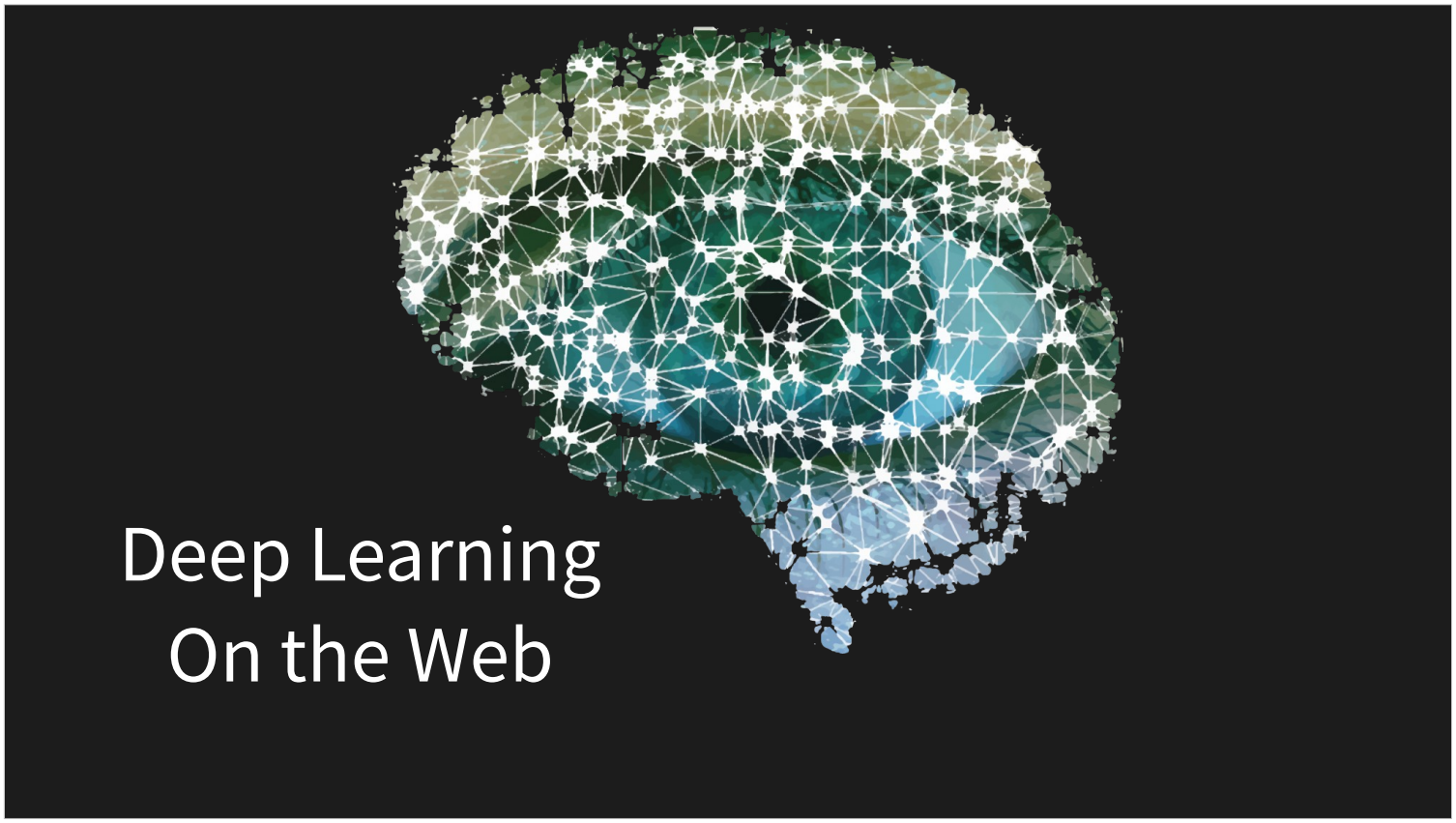
Contemporary Web Development
Lesson 10

https://www.youtube.com/channel/UCg3qsVzHeUt5_cPpcRtoaJQ


Deep Learning
On the Web

Three main approaches:

1) Load/train your model on a server, send the input from the browser and receive the result - Can use specialized cloud hosts and platforms.
2) Use a cloud based dedicated service from one of the major providers.
3) Run a model completely on the browser.

Three main approaches:

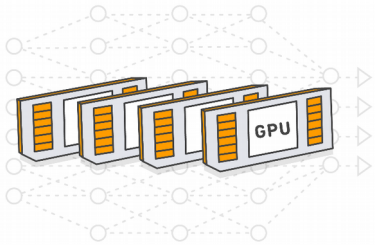
1) Load/train your model on a server, send the input from the browser and receive the result - Can use specialized cloud hosts and platforms.
2) Use a cloud based dedicated service from one of the major providers.
3) Run a model completely on the browser.

Amazon EC2 P2 Instances

Powerful, Scalable GPU instances for high-performance computing

Amazon EC2 P2 Instances are powerful, scalable instances that provide GPU-based parallel compute capabilities. For customers with graphics requirements, see G2 instances for more information.

P2 instances, designed for general-purpose GPU compute applications using CUDA and OpenCL, are ideally suited for machine learning, high performance databases, computational fluid dynamics, computational finance, seismic analysis, molecular modeling, genomics, rendering, and other server-side workloads requiring massive parallel floating point processing power.



Cloud ML Engine

Check out also Elastic Inference.

Google Colab Python Notebook

BigGAN demo.

Three main approaches:

1) Load/train your model on a server, send the input from the browser and receive the result - Can use specialized cloud hosts and platforms.
2) Use a cloud based dedicated service from one of the major providers.
3) Run a model completely on the browser.

Main Providers – All have free trials

1) Google - Requires credit card but no auto charge **(Usually highest quality)**.
2) IBM Watson – Doesn't require credit card.
3) Microsoft Azure – Doesn't require credit card **(Good value for money).**
4) AWS Free Tier – Requires credit card and may auto charge.

Magic Mirror

Webcam Access
and much more

WebRTC facilitates peer to peer connections,

./webcam.js

```
export async function start(element) {
  try {
    let stream = await navigator.mediaDevices.getUserMedia({ video: true, audio: false })
    element.srcObject = stream;
    element.play();
    console.log("Webcam capture active");
  }
  catch (err) {
    console.warn("Webcam capture error", err);
  }
}
```

./webcam.js

```
export async function snap(camera, film) {
  return new Promise((resolve, reject)  => {
    let videoRect = camera.getBoundingClientRect();
    film.width = videoRect.width;
    film.height = videoRect.height;
    let context = film.getContext('2d');
    context.drawImage(camera, 0, 0, film.width, film.height);
    film.toBlob((blob) => {
      resolve(blob);
    });
  });
}
```

Posting image "Blobs"

XMLHTTPRequest

it’s not recommended to use "fetch" for sending form data.

./routes.js

```
import * as Forms from './forms'
import * as Vision from './vision'

export default function(app) {
  app.post('/image/labels', async function(req, res) {
    try {
      let file = await Forms.getFile(req, 'image');
      let labels = await Vision.tagImage(file);
      res.send(labels)
    }
    catch (err) {
      res.status(500).send({message: err.toString()})
    }
  })
}
```

Server side

./forms.js

```
import formidable from 'formidable'
import fs from 'fs'

export function getFile(req,field) {
   return new Promise((resolve, reject) => {
      let form = new formidable.IncomingForm();
      form.keepExtensions = true;
      form.parse(req, (err, fields, files) => {
        let file = files[field]
        if (!file) {
           reject(new Error("Missing file field"))
        }
        resolve(file.path);
      })
   })
}
```

Processing forms with ‘Formidable’

./vision.js

```
import vision from '@google-cloud/vision';

export async function tagImage(file) {
   const client = new vision.ImageAnnotatorClient();
   let results = await client.labelDetection(file)
   return results[0].labelAnnotations;
}
```

Using Google's Node JS Library

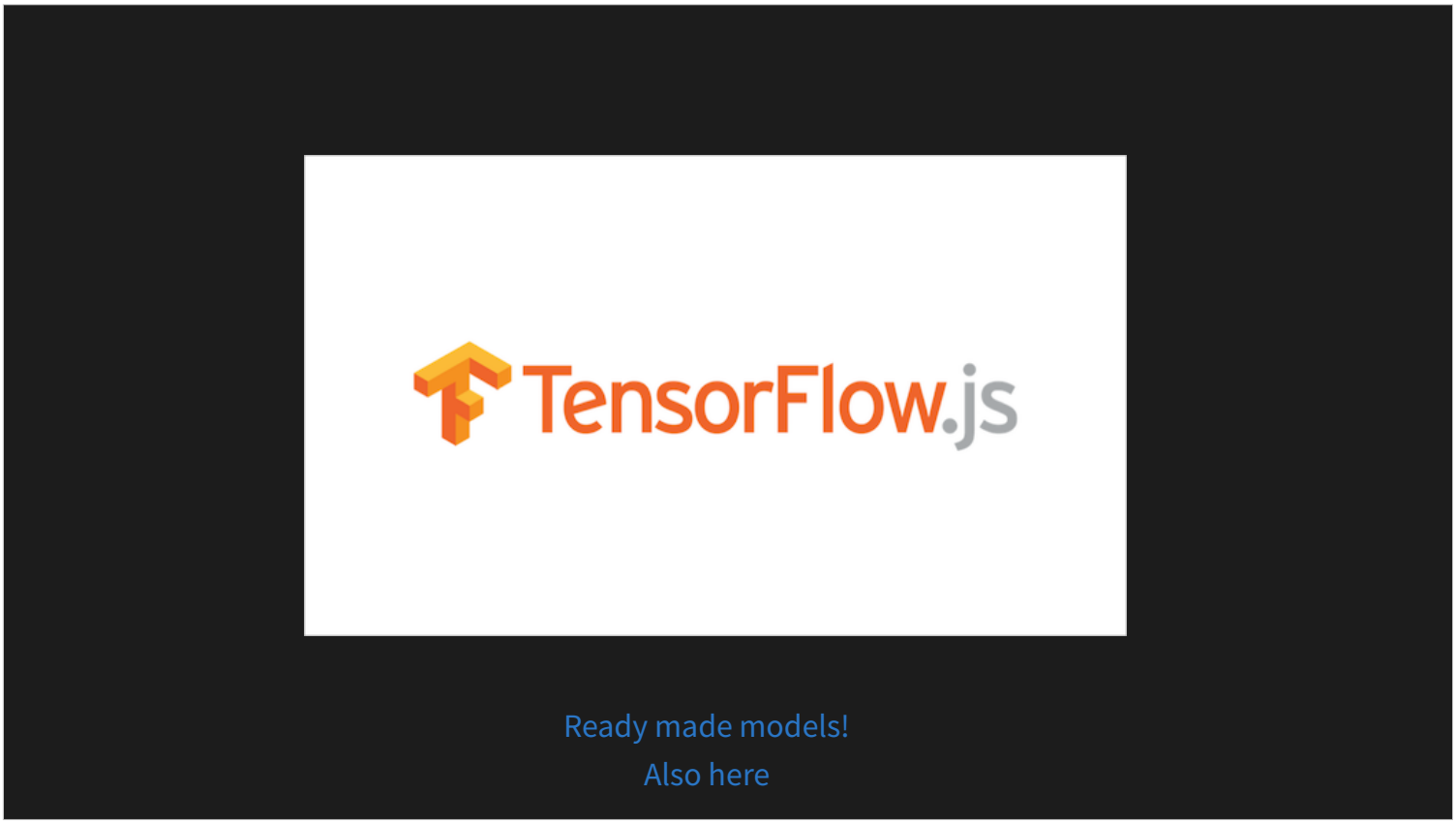
./webcam.js

```
export async function snap(camera, film) {
  return new Promise((resolve, reject)  => {
    let videoRect = camera.getBoundingClientRect();
    film.width = videoRect.width;
    film.height = videoRect.height;
    let context = film.getContext('2d');
    context.drawImage(camera, 0, 0, film.width, film.height);
    film.toBlob((blob) => {
      resolve(blob);
    });
  });
}
```

Here’s my key for trying.
(ask me for the password)

Three main approaches:

1) Load/train your model on a server, send the input from the browser and receive the result - Can use specialized cloud hosts and platforms.
2) Use a cloud based dedicated service from one of the major providers.
3) Run a model completely on the browser.



It actually uses WebGL for GPU acceleration!

It actually uses WebGL for GPU acceleration!

It actually uses WebGL for GPU acceleration!

It actually uses WebGL for GPU acceleration!

Reactive Exercise

1) **The goal: if a bottle is detected – the camera image turns translucent blue.**
2) In the *State*, add a variable named *Render.backgroundColor,* defaults to transparent.
3) Add a function in the *State* named *updateBackground(color),* it updates the states and triggers *Events.trigger('background-updated');*
4) Add a function to *Renderer* named *changeBackground(color,elements)* that would change the background.
5) In *Vision* , if a bottle is detected, *Events.trigger*('bottle-detected')
6) In *Index, r*eact to 'bottle-detected' (*Events.on)* to call State.changeBackground
7) As well in *Index, react to* 'background-updated' and call *Renderer.changeBackground* with the new *Render.backgroundColor*
8) Bonus: Have the background return to transparent when the bottle disappears.

Deep Learning Exercise

1) Fork any one of the ai projects and use a different ML model.