EXPT NO : 7	TensorFlow.js – RUNNING MODELS IN THE BROWSER
DATE : 29.08.2025	

AIM :

To perform and implement the programs based on TensorFlow.js

Procedure:

1. Create an HTML page and include TensorFlow.js and MobileNet libraries.
2. Add an Upload button to let the user choose an image.
3. Display the uploaded image on the screen.
4. Load the MobileNet pre-trained model in the browser.
5. Pass the uploaded image to the model.
6. Get predictions (object names + confidence %) from the model.
7. Show the predictions to the user.

1. Write a HTML program to demonstrate how to load and run pre-trained machine learning models in the browser using TensorFlow.js for:

Image Classification (MobileNet)

PROGRAM CODE:

```
<!DOCTYPE html>
<html lang="en">
<head>
  <meta charset="UTF-8" />
  <meta name="viewport" content="width=device-width, initial-scale=1.0" />
  <title>TensorFlow.js - Run Model in Browser</title>
  <script src="https://cdn.jsdelivr.net/npm/@tensorflow/tfjs"></script>
  <script src="https://cdn.jsdelivr.net/npm/@tensorflow-models/mobilenet"></script>
  <style>
    body { font-family: Arial, sans-serif; text-align: center; margin-top: 40px; }
    #image { max-width: 300px; margin: 20px auto; display: block; border: 2px solid #ccc; }
    #predictions { margin-top: 20px; font-size: 18px; color: blue; }
    #drop-area {
      border: 2px dashed #888;
```

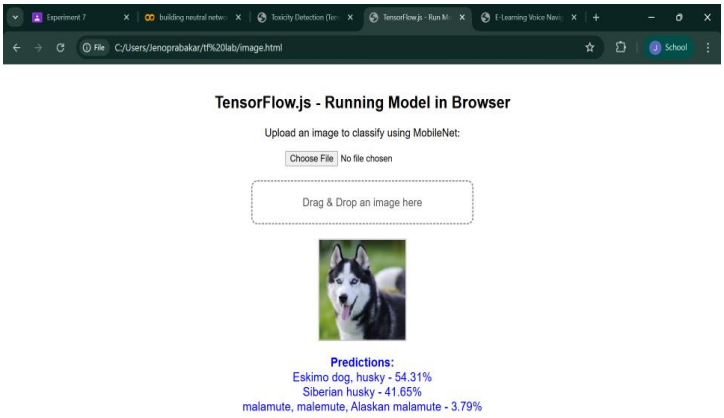
```
      border-radius: 10px;
      width: 320px;
      margin: 20px auto;
      padding: 20px;
      color: #555;  }
    #drop-area.dragover {
      border-color: #007bff;
      background-color: #f0f8ff;
      color: #007bff; }
  </style>
</head>
<body>
  <h2>TensorFlow.js - Running Model in Browser</h2>
  <p>Upload an image to classify using MobileNet:</p>
  <input type="file" id="upload" accept="image/*" />
  <div id="drop-area">Drag & Drop an image here</div>
  <img id="image" />
   <div id="predictions"></div>
 <script>
    let model;
 async function loadModel() {
      document.getElementById('predictions').innerText = "Loading model...";
      model = await mobilenet.load();
      document.getElementById('predictions').innerText = "Model loaded. Upload or drag an
image.";}
 async function classifyImage() {
      const img = document.getElementById('image');
```

```
    if (model && img.src) {
      const predictions = await model.classify(img);
      let resultHTML = "<b>Predictions:</b><br>";
      predictions.forEach(p => {
        resultHTML += `${p.className} - ${(p.probability * 100).toFixed(2)}% <br>`; });
      document.getElementById('predictions').innerHTML = resultHTML;}}
  function handleFile(file) {
    if (file) {
      const reader = new FileReader();
      reader.onload = function(e) {
        document.getElementById('image').src = e.target.result;};
      reader.readAsDataURL(file);  }}
  document.getElementById('upload').addEventListener('change', (event) => {
    handleFile(event.target.files[0]);});
  const dropArea = document.getElementById('drop-area');
  dropArea.addEventListener('dragover', (event) => {
    event.preventDefault();
    dropArea.classList.add('dragover');  });
dropArea.addEventListener('dragleave', () => {
    dropArea.classList.remove('dragover');});
dropArea.addEventListener('drop', (event) => {
    event.preventDefault();
    dropArea.classList.remove('dragover');
    const file = event.dataTransfer.files[0];
    handleFile(file);});
 loadModel();
  </script>
```

</body>

</html>

OUTPUT:



Text Classification (Toxicity Detection)

PROGRAM CODE:

<!DOCTYPE html>

<html lang="en">

<head>

<meta charset="utf-8" />

<meta name="viewport" content="width=device-width,initial-scale=1" />

<title>Toxicity Detection (TensorFlow.js)</title>

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

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

<style>

body { font-family: Arial, sans-serif; max-width: 760px; margin: 40px auto; padding: 0 16px;
}

h1 { font-size: 22px; margin-bottom: 6px; }

textarea { width: 100%; min-height: 120px; padding: 8px; font-size: 14px; }

.controls { margin: 12px 0; display:flex; gap:12px; align-items:center; flex-wrap:wrap; }

button { padding: 8px 14px; font-size: 14px; cursor: pointer; }

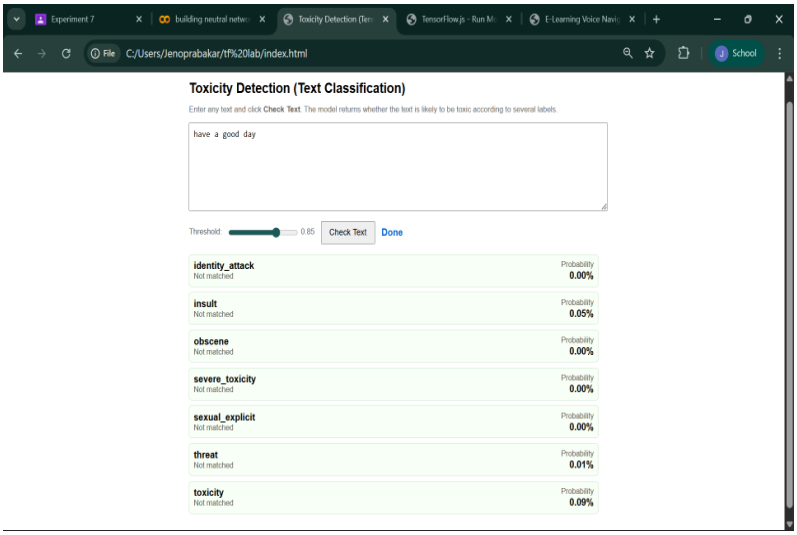
```
    #results { margin-top: 16px; }
    .label-row { padding: 8px; border-radius: 6px; border: 1px solid #e0e0e0; margin-bottom: 8px; display:flex; justify-content:space-between; gap:12px; align-items:center; }
    .match-true { background: #ffecec; border-color: #ffb3b3; }
    .match-false { background: #f7fff7; border-color: #cfe8cf; }
    .small { font-size: 13px; color: #555; }
    #loading { color: #0366d6; font-weight: 600; }
    .muted { color: #666; font-size: 13px; }
  </style></head><body>
  <h1>Toxicity Detection (Text Classification)</h1>
  <p class="muted">Enter any text and click <b>Check Text</b>. The model returns whether the text is likely to be toxic according to several labels.</p>
<textarea id="inputText" placeholder="Type or paste text here..."></textarea>
  <div class="controls">
    <label class="small">Threshold:
      <input id="threshold" type="range" min="0.5" max="0.99" step="0.01" value="0.85" style="vertical-align:middle; margin-left:8px;">
      <span id="thresholdVal" class="small">0.85</span>
    </label>
<button id="checkBtn">Check Text</button>
    <div id="loading">Model not loaded</div>
  </div>
  <div id="results"></div>
  <script>
    let toxicityModel = null;
    const loadingEl = document.getElementById('loading');
    const thresholdInput = document.getElementById('threshold');
    const thresholdVal = document.getElementById('thresholdVal');
```

```
    const checkBtn = document.getElementById('checkBtn');
    const resultsEl = document.getElementById('results');
    const inputText = document.getElementById('inputText');
    thresholdInput.addEventListener('input', () => {
      thresholdVal.innerText = parseFloat(thresholdInput.value).toFixed(2); });
    async function loadToxicityModel() {
      loadingEl.innerText = 'Loading toxicity model...';
      const threshold = parseFloat(thresholdInput.value);
      toxicityModel = await toxicity.load(threshold);
      toxicityModel.threshold = threshold; // save for later
      loadingEl.innerText = 'Model loaded. Enter text and press "Check Text".';  }
    async function classifyText(text) {
      if (!toxicityModel) {
        alert('Model not loaded yet. Please wait a moment.');
        return;}
      resultsEl.innerHTML = '';
      loadingEl.innerText = 'Classifying...';
const predictions = await toxicityModel.classify([text]);
predictions.forEach(pred => {
        const label = pred.label;
        const res = pred.results[0];
        const probTrue = (res.probabilities && res.probabilities[1]) ? res.probabilities[1] : 0;
        const match = res.match;
const row = document.createElement('div');
        row.className = 'label-row ' + (match ? 'match-true' : 'match-false');
const left = document.createElement('div');
        left.innerHTML = `
```

```
        <strong>${label}</strong>
        <div class="small">${match ? 'Matched (>= threshold)' : 'Not matched'}</div>   `;
const right = document.createElement('div');
        right.style.textAlign = 'right';
        right.innerHTML = `
        <div class="small">Probability</div>
        <div><strong>${(probTrue * 100).toFixed(2)}%</strong></div>    `;
row.appendChild(left);
        row.appendChild(right);
        resultsEl.appendChild(row);   });
loadingEl.innerText = 'Done';}
    checkBtn.addEventListener('click', async () => {
      const text = inputText.value.trim();
      if (!text) {
        alert('Please enter some text to classify.');
        return; }
      const currentThreshold = parseFloat(thresholdInput.value);
      if (!toxicityModel || Math.abs((toxicityModel.threshold || 0) - currentThreshold) > 1e-6) {
        toxicityModel = null;
        loadingEl.innerText = 'Reloading model with threshold ' + currentThreshold + ' ...';
        toxicityModel = await toxicity.load(currentThreshold);
        toxicityModel.threshold = currentThreshold;
        loadingEl.innerText = 'Model loaded with threshold ' + currentThreshold + '.';
      }
      await classifyText(text);
    });
```

```
    // Load model on page open
    loadToxicityModel();
  </script>
</body>
</html>
```

OUTPUT :



2. An e-learning web

Procedure:

1. Create an HTML page with lesson content (example: Lesson 1, Lesson 2, etc.).
2. Add **Start Listening** and **Stop Listening** buttons.
3. Use the **Web Speech API** to capture the user's voice.
4. Convert the spoken words into text.
5. If the recognized word is:
    - **"next" → move to the next lesson**
    - **"previous" → go back to the previous lesson**
    - **"stop" → pause navigation**
6. Update the lesson displayed on the screen.

2. An e-learning web application wants to allow students to navigate lessons using simple voice commands instead of keyboard/mouse.
Saying "next" → moves to the next lesson.
Saying "previous" → goes back to the previous lesson.

Saying "stop" → pauses navigation.
This feature should run directly in the browser without requiring extra software installation.
PROGRAM CODE :

```
<!DOCTYPE html>
<html lang="en">
<head>
  <meta charset="UTF-8">
  <title>E-Learning Voice Navigation</title>
  <script src="https://cdn.jsdelivr.net/npm/@tensorflow/tfjs"></script>
  <script src="https://cdn.jsdelivr.net/npm/@tensorflow-models/speech-commands"></script>
<style>
    body {
      font-family: Arial, sans-serif;
      text-align: center;
      margin: 0;
      padding: 0;
      background: linear-gradient(135deg, #e3f2fd, #bbdefb);
      height: 100vh;
      display: flex;
      flex-direction: column;
      justify-content: center;
      align-items: center;   }
   h2 { color: #0d47a1; margin-bottom: 10px; }
    p { font-size: 16px; color: #1565c0; margin-bottom: 20px; }
   #lesson-card {
      background: white;
      padding: 30px;
      border-radius: 15px;
      box-shadow: 0 6px 15px rgba(0,0,0,0.2);
      width: 350px;
      transition: transform 0.3s ease-in-out; }
#lesson {
      font-size: 22px;
      margin: 20px 0;
      font-weight: bold;
      color: #0d47a1; }
 .progress { margin: 10px 0; font-size: 14px; color: #555; }
    .nav-arrows { display: flex; justify-content: space-between; margin-top: 15px; }
```

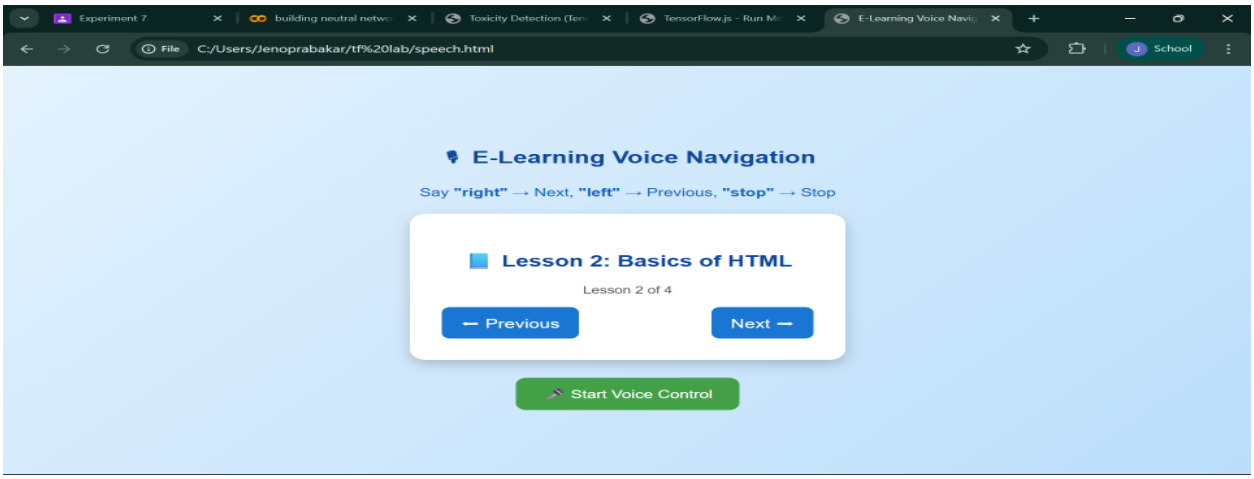
```
.arrow {
  background: #1976d2;
  color: white;
  border: none;
  padding: 10px 18px;
  font-size: 18px;
  border-radius: 8px;
  cursor: pointer;
  transition: background 0.2s;
}

.arrow:hover { background: #0d47a1; }

#start-btn {
  background: #43a047;
  color: white;
  border: none;
  padding: 12px 25px;
  font-size: 16px;
  border-radius: 10px;
  margin-top: 25px;
  cursor: pointer;
  transition: background 0.2s;
}

#start-btn:hover { background: #2e7d32; }

/* Toast notification */
#toast {
  visibility: hidden;
  min-width: 250px;
  background-color: #333;
  color: #fff;
  text-align: center;
  border-radius: 8px;
  padding: 12px;
  position: fixed;
  bottom: 30px;
  left: 50%;
  transform: translateX(-50%);
```

```
    z-index: 1000;
    font-size: 14px;
    opacity: 0;
    transition: opacity 0.5s, visibility 0.5s;
   }

   #toast.show {
    visibility: visible;
    opacity: 1;
   }
  </style>
</head>
<body>
  <h2>🎙️ E-Learning Voice Navigation</h2>
  <p>Say <b>"right"</b> → Next, <b>"left"</b> → Previous, <b>"stop"</b> → Stop</p>

  <div id="lesson-card">
   <div id="lesson">📘 Lesson 1: Introduction</div>
   <div class="progress" id="progress">Lesson 1 of 4</div>

   <div class="nav-arrows">
    <button class="arrow" onclick="prevLesson()">⬅ Previous</button>
    <button class="arrow" onclick="nextLesson()">Next ➡</button>
   </div>
  </div>

  <button id="start-btn" onclick="startListening()">🎤 Start Voice Control</button>

  <!-- Toast element -->
  <div id="toast"></div>

  <script>
   let recognizer;
   let lessons = [
    "📘 Lesson 1: Introduction",
    "📘 Lesson 2: Basics of HTML",
    "📘 Lesson 3: Basics of CSS",
    "📘 Lesson 4: Basics of JavaScript"
   ];
```

```
let currentLesson = 0;

function showToast(message) {
  const toast = document.getElementById("toast");
  toast.innerText = message;
  toast.className = "show";
  setTimeout(() => toast.className = toast.className.replace("show", ""), 2000);
}

function updateLesson() {
  document.getElementById("lesson").innerText = lessons[currentLesson];
  document.getElementById("progress").innerText =
    `Lesson ${currentLesson + 1} of ${lessons.length}`;

  let card = document.getElementById("lesson-card");
  card.style.transform = "scale(1.05)";
  setTimeout(() => card.style.transform = "scale(1)", 200);
}

function nextLesson() {
  if (currentLesson < lessons.length - 1) {
    currentLesson++;
    updateLesson();
  } else {
    showToast("⚠ You are already on the last lesson!");
  }
}

function prevLesson() {
  if (currentLesson > 0) {
    currentLesson--;
    updateLesson();
  } else {
    showToast("⚠ You are already on the first lesson!");
  }
}

async function loadModel() {
  recognizer = speechCommands.create("BROWSER_FFT");
  await recognizer.ensureModelLoaded();
```

```
    console.log("Model loaded. Labels:", recognizer.wordLabels());
  }

  async function startListening() {
    if (!recognizer) await loadModel();

    // ensure mic permission
    await navigator.mediaDevices.getUserMedia({ audio: true });

    recognizer.listen(result => {
      const scores = result.scores;
      const labels = recognizer.wordLabels();
      const maxIndex = scores.indexOf(Math.max(...scores));
      const command = labels[maxIndex];
      console.log("Heard:", command);

      if (command === "right") nextLesson();
      else if (command === "left") prevLesson();
      else if (command === "stop") {
        recognizer.stopListening();
        showToast("Voice control stopped.");
      }
    }, {
      includeSpectrogram: false,
      probabilityThreshold: 0.6
    });
  }

  loadModel();
 </script>
</body>
</html>
```

OUTPUT :



RESULT :

The implementation of the programs based on TensorFlow.js has been executed successfully