UNIVERSITATEA “POLITEHNICA” DIN BUCUREȘTI
FACULTATEA DE ELECTRONICĂ, TELECOMUNICAȚII ȘI TEHNOLOGIA INFORMAȚIEI

Proiect Tehnologii de Programare în Internet

Facial Recognition in JavaScript using TensorFlow.js

Student:Petrescu Georgian Cristian

Grupa: 431A

Cuprins

I.Introducere

În ultimii ani, recunoașterea facială a câștigat o atenție semnificativă și a fost apreciată ca una dintre cele mai promițătoare aplicații în domeniul analizei imaginilor. Detectarea fețelor poate reprezenta o parte semnificativă a operațiilor de recunoaștere facială, datorită capacității sale de a concentra resursele computaționale asupra secțiunii unei imagini care conține o față. Metoda de detectare a fețelor în imagini este complexă din cauza variabilității existente în cadrul fețelor umane, cum ar fi poziția, expresia, orientarea, culoarea pielii, prezența ochelarilor sau a părului facial, diferențele în sensibilitatea camerei, condițiile de iluminare și rezoluția imaginii.

Detectarea obiectelor este una dintre tehnologiile calculatoarelor, care este legată de prelucrarea imaginilor și de viziunea computerizată și interacționează cu detectarea instanțelor unui obiect, cum ar fi fețele umane, clădiri, copaci, mașini, etc. Scopul principal al algoritmilor de detectare a fețelor este de a determina dacă există sau nu o față într-o imagine.

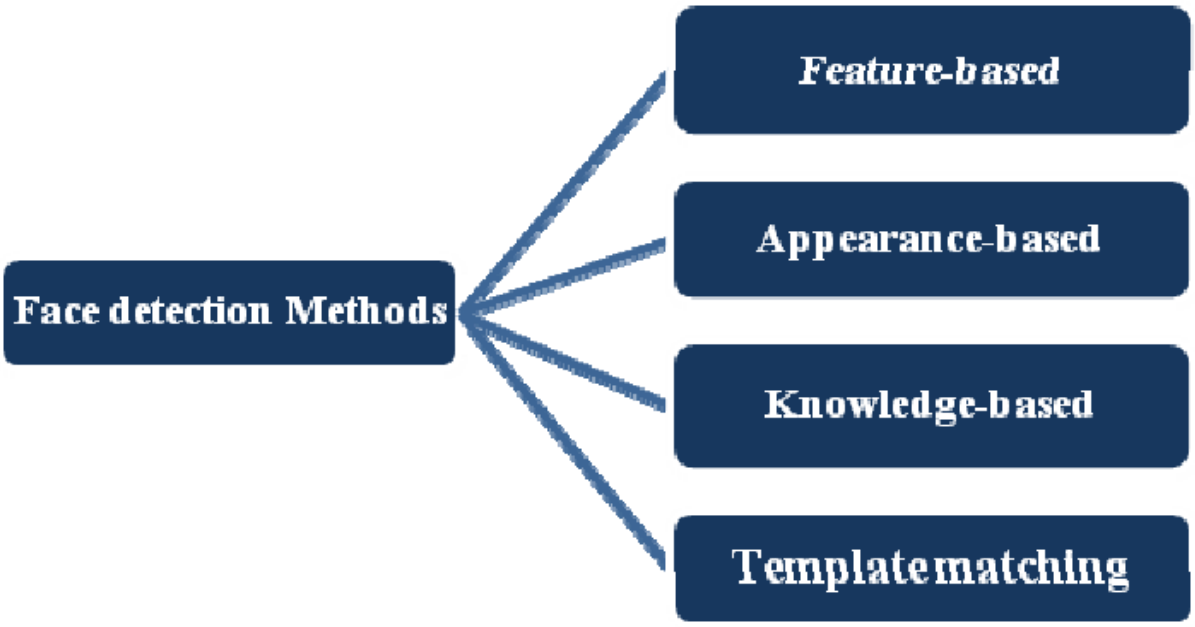
În ultima perioadă, s-au propus multe lucrări de studiu în domeniul recunoașterii feței și detectării feței pentru a o face mai avansată și mai precisă, dar a făcut o revoluție în acest domeniu când Viola-Jones a venit cu detectorul său de față în timp real, care este capabil să detecteze fețele în timp real cu o precizie mare.

Detectarea feței este primul și cel mai important pas pentru recunoașterea feței și este utilizată pentru detectarea fețelor în imaginile. Este o parte a detectării obiectelor și poate fi utilizată în multe domenii, cum ar fi securitatea, biometria, aplicarea legii, divertismentul, siguranța personală, etc. Este folosit pentru a detecta fețe în timp real în scopul supravegherii și urmăririi persoanelor sau obiectelor. Este utilizat pe scară largă în camerele foto pentru a identifica apariții multiple în cadru, cum ar fi camerele mobile și camerele DSLR. Facebook utilizează, de asemenea, algoritmul de detectare a feței pentru a detecta fețele din imaginile și a le recunoaște.

În principiu, aplicațiile de detectare a fețelor utilizează tehnici și algoritmi de învățare automată pentru a detecta fețele umane în imagini mai mari. Aceste imagini mai mari pot conține numeroase obiecte care nu sunt fețe, cum ar fi peisaje, clădiri și alte părți ale corpului uman (de exemplu, picioare, umeri și brațe).

II.METODE ALE RECUNOAȘTERII FACIALE

Yan, Kriegman și Ahuja(oameni de știință în domeniul computer vision și în special al detectării feței) au prezentat o clasificare a metodelor de detectare a feței. Aceste metode sunt împărțite în patru categorii, iar algoritmii de detectare a feței pot aparține la două sau mai multe grupuri. Aceste categorii sunt următoarele:



1. Metoda bazată pe cunoștințe:

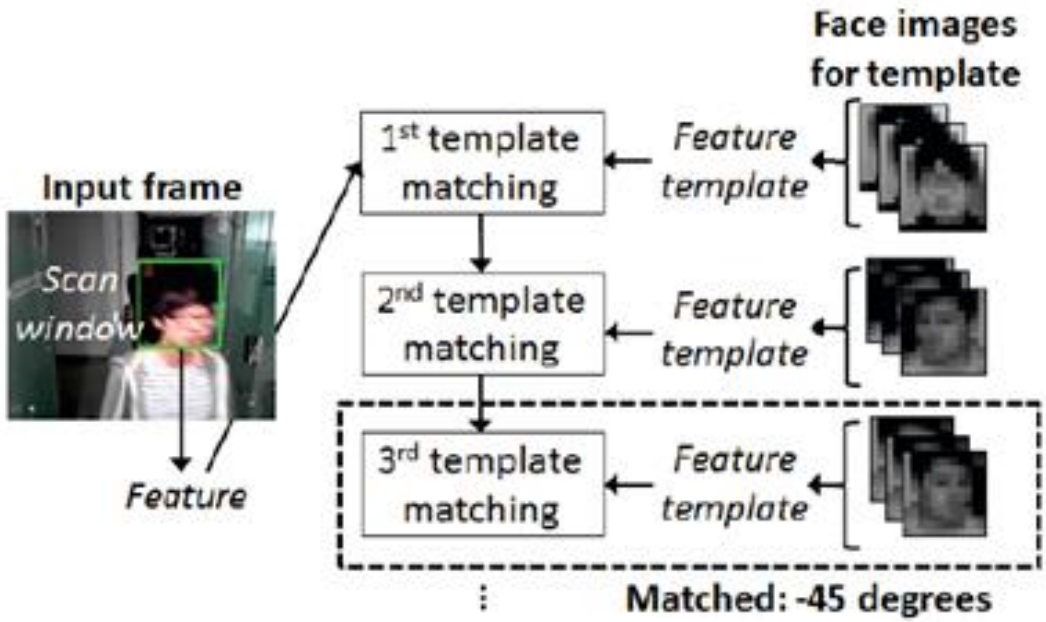
Metoda bazată pe cunoștințe se bazează pe un set de reguli și pe cunoștințele umane pentru a detecta fețele. De exemplu, o față trebuie să aibă un nas, ochi și gură la anumite distanțe și poziții unul față de celălalt. Problema majoră a acestor metode constă în dificultatea de a construi un set adecvat de reguli. Pot exista multe rezultate pozitive false dacă regulile sunt prea generale sau prea detaliate. Această abordare singură este insuficientă și nu poate găsi multe fețe în imagini multiple.

2. Metoda bazată pe caracteristici:

Metoda bazată pe caracteristici constă în localizarea fețelor prin extragerea caracteristicilor structurale ale feței. Aceasta este antrenată mai întâi ca un clasificator și apoi utilizată pentru a face distincția între regiunile faciale și cele non-faciale. Ideea este de a depăși limitele cunoașterii noastre instinctive despre fețe. Această abordare este împărțită în mai multe etape și chiar și în fotografii cu mai multe fețe, raportează o rată de succes de 94%.

3. Metoda potrivirii șabloanelor:

Metoda potrivirii șabloanelor utilizează șabloane de față predefinite sau parametrizate pentru a localiza sau detecta fețele prin corelație între șabloane și imaginile de intrare. De exemplu, o față umană poate fi împărțită în ochi, contur facial, nas și gură. De asemenea, un model de față poate fi construit prin utilizarea muchiilor prin metoda de detectare a marginilor. Această abordare este simplă de implementat, dar este inadecvată pentru detectarea feței. Cu toate acestea, au fost propuse șabloane deformabile pentru a face față acestor probleme.



Metoda potrivirii șabloanelor

4. Metoda bazată pe aspect:

Metoda bazată pe aspect se bazează pe un set de imagini de antrenament delegate pentru a descoperi modelele feței. Abordarea bazată pe aspect este mai bună decât alte modalități în ceea ce privește performanța. În general, metoda bazată pe aspect se bazează pe tehnici de analiză statistică și învățare automată pentru a găsi caracteristicile relevante ale imaginilor feței. Această metodă este, de asemenea, folosită pentru extragerea caracteristicilor în recunoașterea feței.

Modelul bazat pe aspect este împărțit în sub-metode pentru utilizarea detectării feței, care sunt următoarele:

4.1. Bazat pe Eigenface:

Algoritmul bazat pe Eigenface este utilizat pentru recunoașterea feței și este o metodă de reprezentare eficientă a fețelor folosind analiza componentelor principale.

4.2. Bazat pe distribuție:

Algoritmi precum PCA (Principal Component Analysis) și Fisher's Discriminant pot fi utilizați pentru a defini subspațiul care reprezintă modelele faciale. Există un clasificator antrenat, care identifică corect instanțele clasei de modele țintă din modelele de fundal ale imaginilor.

4.3. Rețele neurale:

Mulți algoritmi de detectare precum detectarea obiectelor, detectarea feței, detectarea emoțiilor și recunoașterea feței, etc., au fost abordate cu succes folosind rețele neurale.

4.4. Mașină cu vectori suport:

Mașinile cu vectori suport sunt clasificatoare lineare care maximizează marginea între hiperplanul de decizie și exemplele din setul de antrenament. Osuna et al. au aplicat pentru prima dată acest clasificator la detectarea feței.

4.5. Rețea rară de winnows:

A fost definită o rețea rară formată din două unități liniare sau noduri țintă; una reprezintă modelele feței, iar cealaltă modelele non-feței. Este o metodă eficientă și care necesită mai puțin timp.

4.6. Clasificatori Naive Bayes:

Aceștia calculează probabilitatea ca o față să fie prezentă în imagine prin numărarea frecvenței de apariție a unei serii de modele în imaginile de antrenament. Clasificatorul capturează statistica comună a aspectului local și poziției fețelor.

4.7. Hidden Model Markov :

Stările modelului ar fi caracteristicile faciale, care sunt de obicei descrise ca benzi de pixeli. Modelele Markov ascunse sunt utilizate în mod obișnuit împreună cu alte metode pentru a construi algoritmi de detectare.

4.8. Abordare teoretică a informațiilor:

Câmpurile aleatorii Markov (MRF) pot fi utilizate pentru modelele faciale și caracteristicile corelate. Procesul Markov maximizează discriminarea între clase folosind divergența Kullback-Leibler. Prin urmare, această metodă poate fi folosită în detectarea feței.

4.9. Învățare inductivă:

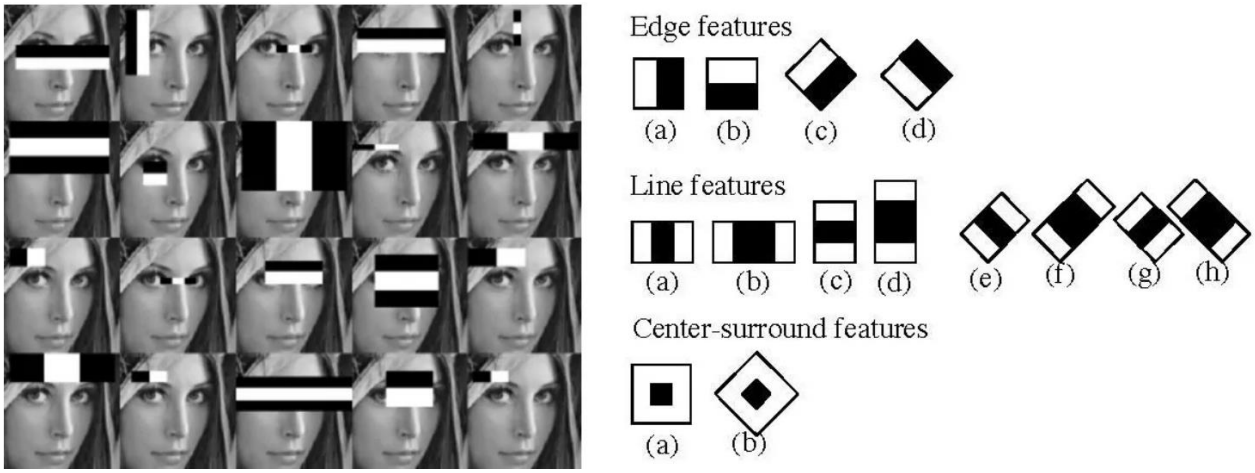
Această abordare a fost utilizată pentru detectarea feței. Algoritmi precum C4.5 al lui Quinlan sau FIND-S al lui Mitchell sunt utilizați în acest scop.

III.CUM FUNCȚIONEAZĂ RECUNOAȘTEREA FACIALĂ

Există câțiva pași prin care funcționează detectarea feței, care sunt următorii:

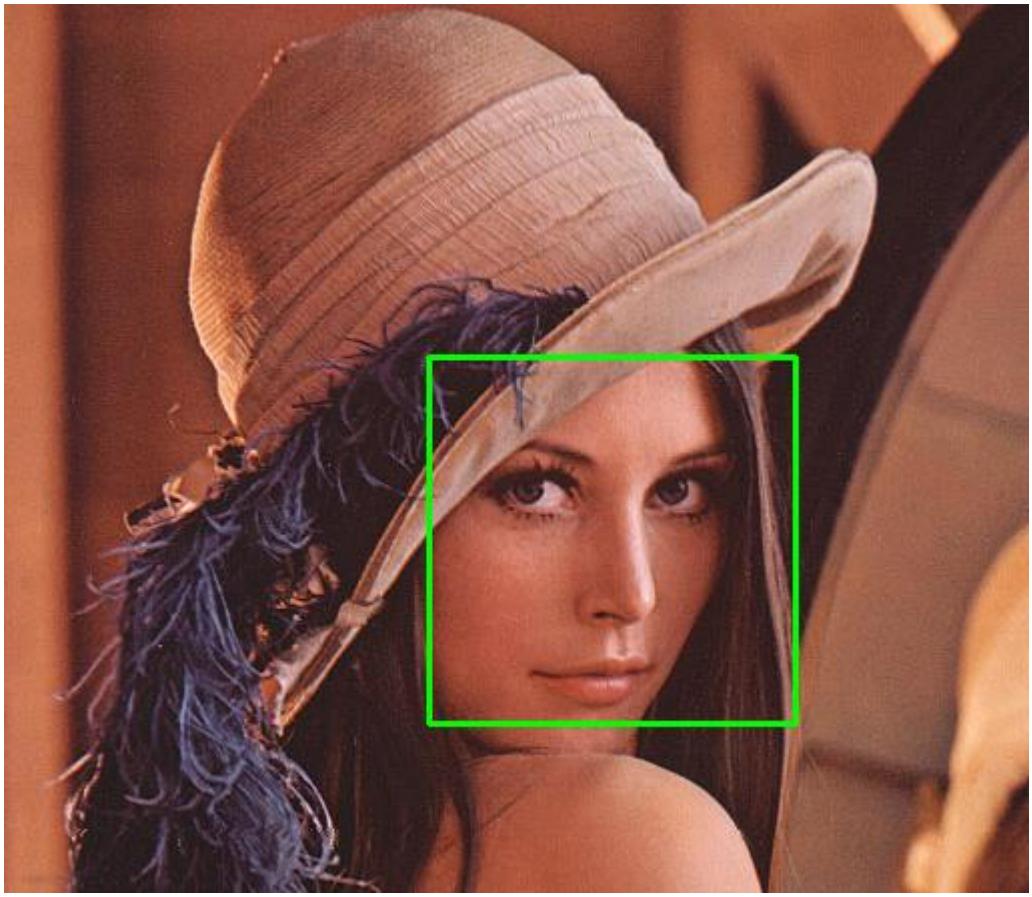
În primul rând, se obține imaginea și se convertește din RGB (Roșu-Verde-Albastru) în alb-negru (grayscale), deoarece este mai ușor să se detecteze obiectele într-o compatibilitate alb-negru.

Următorul pas constă în utilizarea algoritmului Haar-Like features, pe care framework-ul de detectare a obiectelor Viola-Jones îl folosește pentru detectarea fețelor. Acest algoritm este utilizat pentru a găsi locația fețelor umane într-un cadru sau imagine. Toate fețele umane au anumite proprietăți universale. Regiunea ochilor este mai întunecată decât vecinii săi, iar regiunea nasului este mai luminoasă decât regiunea ochilor. De asemenea, acest algoritm este folosit pentru selecția caracteristicilor pentru un obiect într-o imagine. Detectarea marginilor, detectarea liniilor și detectarea centrului sunt utilizate pentru detectarea ochilor, nasului, gurii etc. în imagine. El este folosit pentru a selecta caracteristicile esențiale dintr-o imagine și pentru a extrage aceste caracteristici în scopul detectării feței.



Caracteristicile Haar-like reprezentate pentru detectarea feței

Următorul pas constă în furnizarea coordonatelor x, y, w și h, care formează un dreptunghi în imagine pentru a indica locația feței sau, putem spune, pentru a arăta regiunea de interes în imagine. După aceasta, se poate crea un dreptunghi oriunde se detectează fața.



Recunoaștere facială pe o imagine

IV.TENSORFLOW

TensorFlow este o platformă open-source pentru mașinărie învățată dezvoltată de Google. A fost creată pentru a facilita construirea, antrenarea și desfășurarea modelelor de învățare automată, în special a rețelelor neuronale profunde. TensorFlow utilizează o abordare bazată pe grafuri, unde operațiile matematice sunt reprezentate ca și noduri într-un graf.

Iată câteva aspecte cheie despre TensorFlow:

1. Model de programare flexibil: TensorFlow oferă o suită largă de API-uri care permit dezvoltatorilor să construiască și să antreneze modele de învățare automată. Acestea includ TensorFlow Core, care este API-ul de bază, dar și API-uri specializate pentru diferite aplicații, cum ar fi TensorFlow.js pentru dezvoltarea în browser și TensorFlow Lite pentru dispozitive mobile.
2. Rețele neuronale profunde: TensorFlow este puternic orientat către rețele neuronale profunde. Puteți construi și antrena modele complexe, cum ar fi rețele neuronale convoluționale (CNN) pentru viziune artificială, rețele neuronale recurente (RNN) pentru prelucrarea limbajului natural și multe altele.

3. Eficiență și scalabilitate: TensorFlow este proiectat pentru a profita de puterea calculatoarelor moderne și a resurselor distribuite. Puteți utiliza TensorFlow pentru a rula antrenamente intensive pe seturi de date mari, pe mai multe mașini sau în cluster.
4. Suport pentru multiple limbaje de programare: TensorFlow oferă suport pentru mai multe limbaje de programare, inclusiv Python, C++, Java și JavaScript. Python este cel mai utilizat limbaj pentru dezvoltarea cu TensorFlow, datorită comunității puternice și a bibliotecilor de susținere.

TensorFlow poate fi folosit într-o varietate de domenii și aplicații, cum ar fi:

1. Viziune artificială: TensorFlow este adesea folosit pentru construirea sistemelor de recunoaștere a imaginilor și detecție a obiectelor în imagini. Acesta poate fi utilizat pentru clasificarea imaginilor, segmentarea semantică și multe alte sarcini.
2. Prelucrarea limbajului natural: TensorFlow este utilizat pe scară largă în dezvoltarea de modele pentru traducerea automată, recunoașterea vorbirii, generarea de texte și multe alte sarcini de prelucrare a limbajului natural.
3. Învățare automată în general: TensorFlow este potrivit pentru dezvoltarea și antrenarea de modele de învățare automată în diverse domenii, cum ar fi recomandări personalizate, analiză a sentimentelor, procesare de semnale și multe altele.
4. Robotică și Internetul Lucrurilor (IoT): TensorFlow poate fi utilizat pentru a dezvolta aplicații de învățare automată în domenii precum robotică, casă inteligentă sau vehicule autonome.

Acestea sunt doar câteva exemple ale utilizării TensorFlow. Datorită flexibilității și puterii sale, TensorFlow poate fi aplicat într-o gamă largă de proiecte care implică învățarea automată.

Câteva exemple concrete unde a fost folosit TensorFlow:

- Echipa de inginerie și știință a datelor de la Airbnb aplică învățarea automată folosind TensorFlow pentru a clasifica imaginile și a detecta obiecte la scară largă, contribuind la îmbunătățirea experienței oaspeților.

- Progresele în inteligența artificială și maturizarea platformei TensorFlow au permis companiei Coca Cola să obțină o funcționalitate mult dorită de achiziționare fără efort pentru programul lor de loialitate.

- Google utilizează TensorFlow pentru a alimenta implementările de învățare automată în produse precum Căutare, Gmail și Translate, pentru a ajuta cercetătorii în descoperiri noi, și chiar pentru a realiza progrese în provocările umanitare și de mediu.

- Platforma Lenovo LiCO accelerează antrenamentul AI și computația de performanță înaltă tradițională, și optimizează antrenamentul învățării profunde prin integrarea și optimizarea TensorFlow. LiCO furnizează diverse modele TensorFLow încorporate și suportă antrenamentul distribuit optimizat al acestor modele.

V.TENSORFLOW.JS

TensorFlow.js este o bibliotecă JavaScript care permite dezvoltatorilor să ruleze modele de învățare automată în browserul web sau în medii JavaScript.Acesta este o ramură a framework-ului de învățare automată TensorFlow, care a fost creat de Google și lansat inițial în anul 2015. TensorFlow.js a fost lansat în 2018 și a fost dezvoltat pentru a permite rularea de modele de învățare automată în mediul browser-ului web. Acest lucru a deschis uși noi pentru dezvoltarea de aplicații de învățare automată care pot rula direct pe dispozitivele utilizatorilor, fără a necesita o conexiune la server.

De-a lungul timpului, TensorFlow.js a fost îmbunătățit și actualizat pentru a adăuga noi funcționalități și a îmbunătăți performanța algoritmilor de învățare automată în mediul browser-ului. Acesta a devenit o soluție populară pentru dezvoltatorii web care doresc să integreze funcționalități de învățare automată, inclusiv recunoașterea facială, în aplicațiile lor.

Facial recognition cu TensorFlow.js oferă funcționalități avansate pentru identificarea și verificarea persoanelor pe baza caracteristicilor faciale. Aceasta poate detecta și extrage automat trăsăturile distinctive ale feței umane, cum ar fi ochii, nasul, gura și conturul feței. Utilizând aceste caracteristici, algoritmul poate compara fețele detectate cu un set de fețe de referință și determinați dacă există o potrivire.

Acest produs software funcționează pe suport hardware și software standard. Pentru a rula aplicații TensorFlow.js, este necesar un browser web modern, cum ar fi Google Chrome, Mozilla Firefox sau Microsoft Edge, care suportă tehnologiile web relevante, inclusiv WebGL și WebAssembly. Pentru dispozitivele mobile, TensorFlow.js poate fi rulat în browser-ul mobil sau în cadrul unei aplicații hibride, utilizând framework-uri precum React Native sau Ionic

VI.CREAREA UNUI SITE WEB CU RECUNOAȘTERE FACIALĂ FOLOSIND TENSORFLOW.JS

Această rezumă modul de funcționare, iar acum este momentul să creăm propria versiune a aplicației de detectare a feței cu o cameră web în timp real în browser.

Furnizarea accesului la camera web

În primul rând, trebuie să oferim acces la camera web în browser. Vom începe prin crearea unui fișier index.html de bază, unde accesul la camera web va fi oferit cu ajutorul etichetei video.

```html
<!DOCTYPE html>
<html lang="en">
  <head>
    <meta charset="UTF-8" />
    <meta name="viewport" content="width=device-width, initial-scale=1.0" />
    <title>Facial Detection App</title>
  </head>
  <body>
    <h1>Facial Detection</h1>
    <video id="video" autoplay></video>
  </body>

  <script src="script.js"></script>
</html>
```

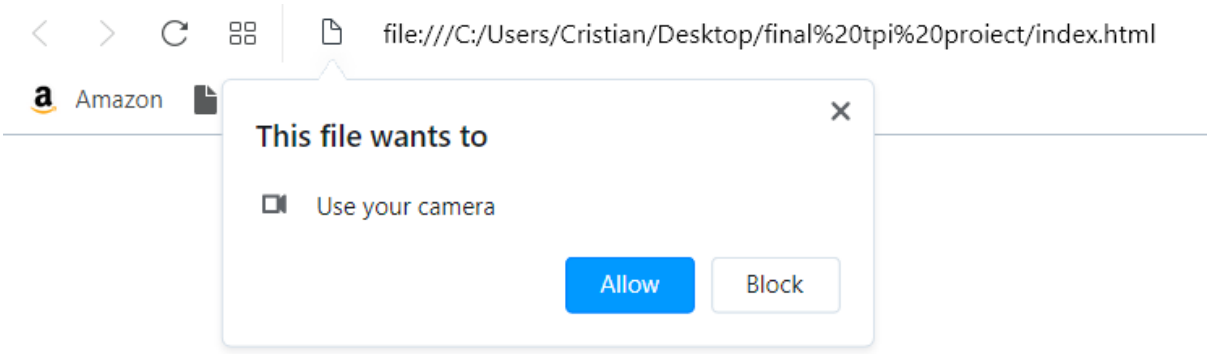
Acum, să creăm un alt fișier numit script.js, unde vom scrie codul JavaScript și îl vom include în index.html.

```javascript
let video = document.getElementById("video");

const accessCamera = () => {
  navigator.mediaDevices
    .getUserMedia({
      audio: false,
      video: { width: 500, height: 400 },
    })
    .then((stream) => {
      video.srcObject = stream;
    });
};

accessCamera();
```

Aici accesăm elementul etichetei video și îl stocăm în variabila "video". Funcția "accessCamera" apelează API-ul "getUserMedia", iar noi specificăm dimensiunile video (am luat 500 px lățime și 400 px înălțime)

Funcția "getUserMedia" returnează o promisiune cu obiectul MediaStream, pe care îl atribuim proprietății "srcObject" a elementului video. În final, apelăm funcția "accessCamera".

Pentru a rula acest cod, se poate folosi fie extensia "Live Server", fie deschide direct fișierul "index.html". Browser-ul va solicita permisiunea de a accesa camera, permiteți accesul și ar trebui să apară fluxul camerei web pe ecran.



Executarea detectării feței

Pentru aceasta, vom folosi modelul Blazeface din cadrul pachetului Simple Face Detection în tensorflow.js. Blazeface este un model ușor folosit pentru detectarea fețelor în imagini.

Acum, vom prelua următoarele etichete script din modelul Blazeface și le vom adăuga în fișierul nostru index.html.

```
<script src="https://cdn.jsdelivr.net/npm/@tensorflow/tfjs"></script>
<script src="https://cdn.jsdelivr.net/npm/@tensorflow-models/blazeface"></script>
```

Ne întoarcem în fișierul script.js, unde vom crea o variabilă numită "model" care va încărca modelul Blazeface în interiorul său.

```
let model;
model = await blazeface.load();
```

În primul rând, este declarată o variabilă numită "model" folosind cuvântul cheie "let". Aceasta lasă loc pentru a atribui o valoare variabilei mai târziu.

Apoi, este utilizat cuvântul cheie "await" pentru a aștepta încărcarea modelului Blazeface. În JavaScript, când se folosește "await" într-o funcție marcată ca "async", aceasta așteaptă finalizarea unei promisiuni. În acest caz, promisiunea este de a încărca modelul Blazeface.

Funcția "blazeface.load()" face parte din biblioteca Blazeface și este responsabilă pentru încărcarea modelului de recunoaștere a feței. După ce modelul este încărcat, acesta este atribuit variabilei "model".

Acum , vom crea o funcție numită "detectFaces" care va efectua detectarea facială și ne va oferi rezultatele necesare.

```
const detectFaces = async () => {
  const prediction = await model.estimateFaces(video, false);
```

Deoarece funcția "blazeface.load()" este o funcție asincronă, durează ceva timp pentru a se încărca. Însă în același timp, funcția "detectFaces" este apelată. Pentru a evita aceasta, vom aștepta mai întâi încărcarea fluxului video de la camera web. Odată ce acest lucru este complet, vom aștepta încărcarea modelului și apoi vom apela funcția "detectFaces". Pentru a detecta încărcarea fluxului video, vom folosi un ascultător de evenimente numit "loadeddata" în acest scop.

Acum, script.js va arata asa:

```
let video = document.getElementById("video");
let model;

const accessCamera = () => {
  navigator.mediaDevices
    .getUserMedia({
      video: { width: 500, height: 400 },
      audio: false,
    })
    .then((stream) => {
      video.srcObject = stream;
    });
};

const detectFaces = async () => {
  const prediction = await model.estimateFaces(video, false);
};

accessCamera();

// this event will be  executed when the video is loaded

video.addEventListener("loadeddata", async () => {
  model = await blazeface.load();
  detectFaces();
});
```

Prezicerea va enumera obiectele în care fiecare obiect corespunde unei fețe detectate. În fiecare obiect, vom găsi coordonatele colțului stânga sus și colțului dreapta jos, precum și cele șase puncte caracteristice ale feței corespunzătoare ochilor, urechilor, nasului și gurii.

Acum, vom utiliza aceste coordonate pentru a afișa rezultatele într-o casetă rectangulară. Mai întâi, trebuie să parcurgem lista de predicții și să desenăm un dreptunghi sau puncte pentru punctele caracteristice ale feței. Vom folosi elementul canvas HTML în acest scop și vom ajusta lățimea și înălțimea pentru a fi la fel cu fluxul video. Deoarece nu mai avem nevoie să afișăm conținutul etichetei video, vom seta proprietatea sa "display" la "none".

Iată fișierul nostru final index.html:

```html
<!DOCTYPE html>
<html lang="en">
  <head>
    <meta charset="UTF-8" />
    <meta http-equiv="X-UA-Compatible" content="IE=edge" />
    <meta name="viewport" content="width=device-width, initial-scale=1.0" />
    <title>Facial Detection App</title>
    <style>
      body {
        display: flex;
        flex-direction: column;
        align-items: center;
        justify-content: center;
        height: 100vh;
        margin: 0;
        font-family: Arial, sans-serif;
      }

      h1 {
        margin-bottom: 20px;
      }

      #video-container {
        position: relative;
        max-width: 500px;
        margin-bottom: 20px;
      }

      #video {
        display: block;
        width: 100%;
        height: auto;
        border: 1px solid #ccc;
        box-shadow: 0 0 10px rgba(0, 0, 0, 0.1);
        border-radius: 10px;
      }

      #frame {
        position: absolute;
        top: 0;
        left: 0;
        width: 100%;
        height: 100%;
        border: 10px solid rgba(255, 255, 255, 0.8);
        box-shadow: 0 0 20px rgba(0, 0, 0, 0.2);
        border-radius: 20px;
        pointer-events: none;
      }

      #canvas {
        display: block;
        max-width: 500px;
        width: 100%;
        height: auto;
        border: 1px solid #ccc;
        box-shadow: 0 0 10px rgba(0, 0, 0, 0.1);
        border-radius: 10px;
      }
    </style>
  </head>
```

```
</head>
<body>
  <h1>Facial Detection</h1>
  <div id="video-container">
    <video id="video" autoplay style="display: none"></video>
    <div id="frame"></div>
  </div>
  <canvas id="canvas" width="500px" height="400px"></canvas>
</body>

<script src="https://cdn.jsdelivr.net/npm/@tensorflow/tfjs"></script>
<script src="https://cdn.jsdelivr.net/npm/@tensorflow-models/blazeface"></script>
<script src="script.js"></script>
</html>
```

Am realizat câteva modificări pe parte de design a site ului și voi explica mai jos semnificatia fiecarui element:

- `display: flex;` - Aceasta setează elementul `<body>` să utilizeze modelul flexbox pentru a dispune elementele copil într-un flux flexibil.
- `flex-direction: column;` - Aceasta specifică direcția fluxului flexibil ca fiind verticală, adică elementele copil vor fi dispuse de sus în jos.
- `align-items: center;` - Aceasta centrează elementele copil orizontal în cadrul elementului `<body>`.
- `justify-content: center;` - Aceasta centrează elementele copil vertical în cadrul elementului `<body>`.
- `height: 100vh;` - Aceasta setează înălțimea elementului `<body>` să fie de 100% din înălțimea vizibilă a ferestrei browser-ului. Astfel, întreaga înălțime a paginii va fi ocupată de elementul `<body>`.
- `margin: 0;` - Aceasta elimină orice margini exterioare ale elementului `<body>`, asigurându-se că nu există spații goale în jurul paginii.
- `font-family: Arial, sans-serif;` - Aceasta specifică fontul folosit pentru textul din interiorul elementului `<body>` să fie Arial sau, dacă acesta nu este disponibil, un font sans-serif implicit al sistemului.

Alte stiluri definite în codul CSS includ:

- `h1` - Acest stil specifică o margine inferioară de 20px pentru elementele `<h1>`, ceea ce creează un spațiu între titlul "Facial Detection" și restul conținutului.
- `#video-container` - Acest stil specifică o poziție relativă pentru containerul video și o lățime maximă de 500px. De asemenea, adaugă o margine inferioară de 20px pentru a crea un spațiu între containerul video și restul conținutului.

- `#video` - Acest stil specifică ca elementul video să fie afișat ca bloc, cu lățimea de 100% și înălțimea auto. De asemenea, adaugă un cadru subțire și o umbră pentru a evidenția elementul video.
- `#frame` - Acest stil specifică o poziție absolută pentru elementul div cu id-ul "frame", astfel încât acesta să se suprapună peste video. Se stabilește o lățime și înălțime de 100% pentru a acoperi întregul cadru video și se adaugă un cadru gros, o umbră și un fundal semi-transparent pentru a evidenția fețele detectate.
- `#canvas` - Acest stil specifică ca elementul canvas să fie afișat ca bloc, cu o lățime maximă de 500px. De asemenea, adaugă un cadru subțire și o umbră pentru a evidenția elementul canvas.

Prin aplicarea acestor stiluri, pagina va fi formatată într-un mod coerent și estetic, cu elementele video și canvas încadrate și evidențiate corespunzător.

În continuare, trebuie să creăm o variabilă pentru obiectul canvas și să creăm un context 2D. După aceea, trebuie să desenăm fluxul video curent pe canvas, urmat de un dreptunghi pentru detectarea feței, ceea ce poate fi realizat cu ajutorul funcțiilor drawImage și rect ale elementului canvas HTML.

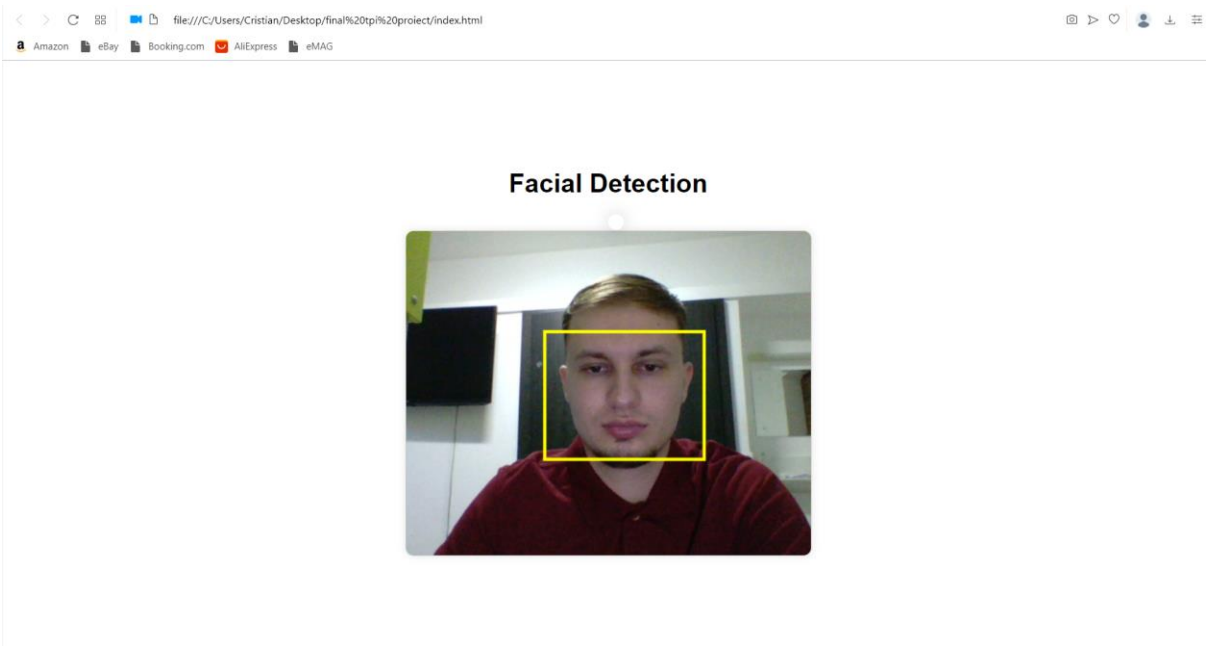
Deoarece funcția "detectFaces" este apelată doar o singură dată, pentru a vedea rezultatele în video, trebuie să o apelăm în mod repetat. Vom folosi metoda setInterval și o vom apela la fiecare 40 de milisecunde pentru a obține un rezultat de aproximativ 24 de cadre pe secundă.

Aici avem codul script.js final:

```
let video = document.getElementById("video");
let model;

// declare the canvas variable and setting up the context

let canvas = document.getElementById("canvas");
let ctx = canvas.getContext("2d");

const accessCamera = () => {
  navigator.mediaDevices
    .getUserMedia({
      video: { width: 500, height: 400 },
      audio: false,
    })
    .then((stream) => {
      video.srcObject = stream;
    });
};

const detectFaces = async () => {
  const prediction = await model.estimateFaces(video, false);

  // Using canvas to draw the video first

  ctx.drawImage(video, 0, 0, 500, 400);
```

```
  prediction.forEach((predictions) => {

    // Drawing rectangle that'll detect the face
    ctx.beginPath();
    ctx.lineWidth = "4";
    ctx.strokeStyle = "yellow";
    ctx.rect(
      predictions.topLeft[0],
      predictions.topLeft[1],
      predictions.bottomRight[0] - predictions.topLeft[0],
      predictions.bottomRight[1] - predictions.topLeft[1]
    );
    // The last two arguments denotes the width and height
    // but since the blazeface models only returns the coordinates
    // so we have to subtract them in order to get the width and height
    ctx.stroke();
  });
};

accessCamera();
video.addEventListener("loadeddata", async () => {
  model = await blazeface.load();
  // Calling the detectFaces every 40 millisecond
  setInterval(detectFaces, 40);
});
```

Rezultatul final al site ului web



VII.CONCLUZII

Prima concluzie a proiectului este că site-ul web implementat utilizează TensorFlow.js pentru recunoașterea facială și demonstrează o performanță excelentă în detectarea și identificarea fețelor, chiar și în cazul fețelor în mișcare și a prezenței multiple a acestora.

Prin atingerea acestui nivel de performanță, site-ul web demonstrează un potențial semnificativ în ceea ce privește utilizarea tehnologiei de recunoaștere facială în scopuri practice și poate oferi o experiență utilizatorilor îmbunătățită și o funcționalitate avansată în cadrul aplicației.

VIII.BIBLIOGRAFIE

https://www.tensorflow.org/

https://www.tensorflow.org/about/case-studies?filter=all#tf-filters

https://towardsdatascience.com/face-detection-for-beginners-e58e8f21aad9

IX.ANEXĂ COD

Cod HTML, CSS

```
<!DOCTYPE html>
<html lang "en">
  <head>
    <meta charset "UTF-8" />
    <meta http-equiv "X-UA-Compatible" content "IE=edge" />
    <meta name "viewport" content "width=device-width, initial-scale=1.0" />
    <title>                    </title>
    <style>
      body
        display  flex
        flex-direction  column
        align-items  center
        justify-content  center
        height  100vh
        margin  0
        font-family  Arial  sans-serif


      h1
        margin-bottom  20px


      #video-container
        position  relative
        max-width  500px
        margin-bottom  20px


      #video
        display  block
        width  100%
        height  auto
        border  1px solid #ccc
        box-shadow  0 0 10px rgba 0  0  0  0.1
        border-radius  10px


      #frame
        position  absolute
        top  0
```

```
        left  0
        width  100%
        height  100%
        border  10px solid rgba 255  255  255  0.8
        box-shadow  0 0 20px rgba 0  0  0  0.2
        border-radius  20px
        pointer-events  none


      #canvas
        display  block
        max-width  500px
        width  100%
        height  auto
        border  1px solid #ccc
        box-shadow  0 0 10px rgba 0  0  0  0.1
        border-radius  10px

    </style>
  </head>
  <body>
    <h1>                </h1>
    <div id "video-container">
      <video id "video" autoplay style "display: none"></video>
      <div id "frame"></div>
    </div>
    <canvas id "canvas" width "500px" height "400px"></canvas>
  </body>

  <script src "https://cdn.jsdelivr.net/npm/@tensorflow/tfjs"></script>
  <script src "https://cdn.jsdelivr.net/npm/@tensorflow-
models/blazeface"></script>
  <script src "script.js"></script>
</html>
```

COD JAVASCRIPT

```
let video = document getElementById "video"
let model

// declare the canvas variable and setting up the context

let canvas = document getElementById "canvas"
let ctx = canvas getContext "2d"

const accessCamera =    =>
  navigator mediaDevices
     getUserMedia
      video:   width: 500  height: 400
      audio: false

     then  stream  =>
      video srcObject = stream



const detectFaces = async    =>
  const prediction = await model estimateFaces video  false

  // Using canvas to draw the video first

  ctx drawImage video  0  0  500  400

  prediction forEach  predictions  =>

    // Drawing rectangle that'll detect the face
    ctx beginPath
    ctx lineWidth = "4"
    ctx strokeStyle = "yellow"
    ctx rect
      predictions topLeft 0
      predictions topLeft 1
      predictions bottomRight 0  - predictions topLeft 0
      predictions bottomRight 1  - predictions topLeft 1

    // The last two arguments denotes the width and height
    // but since the blazeface models only returns the coordinates
    // so we have to subtract them in order to get the width and height
    ctx stroke
```

```
accessCamera
video addEventListener "loadeddata"  async    =>
  model = await blazeface load
  // Calling the detectFaces every 40 millisecond
  setInterval detectFaces  40
```