



Belajar TensorFlow.js bersama Lab AI

2019

Artificial Intelligence Laboratory - Telkom University
Jl. Telekomunikasi No. 1, Sukapura, Bojongsoang
Kab. Bandung, Jawa Barat
40257
Gedung E Fakultas Informatika ruang E107

Belajar TensorFlow.js Bersama lab AI

Modul I Pengenalannya TensorFlow.js



Artificial Intelligence Laboratory
Fakultas Informatika
Universitas Telkom
Bandung, April 2019



Modul I

Pengenalan TensorFlow.js

1 Apa itu TensorFlow.js

TensorFlow.js adalah *library* JavaScript yang memberikan fasilitas pada kita untuk bisa menambahkan dan menggunakan kemampuan Pembelajaran Mesin ke dalam aplikasi web apa pun. Dengan **TensorFlow.js** kita dapat mengembangkan sistem Pembelajaran Mesin dari level tingkat rendah hingga tinggi. *Library* ini telah menyediakan berbagai fungsi API yang dapat kita gunakan untuk membangun model yang ringan dan dapat langsung dilatih di browser atau di aplikasi server **Node.js** kita.



Kita juga dapat menggunakan **TensorFlow.js** untuk menjalankan model yang sudah dilatih dari lingkungan sistem lain ke dalam lingkungan JavaScript di aplikasi web kita. Kita bahkan dapat menggunakan **TensorFlow.js** untuk melatih ulang model Pembelajaran Mesin yang sudah ada dengan data yang tersedia sisi klien di browser. Misalnya, kita dapat menggunakan data gambar dari kamera webcam di komputer kita.

Situs web dan dokumentasi API dapat ditemukan di <https://www.tensorflow.org/js/>

1.1 Sejarah

TensorFlow.js dikembangkan pada **2011** di Google sebagai *library* pendukung pembuatan aplikasi Pembelajaran Mesin / Deep learning di Google. Perpustakaan ini bersumber terbuka pada 2015 di bawah Lisensi Apache.

Pada **2017**, sebuah proyek bernama **DeepLearn.js** muncul, yang bertujuan untuk mengaktifkan ML / DL dalam JavaScript, dengan API yang mudah digunakan.

Tetapi ada pertanyaan tentang kecepatan. Sudah sangat diketahui bahwa kode JavaScript tidak dapat berjalan pada GPU. Untuk mengatasi masalah ini, **WebGL** diperkenalkan. **WebGL** adalah antarmuka browser untuk OpenGL yang memungkinkan untuk mengeksekusi kode JavaScript pada GPU.

Pada bulan **Maret 2018**, tim **DeepLearn.js** bergabung ke Tim TensorFlow di **Google** dan diganti namanya menjadi **TensorFlow.js**.

Dan akhirnya, pada **7 Maret 2019**, TensorFlow resmi merilis **versi 1.0** dari **TensorFlow.js** bertepatan dengan event **TF Dev Summit**. Informasi mengenai rilis versi terbaru dari **TensorFlow.js** dapat dilihat di <https://github.com/tensorflow/tfjs/releases>



2 Istilah Dasar dalam TensorFlow.js

Sebelum memulai dengan contoh praktis, mari kita lihat blok bangunan utama di TensorFlow.js.

2.1 Tensor

Tensor adalah unit pusat data di TensorFlow. Tensor berisi sekumpulan nilai numerik dan dapat dalam bentuk apa pun: satu dimensi atau lebih. Saat kita membuat Tensor baru, kita juga perlu menentukan bentuknya. Kita bisa melakukannya dengan menggunakan fungsi `tf.tensor` dan mendefinisikan bentuk dengan meneruskan argumen kedua seperti yang dapat dilihat pada kode berikut ini:

```
const t1 = tf.tensor([4, 3, 2, 1, 2, 4, 6, 8]), [2, 4])
```

kode tersebut mendefinisikan tensor dengan bentuk **dua** baris dan **empat** kolom. Tensor yang dihasilkan terlihat seperti berikut:

```
[ [4, 3, 2, 1],  
  [2, 4, 6, 8]]
```

Tensor yang sama dapat pula dihasilkan menggunakan kode

```
const t2 = tf.tensor([[4, 3, 2, 1], [2, 4, 6, 8]])
```

Hasilnya akan sama seperti sebelumnya.

Selanjutnya kita dapat menggunakan fungsi-fungsi berikut untuk memudahkan dalam membaca kode:

```
tf.scalar      : Tensor dengan hanya satu nilai  
tf.tensor1d    : Tensor dengan satu dimensi  
tf.tensor2d    : Tensor dengan dua dimensi  
tf.tensor3d    : Tensor dengan tiga dimensi  
tf.tensor4d    : Tensor dengan empat dimensi
```

Terdapat banyak cara untuk menginisialisasi sebuah tensor. Misalnya, jika kita ingin membuat tensor dengan semua nilai diatur ke nilai **0**, kita dapat menggunakan fungsi `tf.zeros()`, seperti contoh di bawah ini:

```
const t_zeros = tf.zeros([2, 3])
```

Baris kode di atas membuat tensor sebagai berikut:

```
[ [0, 0, 0],  
  [0, 0, 0]]
```

Fungsi-fungsi lain untuk menginisialisasi tensor terdapat `tf.ones()`, `tf.eye()`, `tf.randomNormal()`, dan banyak lainnya.

Di dalam TensorFlow.js semua tensor tidak bisa berubah setelah diinisialisasi (*immutable*). Hal ini berarti bahwa tensor yang pernah dibuat, hanya bisa dihapus. Jika kita melakukan operasi yang mengubah nilai tensor, tensorflow akan selalu membuat tensor baru dengan nilai yang dihasilkan dibuat dan dikembalikan.

Karena itu sangat penting untuk mengingat dan menjaga tensor apa saja yang telah dibuat. Jika suatu tensor sudah tidak digunakan, kita dapat menghapus tensor tersebut dengan fungsi `.dispose()`

2.2 Operasi

Dengan menggunakan operasi TensorFlow.js kita dapat memanipulasi data tensor. TensorFlow.js menawarkan banyak operasi yang bermanfaat seperti `square`, `add`, `sub`, `mul`, dan sebagainya. Menerapkan operasi sangat mudah seperti yang bisa dilihat di bawah ini:

```
const t3 = tf.tensor2d([1,2], [3, 4]);
const t3_squared = t3.square();
```

Setelah menjalankan kode ini, tensor baru berisi nilai-nilai berikut:

```
[[1, 4],
 [9, 16]]
```

2.3 Model dan Lapisan

Model dan *Layers* adalah dua blok bangunan terpenting dalam hal Pembelajaran Mesin dan *Deep Learning*. Setiap model dibangun dari satu atau lebih *layer*. TensorFlow.js mendukung berbagai jenis *layer* yang populer digunakan di Pembelajaran *Deep Learning* terkini. Untuk penggunaan kasus Pembelajaran Mesin yang berbeda, kita perlu menggunakan dan menggabungkan berbagai jenis *layer*.

Untuk saat ini cukup untuk memahami bahwa *layer* digunakan untuk membangun jaringan saraf (*model*) yang dapat dilatih menggunakan sekumpulan data latih dan kemudian digunakan lebih lanjut untuk memprediksi nilai baru berdasarkan informasi yang dilatih.

Ada dua cara utama untuk pembangunan jaringan saraf tiruan pada TensorFlow.js. Namun di sini kita hanya fokuskan pada cara yang paling umum dan paling mudah. Pembangunan model diawali dengan membuat objek `sequential()`. Kemudian kita bisa tambahkan beberapa tumpukan *layer* sesuai kebutuhan kita di dalamnya.

Contoh pembangunan model terlihat pada kode di bawah

```
model = tf.sequential()
model.add(
  tf.layers.dense({units: 10, activation: 'sigmoid',
    inputShape: [5]}))
model.add(tf.layers.dense({units: 1, , activation: 'sigmoid'}))
```

Kita akan lihat contoh penerapan lebih lanjut mengenai model saat kita mulai membangun aplikasi pembelajaran mesin.



3 Instalasi Environment

3.1 Instalasi Python

Meskipun sudah mengangkat nama JavaScript, TensorFlow.js masih harus berjalan di atas Interpreter Python. Dan sayangnya, tidak seperti TensorFlow standar python yang harus berjalan di atas Python versi 3 ke atas, TensorFlow.js hingga saat ini hanya bisa dijalankan di atas Python 2.7. Untuk itu, sangat disarankan instalasi Python dilakukan menggunakan Paket Distribusi Anaconda. Instalasi Anaconda dapat diunduh dari <https://www.anaconda.com/distribution/>. Untuk kemudahan, disarankan memasang Anaconda versi 3.x, kemudian mengunduh python 2.7 menggunakan environment baru di dalamnya.

3.2 Pembuatan Python Environment

Setelah menginstal Anaconda, buka Anaconda Prompt untuk membuat environment baru dengan basis python versi 2.7. Syntax umum untuk membuat environment baru adalah sebagai berikut

```
(base) > conda create --name nama_env python=2.7
```

Pada contoh kali ini, kita akan bangun environment python nama environment my_tfjs.

```
(base) > conda create --name my_tfjs python=2.7
```

Syntax tersebut akan mengunduh python versi 2.7. Untuk mengaktifkannya gunakan syntax

```
(base) > activate my_tfjs  
(my_tfjs) >
```

3.3 Instalasi Node.js

Node.js adalah platform perangkat lunak pada sisi-server dan aplikasi jaringan yang mengeksekusi kode JavaScript di luar browser. Untuk *client-side scripting*, pada umumnya digunakan JavaScript, di mana skrip yang ditulis tertanam dalam HTML halaman web dan dijalankan dengan mesin JavaScript di browser web pengguna. **Node.js** memungkinkan pengembang menggunakan JavaScript membangun konten halaman web secara dinamis sebelum halaman dikirim ke browser web pengguna. Meskipun .js adalah ekstensi nama file konvensional untuk kode JavaScript, nama "**Node.js**" tidak merujuk ke file tertentu dan dalam konteks ini dan hanya nama produk. Untuk menggunakan **Node.js**, kita harus menginstall npm sebagai package manager default untuk lingkungan runtime **Node.js**. Download instalasi **Node.js** di <https://nodejs.org/en/>

Setelah instalasi, aktifkan environment my_tfjs melalui Conda Prompt, dan jalankan syntax

```
(my_tfjs) > node -v  
(my_tfjs) > npm -v
```

jika versi dari **Node.js** dan npm sudah bisa tertampil, maka instalasi telah berhasil



4 Membangun Proyek dengan TensorFlow.js

4.1 Memulai Proyek

Mari kita mulai dengan sebuah contoh sederhana. Pada langkah pertama kita perlu mengatur direktori dan modul kebutuhan proyek. Buat direktori kosong baru:

```
> mkdir ai_tfjs
```

Ubah ke folder proyek yang baru dibuat:

```
> cd ai_tfjs
```

Di dalam folder kita sekarang siap untuk membuat file `package.json`, sehingga kita dapat mengelola dependensi dengan menggunakan Package Manager **Node.js**:

```
\ai_tfjs> npm init -y
```

Karena kita akan menginstal dependensi (mis. *Library TensorFlow.js*) secara lokal di folder proyek kita, kita perlu menggunakan bundler modul untuk aplikasi web kita. Untuk menjaga hal-hal semudah mungkin kita akan menggunakan bundler aplikasi web **Parcel** karena **Parcel** bekerja dengan konfigurasi nol. Mari kita pasang bundler **Parcel** dengan menjalankan perintah berikut di direktori proyek:

```
\ai_tfjs> npm install -g parcel-bundler
```

Selanjutnya, mari kita tambahkan *library Bootstrap* sebagai dependensi karena kita akan menggunakan beberapa kelas **Bootstrap CSS** untuk elemen antarmuka pengguna kita:

```
\ai_tfjs> npm install bootstrap
```

Terakhir, kita tambahkan juga *library jQuery* untuk mempersingkat penulisan kode JavaScript

```
\ai_tfjs> npm install jquery
```

Setelah semua *library* siap, mari kita buat dua file kosong baru `index.html` dan `index.js` di dalam folder `modul_1` untuk implementasi kita.

```
\ai_tfjs> mkdir modul_1  
\ai_tfjs > cd > modul_1\index.html  
\ai_tfjs > cd > modul_1\index.js
```



Dalam file `index.html` mari kita masukkan kode berikut dari halaman HTML dasar:

```
File: index.html

<html>
<body>
  <div class="container">
    <div class="card-body">
      <h1>Belajar TensorFlow.js Bersama Lab AI</h1>
    </div><hr>

    <div id="output"/>
  </div>
  <script src="./index.js"></script>
</body>
</html>
```

Setelah itu, tambahkan kode berikut untuk menambahkan dependensi **Bootstrap** dan **jQuery** ke dalam file `index.js`:

```
File: index.js

import 'bootstrap/dist/css/bootstrap.css'
import $ from 'jquery'

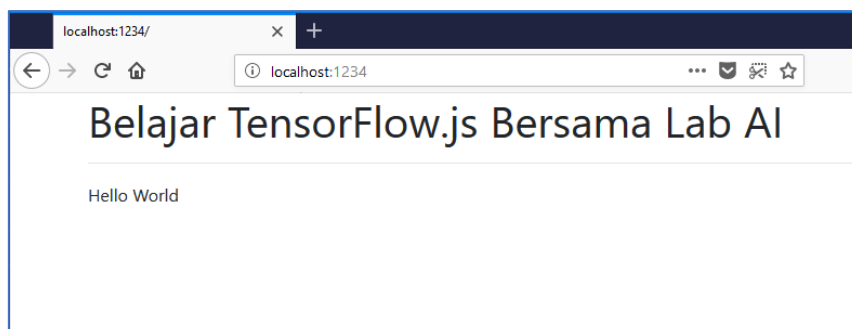
$('#output').text('Hello World')
```

Syntax sederhana tersebut akan menulis teks **"Hello world"** ke dalam elemen dengan ID `output` hanya untuk melihat hasil pertama di layar dan mendapatkan konfirmasi bahwa kode JS sedang diproses dengan benar.

Sekarang mari kita mulai proses membangun server web menggunakan perintah `parcel` sebagai berikut:

```
\ai_tfjs > parcel modul_1\index.html
```

Kita sekarang dapat membuka situs web melalui URL `http://localhost:1234` di browser. Hasilnya akan terlihat seperti berikut:



4.2 Instalasi TensorFlow.js

Untuk menambahkan TensorFlow.js ke proyek kita, kita gunakan NPM lagi dan menjalankan perintah berikut di direktori proyek:

```
\ai_tfjs > npm install @tensorflow/tfjs
```

Perintah tersebut akan mengunduh *library* dan memasangnya ke folder `node_modules`. Setelah menjalankan perintah ini dengan sukses, kita sekarang siap untuk mengintegrasikan *library* Tensorflow.js di dalam file `index.js` dengan menambahkan pernyataan `import` berikut:

```
File: index.js
import 'bootstrap/dist/css/bootstrap.css'
import $ from 'jquery'

import * as tf from '@tensorflow/tfjs'

$('#output').text('Hello World')
```

setelah kita mengimpor TensorFlow.js sebagai `tf`, sekarang memiliki akses ke TensorFlow.js API dengan menggunakan objek `tf` dalam kode kita.



5 Library Visualisasi Data

Terkadang, saat kita bermain-main dengan Pembelajaran Mesin, visualisasi data merupakan alat bantu yang sangat penting dalam memahami data dan model yang akan kita bangun. Dan menggunakan JavaScript, kita bisa memilih berbagai macam *library* untuk menampilkan visualisasi data, misalnya menggunakan `Chart.js`.

TensorFlow sendiri memiliki *library* JavaScript untuk memvisualisasikan data bernama `tfjs-vis`, namun sayangnya *library* ini masih dalam tahap pengembangan dan saat ini belum setangguh *library-library* lainnya. Meski begitu, developer TensorFlow.js berharap nantinya *library* ini dapat berkembang dan menjadi semakin lebih mudah digunakan bersama TensorFlow.js. Untuk mengenal lebih jauh mari kita install kedua *library* tersebut menggunakan `npm`.

5.1 Chart.js

`Chart.js` adalah *open-source library* yang cukup populer digunakan untuk membantu memplot (memvisualisasikan) data dalam aplikasi web. *Library* ini sangat mudah untuk dikustomisasi, tetapi mengonfigurasi beberapa opsi yang ditawarkan terkadang masih cukup rumit.

5.2 tfjs-vis

`tfjs-vis` adalah *library* kecil untuk memvisualisasikan data di browser yang dimaksudkan untuk digunakan dengan TensorFlow.js agar kita dapat lebih mudah untuk memahami apa yang terjadi di dalam proses Pembelajaran Mesin. *Library* ini dibangun dengan tujuan agar bisa fleksibel dan digabung dengan *library* visualisasi lain seperti `d3`, `Chart.js` atau `plotly.js`. Fitur utamanya dari `tfjs-vis` adalah:

- Memvisualisasikan perilaku model saat proses pelatihan
- Memiliki Fungsi khusus untuk memvisualisasikan objek spesifik dari TensorFlow.js
- Memberikan visualisasi yang tidak akan mengganggu aplikasi web yang kita buat
- Memberikan tempat khusus untuk visualisasi bernama `visor` yang tidak mengganggu halaman web.

5.3 Instalasi Chart.js dan tfjs-vis

Untuk memulai instalasi, kita bisa menghentikan dulu hosting parcel yang sedang berjalan, atau buka *command prompt* baru dan navigasikan ke folder project. Setelah itu, lakukan instalasi kedua *library* dengan memanggil

```
\ai_tfjs > npm install @tensorflow/tfjs-vis
\ai_tfjs > npm install chart.js
```

Setelah menjalankan perintah ini dengan sukses, kita sekarang siap untuk mengintegrasikan *library* dengan menambahkan syntax `import` berikut



File: `index.js`

```
import 'bootstrap/dist/css/bootstrap.css'
import $ from 'jquery'
import * as tf from '@tensorflow/tfjs'

import * as tfvis from '@tensorflow/tfjs-vis'
import Chart from 'chart.js'

$('#output').text('Hello World')
```

5.4 Container Visualisasi

Untuk memvisualisasikan data pada halaman web, kita harus menyediakan tempat di `index.html`. `Chart.js` membutuhkan `<canvas>` untuk menempatkan grafik, sementara `tfjs-vis` hanya membutuhkan section `<div>` untuk menampilkan *inline* di dalam halaman. `Tfjs-vis` juga menyediakan fitur window `visor` yang akan menampilkan visualisasi di sebuah tab baru luar halaman utama.

Untuk mencoba semua pilihan, mari kita tambahkan section baru di dalam `index.html` sebagai berikut

File: `index.html`

```
<html>
<body>
  <div class="container">
    <div class="card-body">
      <h1>Belajar TensorFlow.js Bersama Lab AI</h1>
    </div><hr>

    <div class="card-body">
      <h4>Scatter Plot Data (tfjs-vis)</h4>
      <div id="scatter-tfjs"></div>
      <hr><br>

      <h4>Scatter Plot Data (Chart.js)</h4>
      <canvas id="scatter-chartjs" height="400" width="500">
      </canvas>
    </div><hr>

    <div id="output"/>
  </div>
  <script src="./index.js"></script>
</body>
</html>
```



5.5 Contoh Data

Pada teknik Pembelajaran Mesin, terdapat beberapa istilah/penamaan dari data yang digunakan. Secara umum data terbagi menjadi tiga jenis: Data Latih, Data Uji, dan Data Validasi. Data Latih adalah data yang digunakan untuk membangun model pembelajaran mesin, dan Data Validasi adalah data yang digunakan untuk mengecek apakah model sudah pintar (memiliki performa/akurasi bagus). Sementara Data Uji adalah data baru yang belum pernah dipakai selama proses pelatihan.

Pada contoh ini, mari kita buat sekumpulan Data Latih sederhana dengan menambahkan kode berikut ke dalam `index.js`.

```
const ar_data    = [-1, 0, 1, 2, 3, 4]
const ar_target = [-3, -1, 1, 3, 5, 7]
```

Selanjutnya, kita ubah array JavaScript tersebut menjadi sebuah tensor agar bisa diproses oleh TensorFlow.js. Untuk itu, masukkan `ar_data` dan `ar_target` ke dalam tensor berukuran `[6, 1]` sebagai berikut

```
const x = tf.tensor2d(ar_data, [6, 1])
const y = tf.tensor2d(ar_target, [6, 1])
```

5.6 Visualisasi Data

Untuk memvisualisasikan data sebagai plot *scatter*, pada umumnya *library* JavaScript mengharuskan kita mengubah bentuk data menjadi format *dictionary* titik *x* dan *y*. Untuk itu, kita buat sebuah fungsi yang akan memetakan data dari 2 array menjadi format titik sebagai berikut ke dalam `index.js`.

Selanjutnya kita panggil fungsi tersebut untuk mengubah data `ar_data` dan `ar_target`

```
let zip = (ar1, ar2) => ar1.map((x, i) => { return { 'x': x, 'y': ar2[i], } })

const data_train = zip(ar_data, ar_target)
const label_train = ['trainset'] //label tertampil pada chart
```

Untuk menampilkan scatter plot data pada container menggunakan `tfjs-vis`, tambahkan kode berikut

```
//TFJS-VIS
let data = { values: [data_train], series: label_train }

const container = $('#scatter-tfjs')[0]
tfvis.render.scatterplot(container, data, { width:500, height:400 } )
```



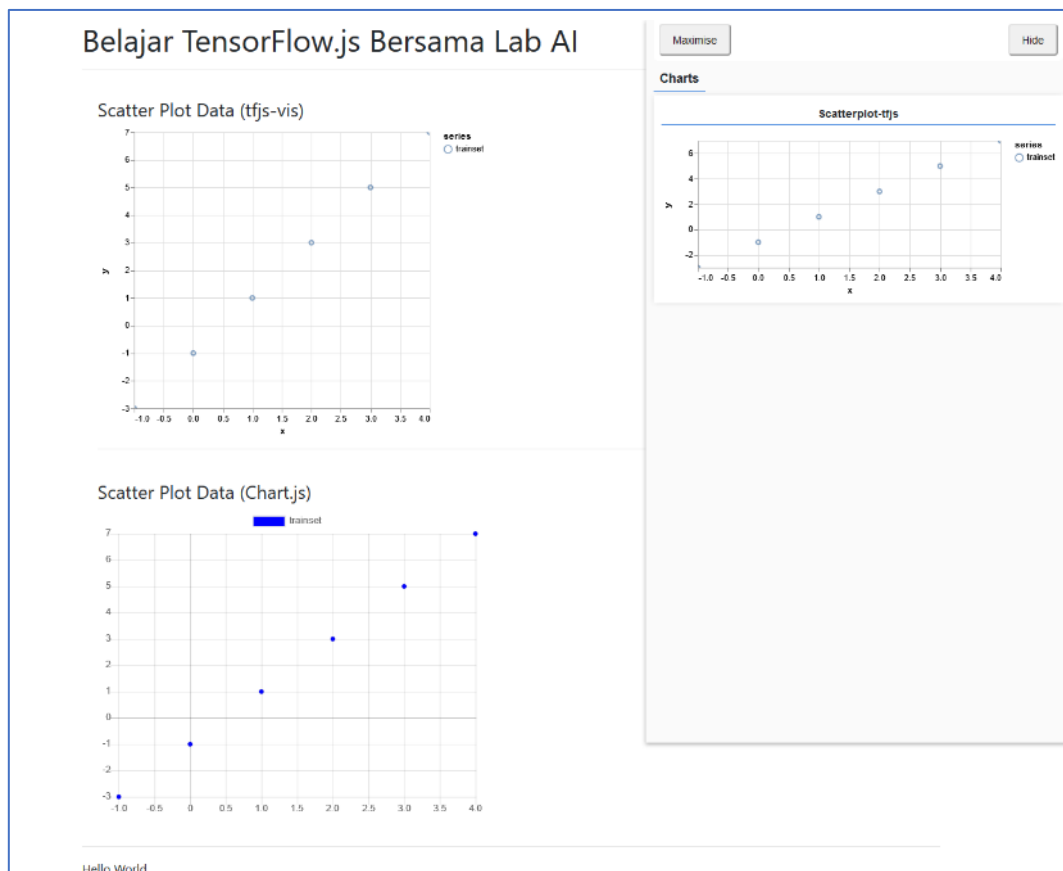
Untuk menampilkan scatter plot data ke dalam **visor** menggunakan **tfjs-vis**, gunakan kode berikut

```
//TFJS-VIS VISOR
const surface = tfvis.visor().surface({
  name: 'Scatterplot-tfjs', tab: 'Charts' })
tfvis.render.scatterplot(surface, data);
```

Untuk menampilkan scatter plot ke dalam canvas menggunakan **Chart.js**, tambahkan kode berikut

```
// CHART.JS
var ctx = $('#scatter-chartjs')
var scatterChart = new Chart(ctx, {
  type: 'scatter',
  data: {
    datasets: [{
      data: data_train,
      label: label_train,
      backgroundColor: 'red' }]
  },
  options: { responsive: false }
})
```

Hasil tampilan dari ketiga scatterplot dapat dilihat sebagai berikut



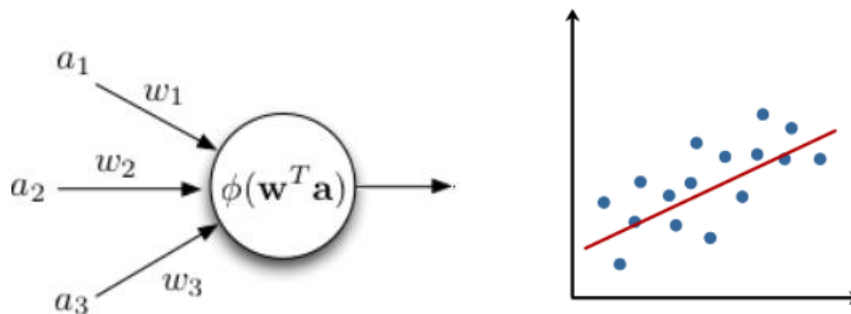
Untuk selanjutnya, kita akan lebih gunakan **Chart.js** untuk menampilkan visualisasi data, sementara menggunakan **tfjs-vis** untuk menampilkan visualisasi proses pembelajaran

6 Regresi Linier dengan TensorFlow.js

Kini mari kita mulai dengan membangun Pembelajaran Mesin yang paling sederhana. Kita mulai dengan regresi linier, hanya untuk mengecek apakah TensorFlow.js benar-benar telah berjalan.

6.1 Regresi Linier

Regresi Linier adalah fungsi statistik yang menjadi dasar hampir segala algoritma pembelajaran mesin. Fungsi ini bertujuan untuk membuat sebuah fungsi garis lurus yang dapat menebak/memperkirakan output dari suatu input berdasarkan data-data yang diberikan. Jika kita lihat data yang sudah kita buat sebelumnya, regresi linier akan berusaha untuk membuat garis lurus yang bisa mendekati atau menyentuh semua data.



Sebelum melatih model jaringan, mari kita percantik tampilan untuk mempermudah pemahaman.

6.2 Tampilan Muka

Mula-mula kita hapus bagian visualisasi tfjs-vis dan hanya sisakan tempat visualisasi untuk Chart.js

File: `index.html`

```
<body>
  <div class="container">
    <div class="card-body">
      <h1>Belajar TensorFlow.js Bersama Lab AI</h1>
    </div><hr>

    <div class="card-body">
      <h4>Scatter Plot Data (Chart.js)</h4>
      <canvas id="scatter-chartjs" height="400" width="500">
      </canvas>
    </div><hr>

    <!-- Training Process -->
    ...

  </div>
  <script src="./index.js"></script>
</body>
```

Proses Pelatihan terdiri dari tahap inialisasi model, pelatihan model (*training*), dan pengujian (*testing*). Karena input setiap data hanya memiliki satu atribut, maka mari kita ubah dan tambahkan kode berikut ke dalam `index.html`

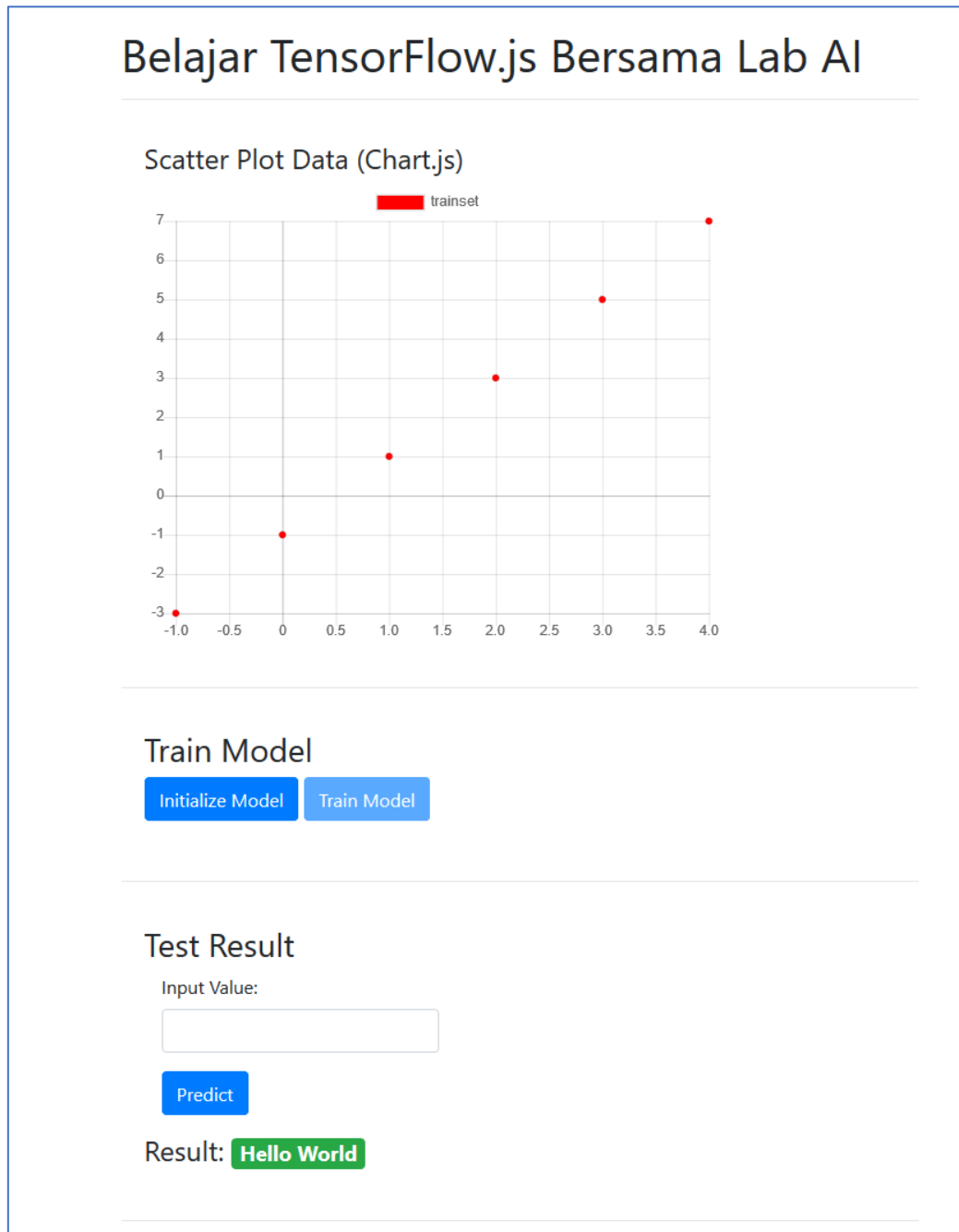
File: `index.html`

```
<!-- Training Process -->
<div class="card-body">
  <div class="form-group">
    <h3>Train Model</h3>
    <button type="button" class="btn btn-primary"
      id="init-btn">Initialize Model</button>
    <button type="button" class="btn btn-primary"
      id="train-btn" disabled>Train Model</button>
  </div>
  <h4><span id="is-training" class="badge"/></h4>
</div><hr>

<!-- Testing Process -->
<div class="card-body" id="predict" style="display:none;">
  <h3>Test Result</h3>
  <div class="col-5 form-group">
    <label>Input Value:</label>
    <input type="text" id="inputValue" class="form-control">
  </div>
  <div class="col-5 form-group">
    <button type="button" class="btn btn-primary"
      id="predict-btn">Predict</button>
  </div>
  <h4>Result: <span class="badge badge-success"
    id="output"/></h4>
</div>
```

Terdapat 3 tombol masing-masing untuk inialisasi model, melatih model, dan menguji model (*predict*). Kita desain tombol **Train disabled** agar proses pelatihan tidak dapat dimulai sampai model sudah diinisialisasi, dan body bagian pengujian kita sembunyikan hingga model selesai dilatih.

Tampilan halaman [index.html](#) jika body pengujian ditampilkan akan menjadi seperti gambar di bawah



6.3 Inisialisasi Model

Sekarang mari kita tambahkan fungsi pada tombol **Initialize Model** untuk menginisialisasi jaringan kita. Mula-mula, kita tambahkan variable global penampung model jaringan. Selanjutnya kita buat *event Listener* untuk tombol **init-btn**.

Di dalamnya, kita definisikan arsitektur jaringan kita sebagai model **sequential()**. Karena data yang sangat mudah dengan input 1 dimensi, maka kita hanya membutuhkan satu layer jaringan saraf dengan satu neuron saja. Kemudian model akan kita latih dengan *Stochastic Gradient Descent* dengan perhitungan *Mean Squared Error*. Maka, kita tambahkan kode sebagai berikut

```
let model
$('#init-btn').click(function() {
  model = tf.sequential()
  model.add(tf.layers.dense({units: 1, inputShape: [1]}))
  model.compile({loss: 'meanSquaredError', optimizer: 'sgd'})
  ...
})
```

Dengan kode di atas, saat tombol ditekan **TensorFlow.js** akan menginisialisasi model jaringan secara acak. Untuk melihat bagaimana garis regresi yang dibuat pada saat inisialisasi, mari kita tambahkan sebuah fungsi untuk menambahkan garis prediksi ke dalam scatter plot data latih sebagai berikut.

```
function viewPrediction(scatterDt, lineDt ){
  scatterChart.destroy()
  scatterChart = new Chart(ctx, {
    type: 'line',
    data: {
      labels: ar_data,
      datasets: [{
        type: 'line',
        label: 'prediction',
        data: lineDt,
        fill: false,
        borderColor: 'blue',
        pointRadius: 0
      }, {
        type: 'bubble',
        label: 'training data',
        data: scatterDt,
        backgroundColor: 'red',
        borderColor: 'transparent'
      }
    ]
  },
  options: { responsive: false }
})
}
```

Selanjutnya kita tambahkan kode di dalam *Event Listener* tombol **Init-btn**. Mula-mula kita lakukan prediksi menggunakan model yang belum dilatih, terhadap tensor data latih *x* dengan memanggil fungsi **predict()**.



Hasil dari fungsi `predict()` merupakan sebuah tensor, dan untuk mengubah kembali menjadi array JavaScript, gunakan fungsi `dataSync()`.

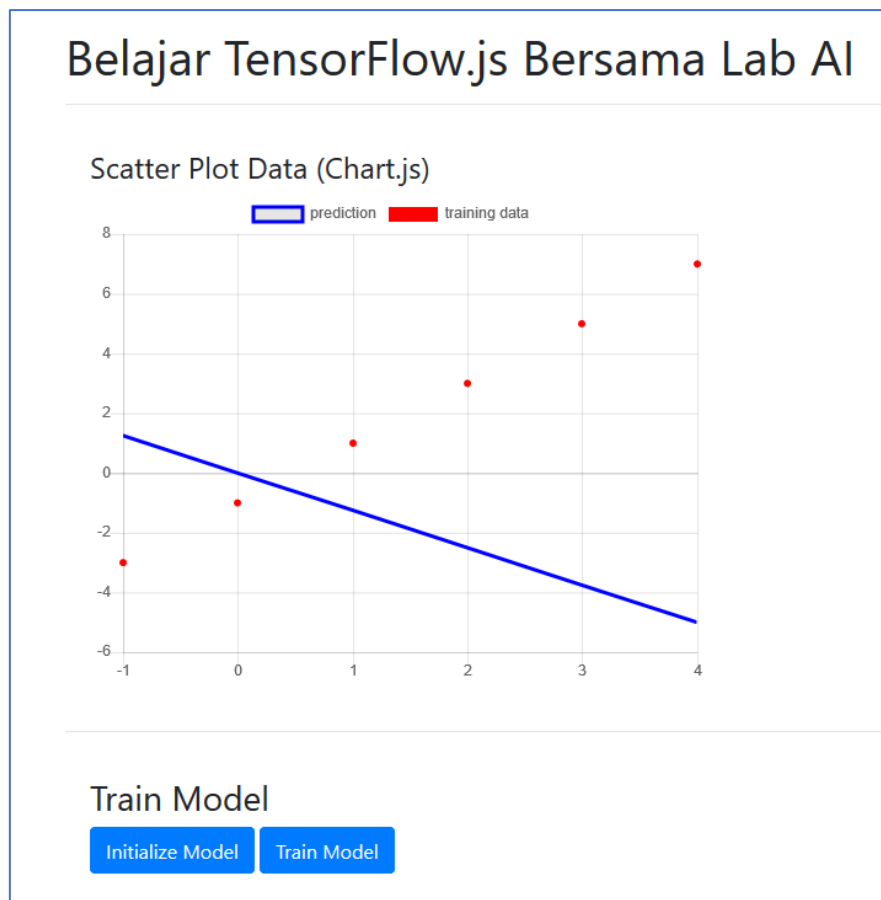
Kemudian, untuk bisa menampilkan ke grafik, kita gabungkan array `y_pred` dengan array `ar_data` menjadi data point. Terakhir, kita panggil fungsi `viewPrediction()` yang sudah dibuat sebelumnya dan nyalakan tombol Train.

```
let model
$('#init-btn').click(function() {
  model = tf.sequential()
  model.add(tf.layers.dense({units: 1, inputShape: [1]}))
  model.compile({loss: 'meanSquaredError', optimizer: 'sgd'})

  let t_pred = model.predict(x)
  let y_pred = t_pred.dataSync()
  let data_pred = zip(ar_data, y_pred)

  viewPrediction(data_train, data_pred)
  $('#train-btn').prop('disabled', false)
})
```

Sekarang, dengan menekan tombol Initialize Model, grafik akan berubah dan bertambah dengan garis prediksi acak dari model yang belum dilatih.



6.4 Melatih Model

Proses Pelatihan Jaringan pada dasarnya dapat diartikan bahwa jaringan akan “memutar” dan “menggeser” garis regresi untuk bisa mendapatkan garis yang memiliki jarak terkecil terhadap semua data latih yang ada. Untuk memulai Proses Pelatihan Jaringan, mari kita tambahkan *Event Listener* untuk tombol train. Di dalam *Event Listener*, kita tambahkan pesan bahwa model sedang dilatih agar user bisa menunggu.

```
$('#train-btn').click(function() {
  var msg = $('#is-training')
  msg.toggleClass('badge-warning')
  msg.text('Training, please wait...')
  ...
})
```

Selanjutnya kita panggil fungsi `fit()` untuk melatih model. Fungsi `fit()` dalam *TensorFlow.js* merupakan fungsi *asynchronous*, yang berarti fungsi berjalan menggunakan loop event sehingga fungsi dapat digunakan tanpa memblokir/menghambat thread UI utama saat berjalan di browser. Karena fungsi `fit()` bersifat *async*, kita bisa menambahkan method `then()` untuk mendefinisikan blok kode yang akan dijalankan begitu proses fungsi *async* `fit()` selesai.

Di sini, mari kita set agar model untuk dilatih selama *20 epoch* (iterasi). Kemudian kita tambahkan method `then()` yang akan menggambarkan garis regresi hasil pelatihan seperti sebelumnya. Selain itu, kita tambahkan kode untuk memanggil fungsi `evaluate()` untuk menghitung dan menampilkan error MSE dari model yang sudah dilatih. Setelah proses pelatihan selesai, kita tampilkan blok body *predict*.

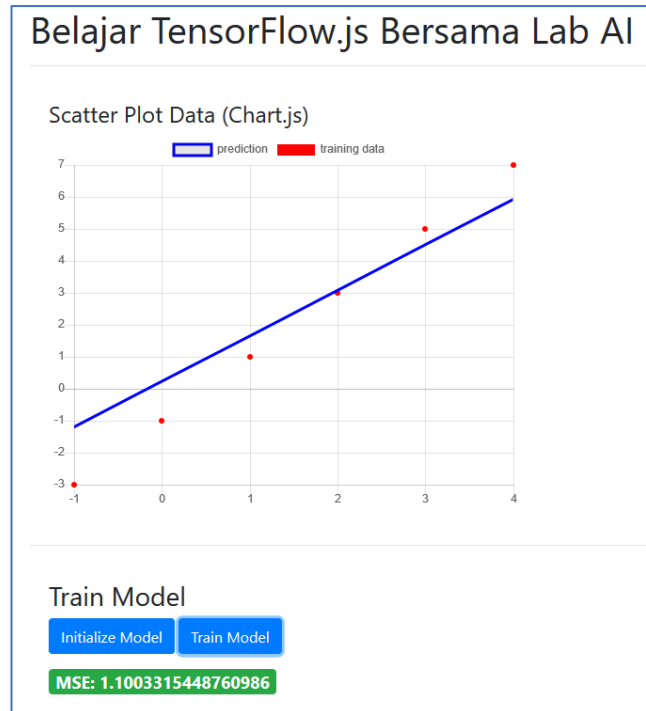
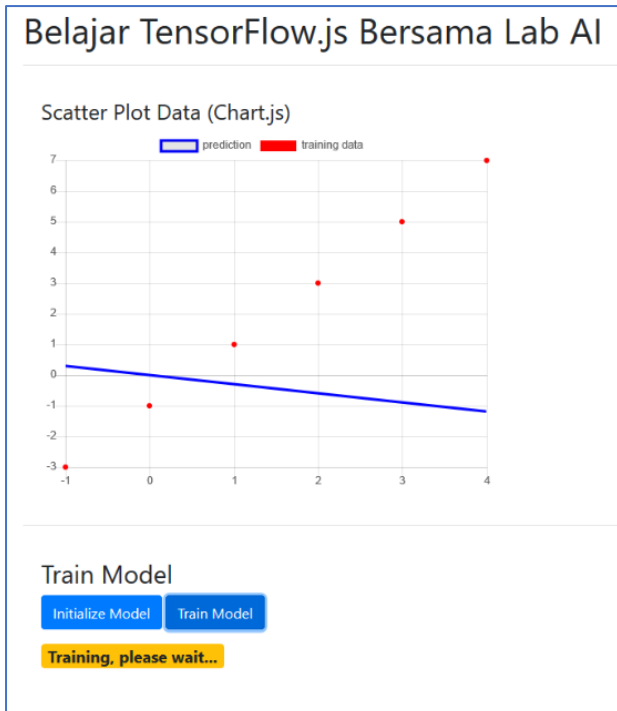
```
$('#train-btn').click(function() {
  var msg = $('#is-training')
  msg.toggleClass('badge-warning')
  msg.text('Training, please wait...')

  model.fit(x, y, {epochs: 20}).then((hist) => {
    let t_pred = model.predict(x)
    let y_pred = t_pred.dataSync()
    let data_pred = zip(ar_data, y_pred)
    viewPrediction(data_train, data_pred )

    let mse = model.evaluate(x, y);

    msg.removeClass('badge-warning').addClass('badge-success')
    msg.text('MSE: ' + mse.dataSync())
    $('#predict').show()
  })
})
```

Gambar proses pelatihan hingga selesai akan terlihat seperti di bawah



6.5 Melatih Model Lebih Lanjut

Dapat dilihat bahwa dari hasil pelatihan, model sudah mulai menyesuaikan dengan data latih, namun mungkin terlihat belum cukup baik. Kalian dapat meneruskan proses pelatihan dengan menekan ulang tombol train hingga MSE mencapai sekecil mungkin, atau tingkatan nilai *epoch* pada *Event Listener* tombol Train.

Cara lain yang bisa dilakukan adalah mendefinisikan nilai *learning rate* pada inialisasi jaringan. *Learning rate* pada jaringan saraf tiruan menyatakan besarnya pergerakan/pergeseran dari model jaringan. Saat kita menggunakan *Stochastic Gradient Descent*, TensorFlow.js secara default menetapkan nilai *learning rate* sebesar *0.01*. Untuk mengubah nilai *learning rate* tersebut, kita dapat ubah kode pada *Event Listener* tombol inialisasi sebagai berikut

```
let model = tf.sequential()
$('#init-btn').click(function() {

  model.add(tf.layers.dense({units: 1, inputShape: [1]}))

  //model.compile({loss: 'meanSquaredError', optimizer: 'sgd'})

  const mySGD = tf.train.sgd(0.1)
  model.compile({loss: 'meanSquaredError', optimizer: mySGD})

  let t_pred = model.predict(x)
  let y_pred = t_pred.dataSync()
  let data_pred = zip(ar_data, y_pred)

  viewPrediction(data_train, data_pred)
  $('#train-btn').prop('disabled', false)
})
```

6.6 Menampilkan History Pelatihan

Selama proses pelatihan, model jaringan akan menggeser garis regresi ke posisi yang memberikan *error* MSE yang lebih kecil. Perubahan *error* atau *loss* ini tersimpan dalam history sebagai output dari fungsi `fit()` yang telah kita simpan dalam variabel `hist`. Kita dapat menampilkan grafik history tersebut ke dalam visor tfjs-vis dengan menambahkan kode berikut di dalam fungsi fit pada Event Listener tombol Train

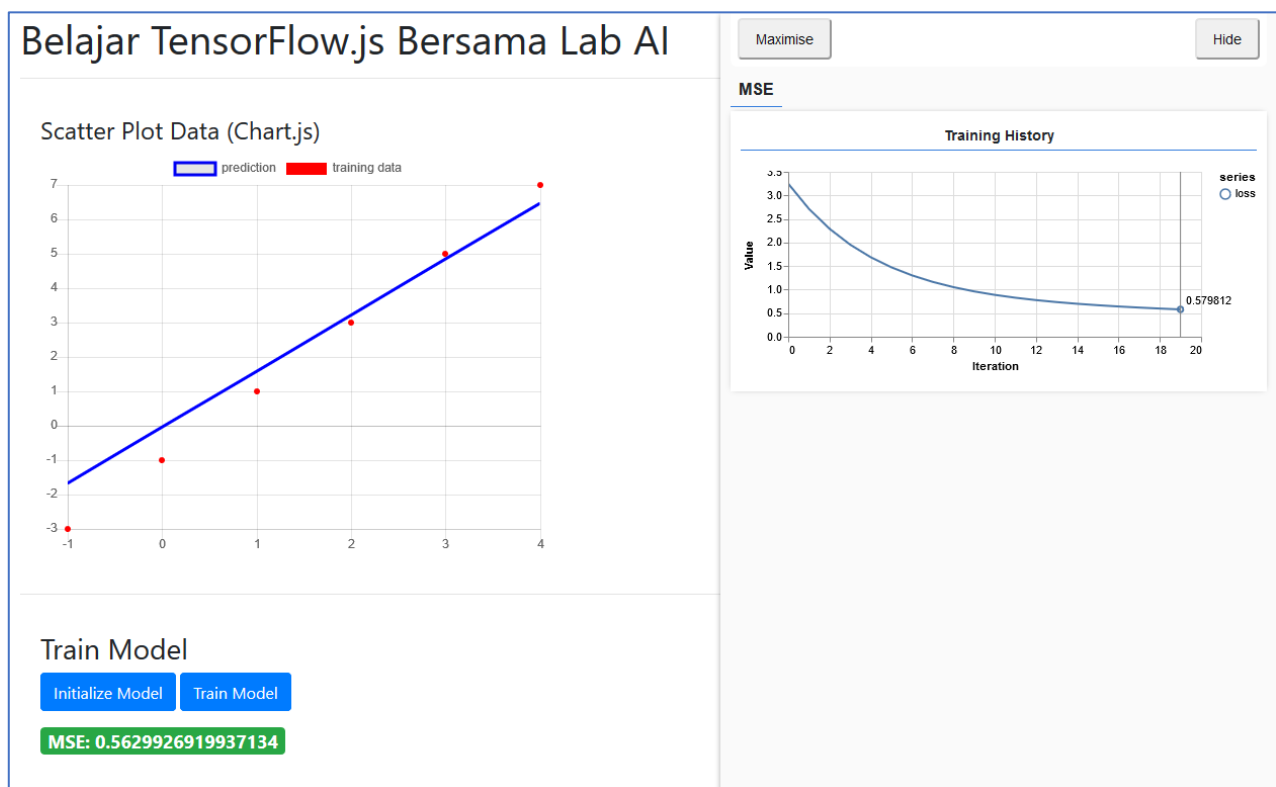
```
model.fit(x, y, {epochs: 20}).then((hist) => {
  let t_pred = model.predict(x)
  let y_pred = t_pred.dataSync()
  let data_pred = zip(ar_data, y_pred)
  viewPrediction(data_train, data_pred)

  let mse = model.evaluate(x, y)

  msg.removeClass('badge-warning').addClass('badge-success')
  msg.text('MSE: ' + mse.dataSync())
  $('#predict').show()

  const surface = tfvis.visor().surface({
    name: 'Training History', tab: 'MSE' })
  tfvis.show.history(surface, hist, ['loss'])
})
```

Berikut adalah contoh tampilan dari visor history yang terbentuk



6.7 Menguji Model

Setelah model dilatih dengan baik, kita dapat melakukan pengujian terhadap data baru. Mari kita tambahkan *Event Listener* untuk tombol *predict*. Pertama, saat tombol ditekan, kita ambil nilai dari dialog input, kemudian ubah input menjadi tensor dan panggil fungsi `predict()`. Selanjutnya kita kembalikan nilai hasil prediksi ke halaman

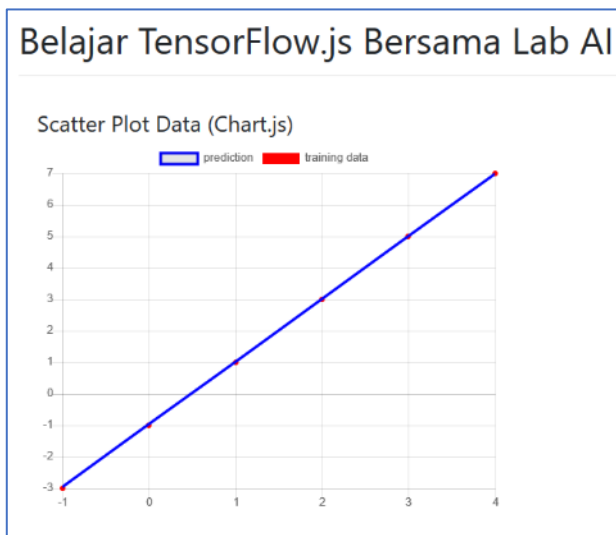
```
$('#predict-btn').click(function() {

    var num = parseFloat($('#inputValue').val())
    let y_pred = model.predict(tf.tensor2d([num], [1,1]))

    $('#output').text(y_pred.dataSync())

})
```

Jika kita coba, maka model akan mengembalikan hasil prediksi dengan baik



Train Model

Initialize Model Train Model

MSE: 0.0007086227997206151

Test Result

Input Value:

5

Predict

Result: 8.100218772888184

Belajar TensorFlow.js Bersama lab AI

Modul II

Klasifikasi Biner Jaringan Saraf Tiruan 2 Layer dengan TensorFlow.js



Artificial Intelligence Laboratory
Fakultas Informatika
Universitas Telkom
Bandung, April 2019

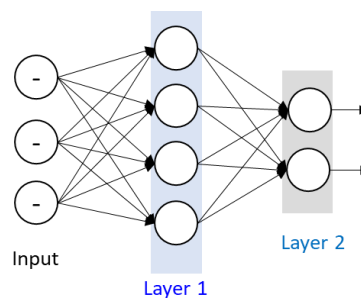
Modul II

JST 2 Layer untuk Klasifikasi Biner dengan TensorFlow.js

Setelah kita mencoba melatih model regresi linier dengan [TensorFlow.js](#), mari kita coba melatih model untuk kasus selanjutnya, yaitu klasifikasi untuk dua kelas data dengan Jaringan Saraf Tiruan 2 layer.

1 Jaringan Saraf Tiruan 2 Layer

Secara singkat, Jaringan Saraf Tiruan dapat dilihat sebagai tumpukan beberapa fungsi linier. Di antara setiap fungsi linier diberikan fungsi non linier disebut fungsi aktivasi. Dengan menumpuk beberapa fungsi linier menjadi satu, kita akan mendapatkan sebuah model yang memiliki kemampuan lebih baik daripada hanya menggunakan satu layer saja.



Jaringan Saraf Tiruan 2 Layer artinya terdapat 2 tumpukan fungsi linier ([dense](#)) dengan fungsi aktivasi umumnya yang dipakai adalah fungsi [sigmoid](#).

2 Data Klasifikasi Biner (dua kelas)

Untuk contoh klasifikasi kali ini, mari kita bangkitkan sekumpulan Data Latih. Karena JavaScript hanya bisa melakukan random normal, maka telah diberikan fungsi-fungsi bantuan di dalam file JavaScript [data.js](#) untuk melakukan random pseudo-gaussian.

File: [data.js](#)

```
function gaussianRand(a,b) {
  var rand = 0
  for (var i = 0; i < 6; i++) {
    rand += Math.random()
  }
  return (rand / 6)+(Math.random()*a+b)
}
...
```


Fungsi `generateData` akan membangkitkan sejumlah `numPoint` data dua dimensi dengan proporsi antara kelas 0 dan 1 sebesar `class_frac`. Data dibangkitkan dalam bentuk data point. Kemudian dari sejumlah `numPoint` data yang dibangkitkan, 30% data akan dijadikan sebagai Data Validasi.

Data dikembalikan dalam bentuk array untuk proses pelatihan dan dalam bentuk data titik untuk visualisasi.

File: `data.js`

```
...

// convert array of data point into 2d array
function toArray(arr){
  var out = [];
  for(var i = 0; i < arr.length; i++){
    out.push([arr[i].x, arr[i].y])
  }
  return out
}

export function generateData(numPoints, class_frac){
  // class proportion
  let nx1 = Math.round(numPoints*class_frac)
  let nx2 = numPoints - nx1

  // create random 2 dimension data
  let data1 = Array(nx1).fill(0).map(() =>
    { return{'x':gaussianRand(10,7),'y':gaussianRand(10,7) } })
  let class1 = Array(nx1).fill(0)

  let data2 = Array(nx2).fill(0).map(() =>
    { return{'x':gaussianRand(9,0),'y':gaussianRand(9,0) } })
  let class2 = Array(nx2).fill(1)

  // split 30% for data validation
  let nv1 = Math.round(nx1*.7)
  let nv2 = Math.round(nx2*.7)

  // data point for visualization
  let trainPt = [data1.slice(0,nv1), data2.slice(0,nv2)]
  let valPt = [data1.slice(nv1), data2.slice(nv2)]

  // array data for TensorFlowJS
  let x_train = toArray(trainPt[0].concat(trainPt[1]))
  let y_train = class1.slice(0,nv1).concat(class2.slice(0,nv2))
  let x_val = toArray(valPt[0].concat(valPt[1]))
  let y_val = class1.slice(nv1).concat(class2.slice(nv2))

  return { x_train, y_train, x_val, y_val, trainPt, valPt }
}
```



3 Tampilan Muka

Berikutnya, mari kita bangun tampilan muka baru untuk kasus klasifikasi ini pada file html baru. Di sini, mari kita beri nama file `index2.html`. Tampilan muka akan terdiri dari **3 bagian**. Bagian pertama merupakan tempat kita menampilkan persebaran data latih dan data validasi yang akan kita gunakan. Bagian kedua akan kita gunakan untuk tempat menginisiasi, melatih, dan menguji model jaringan. Sedangkan bagian ketiga akan kita gunakan untuk menampilkan progres selama proses pelatihan.

File: `index2.html`

```
<html>
  <body>
    <div class="container">
      <div class="card-body">
        <h1>Binary Classifier menggunakan TensorFlow.js</h1>
      </div><hr>

      <!-- Chart Data -->
      <div class="row" >
        ...
      </div><hr>

      <!-- Train and Test -->
      <div class="row">
        ...
      </div><hr>

      <!-- Training Progress -->
      <div class="card-body" id="training-progress">
        ...
      </div>

    </div>

    <script src="./index2.js"></script>

  </body>
</html>
```



3.1 Tampilan Visualisasi

Untuk bagian visualisasi data, kita sediakan dua **canvas** di dalam dua kolom untuk visualisasi Data Latih dan Data Validasi.

File: `index2.html`

```
...
<!-- Chart Data -->
<div class="row" >
  <div class="col-5">
    <canvas id="trainset" height="400" width="400"></canvas>
    <figcaption class="text-center">Data Latih</figcaption>
  </div>

  <div class="col-5">
    <canvas id="valset" height="400" width="400"></canvas>
    <figcaption class="text-center">Data Validasi</figcaption>
  </div>
</div><hr>
...
```

Sementara pada **BAGIAN KETIGA** untuk visualisasi progres pelatihan, sediakan juga dua **div** masing-masing untuk graf **loss** dan akurasi seperti berikut. Bagian proses pelatihan dan pengujian akan kita isi pada tahap berikutnya.

```
...
<!-- Training Progress -->
<div class="card-body" id="training-progress">
  <h3>Training Progress</h3>
  <div class="row">
    <div class="col-5 border-right">
      <div id="loss-graph"></div>
      <figcaption class="text-center">Loss</figcaption>
    </div>

    <div class="col-5">
      <div id="acc-graph"></div>
      <figcaption class="text-center">Accuracy</figcaption>
    </div>
  </div><br><hr>
</div>
...
```

3.2 Tampilan Menu Pelatihan dan Pengujian

Pada bagian pelatihan dan pengujian, tambahkan masing-masing satu kolom. Dalam kasus klasifikasi kali ini kita akan menggunakan jaringan dengan arsitektur 2 layer. Layer terakhir memiliki 1 *neuron* (kelas biner), sementara pada layer pertama (*hidden layer*) kita set jumlah *neuron* dinamis sesuai input user.

Untuk itu, pada kolom pelatihan kita tambahkan input text untuk mengubah jumlah *hidden neuron*. Lalu kita berikan juga input text untuk mengubah *learning rate* dan maksimum *epoch*. Pada bagian ini kita juga tambahkan dua tombol untuk inisialisasi model dan memulai pelatihan, serta dua **badge** untuk menampilkan akurasi data latih dan juga akurasi data validasi hasil pelatihan.

File: `index2.html`

```
...
<!-- Train and Test -->
<div class="row">

  <!-- Train Col -->
  <div class="col-5">
    <h3>Train Model</h3>
    <div class="form-inline form-group">
      <label class="col-3">Hidden Neuron:</label>
      <input id='num-hid' class="form-control"
        type="text" value="20">
    </div>
    <div class="form-inline form-group">
      <label class="col-3">Learning Rate:</label>
      <input id='lr' class="form-control"
        type="text" value="0.1">
    </div>
    <div class="form-inline form-group">
      <label class="col-3">Max Epoch:</label>
      <input id='epoch' class="form-control"
        type="text" value="100">
    </div>
    <div class="form-group">
      <button type="button" class="btn btn-primary"
        id="init-btn">Initialize Model</button>
      <button type="button" class="btn btn-primary"
        id="train-btn" disabled>Train Model</button>
    </div>
    <h4><span id="is-training" class="badge"/></h4>
    <h4><span id="eval-train" class="badge badge-info"/></h4>
    <h4><span id="eval-val" class="badge badge-info"/></h4>
  </div>

  <!-- Test Col -->
  <div class="col-5" id="testing">
    ...
  </div>

</div><hr>
...
```

Pada kolom pengujian, kita tambahkan dua input text untuk menerima sebuah data baru dan tombol untuk melakukan pengujian. Kita sertakan juga sebuah badge untuk menampilkan hasil prediksi kelasnya.

File: `index2.html`

```
...
<!-- Train and Test -->
<div class="row">
  <!-- Train Col -->
  <div class="col-5">
    ...
  </div>

  <!-- Test Col -->
  <div class="col-5" id="testing">
    <h3>Test New Data</h3>
    <div class="form-inline form-group">
      <label class="col-2">X :</label>
      <input id='input-x' class="form-control" type="text">
    </div>
    <div class="form-inline form-group">
      <label class="col-2">Y :</label>
      <input id='input-y' class="form-control" type="text">
    </div>
    <div class="form-inline form-group">
      <button type="button" class="btn btn-primary"
        id="predict-btn" disabled>Test Model</button>
    </div>
    <h4><span id="class-pred" class="badge badge-success"/></h4>
  </div>
</div><hr>
...
```

Perhatikan bahwa tombol Train dan Test kita set *disabled*. Tombol Train akan kita aktifkan nanti jika model sudah diinisialisasi, sedangkan tombol Test akan kita aktifkan saat model sudah dilatih.



4 Index2.js

4.1 Import library dan Generate Data

Sama seperti sebelumnya, pada file `index2.js` kita import semua *library* yang kita butuhkan. Jangan lupa juga kita import fungsi `generateData` dari `data.js`.

Kemudian karena kita akan menampilkan grafik progres pelatihan selama TensorFlowJS melatih jaringan, kita akan menggunakan fungsi `await` dari fungsi `asynchronous fit()`. Agar kita bisa menggunakannya, kita membutuhkan *library* `babel` untuk memproses fungsi asynchronous.

Untuk mengunduh dan menggunakan *library* tersebut, kita tambahkan syntax `require` seperti di bawah ini.

```
File: index.js

import 'bootstrap/dist/css/bootstrap.css'
import * as tf from '@tensorflow/tfjs'
import * as tfvis from '@tensorflow/tfjs-vis'
import Chart from 'chart.js'
import $ from 'jquery'

import {generateData} from './data'
require('babel-polyfill')
...
```

Syntax `require` tersebut akan mengunduh paket *library* `babel` saat dijalankan pertama kali.

Pada tahap ini kita sudah bisa mencoba melihat tampilan muka aplikasi web kita dengan memanggil sintaks

```
\ai_tfjs > parcel modul_2\index2.html
```

Namun dapat dilihat bahwa belum ada data yang tertampil.

Selanjutnya mari kita bangkitkan dataset sebanyak 100 data menggunakan fungsi `generateData`. Di sini misal kita set rasio kelas 0 dan kelas 1 sebesar 60:40. Kemudian untuk masing-masing data latih dan data uji, kita bentuk menjadi tensor sesuai ukurannya masing-masing

```
...
const dataset = generateData(100, 0.6)
const x_train = tf.tensor2d(dataset.x_train, [dataset.x_train.length, 2])
const y_train = tf.tensor2d(dataset.y_train, [dataset.y_train.length, 1])
const x_val   = tf.tensor2d(dataset.x_val,   [dataset.x_val.length, 2])
const y_val   = tf.tensor2d(dataset.y_val,   [dataset.y_val.length, 1])
...
```

4.2 Visualisasi Data

Setelah kita bangkitkan dataset, mari kita visualisasikan kedua data ke `canvas` trainset dan validationset dengan syntax berikut. Perhatikan bahwa pada visualisasi data uji, kita berikan tempat untuk tiga data masing-masing untuk memvisualisasikan data kelas 0, kelas 1, dan data baru input user saat pengujian.



```
...
// Visualisasi Data Latih
var ctx = $('#trainset')
var scatter_train = new Chart(ctx, {
  type: 'scatter',
  data: {
    datasets: [{
      data: dataset.trainPt[0],
      label: 'class 0',
      backgroundColor: 'black'
    }, {
      data: dataset.trainPt[1],
      label: 'class 1',
      backgroundColor: 'red'
    }
  ],
  options: { responsive: false }
})

// Visualisasi Data Validasi
var ctx = $('#valset')
var scatter_val = new Chart(ctx, {
  type: 'scatter',
  data: {
    datasets: [{
      data: dataset.valPt[0],
      label: 'class 0',
      backgroundColor: 'black'
    }, {
      data: dataset.valPt[1],
      label: 'class 1',
      backgroundColor: 'red'
    }, {
      type: 'bubble',
      data: [],
      label: 'new data',
      backgroundColor: '#32fa32',
      borderColor: 'green',
    }
  ],
  options: { responsive: false }
})
...
```



4.3 Event Listener Tombol Initialize

Sekarang mari kita tambahkan Event Listener untuk menginisialisasi model berdasarkan input user pada tombol `init-btn`. Model jaringan kita set dengan input 2 dimensi dan output 1 neuron karena klasifikasi biner (0/1). Sama seperti sebelumnya, kita definisikan variabel global `model` terlebih dahulu. Saat tombol ditekan, ambil integer jumlah *hidden neuron* dan float *learning rate*. Kemudian kita definisikan modelnya.

Setelah terdefinisi, selanjutnya kita `compile` model dengan metode `SGD`, lalu kita aktifkan tombol train. Perhitungan `loss` untuk klasifikasi biner kita gunakan `binaryCrossentropy`, dan metrik perhitungan performansi kita gunakan `accuracy`.

```
...
let model

$('#init-btn').click(function() {
  var num_h = parseInt($('#num-hid').val())
  var lr = parseFloat($('#lr').val())

  model = tf.sequential()
  model.add(tf.layers.dense({units: num_h,
    activation: 'sigmoid', inputShape: [2]}))
  model.add(tf.layers.dense({units: 1, activation: 'sigmoid'}))

  const mySGD = tf.train.sgd(lr)
  model.compile({loss: 'binaryCrossentropy',
    optimizer: mySGD, metrics: ['accuracy']})

  $('#train-btn').prop('disabled', false)
})
...
```

4.4 Event Listener Tombol Train

Untuk menampilkan history progres pelatihan selama fungsi asynchronous `fit` berjalan, kita akan menggunakan fungsi keyword `await`. Pemanggilan fungsi async `fit` menggunakan keyword `await` haruslah dilakukan di dalam fungsi `async` juga. Karena itu kita definisikan fungsi `async` di dalam action event `click` pada tombol `train-btn`.

Di dalam action click tersebut, pertama-tama mari kita tampilkan pesan bahwa model jaringan sedang dilatih seperti berikut ini

```
...
$('#train-btn').click(async() => {
  var msg = $('#is-training')
  msg.toggleClass('badge-warning')
  msg.text('Training, please wait...')
  ...
})
...
```


Selanjutnya untuk menampilkan progres pelatihan, ambil div `loss` dan `acc` sebagai tempat visualisasi. Di samping itu, kita sediakan sebuah array `trainLogs` untuk menampung data log pelatihan.

```
...
const loss = $('#loss-graph')[0]
const acc = $('#acc-graph')[0]
const trainLogs = []
...
```

Kemudian kita ambil value *max epoch* dari input user, dan jalankan fungsi async `fit()` dengan keyword `await`. Pada fungsi `fit()` tersebut, kita berikan data latih `x_train`, `y_train`, dan data validasi `x_val`, `y_val`. Di sini kita berikan juga `callbacks` yang akan memonitor progres dari fungsi setiap epoch.

Pada callback, kita panggil `history` loss dan akurasi, yang kemudian menggunakan fungsi dari library `tfjs-vis`, kita tampilkan history tersebut ke div visualisasi loss dan acc

```
...
const epoch = parseInt($('#epoch').val())
const history = await model.fit(x_train, y_train, {
  epochs: epoch,
  validationData: [x_val, y_val],
  callbacks: {
    onEpochEnd: async (epoch, logs) => {
      trainLogs.push(logs)
      tfvis.show.history(loss, trainLogs,
        ['loss', 'val_loss'], { width: 400, height: 250 })
      tfvis.show.history(acc, trainLogs,
        ['acc', 'val_acc'], { width: 400, height: 250 })
    },
  },
})
...
```

Terakhir, setelah fungsi `fit()` selesai dijalankan, artinya model telah dilatih. Kita tambahkan kode untuk mengevaluasi model guna mendapatkan akurasi data latih dan data validasinya. Selanjutnya kita ubah pesan pada `is-training` bahwa pelatihan telah selesai, tampilkan akurasinya, dan nyalakan tombol prediksi.

```
...
let eval_train = model.evaluate(x_train, y_train)
let eval_val = model.evaluate(x_val, y_val)

msg.toggleClass('badge-warning badge-success')
msg.text('Training Done')

let round = (num) => parseFloat(num*100).toFixed(2)
$('#eval-train').text('Trainset Accuracy : '
  + round(eval_train[1].dataSync())+'%')
$('#eval-val').text('Validation Accuracy : '
  + round(eval_val[1].dataSync())+'%')
$('#predict-btn').prop('disabled', false)
})
...
```

4.5 Event Listener Tombol Testing/Predict

Sekarang kita akan menambahkan *event listener* pada tombol *predict*. Saat tombol ditekan, kita ambil data baru dari user kemudian kita tampilkan posisi data baru di tempat visualisasi data validasi. Selanjutnya kita prediksi kelas dari data baru tersebut menggunakan model yang telah kita latih menggunakan fungsi `predict()`.

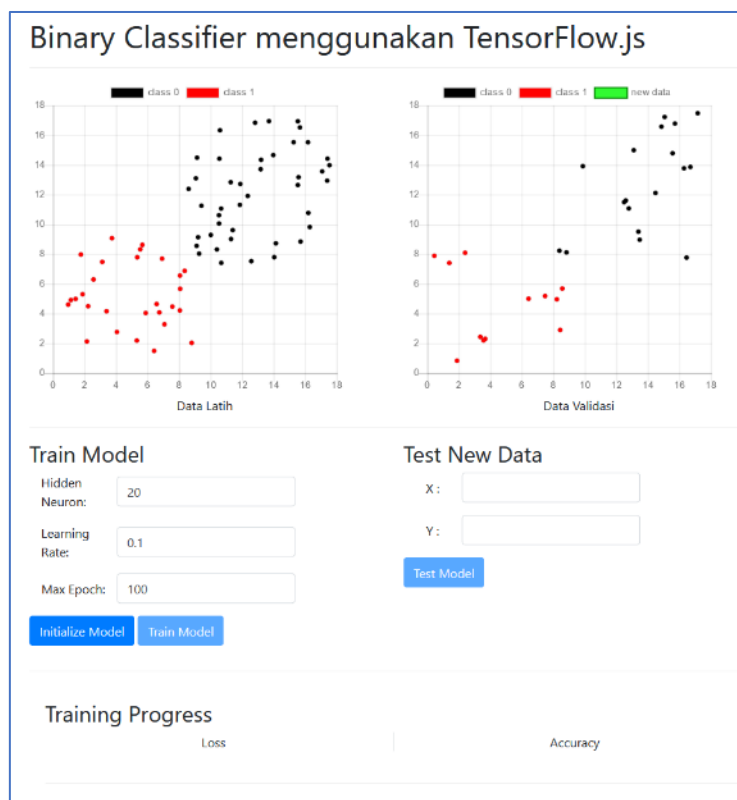
```
...
$('#predict-btn').click(function() {
  var x = parseFloat($('#input-x').val())
  var y = parseFloat($('#input-y').val())

  let new_dt = {'x':x, 'y':y, 'r':5}
  scatter_val.data.datasets[2].data[0] = new_dt
  scatter_val.update()

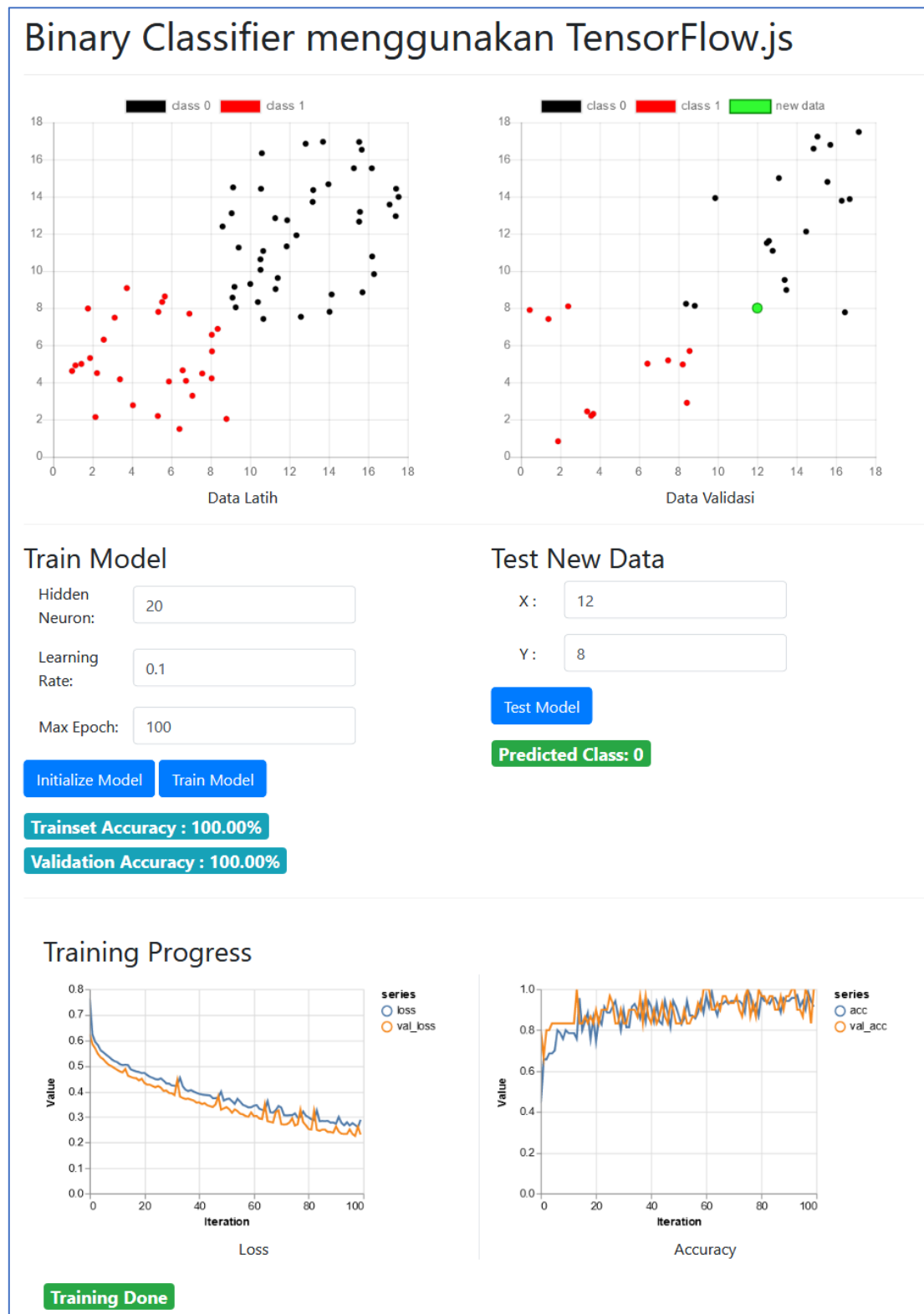
  let y_pred = model.predict(tf.tensor2d([[x,y]], [1,2]))
  let class_pred = 'Predicted Class: '+Math.round(y_pred.dataSync())
  $('#class-pred').text(class_pred)
})
...
```

5 Hasil akhir

Binary classifier kita siap untuk dicoba. Tampilan akhir dari [index2.html](#) jika dijalankan menggunakan syntax `parcel modul_2\index2.html`, maka akan tertampil seperti gambar berikut



Kemudian, jika kita jalankan pelatihan dan pengujian, maka akan terlihat seperti gambar berikut



Belajar TensorFlow.js Bersama lab AI

Modul III

Klasifikasi Multi-Kelas dengan TensorFlow.js



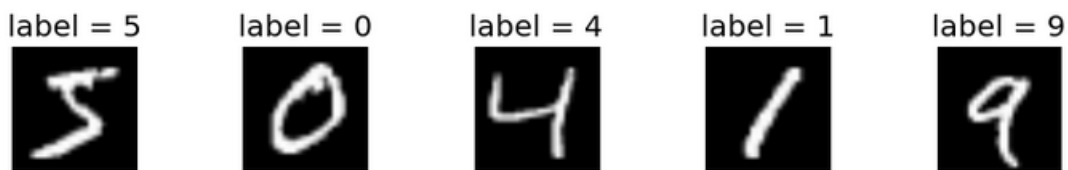
Artificial Intelligence Laboratory
Fakultas Informatika
Universitas Telkom
Bandung, April 2019

Modul III

Klasifikasi Multi-Kelas dengan TensorFlow.js

1 MNIST Dataset

Setelah kita bisa membangun sistem klasifikasi untuk dua kelas data, mari kita naik level ke kasus yang lebih menantang, yaitu jika kelas yang akan diklasifikasi lebih dari dua. Data yang sangat populer digunakan untuk berlatih klasifikasi *multiclass* adalah data angka tulisan tangan atau yang terkenal dengan nama **MNIST dataset**.



Untuk menggunakan dataset tersebut, kita sudah disediakan sebuah kelas `MnistData` di dalam file javascript `mnist_data.js` yang akan mengunduh dataset **MNIST** menggunakan fungsi `load()`. Setelah data terunduh, menggunakan fungsi `getTrainData()` dan `getTestData()`, kita dapat langsung mengubahnya menjadi tensor data latih dan data uji.

2 Tampilan Muka

Seperti sebelumnya, kita mulai aplikasi ini dengan membuat file tampilan baru di dalam direktori `modul_3`, yang kini kita beri nama `index3.html`. kemudian, untuk contoh ketiga ini, mari kita buat aplikasi selengkap mungkin, dari proses pembangunan arsitektur jaringan, pelatihan, pengujian, hingga penyimpanan model yang sudah dilatih, sehingga nantinya jika kita ingin menggunakan aplikasi ini, kita tidak perlu melatih model dari awal.

Secara keseluruhan, akan ada **6 (enam)** sub bagian (`div`) di dalam tampilan utama, yang masing-masing akan berisi:

1. Bagian mengunduh dataset MNIST dan menampilkan beberapa contoh gambarnya
2. Bagian mendefinisikan/mendesain arsitektur model jaringan
3. Bagian menampilkan menu pelatihan beserta progres pelatihannya
4. Bagian mengevaluasi model yang telah dilatih lalu menampilkan dalam bentuk akurasi dan confusion matrix, serta menampilkan beberapa contoh gambar hasil prediksi data uji
5. Bagian untuk menyimpan dan memuat model yang sudah dilatih
6. Bagian untuk menguji model menggunakan input tulisan tangan langsung dari user

Mari kita mulai membuat satu per satu bagiannya

Mula-mula, kita sediakan dahulu container utama halaman. Di sini akan kita gunakan CSS tambahan [mnist.css](#) untuk mempercantik tampilan. Tidak lupa kita muat file JavaScript [index3.js](#) yang nanti akan kita buat. Lalu kita seikan enam div untuk masing-masing bagian sebagai berikut

```
<html>
<head><link rel="stylesheet" type="text/css" href="mnist.css"></head>
<body>
  <div class="container">
    <div class="card-body">
      <h1>MNIST Classifier menggunakan TensorFlow.js</h1>
    </div><hr>

    <!-- Load Dataset -->
    <div class="card-body" >
      ...
    </div><hr>

    <!-- Initialize Model -->
    <div class="card-body ">
      ...
    </div><hr>

    <!-- Train Model -->
    <div class="card-body">
      ...
    </div><hr>

    <!-- Evaluate Model -->
    <div class="card-body">
      ...
    </div><hr>

    <!-- Save/Load Model -->
    <div class="card-body">
      ...
    </div><hr>

    <!-- Test Model -->
    <div class="card-body">
      ...
    </div><hr>

  </div><script src="./index3.js"></script>
</body>
</html>
```

2.1 Index3.js

Agar kita bisa segera melihat hasil tampilan selagi kita membangunnya, mari kita buat file `index3.js`. Untuk saat ini, cukup kita lengkapi bagian `import` kebutuhan framework dan tfjs. Kita akan kembali melengkapi file ini setelah tampilan terbentuk

```
import 'bootstrap/dist/css/bootstrap.css'
import * as tf from '@tensorflow/tfjs'
import * as tfvis from '@tensorflow/tfjs-vis'
import $ from 'jquery'
require('babel-polyfill')

import {MnistData} from './mnist_data'
...
```

Nyalakan aplikasi web kita dengan memanggil sintaks

```
\ai_tfjs > parcel modul_3\index3.html
```

2.2 Tampilan Load Dataset MNIST

Pada bagian ini, kita sediakan sebuah tombol **Load Data** untuk mengunduh dataset. Karena proses mengunduh data membutuhkan waktu, kita berikan sebuah `badge` untuk menampilkan pesan apakah proses mengunduh data sudah selesai atau belum. Terakhir, kita sediakan sebuah `div` untuk menampilkan contoh beberapa gambar yang sudah diunduh

```
...
<!-- Load Dataset -->
<div class="card-body" >

  <h3>MNIST Dataset</h3>
  <button type="button" id="load-data-btn"
    class="btn btn-primary form-group">Load Data</button>
  <h4><div id="loading-data" class="badge badge-warning"/></h4>

  <h4>Test Data Example</h4>
  <div id="mnist-preview"></div>

</div><hr>
...
```



2.3 Tampilan Inisialisasi Model

Pada bagian inisialisasi model, kita sediakan sebuah **text area** agar user bisa mendesain hingga mengcompilanya sendiri model yang akan digunakan. Di sini kita tambahkan juga **radio button** untuk menampilkan contoh pilihan arsitektur yang sudah didefinisikan dan siap digunakan. Kita berikan dua pilihan: model Jaringan Saraf Tiruan "biasa" (**ANN**), dan model Jaringan Saraf Konvolusi (**CNN**).

CNN adalah model arsitektur Jaringan Saraf Tiruan yang saat ini populer digunakan karena kemampuannya yang lebih tinggi dengan parameter lebih sedikit dibandingkan JST biasa.

Jangan lupa menambahkan sebuah tombol untuk memproses text desain arsitektur. Terakhir, kita tambahkan sebuah **div** untuk menampilkan summary dari arsitektur yang dibentuk.

```
...
<!-- Initialize Model -->
<div class="card-body">

  <h3>Model Architecture</h3>
  <div class="form-group">
    <textarea class="form-control" rows="11"
      id="model">model = tf.sequential(); </textarea>
  </div>

  <div class="row form-group">
    <div class="col">
      <label class="radio-inline">
        <input type="radio" name="optmodel" value='ANN'> ANN
      </label>
      <label class="radio-inline">
        <input type="radio" name="optmodel" value='CNN'> CNN
      </label>
    </div>

    <div class="col">
      <button type="button" class="float-right btn btn-primary"
        id="init-btn">Initialize Model</button>
    </div>
  </div><hr>

  <h3>Model Summary</h3>
  <div id="summary"></div>

</div><hr>
...
```



2.4 Tampilan Proses Pembelajaran

Untuk bagian pembelajaran, kita sediakan tiga kolom. Kolom pertama akan kita isi field untuk menerima input **max epoch** dan **batch size**, sementara kolom kedua dan ketiga akan kita gunakan untuk menampilkan progres **loss** dan **akurasi** selama pelatihan.

Apa itu **batch size**?

Untuk memahami intuisi dari istilah-istilah baru ini, mari kita sedikit simak teori singkat dari proses pembelajaran di dalam Jaringan Saraf Tiruan berikut ini.

a. Proses Pelatihan Jaringan Saraf Tiruan

Proses pembelajaran di dalam JST terdiri dari (1) fungsi maju (**forward pass**) untuk membaca seluruh data latih hingga menghitung outputnya, dan (2) fungsi mundur (**backward pass**) untuk memperbaiki bobot (angka-angka di dalam model) berdasarkan kesalahan output yang didapat

b. Epoch

Epoch merupakan hitungan iterasi pembelajaran (**forward** dan **backward**) terhadap keseluruhan data latih. Artinya pembelajaran dikatakan sudah melalui satu **epoch** jika sudah melakukan proses maju dan mundur untuk setiap data yang ada di dalam data latih.

Pada algoritma dasarnya, JST akan menggunakan seluruh data latih sekaligus dalam satu waktu pada setiap iterasi pelatihan. Sehingga 1 iterasi = 1 **epoch**. Proses pelatihan ini biasa disebut sebagai Pembelajaran **Full Batch Gradient Descent**. Misalnya pada modul sebelumnya, jika terdapat 100 data 2 dimensi, maka dalam sekali proses maju dan mundur dengan pembelajaran Full Batch, JST akan langsung membaca matrix berukuran **[100x2]**.

c. Batch Size

Batch size adalah jumlah data yang digunakan dalam satu kali proses pelatihan (**forward pass** dan **backward pass**). Pada mekanisme **Full Batch**, jumlah data yang digunakan akan sama dengan jumlah data latih yang ada. Permasalahan dari mekanisme Full Batch adalah jika ternyata data latih yang digunakan cukup besar, membaca keseluruhan matrix sekaligus mungkin tidak akan mampu dilakukan dengan **Memory GPU** yang kita miliki.

Misalnya untuk MNIST lengkap, setiap gambar berukuran **[28x28]** grayscale sebanyak **50.000** gambar data latih. Untuk membaca sebuah matrix **float64** berukuran **[50000 x 784]** tentunya akan sangat berat. Untuk itu, proses pembelajaran kita pecah-pecah menjadi beberapa potong data latih yang disebut **batch**.

Dengan menggunakan **Batch**, setiap proses maju dan mundur pelatihan, hanya membaca sebagian data dalam satu waktu. Ukuran batch biasanya disesuaikan dengan kemampuan komputasi (**Memory**) komputer masing-masing.

Nilai standar **batch size** pada **TensorFlow.js** adalah **32**. Nilai **batch** yang terlalu kecil dapat membuat proses pelatihan berjalan terlalu lama, karena semakin kecil nilai batch artinya semakin banyak iterasi yang harus dilatih.



Kembali ke tampilan proses pembelajaran, kita tambahkan sebuah `div row`, dengan tiga kolom. Pada kolom pertama, berikan input untuk `epoch` dan `batch`. Tambahkan juga sebuah tombol untuk memulai proses pelatihan. Kita set tombol tidak bisa digunakan hingga model telah terinisialisasi. Di sini, mari kita beri nilai standar `epoch = 1` dan `batch size = 100`

Pada kolom kedua dan ketiga, masing-masing berikan sebuah `div` untuk menampilkan grafik loss dan akurasi. Tambahkan juga masing-masing 2 `div badge` untuk menampilkan status pelatihan, jumlah iterasi yang akan dilakukan, progres pelatihan, dan akurasi data latih pada iterasi saat ini.

```
...
<!-- Train Model -->
<div class="card-body">
  <div class="row">

    <div class="col border-right">
      <h3>Train Model</h3>
      <div class="form-group">
        <label>Max Epoch:</label>
        <input id='epoch' class="form-control"
              type="text" value="1">
        <label>Batch Size:</label>
        <input id='batch' class="form-control"
              type="text" value="100">
      </div>
      <button type="button" class="btn btn-primary"
            id="train-btn" disabled>Train Model</button>
    </div>

    <div class="col">
      <h3>Training Progress</h3>
      <div id="loss-graph"></div>
      <div class="caption">Loss</div><hr>
      <h4><div id="training" class="badge badge-success"/></h4>
      <h4><div id="num-iter" class="badge"/></h4>
    </div>

    <div class="col">
      <h3><br></h3>
      <div id="acc-graph"></div>
      <div class="caption">Accuracy</div><hr>
      <h4><div id="train-iter" class="badge badge-warning"/></h4>
      <h4><div id="train-acc" class="badge badge-info"/></h4>
    </div>

  </div>
</div><hr>
...
```



2.5 Tampilan Proses Evaluasi

Pada bagian keempat, kita akan menampilkan performa model yang sudah kita latih terhadap data uji dalam bentuk **Class Accuracy** dan **Confusion Matrix**. Kedua teknik visualisasi performa tersebut dapat membantu kita untuk memahami tingkat kemampuan model jaringan.

Untuk itu, kita berikan sebuah tombol untuk memulai evaluasi, dan masing-masing sebuah **div** di dalam dua kolom untuk menampilkannya. Di dalam kolom **Class Accuracy**, tambahkan juga sebuah **badge** untuk menampilkan akurasi keseluruhan.

Di bagian kedua, kita berikan sebuah tombol dan sebuah **div** untuk menampilkan beberapa contoh gambar data uji dan hasil prediksi kelasnya

Bagian evaluasi akan menjadi seperti sintaks berikut

```
...
<!-- Evaluate Model -->
<div class="card-body">

  <h3>Evaluate Model</h3>
  <div class="form-group">
    <button type="button" class="btn btn-primary"
      id="eval-btn" disabled>Evaluate Model</button>
  </div>

  <div class="row">
    <div class="col">
      <h3>Class Accuracy</h3>
      <div class="form-group" id="class-accuracy"></div>
      <h4><div id="test-acc" class="badge badge-success"/></h4>
    </div>
    <div class="col">
      <h3>Confusion Matrix</h3>
      <div class="form-group" id="confusion-matrix"></div>
    </div>
  </div><hr>

  <h3>Prediction Example</h3>
  <button type="button" class="btn btn-primary"
    id="show-example-btn" disabled>Show Prediction</button>
  <div id="example-preview"></div>

</div><hr>
...
```

2.6 Tampilan Penyimpanan Model

Setelah kita latih model jaringan dengan baik, selanjutnya kita bisa menyimpan model tersebut agar kita bisa gunakan di kemudian hari tanpa harus melatih ulang model dari awal. Saat menyimpan sebuah model, TensorFlow.js akan membuat dua file dalam format ekstensi `.json` dan `.weights.bin`

File **JSON** yang dibentuk akan menyimpan struktur/arsitektur model yang dibangun, sementara file **BIN** akan menyimpan angka-angka bobot model yang sudah dilatih.

Sekarang, mari kita bangun dua kolom masing-masing untuk bagian menyimpan dan memuat model jaringan. Pada bagian penyimpanan, kita hanya membutuhkan sebuah tombol Save Model. Sementara pada bagian memuat model, kita membutuhkan tombol untuk memuat arsitektur model dari file **JSON**, sebuah tombol untuk memuat bobot dari file **BIN**, dan sebuah tombol untuk memprosesnya. Tambahkan juga masing-masing sebuah **badge** untuk memberikan pesan jika model berhasil disimpan atau dimuat.

Berikut akhir dari isi sintaks di bagian kelima

```
...
<!-- Save/Load Model -->
<div class="card-body">
  <div class="row">

    <div class="col border-right">
      <h3>Save Model</h3>
      <button type="button" class="btn btn-primary form-group"
        id="save-btn" disabled>Save Model</button>
      <h4><div class="badge badge-success" id="saved"
        style="display:none;">Model Saved</div></h4>
    </div>

    <div class="col">
      <h3>Load Model</h3>
      <label class="btn btn-primary btn-file">
        Browse JSON <input type="file" id="json-upload">
      </label>
      <label class="btn btn-primary btn-file">
        Browse Weights <input type="file" id="weights-upload">
      </label>
      <button type="button" class="btn btn-primary form-group"
        id="load-model-btn">Load Model</button>
      <h4><div class="badge badge-success" id="loaded"
        style="display:none;">Model Loaded</div></h4>
    </div>

  </div>
</div><hr>
...
```

2.7 Tampilan Pengujian Model

Terakhir, adalah bagian tempat kita bisa mencoba secara langsung hasil model pelatihan, dengan memberikan input baru gambar angka tulisan tangan. Di sini mari kita desain tampilan dalam tiga kolom. Kolom pertama akan berisi canvas tempat user dapat menggambarkan/menuliskan input angka baru. Kolom kedua akan menampilkan hasil crop dan konversi gambar, serta tempat menampilkan hasil prediksinya. Kolom ketiga kita tempatkan untuk menampilkan histogram nilai prediksi model terhadap keseluruhan kelas **0-9**

```
...
<!-- Test Model -->
<div class="card-body">
  <h3>Test Model</h3>
  <div class="row">

    <div class="col-2 offset-0 form-group">
      <canvas id="predict-canvas" width="140" height="140"
        style="border:1px solid #000000;" ></canvas>
    </div>

    <div class="col-2" style="text-align:center">
      <div class="form-group">
        preview <br>
        <canvas id="preview-canvas" width="28" height="28"
          style="border:1px solid #000000;"></canvas>
      </div>
      <h4><div class="badge badge-success" id="prediction"/></h4>
    </div>

    <div class="col-5">
      Prediction Class Histogram
      <div id="predict-graph"></div>
    </div>

  </div>

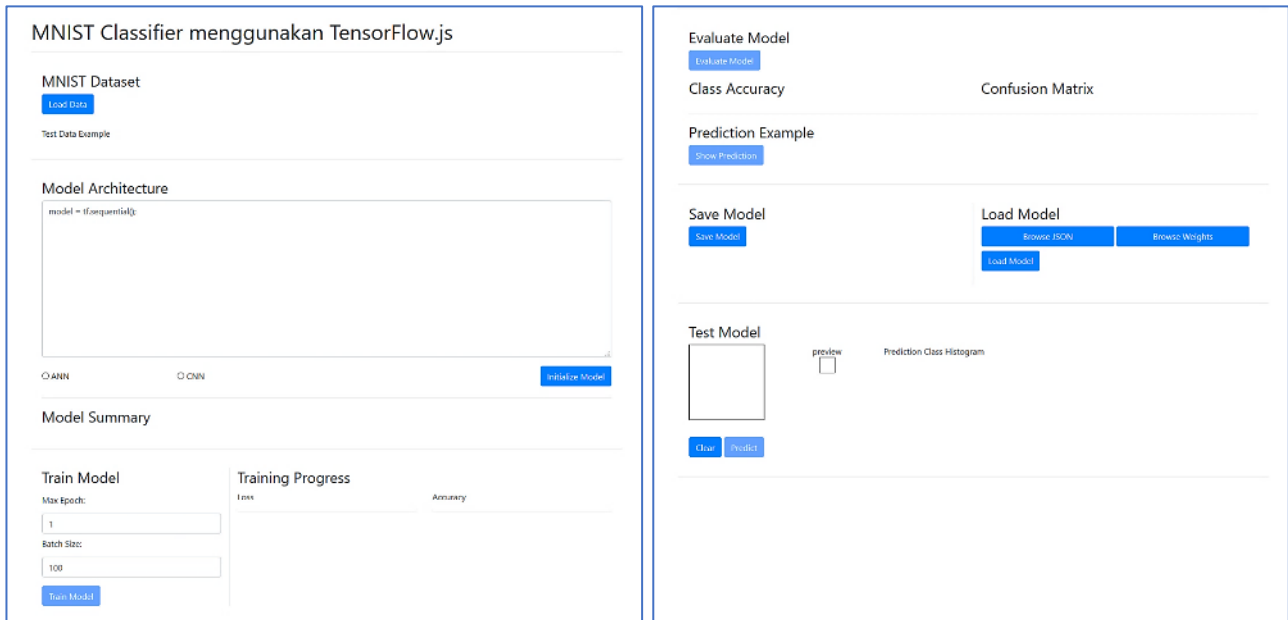
  <button type="button" class="btn btn-primary"
    id="clear-btn">Clear</button>
  <button type="button" class="btn btn-primary"
    id="predict-btn" disabled>Predict</button>

</div><hr>
...
```

2.8 Preview Tampilan Muka

Seluruh kode tampilan muka dalam [index3.html](#) sudah kita selesaikan.

Jika kita lihat tampilan akhir dengan menjalankan sintaks `parcel modul_3\index3.html`, maka akan terlihat seperti gambar berikut



The screenshot displays the MNIST Classifier web application interface, which is divided into two main panels. The left panel, titled "MNIST Classifier menggunakan TensorFlow.js", contains sections for "MNIST Dataset" (with a "Load Data" button), "Model Architecture" (with a text area for defining the model and a "Initialize Model" button), and "Train Model" (with input fields for "Max Epochs" and "Batch Size", and a "Train Model" button). The right panel, titled "Evaluate Model", includes sections for "Class Accuracy" and "Confusion Matrix", "Prediction Example" (with a "Show Prediction" button), "Save Model" (with a "Save Model" button), "Load Model" (with "Browse JSON" and "Browse Weights" buttons, and a "Load Model" button), and "Test Model" (with a "preview" button and a "Prediction Class Histogram" label).

3 MNIST Utility Functions

Berikutnya, kita jika sudah disediakan sebuah file javascript `mnist_utils.js` yang berisi berbagai fungsi bantuan yang akan digunakan dalam aplikasi kita. Beberapa fungsi bantuan tersebut antara lain:

a. `function getModel(name)`

Fungsi ini bertujuan untuk memberikan template awal arsitektur jaringan ANN/CNN yang bisa dipilih dan diubah lebih lanjut oleh user. Fungsi mengembalikan teks arsitektur 2 layer dense untuk pilihan opsi ANN dan 3 layer konvolusi (2 conv, 1 dense) untuk opsi CNN. Setiap layer menggunakan fungsi aktivasi relu dan output aktivasi softmax. Kedua pilihan arsitektur dicompile dengan optimasi rmsprop menggunakan loss categoricalCrossentropy dan accuracy sebagai metrics perhitungan performanya

b. `function showExample(elementId, data, labels, predictions=null)`

Fungsi `showExample` akan membantu kita untuk membuat tampilan saat menampilkan contoh gambar angka tulisan tangan dari dataset secara dinamis sesuai banyaknya data yang diberikan

c. `function cropImage(img, width=140)`

Fungsi `cropImage` digunakan saat proses pengujian untuk memotong input gambar tulisan tangan dari user dan membuang bagian putih yang berlebih untuk meningkatkan performa model dalam mengklasifikasi input baru tersebut. Nilai `width` pada fungsi menyatakan ukuran canvas yang kita sediakan untuk menerima input gambar angka tulisan tangan dari user, di mana sebelumnya kita desain berukuran 140x140

d. `function firefoxSave(model)`

TensorFlow.js telah menyediakan modul fungsi untuk menyimpan dan memuat kembali model yang sudah dilatih. Sayangnya, hingga sampai saat tutorial ini dibuat, modul yang diberikan hanya dapat berfungsi dengan baik pada aplikasi browser Chrome. Untuk penggunaan browser lain seperti Firefox, fungsi untuk menyimpan model masih mengalami beberapa kekurangan. Maka dari itu, disediakan fungsi tambahan untuk membantu menyimpan model yang telah dilatih jika kita menggunakan browser lain seperti Firefox.

4 Drawing Utility Function

File JavaScript bantuan terakhir yang diberikan adalah file `draw_util.js`. File ini mengandung fungsi `initCanvas()` yang akan memberikan fungsionalitas pada canvas yang akan menerima aksi dari event mouse sehingga user bisa menggambar pada canvas tersebut. Fungsi ini menerima input id canvas yang akan dijadikan tempat menggambar, serta ukuran dan warna pena.



5 Index3.js

Sekarang, mari kita mulai memberikan fungsionalitas pada aplikasi web kita dengan mulai melengkapi file `index3.js`. Mula-mula, kita lengkapi bagian `import` dengan menambahkan fungsi utilitas yang dibutuhkan dari file `mnist_data`, `mnist_utils`, dan `draw_utils`

```
import 'bootstrap/dist/css/bootstrap.css'
import * as tf from '@tensorflow/tfjs'
import * as tfvis from '@tensorflow/tfjs-vis'
import $ from 'jquery'
require('babel-polyfill')

import {MnistData} from './mnist_data'
import * as util from './mnist_utils'
import {initCanvas} from './draw_utils'

...
```

5.1 Event Listener Load Data

Berikutnya, kita berikan `Event Listener` untuk tombol Load Data dengan fungsi `asynchronous`. Mula-mula kita definisikan objek global `MnistData`. Kemudian, saat tombol ditekan, berikan pesan bahwa data MNIST sedang diunduh, lalu panggil fungsi `load()` untuk mulai mengunduh.

Google menyediakan total data sebanyak **65.000** data MNIST. Agar proses pelatihan tidak terlalu lama nantinya, mari kita hanya unduh **40.000** data untuk data latih, dan **10.000** data untuk data uji. Ketika data telah terunduh, ubah pesan yang ditampilkan. Terakhir, mari kita ambil **8** gambar pertama dari data uji untuk kita tampilkan.

Ambil data menggunakan fungsi `getTestData()`, kemudian tampilkan gambar-gambar tersebut pada `div mnist-preview` menggunakan fungsi bantuan `util.showExample()`.

```
...
let data = new MnistData()
$('#load-data-btn').click(async() => {
  let msg = $('#loading-data')
  msg.text('Downloading MNIST data. Please wait...')
  await data.load(40000, 10000)

  msg.toggleClass('badge-warning badge-success')
  msg.text('MNIST data Loaded')
  $('#load-btn').prop('disabled', true)

  const [x_test, y_test] = data.getTestData(8)
  const labels = Array.from(y_test.argmax(1).dataSync())
  util.showExample('mnist-preview', x_test, labels)
})
...
```



5.2 Event Listener Inisialisasi Model

Pada bagian ini, berikan **Action Event** pada **radio button** untuk menampilkan text arsitektur model yang telah disediakan melalui fungsi bantuan `util.getModel()`. Selanjutnya buat variabel global model, lalu berikan action **event click** pada tombol `init-btn`. Di dalam action event, ambil text arsitektur dari text area model dan eksekusi sintaks yang dituliskan user menggunakan fungsi `eval()`.

Jika inisialisasi model berhasil, tampilkan arsitektur yang dibentuk menggunakan fungsi `modelSummary()` pada `div summary`, dan nyalakan tombol-tombol pada tampilan utama.

```
...  
$('input[name=optmodel]:radio').click(function() {  
    $('#model').text(util.getModel(this.value))  
})  
  
let model  
$('#init-btn').click(function() {  
    var md = $.trim($('#model').val())  
    eval(md)  
  
    tfvis.show.modelSummary($('#summary')[0], model)  
  
    $('#train-btn').prop('disabled', false)  
    $('#predict-btn').prop('disabled', false)  
    $('#eval-btn').prop('disabled', false)  
    $('#show-example-btn').prop('disabled', false)  
})  
...
```



5.3 Event Listener Tombol Train

Sama seperti modul sebelumnya, pada tombol Train kita berikan **Action Event** menggunakan fungsi **asynchronous**. Mula-mula kita berikan pesan bahwa model sedang dilatih, lalu ambil nilai **epoch** dan **batch** dari input user beserta **div** tempat menampilkan grafik akurasi dan loss selama pelatihan

```
...
let round = (num) => parseFloat(num*100).toFixed(1)

$('#train-btn').click(async() => {
  var msg = $('#training')
  msg.toggleClass('badge-warning badge-success')
  msg.text('Training, please wait...')

  const trainLogs = []
  var epoch = parseInt($('#epoch').val())
  var batch = parseInt($('#batch').val())
  const loss = $('#loss-graph')[0]
  const acc = $('#acc-graph')[0]

  ...
})
...
```

Berikutnya kita ambil data latih MNIST menggunakan fungsi **getTrainData()**. Seperti yang telah dijelaskan sebelumnya, dari keseluruhan data latih, akan dibagi menjadi beberapa **batch** sesuai ukuran yang diberikan user. Sehingga di sini bisa kita hitung akan ada berapa iterasi (fungsi maju dan mundur) yang akan dilakukan sejumlah **epoch** yang diberikan, lalu tampilkan ke halaman.

Total iterasi dapat dihitung sebagai **jumlah data / ukuran batch * max epoch**.

```
...
$('#train-btn').click(async() => {
  ...
  const [x_train, y_train] = data.getTrainData()
  let nIter = 0
  const numIter = Math.ceil(x_train.shape[0] / batch) * epoch
  $('#num-iter').text('Num Training Iteration: '+ numIter)
  ...
})
...
```



Kini kita panggil fungsi `fit()` untuk melatih model menggunakan data latih, `epoch`, dan ukuran `batch` yang diberikan. Selama pelatihan, setiap iterasi batch selesai dilakukan, kita akan tampilkan progres loss dan akurasi. Bersama itu, tampilkan juga pesan progres berapa persen proses pembelajaran sudah dijalankan.

```
...
$('#train-btn').click(async() => {
    ...
    const history = await model.fit(x_train, y_train, {
        epochs: epoch,
        batchSize: batch,
        shuffle: true,
        callbacks: {
            onBatchEnd: async (batch, logs) => {
                nIter++
                trainLogs.push(logs)
                tfvis.show.history(loss, trainLogs, ['loss'],
                    { width: 300, height: 160 })
                tfvis.show.history(acc, trainLogs, ['acc'],
                    { width: 300, height: 160 })
                $('#train-iter').text(`Training..
                    ( ${round(nIter / numIter)}% )`)
                $('#train-acc').text('Training Accuracy : '
                    + round(logs.acc) + '%')
            },
        }
    })
    ...
})
...
```

Di akhir fungsi, setelah proses pelatihan selesai dijalankan, berikan status selesai, lalu nyalakan tombol Save

```
...
$('#train-btn').click(async() => {
    ...
    $('#train-iter').toggleClass('badge-warning badge-success')
    msg.toggleClass('badge-warning badge-success')
    msg.text('Training Done')
    $('#save-btn').prop('disabled', false)
})
...
```

5.4 Event Listener Tombol Evaluasi

Pada **Action Event** tombol evaluasi juga kita berikan fungsi **asynchronous**. Di dalamnya kita ambil keseluruhan data uji menggunakan fungsi **getTestData()**. Setelah itu kita panggil fungsi **predict()** untuk menguji model yang sudah dilatih. Kemudian kita tampilkan akurasi dari keseluruhan data latih ke dalam **badge test-acc**

Untuk menampilkan akurasi per kelas dan detail kesalahan dalam bentuk confusion matrix, kita ambil kedua **div** tempat menampilkannya. Berikutnya kita panggil fungsi untuk menghitung akurasi per kelas dan confusion matrix menggunakan modul metrics dari library **tfjs-vis**. Terakhir, kita tampilkan objek hasil perhitungan kedalam div masing-masing menggunakan modul **show()** yang juga dari library **tfjs-vis**

```
...
$('#eval-btn').click(async() => {
  let [x_test, y_test] = data.getTestData()

  let y_pred = model.predict(x_test).argMax(1)
  let y_label = y_test.argMax(1)

  let eval_test = await tfvis.metrics.accuracy(y_label, y_pred)
  $('#test-acc').text( 'Testset Accuracy : '+ round(eval_test)+'%')

  const acc = $('#class-accuracy')[0]
  const conf = $('#confusion-matrix')[0]
  const clsAcc = await tfvis.metrics.perClassAccuracy(y_label, y_pred)
  const confMt = await tfvis.metrics.confusionMatrix(y_label, y_pred)

  const classNames = ['Zero', 'One', 'Two', 'Three', 'Four',
    'Five', 'Six', 'Seven', 'Eight', 'Nine']
  tfvis.show.perClassAccuracy(acc, clsAcc, classNames)
  tfvis.render.confusionMatrix(conf,
    { values: confMt , tickLabels: classNames })
})
...
```

Kemudian pada tombol Show Example, kita berikan **Action Event** untuk kembali menampilkan beberapa gambar data uji namun sekarang beserta output label prediksi dari model. Pertama kita ambil **16** data gambar dari data uji menggunakan fungsi **getTestData()**. Sama seperti pada tombol evaluasi, selanjutnya kita panggil fungsi **predict()** dan ambil label hasil prediksinya. Selanjutnya kita kembali gunakan fungsi bantuan **util.showExample()** untuk menampilkan hasil prediksi/klasifikasi ke dalam **div** example-preview

```
...
$('#show-example-btn').click(function(){
  let [x_test, y_test] = data.getTestData(16)

  let y_pred = model.predict(x_test)
  const labels = Array.from(y_test.argMax(1).dataSync())
  const preds = Array.from(y_pred.argMax(1).dataSync())

  util.showExample('example-preview', x_test, labels, preds)
})
...
```

5.5 Event Listener Tombol Save dan Load

Untuk tombol Save, kita berikan **Action Event** fungsi **asynchronous**, kemudian panggil fungsi **save()** dari objek model dan masukkan opsi argumen untuk menunduh model ke direktori lokal. Seperti yang telah dijelaskan sebelumnya, hingga saat modul ini dibuat, fungsi **save()** hanya bisa berjalan di browser Chrome. Untuk menyimpan model yang dijalankan di browser lain misalnya Firefox, aktifkan baris kode **util.firefoxSave(model)**.

Setelah model diunduh, tampilkan pesan bahwa model telah tersimpan. Kemudian mari kita berikan fungsionalitas untuk mempercantik pesan tersebut agar otomatis hilang setelah satu detik.

```
...
$('#save-btn').click(async() => {
  const saveResults = await model.save('downloads://')

  //For Firefox browser
  //util.firefoxSave(model)

  $('#saved').show()
  setTimeout(function() {
    $('#saved').fadeOut()
  }, 1000)
})
...
```

Kemudian untuk tombol Load, tambahkan juga **Action Event** fungsi **asynchronous**. Di dalamnya, ambil file **JSON** dan **BIN** yang diberikan dari input file json-upload dan weights-upload. Lalu panggil fungsi **loadLayersModel()** untuk mulai membentuk model baru. Berikutnya nyalakan tombol Predict untuk pengujian jika memang belum dinyalakan, dan berikan pesan yang muncul selama satu detik bahwa model berhasil dimuat.

```
...
$('#load-model-btn').click(async() => {
  const jsonUpload = $('#json-upload')[0]
  const weightsUpload = $('#weights-upload')[0]
  model = await tf.loadLayersModel(tf.io.browserFiles(
    [jsonUpload.files[0], weightsUpload.files[0]] ))

  $('#predict-btn').prop('disabled', false)

  $('#loaded').show()
  setTimeout(function() { $('#loaded').fadeOut() }, 1000)
  if (data.isDownloaded){
    $('#eval-btn').prop('disabled', false)
    $('#show-example-btn').prop('disabled', false)
  }
})
...
```

5.6 Event Listener Canvas dan Tombol Predict

Yang terakhir, untuk bagian pengujian, mula-mula mari kita inisialisasi `canvas` `predict-canvas` agar bisa digambar secara langsung oleh user dengan memanggil fungsi `initCanvas()`. Kemudian untuk tombol `clear-btn`, berikan `action event` untuk menghapus isi canvas dengan memanggil fungsi `clearRect()`.

```
...
initCanvas('predict-canvas')

$('#clear-btn').click(function(){
  var canvas = $('#predict-canvas')[0]
  var context = canvas.getContext('2d')
  context.clearRect(0, 0, canvas.width, canvas.height)
})
...
```

Untuk tombol `predict-btn`, tambahkan `Action Event` fungsi `asynchronous`. Di dalamnya, kita ambil `canvas` input user dan canvas untuk menampilkan hasil crop dan resize input. Ambil gambar dari canvas input dan resize menggunakan fungsi bantuan `util.cropImage()`. Selanjutnya, ubah image hasil resize menjadi tensor agar bisa diprediksi, sambil menampilkan image hasil resize ke canvas.

Panggil fungsi `predict()` untuk memprediksi kelas input user, ambil indeks kelasnya, lalu tampilkan ke halaman. Selain itu proses hasil prediksi model agar bisa ditampilkan dalam bentuk grafik batang, lalu tampilkan histogram ke halaman utama.

```
...
$('#predict-btn').click(async() => {
  var canvas = $('#predict-canvas')[0]
  var preview = $('#preview-canvas')[0]

  var img = tf.browser.fromPixels(canvas, 4)
  var resized = util.cropImage(img, canvas.width)

  tf.browser.toPixels(resized, preview)
  var x_data = tf.cast(resized.reshape([1, 28, 28, 1]), 'float32')

  var y_pred = model.predict(x_data)

  var prediction = Array.from(y_pred.argMax(1).dataSync())
  $('#prediction').text('Predicted: ' + prediction)

  const chartData = Array.from(y_pred.dataSync()).map((d, i) => {
    return { index: i, value: d }
  })
  tfvis.render.barchart($('#predict-graph')[0], chartData,
    { width: 400, height: 140 } )
})
```

6 Hasil akhir

MNIST classifier kita siap untuk dicoba. Jika kita jalankan pelatihan dan pengujian, maka akan terlihat seperti gambar berikut


MNIST Classifier menggunakan TensorFlow.js

MNIST Dataset

[Load Data](#)

MNIST data loaded

Test Data Example



Model Architecture

```

model = tf.sequential({
  model.add(tf.layers.conv2d({ kernelSize: 3, filters: 16, activation: 'relu', inputShape: (28, 28, 1) })),
  model.add(tf.layers.conv2d({ kernelSize: 3, filters: 16, activation: 'relu' })),
  model.add(tf.layers.maxPooling2d({ poolSize: 2, strides: 2 })),
  model.add(tf.layers.flatten({ })),
  model.add(tf.layers.dense({ units: 10, activation: 'softmax' })),
});

const myOptim = tf.train.adam(0.001);
model.compile({ loss: 'categoricalCrossentropy', optimizer: myOptim, metrics: ['accuracy'] });

```

☐ ANN ☒ CNN [Initialize Model](#)

Model Summary

Layer Name	Output Shape	# Of Params	Trainable
conv2d_Conv2D1	[batch, 26, 26, 16]	160	true
conv2d_Conv2D2	[batch, 24, 24, 16]	2,320	true
max_pooling2d_MaxPooling2D1	[batch, 12, 12, 16]	0	true
flatten_Flatten1	[batch, 2304]	0	true
dense_Dense1	[batch, 10]	23,050	true


Train Model

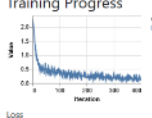
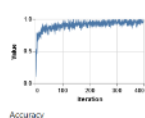
Max Epoch:

Batch Size:

[Train Model](#)

Training Progress



Loss:  Accuracy: 

Training Done

Num Training Iteration: 400

Training Accuracy: 96.0%

Evaluate Model

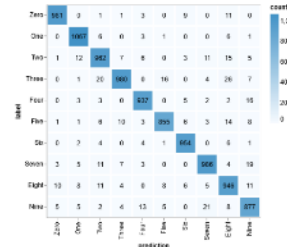
[Evaluate Model](#)

Class Accuracy

Class	Accuracy	# Samples
Zero	0.9751	1,006
One	0.9863	1,084
Two	0.9413	1,022
Three	0.9399	1,054
Four	0.958	958
Five	0.9427	927
Six	0.9815	972
Seven	0.9499	1,038
Eight	0.9376	1,009
Nine	0.933	940


Testset Accuracy: 95.5%

Confusion Matrix



Prediction Example

[Show Prediction](#)



Save Model

[Save Model](#)


Load Model

[Browse JSON](#) [Browse Weights](#)

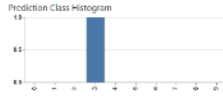
[Load Model](#)

Test Model

[Clear](#) [Predict](#)

preview  **Predicted: 3**

Prediction Class Histogram



Belajar TensorFlow.js Bersama lab AI

Modul IV **Contoh Aplikasi Web menggunakan** **TensorFlow.js**



Artificial Intelligence Laboratory
Fakultas Informatika
Universitas Telkom
Bandung, April 2019

Modul IV

Contoh-contoh Aplikasi Web menggunakan TensorFlow.js

Kita telah belajar bagaimana cara membangun aplikasi web yang menerapkan kemampuan Pembelajaran Mesin menggunakan TensorFlow.js dari mulai tahap pelatihan, pengujian, hingga menyimpan dan menggunakan model yang sudah dilatih. Namun dapat kita lihat bahwa tahap pelatihan Pembelajaran Mesin adalah suatu proses yang berat dan memakan waktu yang tidaklah sebentar. Tujuan utama dibangunnya library TensorFlow.js sebenarnya bukan untuk proses pelatihan, namun lebih kepada proses pengujian dan implementasi model yang sudah dilatih.

Proses pelatihan model itu sendiri umumnya dilakukan pada komputer berkemampuan tinggi dengan bahasa pemrograman yang lebih cocok dan efisien misalnya bahasa Python atau C++. Setelah model Pembelajaran Mesin dilatih, TensorFlow.js memberikan fasilitas kemudahan untuk bisa menggunakannya langsung dengan antar muka aplikasi web. Hal ini akan membuat aplikasi Pembelajaran Mesin bisa langsung dengan mudah diakses pengguna melalui browser baik melalui komputer maupun *mobile devices*.

Berikut ini mari kita lihat beberapa contoh aplikasi web populer dan mengagumkan yang dibangun menggunakan TensorFlow.js. Model-model yang digunakan di sini adalah model ***state-of-the-art*** Pembelajaran Mesin yang telah dilatih menggunakan *Framework Library* Pembelajaran Mesin populer TensorFlow dan Keras.

Pertama, kita dapat melatih sendiri suatu model Pembelajaran Mesin menggunakan TensorFlow dan Keras dalam bahasa Python. Kemudian model yang telah dilatih bisa kita konversi dan simpan sebagai file **.JSON** dan bobot yang dikenali TensorFlow.js. Untuk menggunakannya, kita hanya perlu mengunggah model tersebut ke dalam suatu server cloud storage, dan model siap digunakan.

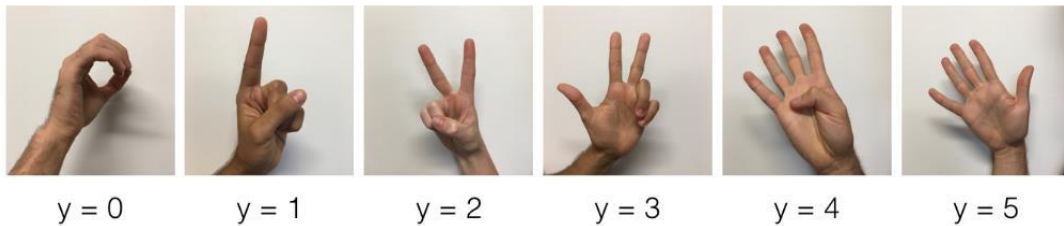
Cara lain adalah dengan menggunakan model yang telah disediakan TensorFlow.js sebagai contoh model aplikasi menarik yang bisa langsung kita gunakan. Selain itu, juga terdapat banyak sekali pengguna TensorFlow.js yang telah membagikan model-model Pembelajaran Mesin yang sudah mereka latih untuk bisa kita gunakan secara cuma-cuma

Mari kita coba contoh-contoh tersebut



1 Pengenalan Bahasa Isyarat

Contoh pertama kita adalah aplikasi pengenalan bahasa isyarat yang dijalankan secara langsung melalui *webcam*. Model yang digunakan adalah model yang telah dilatih oleh Laboratorium Kecerdasan Buatan Universitas Telkom. Model yang dilatih menggunakan arsitektur **MobileNet** yang dikenal ringan namun masih memiliki performa tinggi. Untuk meringankan kasus yang dihadapi, di sini pengenalan bahasa isyarat dibatasi hanya mengenali isyarat angka **0 hingga 5**



Model dilatih menggunakan *library* Keras dengan backend TensorFlow dalam bahasa Python yang kemudian dikonversi menjadi model untuk TensorFlow.js. Hasil model dapat dilihat pada direktori [/saved_model/handsign_model](#). Jika diperhatikan, tidak seperti model yang kita simpan melalui hasil pembelajaran TensorFlow.js sebelumnya, model di dalam direktori adalah sebuah file **.JSON** dan tiga file potongan bobot yang disebut **SHARD**. Banyaknya *shard* yang dibentuk saat konversi akan sebesar ukuran model yang dilatih.

Seperti yang sudah dijelaskan sebelumnya, untuk menggunakan model tersebut, kita perlu mengunggah seluruh model ke suatu server penyimpanan. Kemudian untuk menggunakannya, kita tinggal panggil tautan menuju file **.JSON** model, dan TensorFlow.js akan secara otomatis membaca seluruh file *shard* bobot yang terdaftar.

Untuk mensimulasikan suatu server penyimpanan secara lokal, mari kita gunakan *library* **http-server** dari **Node.js**. Untuk mengunduh *library* tersebut, buka **command prompt** baru, lalu jalankan sintaks berikut

```
\ai_tfjs > npm install -g http-server
```

Setelah instalasi berhasil dilakukan, pindahkan direktori **cmd** ke dalam direktori [saved_model](#), lalu kita simulasikan server penyimpanan pada direktori tersebut dengan menjalankan sintaks berikut

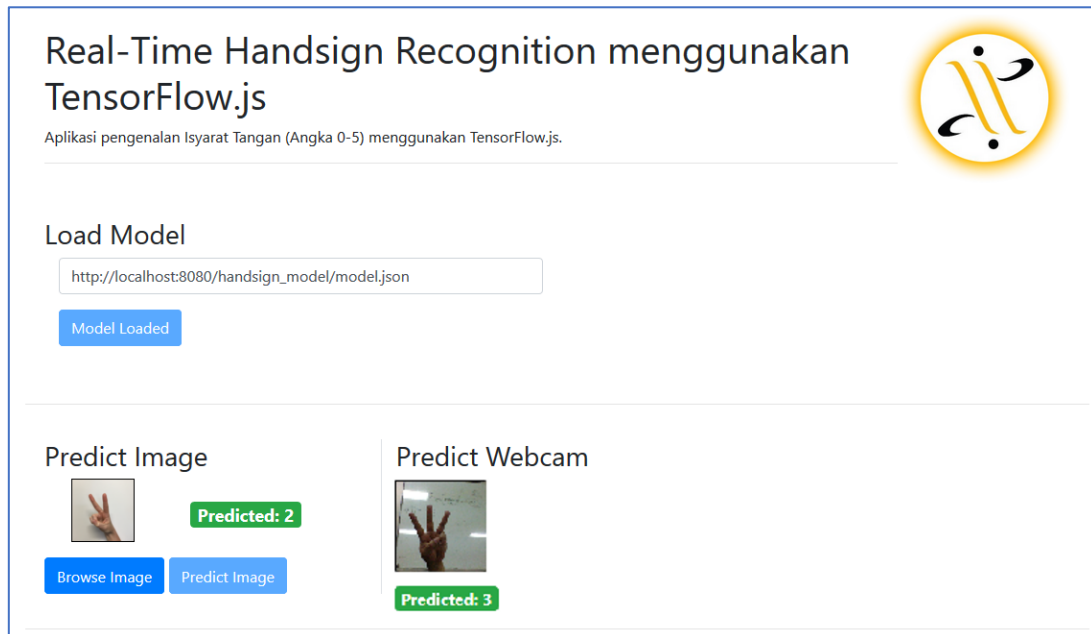
```
\ai_tfjs > cd saved_model  
\ai_tfjs\saved_model > http-server -c1 --cors
```

Kini seluruh isi direktori akan dapat diakses melalui alamat <http://localhost:8080/>

Berikutnya tinggal kita nyalakan aplikasi web ini dengan menjalankan sintaks

```
\ai_tfjs > parcel modul_4\handsign.html
```

Tampilan aplikasi [handsign.html](#) jika dijalankan akan terlihat seperti gambar di bawah



Di dalam aplikasi, setelah memuat model, kita bisa mencoba dengan mengunggah gambar contoh atau mengaktifkan webcam untuk bisa mencoba secara langsung.

2 Melukis menggunakan Style Transfer

Berikutnya mari kita lihat implementasi teknik Pembelajaran Mesin untuk jenis mengeluarkan hasil selain regresi dan klasifikasi, yaitu teknik **Generation** (generasi/pembangkitan/pembentukan). Di sini, kita gunakan dua buah model yang disebut **Style Model** dan **Transformer Model**. Kedua model dilatih menggunakan algoritma/teknik yang dikenal dengan sebutan **Neural Style Transfer** sehingga mampu melukis (membentuk) suatu lukisan baru.

Singkat cara kerja aplikasi ini yaitu kita berikan dua buah citra yang disebut citra konten dan citra *style*. **Style Transfer** akan mencoba melukis ulang obyek yang ada pada citra konten, namun dengan *style* atau gaya lukisan sesuai dengan citra *style* yang diberikan. Gaya lukisan dapat diartikan sebagai gaya pemilihan warna, bentuk benda, garis lukisan, dan sebagainya.



Model yang digunakan pada contoh kali ini berasal dari model yang dilatih oleh seorang kontributor bernama [Reiichiro Nakano](#). Model tersebut kemudian kami konversi untuk bisa digunakan pada **TensorFlow.js**

Mula-mula citra *style* akan dimasukkan ke dalam **Style Model** untuk dikuantifikasi ciri atau gaya lukisannya. Kemudian hasil ciri tersebut, bersama dengan citra konten akan dimasukkan ke dalam **Transformer Model** untuk menghasilkan citra baru. **Transformer Model** akan mencoba membentuk suatu gambar baru yang akan berisi objek yang mirip dengan input citra konten, namun dengan tetap menjaga agar *style* gambar yang dihasilkan yang mirip dengan input *style* yang diberikan.

Untuk menjalankan aplikasi web **Style Transfer**, jalankan sintaks

```
\ai_tfjs > parcel modul_4\styletransfer.html
```


Jangan lupa tetap nyalakan server penyimpanan lokal untuk memuat **Style Model** dan **Transformer Model** dari direktori lokal.

Di dalam aplikasi, kita dapat mengubah kedetilan *style* atau gaya lukisan yang akan ditampilkan dengan mengubah ukuran baik input citra konten dan citra *style*-nya. Kemudian kita juga bisa mengatur detail perubahan atau transformasi citra konten saat menambahkan *style* baru dengan mengubah-ubah nilai (*slider*) *Stylization Strength*

Tampilan aplikasi [handsign.html](#) jika dijalankan akan terlihat seperti gambar di bawah


Style Transfer menggunakan TensorFlow.js

Aplikasi untuk melukis suatu konten dari sebuah gambar dengan gaya lukisan dari gambar lain menggunakan TensorFlow.js.




Load Model

Content Image




Content image size

Style Image



Style image size

☐ Force image to square



Stylization strength

3 Pengenalan Kalimat Kasar

Kita sudah mencoba bagaimana menggunakan model yang sudah kita latih sendiri. Berikutnya, mari kita coba contoh model yang sudah disediakan oleh TensorFlow.js. Beberapa contoh resmi dari TensorFlow.js telah dipublikasi melalui akun **github TensorFlow** pada repositori [tfjs-models](#) dan [tfjs-example](#). Bentuk contoh resmi yang dipublikasi umumnya sudah mengandung berbagai fungsi bantuan yang mempermudah kita agar kita tidak perlu membuat sendiri fungsi-fungsi untuk menggunakannya.

Sejauh ini, kebanyakan kita telah mencoba kasus dengan input citra atau suatu bidang yang biasa disebut *Computer Vision*. Untuk contoh ketiga ini, mari kita coba kasus lain yaitu kasus di mana inputnya adalah teks. Pada 16 Maret 2019, TensorFlow [merilis](#) sebuah model yang mampu mengklasifikasi apakah teks mengandung kalimat kasar (hinaan, ancaman, senonoh, cacian, dsb) atau tidak. Model yang dipublikasi bertajuk *Toxicity Classifier* yang modelnya sudah dibagikan dalam bentuk TensorFlow.js.

Untuk menggunakan model tersebut, kita hanya perlu mengunduh *library* dengan sintaks berikut

```
\ai_tfjs > npm install @tensorflow-models/toxicity
```


Setelah *library* diunduh, jalankan sintaks untuk mencoba aplikasi web tersebut

```
\ai_tfjs > parcel modul_4\toxicity.html
```

Tampilan aplikasi [toxicity.html](#) saat dijalankan akan terlihat seperti gambar di bawah

Pengenalan Kalimat Kasar menggunakan TensorFlow.js

Aplikasi pengenalan kalimat kasar menggunakan TensorFlow.js. - (kalimat yang menyinggung, menghina, senonoh, dll)



Download Model

Download Model
Show Sample

Test Model

Enter text below and click 'Classify' to add it to the table.

I hate you! you are a monkey! you're such a pain. I'll kill you.

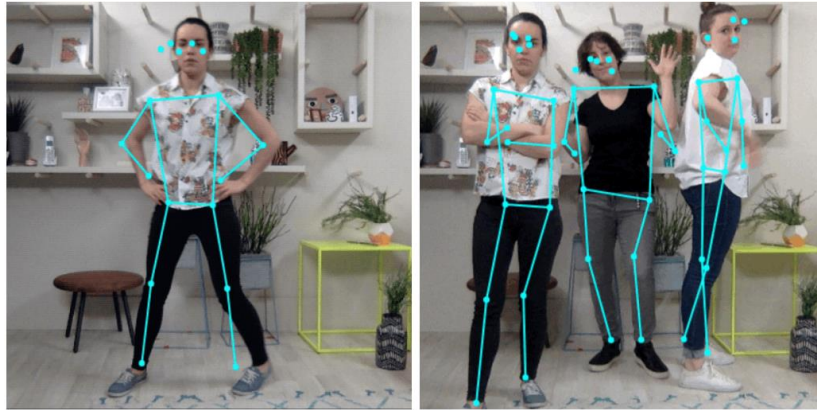
Classify Text

Prediction Results

Input Text	identity attack	insult	obscene	severe toxicity	sexual explicit	threat	toxicity
We're dudes on computers, moron. You are quite astonishingly stupid.	no	yes	no	no	no	no	yes
Please stop. If you continue to vandalize Wikipedia, as you did to Kmart, you will be blocked from editing.	no	no	no	no	no	no	no
I respect your point of view, and when this discussion originated on 8th April I would have tended to agree with you.	no	no	no	no	no	no	no
I hate you! you are a monkey! you're such a pain. I'll kill you.	no	minor	minor	no	no	minor	yes

4 Deteksi Pose Tubuh Manusia menggunakan PoseNet

Contoh menarik lain dari model yang dibagikan oleh TensorFlow.js adalah implementasi **PoseNet** untuk mendeteksi/mengestimasi pose tubuh manusia. Model ini akan mencoba mencari titik-titik sendi dari manusia yang terdeteksi dari *webcam* dan menggambarkan bentuk 'stickman' dari titik-titik sendi yang terdeteksi secara *real-time*. Model ini sangat berguna untuk mendeteksi pose hingga aksi yang dilakukan oleh seseorang



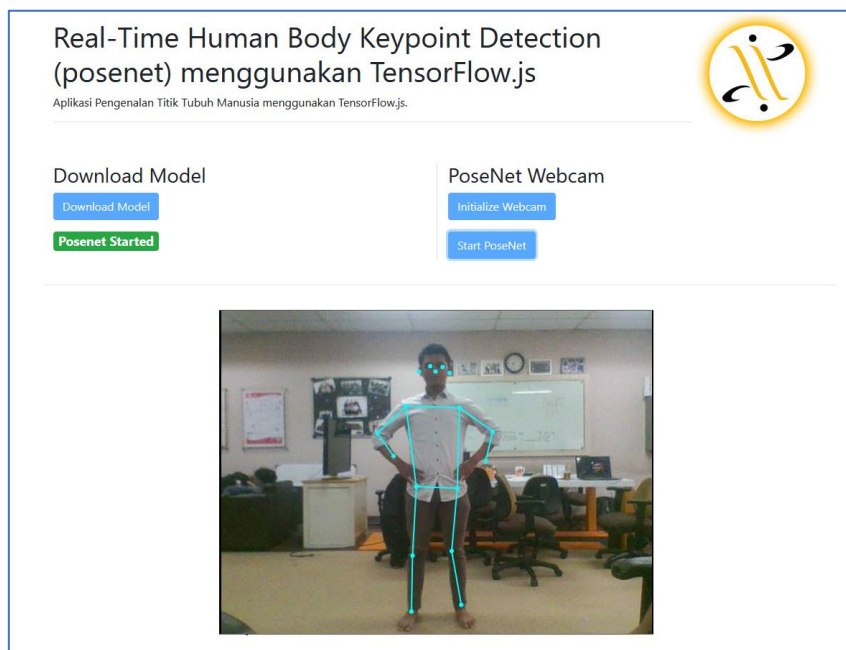
Sama seperti sebelumnya, untuk menggunakan model **PoseNet** tersebut, kita hanya perlu mengunduh *library* dengan sintaks

```
\ai_tfjs > npm install @tensorflow-models/posenet
```

Setelah *library* diunduh, jalankan sintaks untuk mencoba aplikasi web yang disediakan

```
\ai_tfjs > parcel modul_4\posenet.html
```

Tampilan aplikasi **posenet.html** saat dijalankan akan terlihat seperti gambar di bawah



5 Deteksi Benda menggunakan YOLO Detection

Dengan berkembangnya kultur berbagi pada komunitas *Computer Science* dan *Machine Learning*, para peneliti dan pengembang aplikasi Pembelajaran Mesin menjadi sangat mudah untuk membagikan hasil penelitian dan model yang mereka latih. Salah satu model menarik yang banyak digunakan adalah deteksi objek/benda menggunakan teknik yang disebut [YOLO Detection](#). **YOLO** adalah akronim dari teknik deteksi benda **You Only Look Once** yang dibangun oleh peneliti bernama [Joseph Redmon](#).

Joseph Redmon terus mengembangkan model **YOLO** yang bernama **darknet** dari versi 1 hingga versi terkini yaitu versi 3. Dan selama itu Joseph selalu membagikan model yang ia latih. Karena itu banyak kontributor lain yang menggunakan dan ikut mengembangkannya.

Model yang akan kita gunakan pada contoh kali ini adalah model yang telah dikonversi oleh kontributor [shaqian](#) menjadi model **TensorFlow.js**. Berdasarkan model tersebut, selanjutnya kami definisikan model dan fungsionalitas ntuk menggunakan salah satu model **YOLO** ke dalam sebuah package yang sudah kami publikasi ke situs **npm** [berikut](#). Untuk menggunakan fungsionalitas tersebut, kita hanya perlu mengunduhnya menggunakan sintaks **npm** berikut

```
\ai_tfjs > npm install tfjs-tiny-yolov3
```

Model yang kita gunakan di sini disebut sebagai **Tiny YOLO-v3**. Model versi terbaru dari **YOLO** yang dilatih menggunakan arsitektur yang kecil sehingga ringan untuk dijalankan. Setelah *library* diunduh, jalankan sintaks untuk mencoba aplikasi web **YOLO**

```
\ai_tfjs > parcel modul_4\yolo.html
```

Tampilan aplikasi **yolo.html** saat dijalankan akan terlihat seperti gambar di bawah


YOLO: Real-Time Object Detection menggunakan TensorFlow.js

Aplikasi Deteksi Benda YOLO menggunakan TensorFlow.js.

Download Model
Download Model

YOLO Detection Webcam
Initialize Webcam
Start Detection

person	bicycle	car	motorbike
aeroplane	bus	train	truck
boat	traffic light	fire hydrant	stop sign
parking meter	bench	bird	cat
dog	horse	sheep	cow
elephant	bear	zebra	giraffe
backpack	umbrella	handbag	tie
suitcase	frisbee	skis	snowboard
sports ball	kite	baseball bat	baseball glove
skateboard	surfboard	tennis racket	bottle



wine glass	cup	fork	knife
spoon	bowl	banana	apple
sandwich	orange	broccoli	carrot
hot dog	pizza	donut	cake
chair	sofa	pottedplant	bed
diningtable	toilet	tvmonitor	laptop
mouse	remote	keyboard	cell phone
microwave	oven	toaster	sink
refrigerator	book	clock	vase
scissors	teddy bear	hair drier	toothbrush