

---

## BAB VII

# COMPUTER VISION

### 1. Computer Vision

Computer vision adalah cabang dari kecerdasan buatan (AI) yang menggunakan pembelajaran mendalam untuk menganalisis data visual. Teknologi ini memungkinkan komputer untuk melihat, mengenali, dan memahami dunia visual, seperti gambar digital dari kamera dan video. Dengan computer vision, komputer dapat mengidentifikasi dan mengklasifikasikan objek, dan kemudian bereaksi terhadap apa yang "dilihatnya".

#### Komponen Utama dalam Computer Vision:

1. Akuisisi Gambar: Proses mendapatkan gambar atau video dari sensor seperti kamera, kamera video, atau alat pemindai.
2. Pemrosesan Gambar: Melibatkan berbagai teknik untuk meningkatkan kualitas gambar atau mengekstraksi fitur penting dari gambar tersebut. Ini bisa termasuk operasi seperti perbaikan kontras, pengurangan kebisingan, dan deteksi tepi.
3. Analisis Gambar: Melibatkan pengenalan pola, segmentasi gambar (memisahkan gambar menjadi bagian-bagian yang berbeda), dan analisis objek dalam gambar.
4. Pengenalan Objek: Melibatkan identifikasi dan klasifikasi objek dalam gambar atau video berdasarkan fitur yang telah diekstraksi sebelumnya. Ini sering dilakukan dengan menggunakan teknik pembelajaran mesin, khususnya deep learning.
5. Pemahaman Gambar: Menginterpretasikan dan memahami gambar dalam konteks yang lebih luas, seperti mengenali aktivitas dalam video, memahami hubungan antar objek, dan memberikan deskripsi tekstual dari sebuah gambar.

#### Teknologi dan Algoritma:

Computer vision menggunakan berbagai teknologi dan algoritma, termasuk:

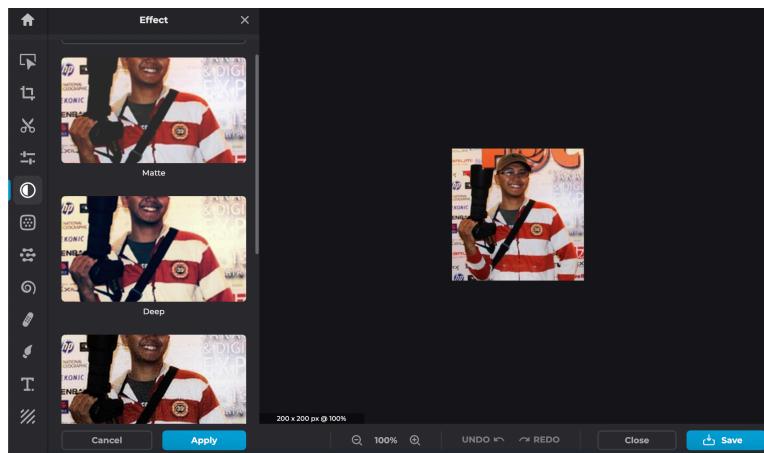
1. Convolutional Neural Networks (CNNs): Digunakan secara luas dalam deep learning untuk pengenalan dan klasifikasi gambar.
2. Optical Flow: Digunakan untuk mendeteksi pergerakan dalam video.
3. Feature Detection and Matching: Algoritma seperti SIFT dan SURF digunakan untuk mendeteksi dan mencocokkan fitur antara gambar yang berbeda.
4. Segmentation Algorithms: Algoritma seperti watershed dan k-means clustering digunakan untuk segmentasi gambar.

Computer vision dan website dapat saling terkait dalam berbagai cara. Integrasi teknologi computer vision ke dalam aplikasi web membuka berbagai kemungkinan menarik dan berguna. Berikut adalah beberapa contoh bagaimana computer vision dapat digunakan dalam konteks website.

#### Aplikasi dan Kaitan Antara Computer Vision dan Website:

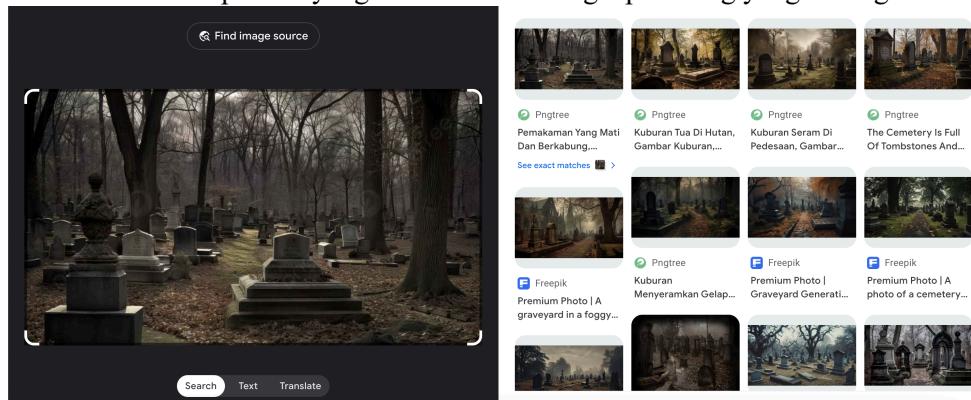
1. Pengenalan Wajah:
  - Keamanan dan Login: Situs web dapat menggunakan teknologi pengenalan wajah untuk autentikasi pengguna, menggantikan kata sandi dengan login biometrik.
  - Personalisasi Konten: Situs e-commerce atau layanan streaming dapat menggunakan pengenalan wajah untuk mengidentifikasi pengguna dan menampilkan konten yang dipersonalisasi berdasarkan preferensi mereka.
2. Pengeditan Gambar dan Video:

- Editor Foto Online: Situs web dapat menyediakan alat pengeditan gambar yang memanfaatkan teknologi computer vision untuk fitur seperti penghapusan latar belakang otomatis, deteksi wajah, dan penyempurnaan gambar.
- Platform Media Sosial: Pengenalan objek dan wajah dapat digunakan untuk menandai orang dalam foto secara otomatis atau menambahkan filter AR (Augmented Reality) ke gambar dan video.



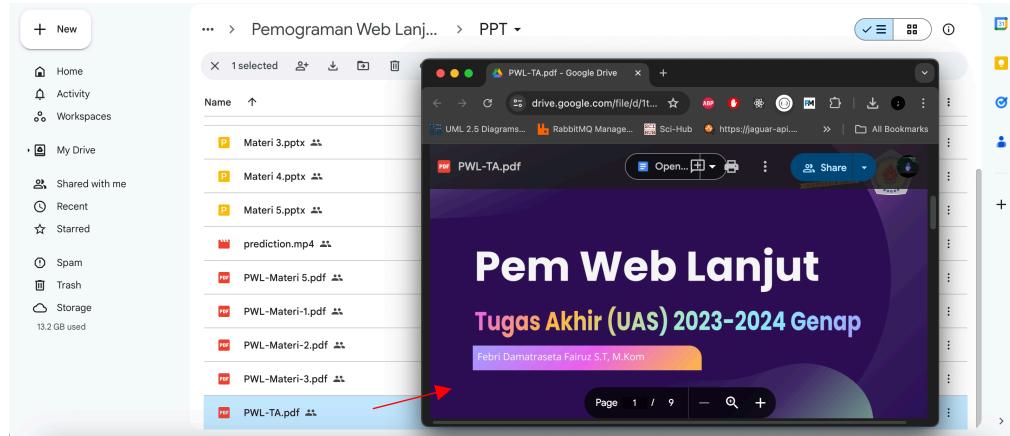
### 3. E-Commerce:

- Pencarian Visual: Situs e-commerce dapat memungkinkan pengguna untuk mencari produk dengan mengunggah gambar produk yang mereka cari. Algoritma computer vision akan menganalisis gambar dan menemukan produk serupa di katalog online.
- Rekomendasi Produk: Analisis gambar yang diunggah pengguna dapat digunakan untuk merekomendasikan produk yang sesuai atau melengkapi barang yang sedang dicari.



### 4. Peningkatan Aksesibilitas:

- Deskripsi Gambar: Situs web dapat menggunakan teknologi computer vision untuk menghasilkan deskripsi otomatis untuk gambar, membantu pengguna dengan gangguan penglihatan memahami konten visual.
- Deteksi Teks dalam Gambar: OCR (Optical Character Recognition) dapat digunakan untuk mengekstrak teks dari gambar atau dokumen yang diunggah, memudahkan pengguna dalam mengakses dan mengedit informasi tersebut.



### 5. Interaksi Pengguna yang Lebih Kaya:

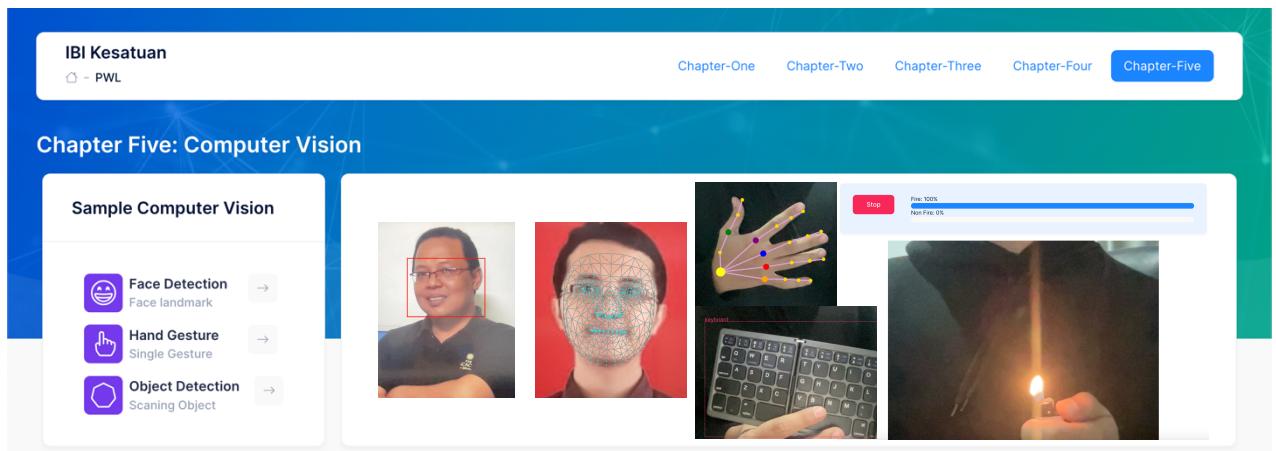
- Gesture Recognition: Situs web dapat menggunakan pengenalan gerakan untuk mengizinkan pengguna berinteraksi dengan situs melalui gerakan tangan atau tubuh, menciptakan pengalaman yang lebih menarik dan interaktif.
- Augmented Reality: Situs web e-commerce atau pendidikan dapat menggunakan teknologi AR untuk menampilkan model 3D produk atau objek pendidikan yang dapat berinteraksi dengan lingkungan pengguna melalui kamera perangkat mereka.

### Teknologi dan Framework yang Digunakan:

Untuk mengintegrasikan computer vision dengan situs web, berbagai teknologi dan framework dapat digunakan, termasuk:

- Webcam Access: Menggunakan API JavaScript seperti getUserMedia() untuk mengakses kamera perangkat pengguna.
- TensorFlow.js: Library JavaScript untuk menjalankan model machine learning di browser.
- OpenCV.js: Porting dari OpenCV ke JavaScript yang memungkinkan pemrosesan gambar langsung di browser.
- Teachable Machine: Platform dari Google yang memungkinkan pembuatan model machine learning yang dapat diintegrasikan dengan situs web untuk berbagai aplikasi computer vision.

### 2. Latihan:Membangun Aplikasi Messenger – Chapter 5



Pada latihan modul kali ini, kita akan mencoba menerapkan computer vision pada website yang telah dibuat sebelumnya, dimana kita akan mengintegrasikan library framework UI React JS dengan computer



vision seperti Tensorflow JS dan Webcam Access. Pelatihan ini akan mencoba mengenali ataupun mendeteksi adanya objek pada citra dalam bentuk real-time, yaitu dengan memanfaatkan webcam yang dimiliki masing-masing perangkat. Pengenalan atau pemanfaatkan computer vision dalam bab ini, kita akan membuat sebuah pengenalan adanya wajah, bentuk muka (landmark) ataupun dapat mengenali benda yang terekam pada citra.

#### A. Pengenalan terhadap wajah

Untuk menerapkan computer vision dengan memanfaatkan Tensorflow JS, pada projek react anda perlu menambahkan beberapa library yang akan digunakan, seperti:

```
npm install @tensorflow/tfjs @tensorflow-models/blazeface @tensorflow-models/face-  
landmarks-detection react-webcam
```

setelah anda menambahkan library diatas, selanjutnya buatlah komponen function react untuk membuat sebuah pengenalan adanya wajah. Contoh dibawah ini ialah script untuk mendeteksi adanya wajah manusia dengan cara membuat bounding box pada wilayah wajah:

```
import React, { useRef, useEffect, useState } from 'react';  
import * as tf from '@tensorflow/tfjs';  
import * as blazeface from '@tensorflow-models/blazeface';  
  
export function FaceDetection() {  
    const videoRef = useRef(null);  
    const canvasRef = useRef(null);  
    const modelRef = useRef(null);  
    const [videoLoaded, setVideoLoaded] = useState(false);  
    const [isDetecting, setIsDetecting] = useState(false);  
    const [stream, setStream] = useState(null);  
  
    useEffect(() => {  
        async function loadModel() {  
            modelRef.current = await blazeface.load();  
        }  
  
        loadModel();  
    }, []);  
  
    useEffect(() => {  
        if (videoLoaded && isDetecting) {  
            const detectFaces = async () => {  
                if (!videoRef.current || !canvasRef.current || !modelRef.current) {  
                    return;  
                }  
  
                const predictions = await  
modelRef.current.estimateFaces(videoRef.current, false);  
  
                // Gambar hasil deteksi di canvas  
                const ctx = canvasRef.current.getContext('2d');  
                if (!ctx) {  
                    return;  
                }  
            };  
        }  
    }, [videoLoaded, isDetecting]);  
}
```



```
        ctx.clearRect(0, 0, canvasRef.current.width,
    canvasRef.current.height);
    predictions.forEach(prediction => {
        const start = prediction.topLeft;
        const end = prediction.bottomRight;
        const size = [end[0] - start[0], end[1] - start[1]];

        // membuat kotak di sekitar wajah
        ctx.beginPath();
        ctx.strokeStyle = "red";
        ctx.lineWidth = 2;
        ctx.rect(start[0], start[1], size[0], size[1]);
        ctx.stroke();
    });

    if (isDetecting) {
        requestAnimationFrame(detectFaces);
    }
};

detectFaces();

}, [videoLoaded, isDetecting]);

const handlePlay = () => {
    navigator.mediaDevices.getUserMedia({ video: true })
        .then((stream) => {
            videoRef.current.srcObject = stream;
            setStream(stream);
            videoRef.current.onloadedmetadata = () => {
                setVideoLoaded(true);
                videoRef.current.play();
                setIsDetecting(true);
            };
        });
};

const handleStop = () => {
    setIsDetecting(false);
    setVideoLoaded(false);
    if (stream) {
        stream.getTracks().forEach(track => track.stop());
        setStream(null);
    }
    const ctx = canvasRef.current.getContext('2d');
    if (ctx) {
        ctx.clearRect(0, 0, canvasRef.current.width, canvasRef.current.height);
    }
};
```



```
return (
  <div className='m-auto'>
    <div className="text-center">
      <div className='btn-group'>
        {stream ? (
          <button
            className='btn btn-lg px-10 btn-danger'
            onClick={handleStop}>
            Stop
          </button>
        ) : (
          <button
            className='btn btn-lg px-10 btn-success'
            onClick={handlePlay}>
            Play
          </button>
        )}
      </div>
    </div>
    <div className='m-auto'
      style={{ position: 'relative', width: '640px', height: '480px' }}>
      <video
        ref={videoRef} width="640" height="480"
        style={{ position: 'absolute', top: 0, left: 0 }} />
      <canvas
        ref={canvasRef} width="640" height="480"
        style={{ position: 'absolute', top: 0, left: 0 }} />
    </div>
  </div>
);
}
```

Penjelasan:

```
async function loadModel() {
  modelRef.current = await blazeface.load();
}
```

Fungsi ini adalah fungsi asinkron yang mendefinisikan dan memuat model Blazeface. blazeface.load() adalah panggilan ke pustaka Blazeface untuk memuat model deteksi wajah. modelRef adalah referensi ke model yang dimuat dan disimpan dalam modelRef.current. modelRef biasanya dibuat menggunakan useRef hook:

```
const modelRef = useRef(null);
```

**Blazeface** adalah model machine learning yang dikembangkan oleh Google untuk deteksi wajah yang sangat cepat dan efisien. Pustaka ini sering digunakan dalam aplikasi web dan mobile untuk mengidentifikasi dan melacak wajah dalam gambar atau video secara real-time.

```
if (videoLoaded && isDetecting)
```

Kondisi ini mengecek apakah video sudah dimuat (videoLoaded) dan deteksi sedang berjalan (isDetecting). Jika keduanya benar, maka fungsi detectFaces akan didefinisikan.

```
const detectFaces = async () => {...}
```



Fungsi asinkron ini bertugas untuk mendeteksi wajah dalam video. Pertama, ia mengecek apakah referensi untuk elemen video (videoRef), canvas (canvasRef), dan model (modelRef) tersedia. Jika salah satu tidak tersedia, fungsi akan keluar (return).

```
const predictions = await modelRef.current.estimateFaces(videoRef.current, false);
```

Panggilan ke estimateFaces pada model yang dimuat (modelRef.current) untuk mendeteksi wajah di dalam elemen video (videoRef.current). Hasilnya adalah array predictions yang berisi informasi tentang wajah yang terdeteksi.

```
const ctx = canvasRef.current.getContext('2d');
if (!ctx) { return; }
ctx.clearRect(0, 0, canvasRef.current.width, canvasRef.current.height);
predictions.forEach(prediction => {
  const start = prediction.topLeft;
  const end = prediction.bottomRight;
  const size = [end[0] - start[0], end[1] - start[1]];
  ctx.beginPath();
  ctx.strokeStyle = "red";
  ctx.lineWidth = 2;
  ctx.rect(start[0], start[1], size[0], size[1]);
  ctx.stroke();
});
```

1. Mendapatkan Konteks 2D dari Canvas:

Mendapatkan konteks 2D (ctx) dari elemen canvas. Jika gagal (misalnya, jika canvas tidak tersedia), fungsi akan keluar.

2. Membersihkan Canvas:

- ctx.clearRect membersihkan seluruh area canvas.

3. Menggambar Kotak di Sekitar Wajah:

Untuk setiap prediksi, menggambar kotak di sekitar wajah yang terdeteksi. prediction.topLeft dan prediction.bottomRight memberikan koordinat untuk sisi kiri atas dan kanan bawah dari kotak wajah.

- ctx.beginPath() memulai jalur baru.
- ctx.strokeStyle menentukan warna garis.
- ctx.lineWidth menentukan ketebalan garis.
- ctx.rect menggambar kotak di sekitar wajah.
- ctx.stroke() menggambar garis pada jalur yang telah ditentukan.

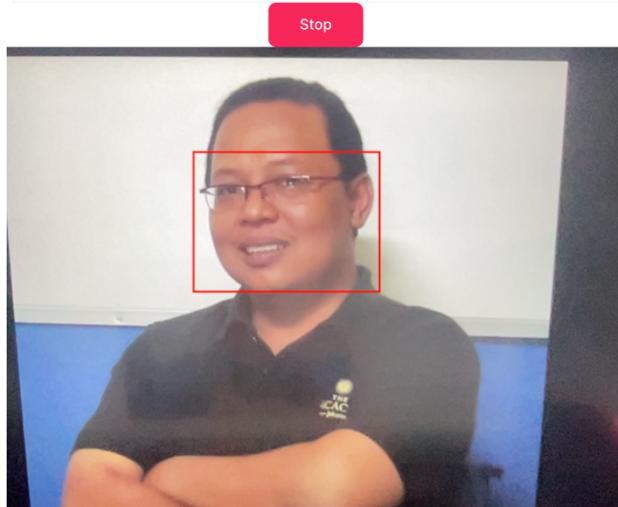
```
if (isDetecting) { requestAnimationFrame(detectFaces); }
```

Jika deteksi masih berjalan (isDetecting), maka requestAnimationFrame akan dipanggil untuk memanggil kembali fungsi detectFaces di frame berikutnya, menciptakan loop yang berjalan terus-menerus selama kondisi isDetecting benar.

```
<video ref={videoRef} width="640" height="480" />
<canvas ref={canvasRef} width="640" height="480" />
```

Komponen ini memuat model Blazeface saat pertama kali dirender. Video dan canvas elemen di-render pada halaman. Ketika tombol "Start" diklik, proses deteksi wajah dimulai dan berjalan terus-menerus hingga

tombol "Stop" diklik. Hasil deteksi wajah digambar di canvas dengan kotak merah di sekitar wajah yang terdeteksi. Maka hasil output dari code yang telah anda buat, akan memiliki tampilan sebagai berikut:



Jika anda ingin membuat sebuah face landmark pada citra wajah tersebut atau ingin membuat titik-titik khusus pada wajah yang secara otomatis dideteksi oleh algoritma computer vision. Titik-titik ini biasanya mewakili fitur-fitur wajah seperti sudut mata, ujung hidung, sudut bibir, dan tepi rahang. Deteksi face landmarks merupakan langkah penting dalam berbagai aplikasi pengenalan wajah, analisis ekspresi wajah, dan aplikasi augmented reality. Library tensorflow yang dapat anda gunakan ialah:

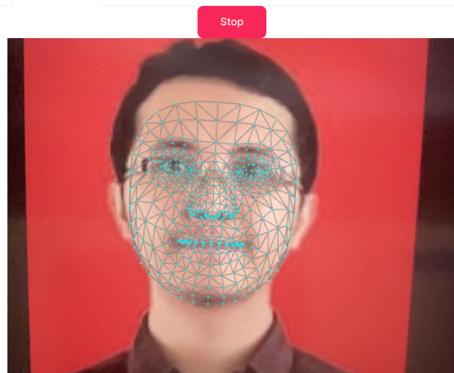
```
npm install @tensorflow-models/face-landmarks-detection
```

sedangkan data model yang dapat anda gunakan ialah:

```
const loadModel = async () => {
  const facemodel = await facemesh.load(
    facemesh.SupportedPackages.mediapipeFacemesh
  );
  setNet(facemodel);
};
```

- facemesh.load: merupakan metode dari library TensorFlow.js untuk memuat model FaceMesh.
- facemesh.SupportedPackages.mediapipeFacemesh: Ini menunjukkan paket model FaceMesh yang didukung. Mediapipe adalah salah satu penyedia model FaceMesh yang populer.
- await: Digunakan untuk menunggu sampai model selesai dimuat sebelum melanjutkan ke baris kode berikutnya.
- setNet: Ini adalah fungsi yang digunakan untuk memperbarui state net dalam komponen React.
- facemodel: Model FaceMesh yang telah dimuat disimpan dalam state untuk digunakan nanti dalam aplikasi.

Jika anda merubah settingan pengaturan data model anda pada code funtion loadModel, maka output yang anda dapatkan seperti dibawah ini:



### B. Pengenalan terhadap benda

jika pada materi diatas mengenai pendektsian sebuah wajah, pada tahap ini kita akan mengenali sebuah objek pada citra. Untuk memuat model deteksi objek dari library TensorFlow.js kita akan menggunakan Coco SSD (Single Shot MultiBox Detector).

```
npm install @tensorflow-models/coco-ssd
```

Dari segi code, sebenarnya tidaklah jauh berbeda penerapannya dengan mendekksi wajah, yang berbeda hanyalah pemanggilan data model yang digunakan.

```
const loadModel = async () => {
  const objectModel = await cocossd.load();
  setNet(objectModel);
};
```

Saat aplikasi dijalankan, fungsi loadModel akan dipanggil untuk memuat model Coco SSD. Setelah model selesai dimuat, model tersebut akan disimpan dalam state, memungkinkan komponen lain dalam aplikasi untuk mengakses dan menggunakan model tersebut. Model Coco SSD yang telah dimuat dapat digunakan untuk mendekksi objek dalam gambar atau video. Model ini berguna untuk berbagai aplikasi seperti keamanan, analisis video, augmented reality (AR), dan banyak lagi.

Berikut adalah contoh sederhana bagaimana model Coco SSD bisa diintegrasikan dalam komponen React:

```
import React, { useEffect, useRef, useState } from "react";
import * as tf from "@tensorflow/tfjs";
import * as cocossd from "@tensorflow-models/coco-ssd";
import Webcam from "react-webcam";
import { drawRect } from "./utilities";

export function ObjectDetections() {
  const webcamRef = useRef(null);
  const canvasRef = useRef(null);
  const [isDetecting, setIsDetecting] = useState(false);
  const [isCameraOn, setIsCameraOn] = useState(false);
  const [net, setNet] = useState(null);

  const loadModel = async () => {
    const objectModel = await cocossd.load();
    setNet(objectModel);
  };
}
```



```
const detect = async () => {
  if (
    typeof webcamRef.current !== "undefined" &&
    webcamRef.current !== null &&
    webcamRef.current.video.readyState === 4
  ) {
    const video = webcamRef.current.video;
    const videoWidth = webcamRef.current.video.videoWidth;
    const videoHeight = webcamRef.current.video.videoHeight;

    webcamRef.current.video.width = videoWidth;
    webcamRef.current.video.height = videoHeight;

    canvasRef.current.width = videoWidth;
    canvasRef.current.height = videoHeight;

    const obj = await net.detect(video);

    const ctx = canvasRef.current.getContext("2d");
    drawRect(obj, ctx);
  }
};

useEffect(() => {
  loadModel();
}, []);

useEffect(() => {
  let intervalId;
  if (isDetecting && net && isCameraOn) {
    intervalId = setInterval(() => {
      detect();
    }, 10);
  }
  return () => clearInterval(intervalId);
}, [isDetecting, net, isCameraOn]);

const handlePlay = () => {
  setIsDetecting(true);
  setIsCameraOn(true);
};

const handleStop = () => {
  setIsDetecting(false);
  setIsCameraOn(false);
};

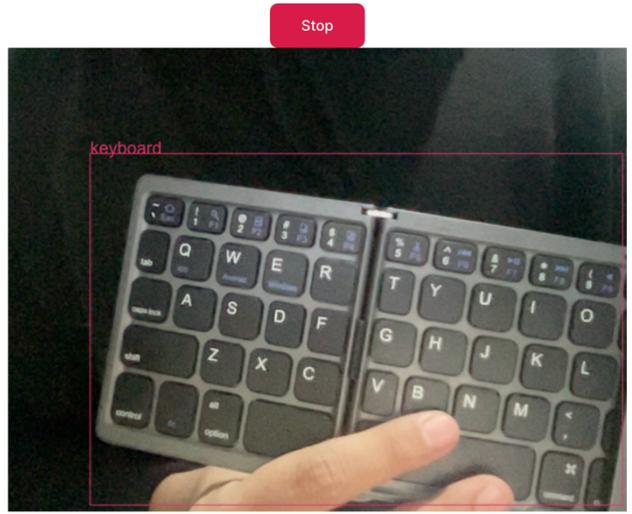
return (
  <div className="m-auto">
```



```
<div className="text-center">
  <div className="btn-group">
    {isDetecting ? (
      <button
        className="btn btn-lg px-10 btn-danger"
        onClick={handleStop}
      >
        Stop
      </button>
    ) : (
      <button
        className="btn btn-lg px-10 btn-success"
        onClick={handlePlay}
      >
        Play
      </button>
    )}
  </div>
</div>
<div
  className="m-auto"
  style={{ position: "relative", width: "640px", height: "480px" }}
>
  {isCameraOn && (
    <Webcam
      ref={webcamRef}
      style={{
        position: "absolute",
        marginLeft: "auto",
        marginRight: "auto",
        left: 0,
        right: 0,
        textAlign: "center",
        zIndex: 9,
        width: 640,
        height: 480,
      }}
    /> )
  <canvas
    ref={canvasRef}
    style={{
      position: "absolute",
      marginLeft: "auto",
      marginRight: "auto",
      left: 0,
      right: 0,
      textAlign: "center",
      zIndex: 9,
      width: 640,
      height: 480,
    }}
  </canvas>
</div>
```

```
    }  
    />  
  </div>  
  </div>  
);  
}
```

Maka bentuk output yang akan didapatkan dari code diatas seperti dibawah ini:



### 3. Tugas Pelatihan

Silakan anda lanjutkan soal Latihan diatas dengan menambahkan contoh kasus dibawah ini:

- 1) Pada halaman sign in sebelumnya hanya menggunakan sistem authentifikasi secara manual dengan memasukan username dan password. Dapatkah anda membuat sebuah sistem sign in dengan memanfaatkan computer vision untuk mendeteksi adanya wajah atau mengenali adanya objek "PERSON"?

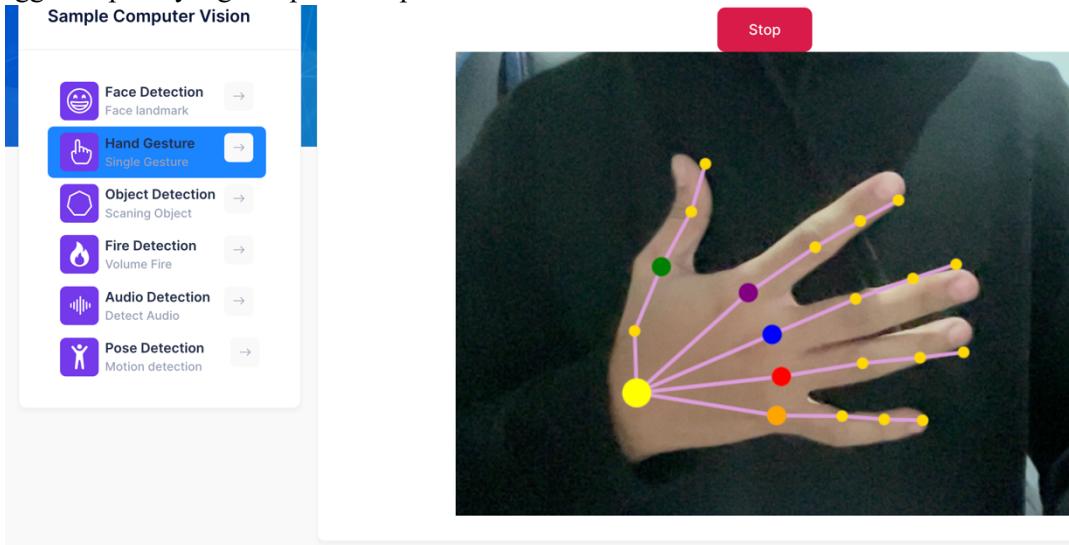


- 2) Dapatkah anda membuat atau mendeteksi adanya tangan pada citra dengan memanfaatkan library tensorflow @tensorflow-models/handpose dimana anda hanya menggunakan model data seperti berikut:



```
const loadModel = async () => {
    const handposeNet = await handpose.load();
    setNet(handposeNet);
};
```

Sehingga tampilan yang didapatkan seperti berikut:



Pengumpulan tugas Latihan praktikum dikumpulkan kedalam GITHUB masing-masing mahasiswa berdasarkan repository yang telah dibuat PWL-TI-21-PA-NPM. File source code disimpan sesuai nama project-praktikum dan masukan kedalam repositori tersebut. Buatkanlah file dokumen dalam bentuk file pdf yang berisi Screen Capture dari hasil program yang telah dikerjakan. Simpan dalam file PDF tersebut kedalam project tersebut.

Tambahkan Collaborator management access pada repository ke [@IrwanRizkyAriansyah](#) dan [@FebryFairuz](#)