

# Copyright Notice

These slides are distributed under the Creative Commons License.

[DeepLearning.AI](#) makes these slides available for educational purposes. You may not use or distribute these slides for commercial purposes. You may make copies of these slides and use or distribute them for educational purposes as long as you cite [DeepLearning.AI](#) as the source of the slides.

For the rest of the details of the license, see

<https://creativecommons.org/licenses/by-sa/2.0/legalcode>

```
model = tf.sequential();

model.add(tf.layers.conv2d({inputShape: [28, 28, 1],
                                kernelSize: 3, filters: 8, activation: 'relu'}));

model.add(tf.layers.maxPooling2d({poolSize: [2, 2]}));

model.add(tf.layers.conv2d({filters: 16,
                                kernelSize: 3, activation: 'relu'}));

model.add(tf.layers.maxPooling2d({poolSize: [2, 2]}));

model.add(tf.layers.flatten());

model.add(tf.layers.dense({units: 128, activation: 'relu'}));

model.add(tf.layers.dense({units: 10, activation: 'softmax'}));
```

```
model = tf.sequential();
```

```
model.add(tf.layers.conv2d({inputShape: [28, 28, 1],  
                             kernelSize: 3, filters: 8, activation: 'relu'}));
```

```
model.add(tf.layers.maxPooling2d({poolSize: [2, 2]}));
```

```
model.add(tf.layers.conv2d({filters: 16,  
                             kernelSize: 3, activation: 'relu'}));
```

```
model.add(tf.layers.maxPooling2d({poolSize: [2, 2]}));
```

```
model.add(tf.layers.flatten());
```

```
model.add(tf.layers.dense({units: 128, activation: 'relu'}));
```

```
model.add(tf.layers.dense({units: 10, activation: 'softmax'}));
```

```
model = tf.sequential();

model.add(tf.layers.conv2d({inputShape: [28, 28, 1],
                                kernelSize: 3, filters: 8, activation: 'relu'}));

model.add(tf.layers.maxPooling2d({poolSize: [2, 2]}));

model.add(tf.layers.conv2d({filters: 16,
                                kernelSize: 3, activation: 'relu'}));

model.add(tf.layers.maxPooling2d({poolSize: [2, 2]}));

model.add(tf.layers.flatten());

model.add(tf.layers.dense({units: 128, activation: 'relu'}));

model.add(tf.layers.dense({units: 10, activation: 'softmax'}));
```

```
model = tf.sequential();

model.add(tf.layers.conv2d({inputShape: [28, 28, 1],
                                kernelSize: 3, filters: 8, activation: 'relu'}));

model.add(tf.layers.maxPooling2d({poolSize: [2, 2]}));

model.add(tf.layers.conv2d({filters: 16,
                                kernelSize: 3, activation: 'relu'}));

model.add(tf.layers.maxPooling2d({poolSize: [2, 2]}));

model.add(tf.layers.flatten());

model.add(tf.layers.dense({units: 128, activation: 'relu'}));

model.add(tf.layers.dense({units: 10, activation: 'softmax'}));
```

```
model = tf.sequential();

model.add(tf.layers.conv2d({inputShape: [28, 28, 1],
    kernelSize: 3, filters: 8, activation: 'relu'}));

model.add(tf.layers.maxPooling2d({poolSize: [2, 2]}));

model.add(tf.layers.conv2d({filters: 16,
    kernelSize: 3, activation: 'relu'}));

model.add(tf.layers.maxPooling2d({poolSize: [2, 2]}));

model.add(tf.layers.flatten());

model.add(tf.layers.dense({units: 128, activation: 'relu'}));

model.add(tf.layers.dense({units: 10, activation: 'softmax'}));
```

```
model = tf.sequential();

model.add(tf.layers.conv2d({inputShape: [28, 28, 1],
                                kernelSize: 3, filters: 8, activation: 'relu'}));

model.add(tf.layers.maxPooling2d({poolSize: [2, 2]}));

model.add(tf.layers.conv2d({filters: 16,
                                kernelSize: 3, activation: 'relu'}));

model.add(tf.layers.maxPooling2d({poolSize: [2, 2]}));

model.add(tf.layers.flatten());

model.add(tf.layers.dense({units: 128, activation: 'relu'}));

model.add(tf.layers.dense({units: 10, activation: 'softmax'}));
```

```
model = tf.sequential();

model.add(tf.layers.conv2d({inputShape: [28, 28, 1],
                                kernelSize: 3, filters: 8, activation: 'relu'}));

model.add(tf.layers.maxPooling2d({poolSize: [2, 2]}));

model.add(tf.layers.conv2d({filters: 16,
                                kernelSize: 3, activation: 'relu'}));

model.add(tf.layers.maxPooling2d({poolSize: [2, 2]}));

model.add(tf.layers.flatten());

model.add(tf.layers.dense({units: 128, activation: 'relu'}));

model.add(tf.layers.dense({units: 10, activation: 'softmax'}));
```



```
model = tf.sequential();

model.add(tf.layers.conv2d({inputShape: [28, 28, 1],
                                kernelSize: 3, filters: 8, activation: 'relu'}));

model.add(tf.layers.maxPooling2d({poolSize: [2, 2]}));

model.add(tf.layers.conv2d({filters: 16,
                                kernelSize: 3, activation: 'relu'}));

model.add(tf.layers.maxPooling2d({poolSize: [2, 2]}));

model.add(tf.layers.flatten());

model.add(tf.layers.dense({units: 128, activation: 'relu'}));

model.add(tf.layers.dense({units: 10, activation: 'softmax'}));
```

```
model = tf.sequential();

model.add(tf.layers.conv2d({inputShape: [28, 28, 1],
                                kernelSize: 3, filters: 8, activation: 'relu'}));

model.add(tf.layers.maxPooling2d({poolSize: [2, 2]}));

model.add(tf.layers.conv2d({filters: 16,
                                kernelSize: 3, activation: 'relu'}));

model.add(tf.layers.maxPooling2d({poolSize: [2, 2]}));

model.add(tf.layers.flatten());

model.add(tf.layers.dense({units: 128, activation: 'relu'}));

model.add(tf.layers.dense({units: 10, activation: 'softmax'}));
```

```
model = tf.sequential();

model.add(tf.layers.conv2d({inputShape: [28, 28, 1],
                                kernelSize: 3, filters: 8, activation: 'relu'}));

model.add(tf.layers.maxPooling2d({poolSize: [2, 2]}));

model.add(tf.layers.conv2d({filters: 16,
                                kernelSize: 3, activation: 'relu'}));

model.add(tf.layers.maxPooling2d({poolSize: [2, 2]}));

model.add(tf.layers.flatten());

model.add(tf.layers.dense({units: 128, activation: 'relu'}));

model.add(tf.layers.dense({units: 10, activation: 'softmax'}));
```

```
model = tf.sequential();

model.add(tf.layers.conv2d({inputShape: [28, 28, 1],
                                kernelSize: 3, filters: 8, activation: 'relu'}));

model.add(tf.layers.maxPooling2d({poolSize: [2, 2]}));

model.add(tf.layers.conv2d({filters: 16,
                                kernelSize: 3, activation: 'relu'}));

model.add(tf.layers.maxPooling2d({poolSize: [2, 2]}));

model.add(tf.layers.flatten());

model.add(tf.layers.dense({units: 128, activation: 'relu'}));

model.add(tf.layers.dense({units: 10, activation: 'softmax'}));
```

```
model = tf.sequential();

model.add(tf.layers.conv2d({inputShape: [28, 28, 1],
                                kernelSize: 3, filters: 8, activation: 'relu'}));

model.add(tf.layers.maxPooling2d({poolSize: [2, 2]}));

model.add(tf.layers.conv2d({filters: 16,
                                kernelSize: 3, activation: 'relu'}));

model.add(tf.layers.maxPooling2d({poolSize: [2, 2]}));

model.add(tf.layers.flatten());

model.add(tf.layers.dense({units: 128, activation: 'relu'}));

model.add(tf.layers.dense({units: 10, activation: 'softmax'}));
```

```
model = tf.sequential();

model.add(tf.layers.conv2d({inputShape: [28, 28, 1],
                                kernelSize: 3, filters: 8, activation: 'relu'}));

model.add(tf.layers.maxPooling2d({poolSize: [2, 2]}));

model.add(tf.layers.conv2d({filters: 16,
                                kernelSize: 3, activation: 'relu'}));

model.add(tf.layers.maxPooling2d({poolSize: [2, 2]}));

model.add(tf.layers.flatten());

model.add(tf.layers.dense({units: 128, activation: 'relu'}));

model.add(tf.layers.dense({units: 10, activation: 'softmax'}));
```

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

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



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

```
model.fit(trainXs, trainYs, {  
    batchSize: BATCH_SIZE,  
    validationData: [testXs, testYs],  
    epochs: 20,  
    shuffle: true,  
    callbacks: fitCallbacks  
});
```

```
model.fit(trainXs, trainYs, {  
    batchSize: BATCH_SIZE,  
    validationData: [testXs, testYs],  
    epochs: 20,  
    shuffle: true,  
    callbacks: fitCallbacks  
});
```

```
model.fit(trainXs, trainYs, {  
    batchSize: BATCH_SIZE,  
    validationData: [testXs, testYs],  
    epochs: 20,  
    shuffle: true,  
    callbacks: fitCallbacks  
});
```

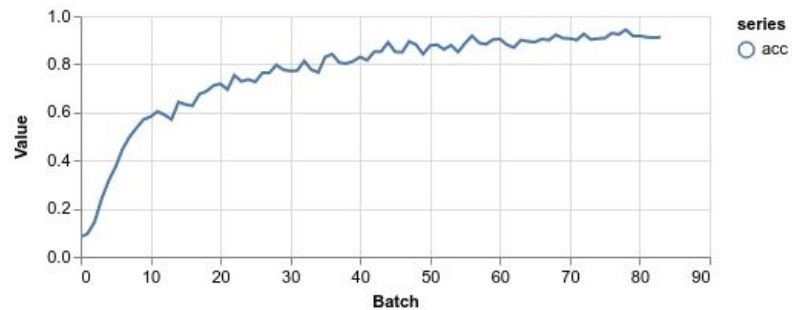
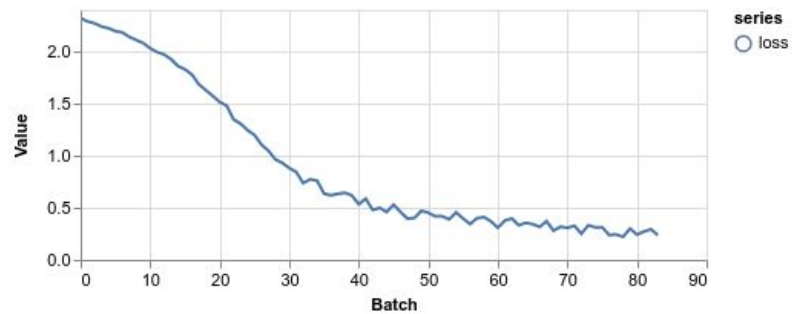
```
model.fit(trainXs, trainYs, {  
    batchSize: BATCH_SIZE,  
    validationData: [testXs, testYs],  
    epochs: 20,  
    shuffle: true,  
    callbacks: fitCallbacks  
});
```

```
model.fit(trainXs, trainYs, {  
    batchSize: BATCH_SIZE,  
    validationData: [testXs, testYs],  
    epochs: 20,  
    shuffle: true,  
    callbacks: fitCallbacks  
});
```

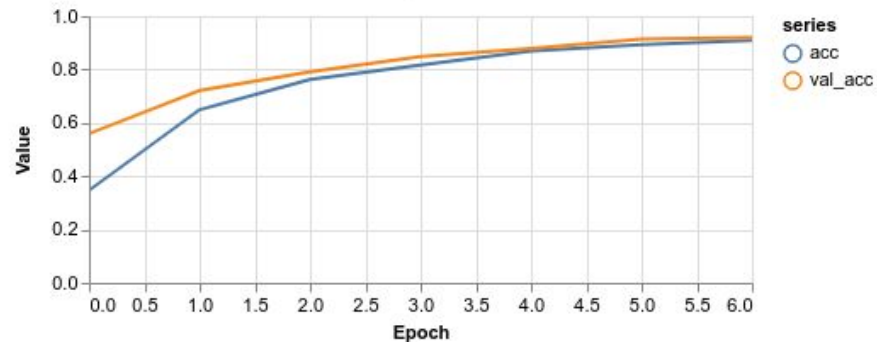
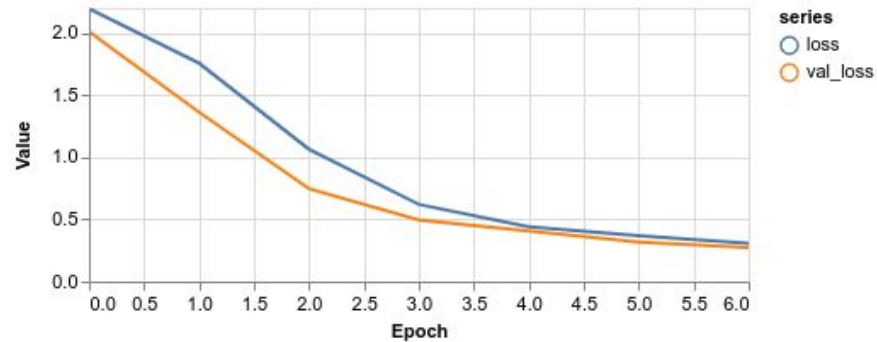
```
model.fit(trainXs, trainYs, {  
    batchSize: BATCH_SIZE,  
    validationData: [testXs, testYs],  
    epochs: 20,  
    shuffle: true,  
    callbacks: fitCallbacks  
});
```

## Model Training

### onBatchEnd



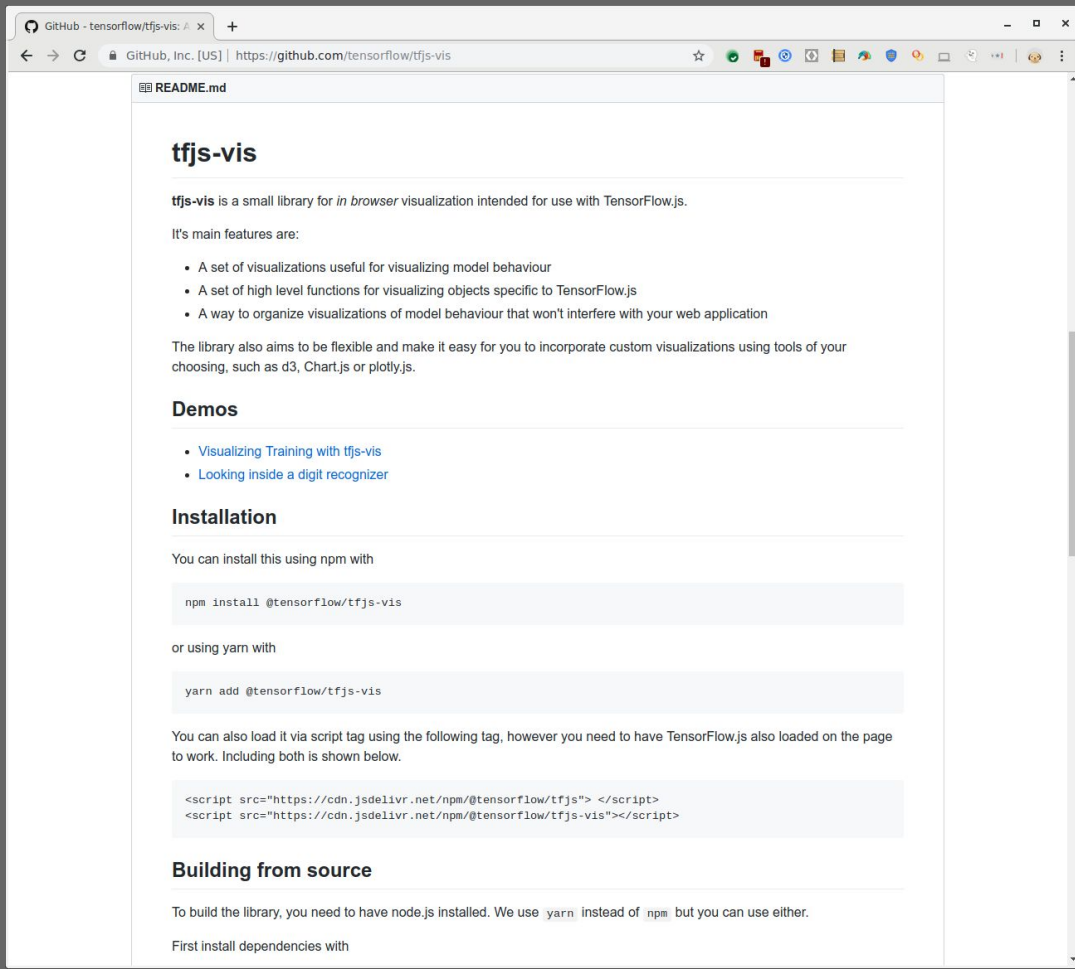
### onEpochEnd







# https://github.com/tensorflow/tfjs-vis



A screenshot of a web browser displaying the GitHub README for the tensorflow/tfjs-vis repository. The browser's address bar shows the URL https://github.com/tensorflow/tfjs-vis. The README content includes a title 'tfjs-vis', a description of the library as a small browser visualization tool for TensorFlow.js, a list of main features, a section for demos with links to training and digit recognizer visualizations, an installation section with npm and yarn commands, a script tag for loading the library, and a section for building from source.

GitHub - tensorflow/tfjs-vis: ✕ +

← → ↻ 🔒 GitHub, Inc. [US] | https://github.com/tensorflow/tfjs-vis

📖 README.md

## tfjs-vis

**tfjs-vis** is a small library for *in browser* visualization intended for use with TensorFlow.js.

It's main features are:

- A set of visualizations useful for visualizing model behaviour
- A set of high level functions for visualizing objects specific to TensorFlow.js
- A way to organize visualizations of model behaviour that won't interfere with your web application

The library also aims to be flexible and make it easy for you to incorporate custom visualizations using tools of your choosing, such as d3, Chart.js or plotly.js.

## Demos

- [Visualizing Training with tfjs-vis](#)
- [Looking inside a digit recognizer](#)

## Installation

You can install this using npm with

```
npm install @tensorflow/tfjs-vis
```

or using yarn with

```
yarn add @tensorflow/tfjs-vis
```

You can also load it via script tag using the following tag, however you need to have TensorFlow.js also loaded on the page to work. Including both is shown below.

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

## Building from source

To build the library, you need to have node.js installed. We use `yarn` instead of `npm` but you can use either.

First install dependencies with

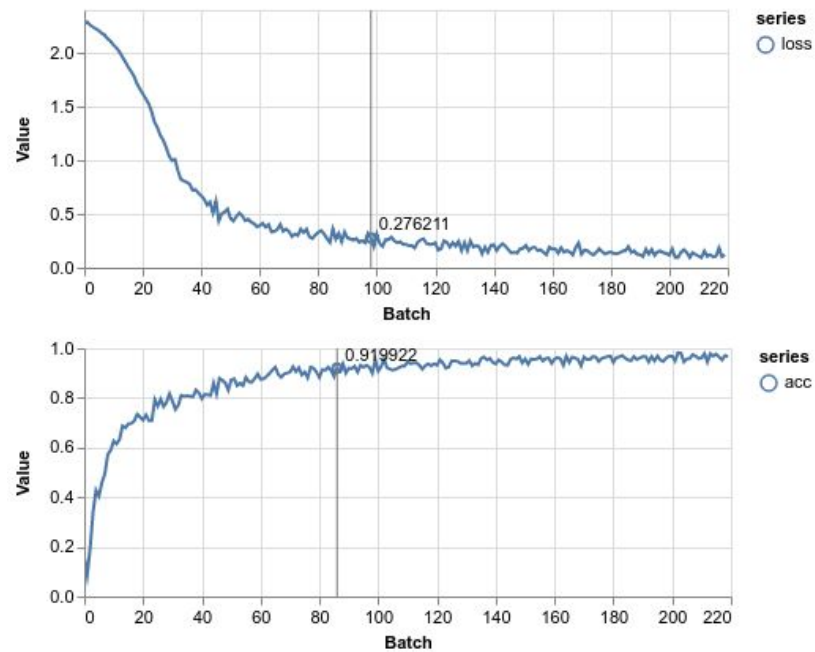
```
model.fit(trainXs, trainYs, {  
    batchSize: BATCH_SIZE,  
    validationData: [testXs, testYs],  
    epochs: 20,  
    shuffle: true,  
    callbacks: fitCallbacks  
});
```

```
model.fit(trainXs, trainYs, {  
    batchSize: BATCH_SIZE,  
    validationData: [testXs, testYs],  
    epochs: 20,  
    shuffle: true,  
    callbacks: fitCallbacks  
});
```

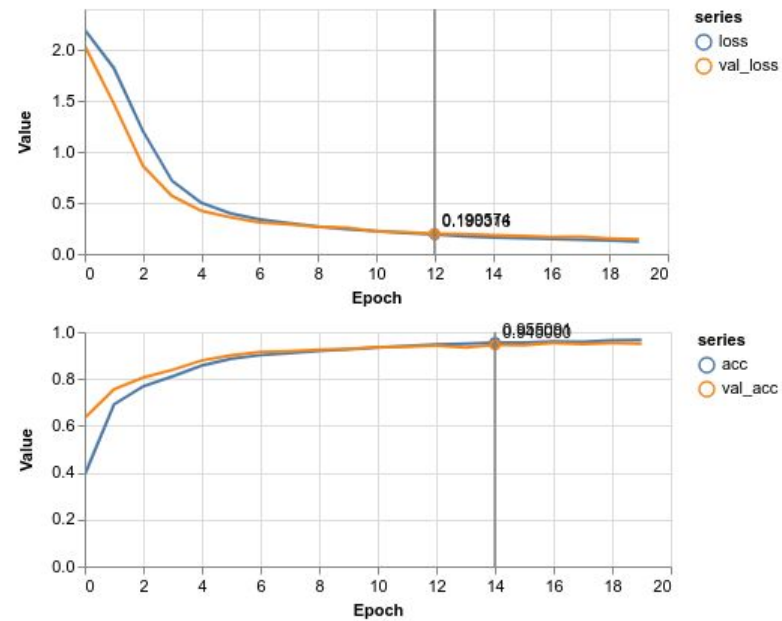


```
const metrics = ['loss', 'val_loss', 'acc', 'val_acc'];  
const container = { name: 'Model Training', styles: { height: '1000px' } };  
const fitCallbacks = tfvis.show.fitCallbacks(container, metrics);
```

### onBatchEnd

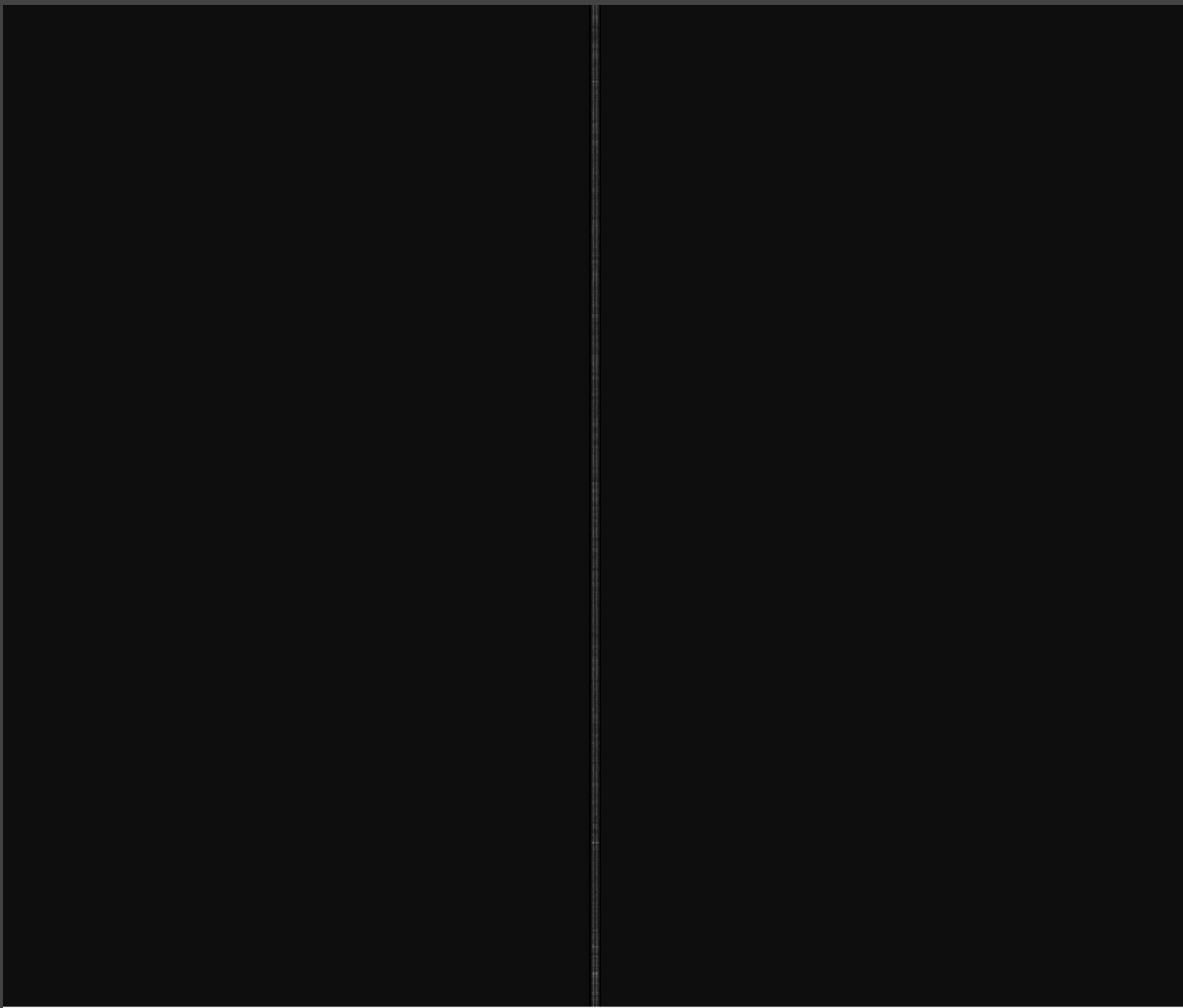


### onEpochEnd









1. The first step in the process is to identify the problem or issue that needs to be addressed. This involves gathering information and understanding the context of the problem.

2. Once the problem is identified, the next step is to define the objectives and goals of the project. This helps to clarify what needs to be achieved and provides a clear direction for the team.

3. The third step is to develop a plan or strategy to address the problem. This involves breaking down the problem into smaller, manageable tasks and determining the resources needed to complete each task.

4. The fourth step is to implement the plan. This involves putting the strategy into action and monitoring progress regularly to ensure that the project is on track.

5. The final step is to evaluate the results of the project. This involves assessing the outcomes against the objectives and goals and identifying any areas for improvement.



00 01 02 03 04 05 06 07 08 09

0000:0000	00	00	00	00	00	00	00	01	00	00
0000:000A	00	00	00	01	00	00	00	00	00	00
0000:0014	00	00	00	00	01	00	00	00	00	00
0000:001E	00	00	00	00	00	00	01	00	00	00
0000:0028	00	01	00	00	00	00	00	00	00	00
0000:0032	00	00	00	00	00	00	00	00	01	00
0000:003C	00	01	00	00	00	00	00	00	00	00
0000:0046	01	00	00	00	00	00	00	00	00	00
0000:0050	00	00	00	00	00	00	00	00	00	01
0000:005A	00	00	00	00	00	00	00	00	01	00
0000:0064	01	00	00	00	00	00	00	00	00	00
0000:006E	00	00	00	01	00	00	00	00	00	00

```
export class MnistData {  
  ...  
  
  async load() {  
    // Download the sprite and slice it  
    // Download the labels and decode them  
  }  
  
  nextTrainBatch(){  
    // Get the next training batch  
  }  
  
  nextTestBatch(){  
    // Get the next test batch  
  }  
}
```

```
export class MnistData {
```

```
...
```

```
  async load() {
```

```
    // Download the sprite and slice it
```

```
    // Download the labels and decode them
```

```
  }
```

```
  nextTrainBatch(){
```

```
    // Get the next training batch
```

```
  }
```

```
  nextTestBatch(){
```

```
    // Get the next test batch
```

```
  }
```

```
}
```

```
export class MnistData {  
  ...  
  
  async load() {  
    // Download the sprite and slice it  
    // Download the labels and decode them  
  }  
  
  nextTrainBatch(){  
    // Get the next training batch  
  }  
  
  nextTestBatch(){  
    // Get the next test batch  
  }  
}
```

```
export class MnistData {  
  ...  
  
  async load() {  
    // Download the sprite and slice it  
    // Download the labels and decode them  
  }  
  
  nextTrainBatch(){  
    // Get the next training batch  
  }  
  
  nextTestBatch(){  
    // Get the next test batch  
  }  
}
```



```
const data = new MnistData();  
await data.load();
```

```
const [trainXs, trainYs] = tf.tidy(() => {  
  const d = data.nextTrainBatch(TRAIN_DATA_SIZE);  
  return [  
    d.xs.reshape([TRAIN_DATA_SIZE, 28, 28, 1]),  
    d.labels  
  ];  
});
```

```
const [trainXs, trainYs] = tf.tidy(() => {  
  const d = data.nextTrainBatch(TRAIN_DATA_SIZE);  
  return [  
    d.xs.reshape([TRAIN_DATA_SIZE, 28, 28, 1]),  
    d.labels  
  ];  
});
```

```
const [trainXs, trainYs] = tf.tidy(() => {  
  const d = data.nextTrainBatch(TRAIN_DATA_SIZE);  
  return [  
    d.xs.reshape([TRAIN_DATA_SIZE, 28, 28, 1]),  
    d.labels  
  ];  
});
```

```
const [trainXs, trainYs] = tf.tidy(() => {  
  const d = data.nextTrainBatch(TRAIN_DATA_SIZE);  
  return [  
    d.xs.reshape([TRAIN_DATA_SIZE, 28, 28, 1]),  
    d.labels  
  ];  
});
```

```
const [trainXs, trainYs] = tf.tidy(() => {  
  const d = data.nextTrainBatch(TRAIN_DATA_SIZE);  
  return [  
    d.xs.reshape([TRAIN_DATA_SIZE, 28, 28, 1]),  
    d.labels  
  ];  
});
```

```
const [trainXs, trainYs] = tf.tidy(() => {  
  const d = data.nextTrainBatch(TRAIN_DATA_SIZE);  
  return [  
    d.xs.reshape([TRAIN_DATA_SIZE, 28, 28, 1]),  
    d.labels  
  ];  
});
```