**Running Models in a Browser**

**Converting a Keras HDF5 model to Tensorflow.js or other models types such as SavedModel to Tensorflow.js**

Running models in a browser can be done using TensorFlow.js, which allows you to run pre-trained models directly in the browser. To do this, you need to convert your Keras HDF5 model or TensorFlow SavedModel to TensorFlow.js format.

**Converting a Keras HDF5 Model to TensorFlow.js**

**Steps:**

1. **Install TensorFlow.js Converter**: First, you need to install the TensorFlow.js converter.

**pip install tensorflowjs** 🡪 in command prompt

1. **Convert the Keras Model**: Use the TensorFlow.js converter to convert the Keras HDF5 model to TensorFlow.js format.

**tensorflowjs\_converter --input\_format keras path\_to\_keras\_model.h5 path\_to\_tfjs\_model** 🡪 in command prompt

1. **Load the Model in the Browser:** Use TensorFlow.js to load and run the model in the browser.

**Example:**

**1.Convert the Model:**

Suppose you have a Keras model saved as model.h5. Convert it using the following command:

**tensorflowjs\_converter --input\_format keras model.h5 tfjs\_model** 🡪 in cmd

**2.Run the Model in the Browser:**

* **Create an HTML File:**

<!DOCTYPE html>

<html lang="en">

<head>

<meta charset="UTF-8">

<meta name="viewport" content="width=device-width, initial-scale=1.0">

<title>TensorFlow.js Example</title>

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

<script src="tfjs\_model/model.json"></script>

</head>

<body>

<h1>Run Keras Model in Browser</h1>

<script>

async function loadModel() {

// Load the model

const model = await tf.loadGraphModel('tfjs\_model/model.json');

// Make a prediction (example input shape: [1, 32])

const input = tf.tensor2d([0.5, 0.5, 0.5, 0.5, 0.5, 0.5, 0.5, 0.5, 0.5, 0.5, 0.5, 0.5, 0.5, 0.5, 0.5, 0.5, 0.5, 0.5, 0.5, 0.5, 0.5, 0.5, 0.5, 0.5, 0.5, 0.5, 0.5, 0.5, 0.5, 0.5, 0.5, 0.5], [1, 32]);

const output = model.predict(input);

output.print();

}

loadModel();

</script>

</body>

</html>

**3.Serve the Files:**

Use a local server to serve the files. You can use Python’s ` http.server `module for this.

**python -m http.server** 🡪 in command prompt

Navigate to ` http://localhost:8000 ` in your browser to see the model running.  
  
U can go through these  references to get better  understanding

<https://www.tensorflow.org/js/guide/conversion> <https://www.tensorflow.org/js/tutorials/conversion/import_keras>

**Hosting a Model File on a Server or VM**

Hosting a TensorFlow or Keras model on a server or VM involves several steps, including saving the model, deploying it with an API framework like FastAPI, and configuring a server to host the API. Here’s a step-by-step explanation:

**1. Prepare Your Model**

* **Save the Model**: Train and save your model in a format that can be served, such as TensorFlow's SavedModel format.

**Python**import tensorflow as tf

from tensorflow.keras.models import Sequential

from tensorflow.keras.layers import Dense

# Create a simple model

model = Sequential([

Dense(10, activation='relu', input\_shape=(32,)),

Dense(3, activation='softmax')

])

# Save the model

model.save('saved\_model/my\_model')

**2. Set Up Your Server**

* **Choose Hosting Platform**: Use cloud providers like AWS, Google Cloud, Azure, or a VPS provider like DigitalOcean.
* **Create and Access VM**: Set up a VM and access it using SSH.

**3. Install Dependencies**

* **Update and Install Required Packages**: Ensure your VM is up-to-date and install Python, pip, and other necessary packages.

**Windows power shell**

sudo apt update

sudo apt upgrade -y

sudo apt install python3-pip python3-venv nginx -y

**4. Deploy the Model with FastAPI**

* **Set Up the Project Environment**: Create a virtual environment and install FastAPI, Uvicorn, and TensorFlow.

**Windows power shell**

mkdir fastapi\_model

cd fastapi\_model

python3 -m venv venv

source venv/bin/activate

pip install fastapi uvicorn tensorflow

**Create FastAPI App**: Write the FastAPI app code to load the model and create API endpoints.

**Python**

from fastapi import FastAPI

import tensorflow as tf

from pydantic import BaseModel

app = FastAPI()

# Load the model

model = tf.keras.models.load\_model('saved\_model/my\_model')

class InputData(BaseModel):

input: list

@app.post("/predict/")

def predict(input\_data: InputData):

data = tf.convert\_to\_tensor([input\_data.input])

prediction = model(data)

return {"prediction": prediction.numpy().tolist()}

**Run the FastAPI App**: Use Uvicorn to run your FastAPI application.

**Shell**

uvicorn main:app --host 0.0.0.0 --port 8000

**5. Set Up Nginx as a Reverse Proxy**

* **Install Nginx**: Install and configure Nginx to act as a reverse proxy.

**Shell**

sudo apt install nginx

**Configure Nginx**: Create and enable an Nginx configuration file that forwards requests to the FastAPI app.

**Nginx**

server {

listen 80;

server\_name your\_domain\_or\_IP;

location / {

proxy\_pass http://127.0.0.1:8000;

proxy\_set\_header Host $host;

proxy\_set\_header X-Real-IP $remote\_addr;

proxy\_set\_header X-Forwarded-For $proxy\_add\_x\_forwarded\_for;

proxy\_set\_header X-Forwarded-Proto $scheme;

}

}

**Restart Nginx**: Apply the configuration changes by restarting Nginx.

**Shell**

sudo systemctl restart nginx

**6. Create a Simple Web Interface**

* **HTML Interface**: Optionally, create a simple HTML form to interact with the API.

**HTML**

<!DOCTYPE html>

<html>

<head>

<title>Model Prediction</title>

</head>

<body>

<h1>Model Prediction</h1>

<form id="prediction-form">

<label for="input">Input:</label>

<input type="text" id="input" name="input">

<button type="submit">Predict</button>

</form>

<p id="result"></p>

<script>

document.getElementById('prediction-form').addEventListener('submit', async (e) => {

e.preventDefault();

const input = document.getElementById('input').value.split(',').map(Number);

const response = await fetch('/predict/', {

method: 'POST',

headers: {

'Content-Type': 'application/json'

},

body: JSON.stringify({ input })

});

const result = await response.json();

document.getElementById('result').innerText = 'Prediction: ' + result.prediction;

});

</script>

</body>

</html>

**7. Testing and Using the API**

* **Access the API**: Open your browser and navigate to your server's IP or domain to access the web interface.
* **Send Requests**: Use the form to send input data to the API and receive model predictions.

By following these steps, you can host a TensorFlow/Keras model on a server or VM, expose it via an API using FastAPI, and optionally create a web interface to interact with the model. This setup allows you to make your machine learning models accessible over the web, enabling integration with various clients and applications.

For reference purpose   
  
<https://medium.com/@shrutipal700/hosting-a-website-on-windows-virtual-machine-61fd427e1ef4>

**Loading and Running the model in a browser**

To load and run a TensorFlow/Keras model in a browser, you can convert the model to TensorFlow.js format and then run it using TensorFlow.js. Below is a step-by-step guide on how to achieve this:

**Step 1: Save Your Model in a Supported Format**

Save your model using the .h5 format (HDF5) since TensorFlow.js supports this format for conversion.

**Python**

import tensorflow as tf

from tensorflow.keras.models import Sequential

from tensorflow.keras.layers import Dense

# Create a simple model

model = Sequential([

Dense(10, activation='relu', input\_shape=(32,)),

Dense(3, activation='softmax')

])

# Save the model in HDF5 format

model.save('model.h5')

**Step 2: Convert the Model to TensorFlow.js Format**

Install the TensorFlow.js converter tool if you haven't already:

**Shell**

pip install tensorflowjs

Then, convert the .h5 model to TensorFlow.js format:

**Shell**

tensorflowjs\_converter --input\_format keras model.h5 web\_model

This command converts the Keras model and saves it in a directory called web\_model.

**Step 3: Set Up a Simple Web Server**

You can use Python's built-in HTTP server to serve your model files. Navigate to the directory containing the web\_model folder and run:

**Shell**

python -m http.server

This will start a local web server at http://localhost:8000.

**Step 4: Load and Run the Model in the Browser**

Create an index.html file to load and run your model using TensorFlow.js. Save the following HTML and JavaScript code in index.html:

**HTML**

<!DOCTYPE html>

<html>

<head>

<title>TensorFlow.js Model Demo</title>

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

</head>

<body>

<h1>TensorFlow.js Model Demo</h1>

<script>

async function runModel() {

// Load the model

const model = await tf.loadLayersModel('http://localhost:8000/web\_model/model.json');

// Print model summary

model.summary();

// Prepare some input data (e.g., a random tensor with the correct input shape)

const input = tf.randomNormal([1, 32]);

// Run the model on the input data

const output = model.predict(input);

// Print the output

output.print();

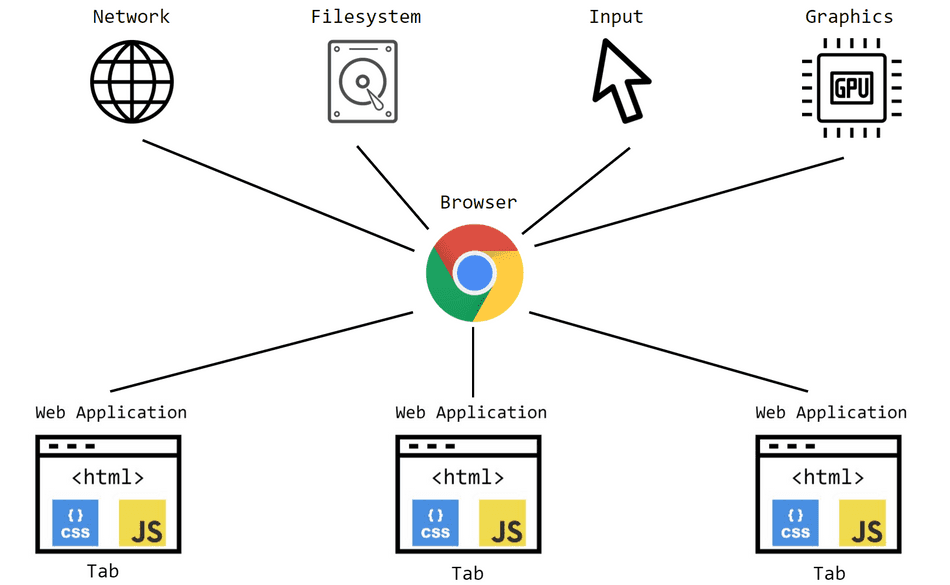
}

runModel();

</script>

</body>

</html>



**Step 5: View the Model in the Browser**

Open a web browser and navigate to http://localhost:8000. You should see the TensorFlow.js model demo page, and the model will load and run, with the summary and output printed to the console.

**Explanation**

1. **TensorFlow.js**: A JavaScript library for training and deploying machine learning models in the browser.
2. **Model Conversion**: The tensorflowjs\_converter tool converts TensorFlow/Keras models to TensorFlow.js format.
3. **Web Server**: A simple HTTP server to serve the model files.
4. **Browser Execution**: The tf.loadLayersModel function loads the model, and model.predict runs the model on input data.

For reference purpose use below link   
  
<https://www.youtube.com/watch?v=m-x_PiZrnec>