基于Kmeans聚类的图片色彩提取与反向填充

数据可视化之JS Canvas 图像处理

柔化 K-means填充

基于 K-means 的聚类
从图像中提取色彩
数据可视化的配色

机器学习的 JavaScript 库

- 基于 Node.JS
- 纯 JavaScript 库
- https://github.com/mljs/kmeans
- **<script src="https://www.lactame.com/lib/ml/4.0.0/ml.min.js"></script>**

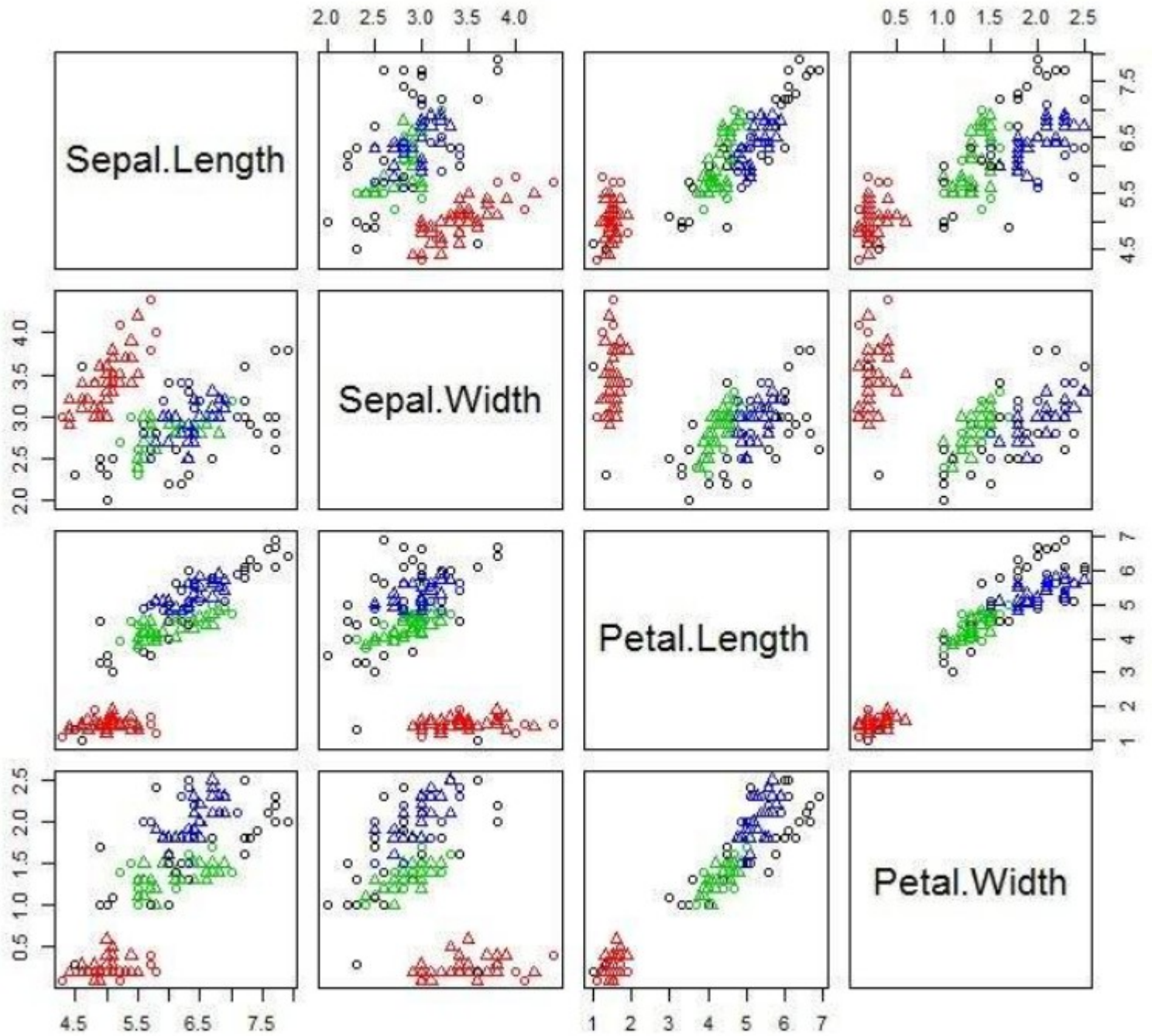
Iris 数据集


张赫
李多海
李凡秀
KBS2TV 2013년 2월 13일
수요일 밤 10시 첫 방송!!
IRIS II
D-24


广告 | 找鸢尾花,上阿里巴巴,海量爆品等你抢批!

IRIS 数据特征

- 描述 & 下载
- http://mirlab.org/jang/books/dcpr/dat



数据格式 csv

- 3 类
- 150 个数据
- 用 D3 直方图改成散点图

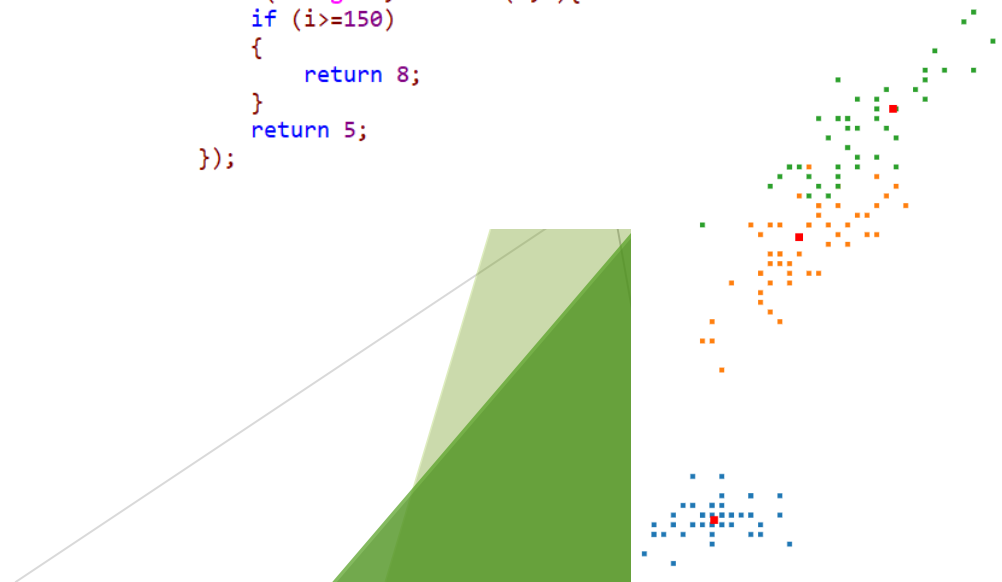
```
Id,SepalLengthCm,SepalWidthCm,PetalLengthCm,PetalWidthCm,Species
1,5.1,3.5,1.4,0.2,Iris-setosa
2,4.9,3.0,1.4,0.2,Iris-setosa
3,4.7,3.2,1.3,0.2,Iris-setosa
4,4.6,3.1,1.5,0.2,Iris-setosa
5,5.0,3.6,1.4,0.2,Iris-setosa
6,5.4,3.9,1.7,0.4,Iris-setosa
7,4.6,3.4,1.4,0.3,Iris-setosa
```

```
d3.csv("data/Iris.csv").then(function(root) {

    //console.log(root);
    var svg = d3.select("body")          //选择<body>
                .append("svg")           //在<body>中添加<svg>
                .attr("width", width)    //设定<svg>的宽度属性
                .attr("height", height);//设定<svg>的高度属性

    var rect = svg.selectAll("rect")
                  .data(root)      //绑定数据
                  .enter()         //获取enter部分
                  .append("rect")      //添加rect元素，使其与绑定数组的长度一致
                  .attr("x", function(d,i){       //设置矩形的x坐标
                      return  d.PetalLengthCm*100;
                  })
                  .attr("transform","translate("+width/4+","+height/2+")")
                  .attr("y", function(d){     //设置矩形的y坐标
                      return height-d.SepalLengthCm*100;
                  })
                  .attr("fill", function(d,i){        //设置矩形的y坐标
                      return color[Math.floor(i/50)];
                  })
                  .attr("width",5)        //设置矩形的宽度
                  .attr("height",5);
```

计算聚类中心 / 显示

```
d3.csv("data/Iris.csv").then(function(root) {

    for (var i=0;i<data.length;i++){
        data[i][0]=parseFloat(root[i].SepalLengthCm);
        data[i][1]=parseFloat(root[i].SepalWidthCm);
        data[i][2]=parseFloat(root[i].PetalLengthCm);
        data[i][3]=parseFloat(root[i].PetalWidthCm);
        //console.log(data[i][0]);
    }

    console.log(typeof(data[0][0]),data[0],data[51],data[101]);
    console.log(data.length);
    var centers = [data[0],data[51],data[101]];

    var ans = ML.KMeans(data,3, {initialization: centers });
    console.log(ans);
    console.log(ans.centroids[0].centroid);
    console.log(typeof(ans.centroids[0]));

    data.push( ans.centroids[0].centroid );
    data.push( ans.centroids[1].centroid );
    data.push( ans.centroids[2].centroid );

    draw();

})
```

```
function draw(){
        var svg = d3.select("body")          //选择<body>
                .append("svg")          //在<body>中添加<svg>
                .attr("width", width)   //设定<svg>的宽度属性
                .attr("height", height);//设定<svg>的高度属性
        var rect = svg.selectAll("rect")
                    .data(data)     //绑定数据
                    .enter()            //获取enter部分
                    .append("rect")     //添加rect元素，使其与绑定数组的长度一致
                    .attr("x", function(d,i){       //设置矩形的x坐标
                        return  d[0]*100;
                    })
                    .attr("transform","translate("+width/8+",0)")
                    .attr("y", function(d){     //设置矩形的y坐标
                        return height-d[2]*100;
                    })
                    .attr("fill", function(d,i){        //设置矩形的y坐标
                        if (i>=150)
                        {
                            return "#FF0000";
                        }
                        return color[Math.floor(i/50)];
                    })
                    .attr("width",function(d,i){
                        if (i>=150)
                        {
                            return 8;
                        }
                        return 5;
                    })
                    .attr("height",function(d,i){
                        if (i>=150)
                        {
                            return 8;
                        }
                        return 5;
                    });
}
```

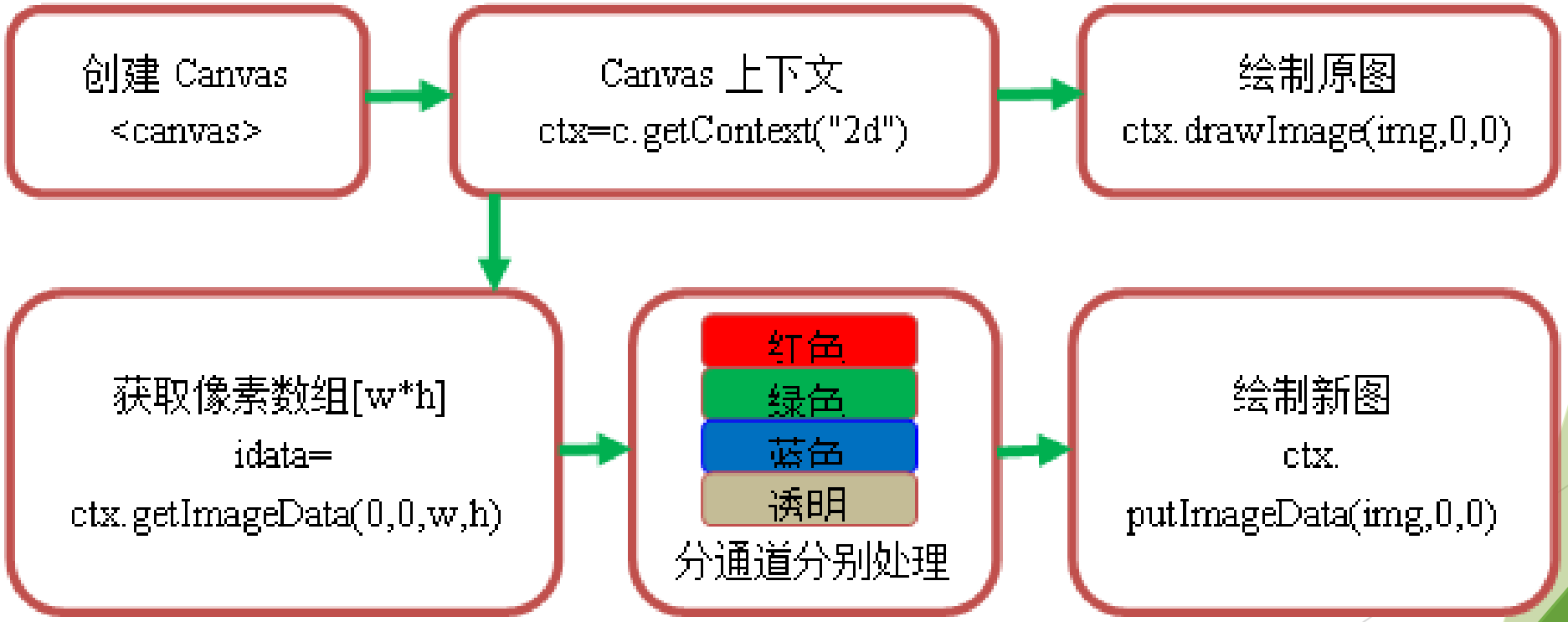


图像处理 Canvas

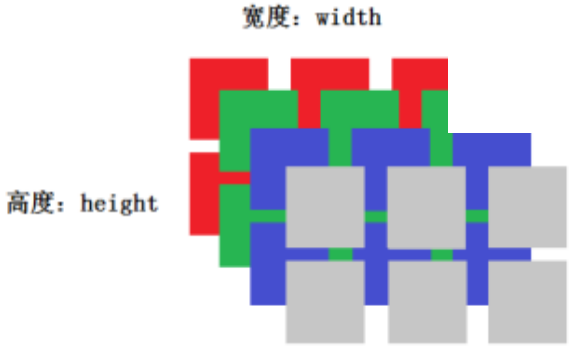
- Canvas 通过 JavaScript 来绘制 2D 图形，是逐像素进行渲染的，一旦图形被绘制完成，它不会继续得到浏览器的关注。如果其位置或色彩发生变化，那么整个场景需要重新绘制。
- Canvas 依赖分辨率，不支持事件处理器，较弱的文本渲染能力，能够以 .png 或 .jpg 格式保存结果图像，适合图像密集型的游戏，对象会被频繁重绘。

Canvas 绘图 - 像素级

图像像素数据后，返回一个数组，其中每一个像素用数组的四段表示，分别表示红（ R ）绿（ G ）蓝（ B ）和透明度（ alpha ）信息。对于图像的操作需要对 RGB 色彩通道分别处理。



RGB 通道分别处理




```
<canvas id="myCanvas" width="800" height="600"></canvas>
<script type="text/javascript">
    var c=document.getElementById("myCanvas");
    var cxt=c.getContext("2d");
    var img=new Image()
    img.src="color.jpg"
    cxt.drawImage(img,0,0);
    var imageData = cxt.getImageData(0,0,c.width, c.height);
    var idata = imageData.data;
    //dataMatrix
    var DM=new Array(c.height);
    var datargb=new Array(c.height*c.width);
    for(var i=0;i<c.height;i++){
            DM[i]=new Array(c.width);
            for(var j=0;j<c.width;j++){
                DM[i][j]=new Array(4);

                DM[i][j][0]=idata[(i*(c.width)+j)*4];
                DM[i][j][1]=idata[(i*(c.width)+j)*4+1];
                DM[i][j][2]=idata[(i*(c.width)+j)*4+2];
                DM[i][j][3]=idata[(i*(c.width)+j)*4+3];

                datargb[i*c.width+j]=new Array(3);
                datargb[i*c.width+j][0]=idata[(i*(c.width)+j)*4];
                datargb[i*c.width+j][1]=idata[(i*(c.width)+j)*4+1];
                datargb[i*c.width+j][2]=idata[(i*(c.width)+j)*4+2];
                //datargb[i*c.height+j][3]=idata[(i*(c.width)+j)*4+3];

            }
    }
```

简单操作：与像素位置无关

- 直接操作各个数值即可
- Grav=R*0.299+G*0.587+B*0.114

图 10-6　Canvas 图像处理灰度图像

阈值=200　　阈值=100

图 10-7　Canvas 图像处理二值图像

图 10-5　Canvas 图像处理负片效果

稍复杂的像素操作：与像素位置相关

- 像素位置变化
- 或者与位置相关

翻转次数=2n+1

翻转次数=2n

柔化次数=1

柔化次数=5

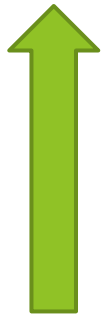
原图

锐化两次

<script src="https://www.lactame.com/lib/ml/4.0.0/ml.min.js"></script>

提取像素数据 - 用 K-means 聚类

与位置相关？



- var cc7 = ML.KMeans(datargb,7, {initialization: cent });
- console.log(cc7);
- console.log(cc7.centroids);

```
var cent=new Array(7);

console.log(datargb.length,datargb[0],datargb[500],datargb[3000]);
//色彩聚类
for(var i=0;i<7;i++){
     cent[i]=datargb[Math.floor(Math.random()*datargb.length)];
     console.log(cent[i]);
}

var cc7 = ML.KMeans(datargb,7, {initialization: cent });
console.log(cc7);

console.log(cc7.centroids);
```

聚类结果

数据可视化之JS Canvas 图像处理

柔化

127.0.0.1:8080/imgcolor/imgcolor2.htm

基于K-Means聚类的图片色彩提取

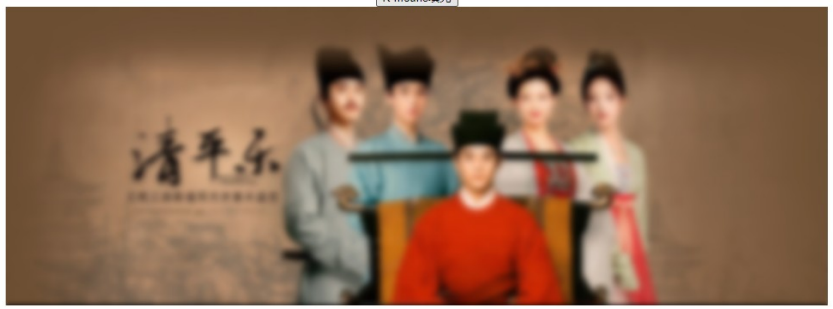
数据可视化之JS Canvas 图像处理

柔化 K-means填充

基于K-Means聚类的图片色彩提取

数据可视化之JS Canvas 图像处理

柔化 K-means填充



基于Kmeans聚类的图片色彩提取与反向填充

数据可视化之JS Canvas 图像处理

柔化 K-means填充

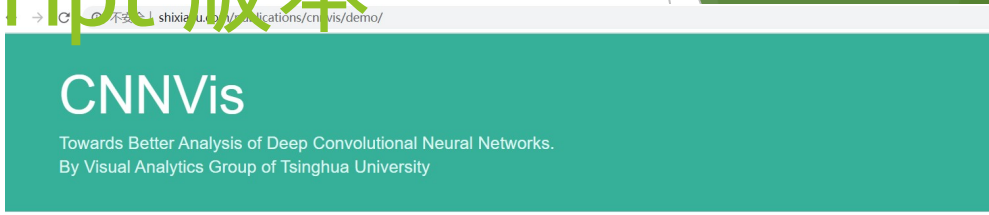
小结

- 1.Iris 数据散点图
- 2. K-Means 算法计算 Iris 数据的聚类中心
- 3. 在 JavaScript 中操作图像像素
- 4. 对像素色彩聚类
- 5. 用调色板显示聚类结果

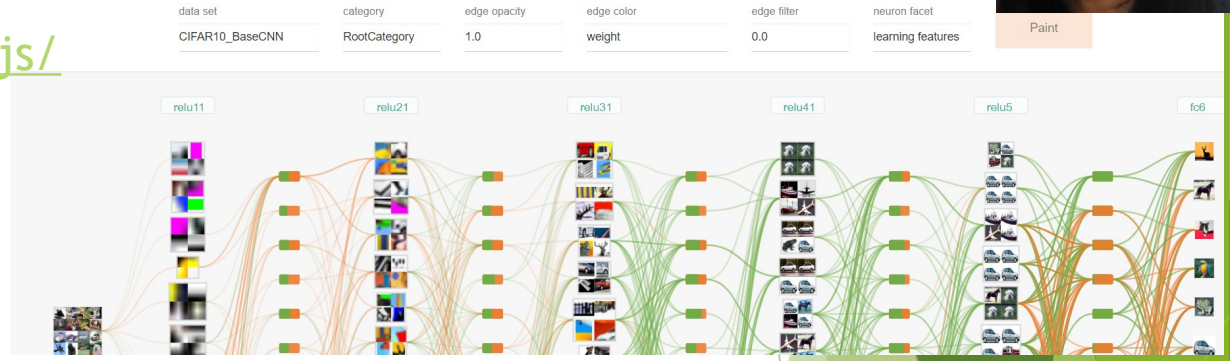
作业

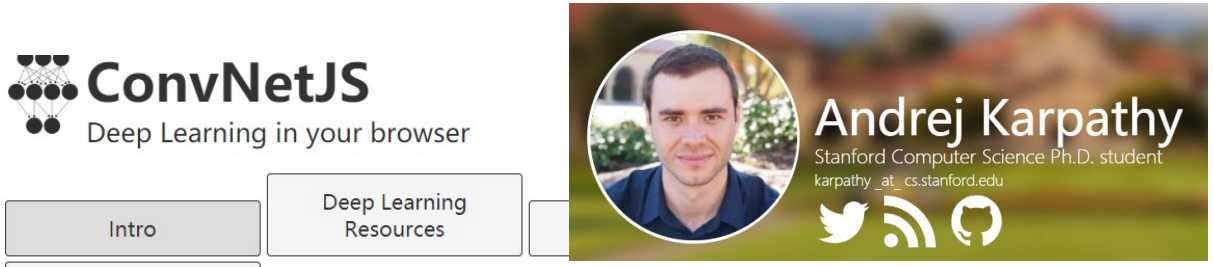
- 6. 第 5 步的像素不在图中，计算聚类中心最近的像素作为配色
- 7. 用最近像素对原图重新填充

机器学习与深度学习 JavaScript 版本 TensorFlow.JS



- https://cs.stanford.edu/people/karpathy/convnetjs/

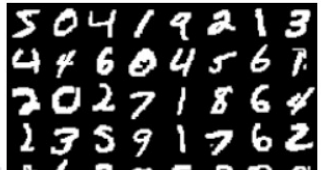




ConvNetJS is a Javascript library for training Deep Learning models (Neural Networks) entirely in your browser. Open a tab and you're training. No software requirements, no compilers, no installations, no GPUs, no sweat.

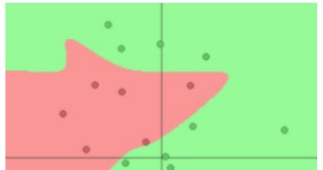
Browser Demos

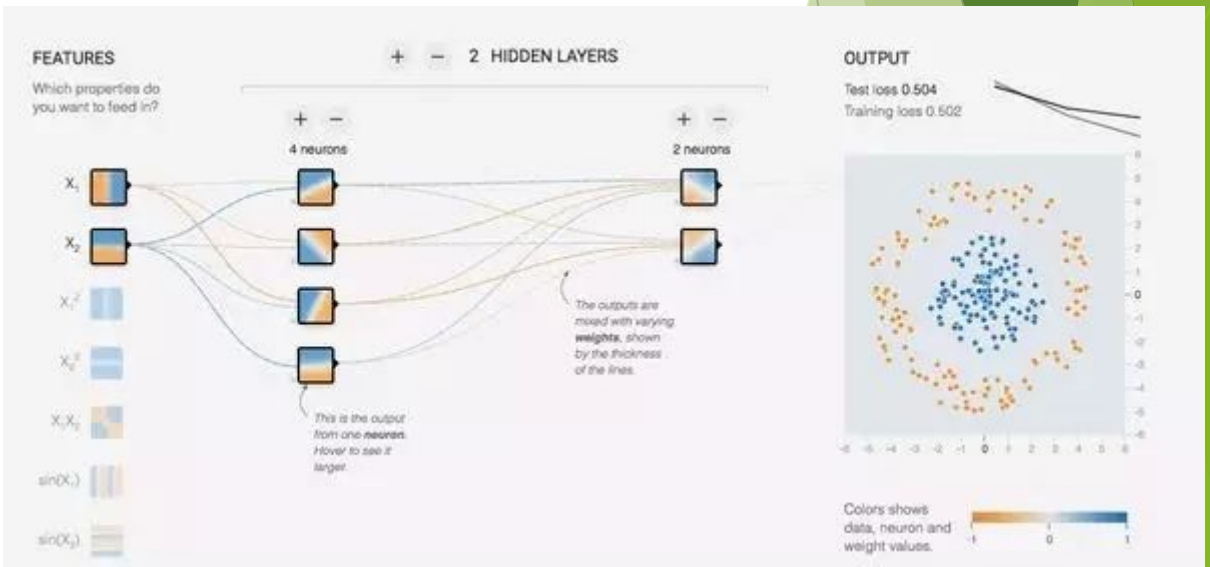
Classify **MNIST** digits with a Convolutional Neural Network



Classify **CIFAR-10** with Convolutional Neural Network

Interactively classify toy 2-D data with a Neural Network





期待佳作！

迁移创新