 640) \* 100}%`,

height: `${((detection.box.ymax - detection.box.ymin) / 480) \* 100}%`,

}}

>

<Badge className="absolute -top-6 left-0 bg-safety-green text-white">

{detection.label} ({Math.round(detection.confidence \* 100)}%)

</Badge>

</div>

))}

{isRecording && (

<div className="absolute top-4 right-4">

<div className="flex items-center gap-2 px-3 py-1 bg-safety-red text-white rounded-full text-sm">

<div className="w-2 h-2 bg-white rounded-full animate-pulse" />

LIVE

</div>

</div>

)}

</div>

<canvas ref={canvasRef} className="hidden" />

<div className="flex gap-2">

{!isRecording ? (

<Button

onClick={startCamera}

disabled={isLoading}

className="bg-gradient-primary hover:opacity-90"

>

<Play className="h-4 w-4 mr-2" />

Start Camera

</Button>

) : (

<Button

onClick={stopCamera}

variant="destructive"

>

<Square className="h-4 w-4 mr-2" />

Stop Camera

</Button>

)}

<Button

onClick={() => fileInputRef.current?.click()}

variant="outline"

disabled={isLoading}

>

<Upload className="h-4 w-4 mr-2" />

Upload Image

</Button>

<input

ref={fileInputRef}

type="file"

accept="image/\*"

onChange={handleImageUpload}

className="hidden"

/>

</div>

</CardContent>

</Card>

</div>

{/\* Detection Results \*/}

<div className="space-y-6">

{/\* Compliance Status \*/}

<Card className={cn(

"shadow-alert transition-all duration-300",

compliance.percentage === 100 ? "border-safety-green" : "border-safety-orange"

)}>

<CardHeader>

<CardTitle className="flex items-center gap-2">

{compliance.percentage === 100 ? (

<CheckCircle className="h-5 w-5 text-safety-green" />

) : (

<AlertTriangle className="h-5 w-5 text-safety-orange" />

)}

Safety Compliance

</CardTitle>

</CardHeader>

<CardContent className="space-y-4">

<div className="text-center">

<div className="text-3xl font-bold">

{compliance.detected}/{compliance.total}

</div>

<p className="text-muted-foreground">PPE Items Detected</p>

</div>

<Progress

value={compliance.percentage}

className="h-3"

/>

<div className="text-center">

<Badge

variant={compliance.percentage === 100 ? "default" : "destructive"}

className={cn(

compliance.percentage === 100

? "bg-safety-green"

: "bg-safety-orange"

)}

>

{compliance.percentage === 100 ? "COMPLIANT" : "NON-COMPLIANT"}

</Badge>

</div>

</CardContent>

</Card>

{/\* PPE Status \*/}

<Card>

<CardHeader>

<CardTitle>PPE Status</CardTitle>

</CardHeader>

<CardContent className="space-y-3">

{[

{ key: 'helmet', label: 'Hard Hat/Helmet', icon: '⛑️' },

{ key: 'vest', label: 'Safety Vest', icon: '🦺' },

{ key: 'gloves', label: 'Safety Gloves', icon: '🧤' },

{ key: 'goggles', label: 'Safety Goggles', icon: '🥽' },

].map(({ key, label, icon }) => (

<div key={key} className="flex items-center justify-between p-3 rounded-lg bg-muted/50">

<div className="flex items-center gap-3">

<span className="text-lg">{icon}</span>

<span className="font-medium">{label}</span>

</div>

<Badge

variant={ppeStatus[key as keyof PPEStatus] ? "default" : "destructive"}

className={cn(

ppeStatus[key as keyof PPEStatus]

? "bg-safety-green"

: "bg-safety-red"

)}

>

{ppeStatus[key as keyof PPEStatus] ? "DETECTED" : "MISSING"}

</Badge>

</div>

))}

</CardContent>

</Card>

{/\* Detection History \*/}

<Card>

<CardHeader>

<CardTitle>Recent Detections</CardTitle>

</CardHeader>

<CardContent>

{detections.length > 0 ? (

<div className="space-y-2">

{detections.map((detection, index) => (

<div key={index} className="flex items-center justify-between p-2 rounded bg-muted/30">

<span className="font-medium capitalize">{detection.label}</span>

<Badge variant="outline">

{Math.round(detection.confidence \* 100)}%

</Badge>

</div>

))}

</div>

) : (

<p className="text-muted-foreground text-center py-4">

No PPE detected yet

</p>

)}

</CardContent>

</Card>

</div>

</div>

</div>

</div>

);

};