РАФАЭЛЬ МУН(RAPHAEL MUN)

Статьи по машинному обучению в браузере с использованием фреймворка TensorFlow.js

УЧЕБНЫЕ РУКОВОДСТВА

Перевод: С. Кузнецов, 2023 г.

Articles on machine training in the browser with use of a framework TensorFlow.js

Raphael Mun

2020 · 2021

https://www.codeproject.com/Articles/instafluff#Article

Статьи по машинному обучению в браузере с использованием фреймворка TensorFlow.js

Рафаэль Мун

2020 · 2021

https://www.codeproject.com/Articles/instafluff#Article

Перевод: С. Кузнецов, 19.11.2023

Статья 6 « Интерпретация жестов рук и языка знаков в веб-камере с AI-системой, используя TensorFlow.js»

Статья **6 Интерпретация жестов рук и языка знаков в веб-камере с AI-системой, используя TensorFlow.js(Interpreting Hand Gestures and Sign Language in the Webcam with AI using TensorFlow.js); https://www.codeproject.com/Articles/5272777/Interpreting-Hand-Gestures-and-Sign-Language-in-th)** является статьей из серии статей **Обнаружение касания лица с помощью Tensorflow.js(Face Touch Detection with Tensorflow.js)**.

15 июля 2020

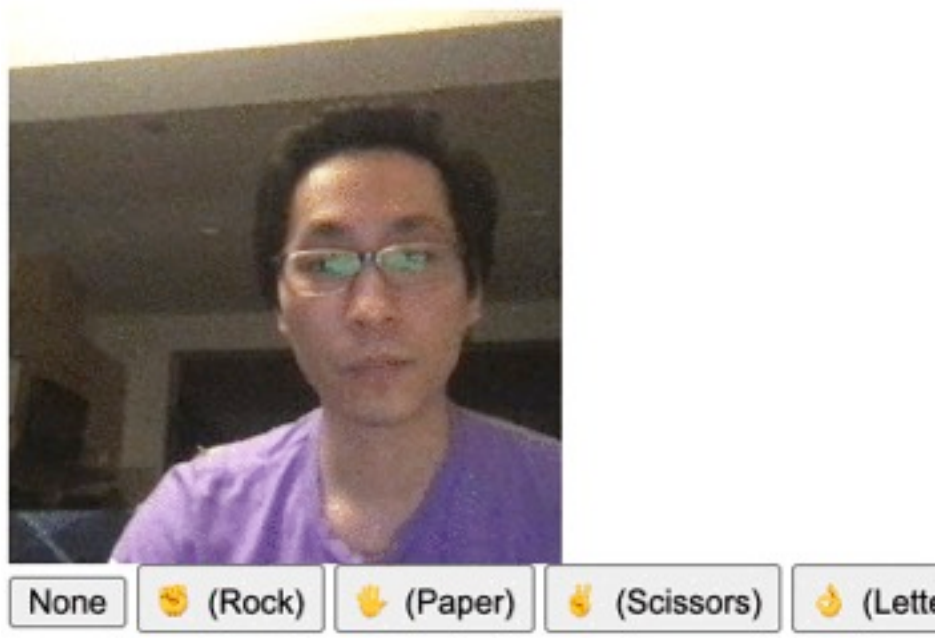
В этой статье мы будем через веб-камеру делать фотографии различных жестов рук и использовать **передачу обучения(transfer learning) в предварительно обученную MobileNet-модели(pre-trained MobileNet model)** для построения **AI-системы машинного зрения(computer vision AI)**, которая может **распознать различные жесты в режиме реального времени(recognize the various gestures in real time)**.

Здесь мы рассмотрим темы: **обнаружение жестов рук(detecting hand gestures)**, создание нашей стартовой точки и использование ее для обнаружения четырех различных категорий жестов: **(Нет/None)**, **(Кулак/Rock)**, **(Ладонь/Paper)**, **(Ножницы/Scissors)**, и добавим некоторые категории жестов **американского языка знаков(American Sign Language; ASL)**, чтобы исследовать, насколько трудно **AI**-системе обнаружить другие жесты.

TensorFlow + JavaScript. Самый популярный, ультрасовременный **AI**-фреймворк(инфраструктура) теперь поддерживает наиболее широко используемый язык программирования на планете, поэтому давайте заставим волшебство произойти посредством **глубокого изучения(deep learning)** прямо в нашем веб-браузере, ускоренном **графическим процессорным**

устройством(ГПУ; GPU) через графическую библиотеку WebGL, используя фреймворк машинного обучения TensorFlow.js!

В этой статье мы будем через веб-камеру делать фотографии различных жестов рук и использовать передачу обучения(transfer learning) в предварительно обученную MobileNet-модели(pre-trained MobileNet model) для построения AI-системы машинного зрения(computer vision AI), которая может распознать различные жесты в режиме реального времени(recognize the various gestures in real time).



Начальная точка

Чтобы распознать множество жестов рук, мы собираемся использовать почти готовый стартовый код и развернуть его, чтобы обнаружить больше категорий объектов. Для этого проекта вы должны сделать следующее:

- Импортируйте TensorFlow.js и tf-data.js от TensorFlow
- Определите элемент текст состояния Касание(Touch) по сравнению с состоянием Нет касания(Not-Touch)
- Добавьте видео элемент для веб-камеры

- Выполните прогноз на модели каждые `200 мс`, вместо выбора изображения, но после того, как модель будет обучена впервые
- Покажите результат прогноза
- Загрузите предварительно обученную `MobileNet`-модель и `подготовьтесь к передаче обученности(prepare for transfer learning)` стольких же категорий, сколько есть меток
- Обучайте и классифицируйте множество нестандартных(пользовательских) объектов в изображениях
- Пропустите изображение расположения и целевых выборок в учебном процессе, чтобы сохранить их для выполнений множества обучения

Вот код нашей начальной точки:

JavaScript

```
<html>
    <head>
        <meta charset="UTF-8">
        <title>Interpreting Hand Gestures and Sign Language in the Webcam
with AI using TensorFlow.js</title>
        <script
src="https://cdn.jsdelivr.net/npm/@tensorflow/tfjs@2.0.0/dist/tf.min.js"><
/script>
        <script
src="https://cdn.jsdelivr.net/npm/@tensorflow/tfjs-data@2.0.0/dist/tf-data
.min.js"></script>
        <style>
            img, video {
                object-fit: cover;
            }
        </style>
    </head>
    <body>
        <video autoplay playsinline muted id="webcam" width="224"
height="224"></video>
        <div id="buttons">
            <button onclick="captureSample(0)">(Нет/None)</button>
            <button onclick="captureSample(1)"> (Кулак/Rock)</button>
            <button onclick="captureSample(2)"> (Ладонь/Paper)</button>
         <button onclick="captureSample(3)"> (Ножницы/Scissors)</button>
              <button onclick="trainModel()">Обучить/Train</button>
        </div>
        <h1 id="status">Загрузка.../Loading...</h1>
```

```
<script>
let trainingData = [];

const labels = [
    "None",
    " (Кулак/Rock)",
    " (Ладонь/Paper)",
    " (Ножницы/Scissors)",
];

function setText( text ) {
    document.getElementById( "status" ).innerText = text;
}

async function predictImage() {
    if( !hasTrained ) { return; } // Пропустите прогноз,
     // пока не обучена
     // Skip prediction until trained
    const img = await getWebcamImage();
    let result = tf.tidy( () => {
        const input = img.reshape( [ 1, 224, 224, 3 ] );
        return model.predict( input );
    });
    img.dispose();
    let prediction = await result.data();
    result.dispose();
    // Получите индекс самого высокого значения в прогнозе
     // Get the index of the highest value in the prediction
    let id = prediction.indexOf( Math.max( ...prediction ) );
    setText( labels[ id ] );
}

function createTransferModel( model ) {
     // Создайте усеченную базовую модель(удалите "верхние" слои,
     // классификация + слои узкого места)
     // Create the truncated base model (remove the "top" layers,
     // classification + bottleneck layers)
    const bottleneck = model.getLayer( "dropout" );
     // Финальный уровень, перед слоем conv_pred,
     // предварительно обученным слоем классификации
     // This is the final layer before
     // the conv_pred pre-trained classification layer
    const baseModel = tf.model({
        inputs: model.inputs,
        outputs: bottleneck.output
    });
     // Заморозьте сверточную базу
     // Freeze the convolutional base

    for( const layer of baseModel.layers ) {
        layer.trainable = false;
    }
     // Добавьте «голову» классификации
     // Add a classification head
```

```
    const newHead = tf.sequential();
    newHead.add( tf.layers.flatten( {
              inputShape: baseModel.outputs[ 0 ].shape.slice( 1 )
              } ) );
    newHead.add( tf.layers.dense( {
              units: 100, activation: 'relu' } ) );
    newHead.add( tf.layers.dense( {
              units: 100, activation: 'relu' } ) );
    newHead.add( tf.layers.dense( {
              units: 10, activation: 'relu' } ) );
    newHead.add( tf.layers.dense( {
        units: labels.length,
        kernelInitializer: 'varianceScaling',
        useBias: false,
        activation: 'softmax'
    } ) );
     // Постройте новую модель
     // Build the new model
    const newOutput = newHead.apply( baseModel.outputs[ 0 ] );
    const newModel = tf.model(
          { inputs: baseModel.inputs, outputs: newOutput } );
    return newModel;
}

async function trainModel() {
    hasTrained = false;
    setText( "Обучение.../ Training..." );
     // Установите данные обучения(тренировки)
     // Setup training data
const imageSamples = [];
const targetSamples = [];
trainingData.forEach( sample => {
    imageSamples.push( sample.image );
    let cat = [];
    for( let c = 0; c < labels.length; c++ ) {
        cat.push( c === sample.category ? 1 : 0 );
    }
    targetSamples.push( tf.tensor1d( cat ) );
});
const xs = tf.stack( imageSamples );
const ys = tf.stack( targetSamples );

// Обучите модель на новых выборках изображений
// Train the model on new image samples
model.compile( { loss: "meanSquaredError",
         optimizer: "adam", metrics: [ "acc" ] } );

await model.fit( xs, ys, {
    epochs: 30,
    shuffle: true,
    callbacks: {
        onEpochEnd: ( epoch, logs ) => {
            console.log( "Эпоха #/Epoch #", epoch, logs );
```

```
                    }
                }
            });
            hasTrained = true;
        }

         // Модель Mobilenet v1 0.25 для изображений размером 224x224
         // Mobilenet v1 0.25 224x224 model
        const mobilenet =
"https://storage.googleapis.com/tfjs-models/tfjs/mobilenet_v1_0.25_224/mod
el.json";

        let model = null;
        let hasTrained = false;

        async function setupWebcam() {
            return new Promise( ( resolve, reject ) => {
                const webcamElement = document.getElementById( "webcam" );
                const navigatorAny = navigator;
                navigator.getUserMedia = navigator.getUserMedia ||
                navigatorAny.webkitGetUserMedia ||
                          navigatorAny.mozGetUserMedia ||
                          navigatorAny.msGetUserMedia;
                if( navigator.getUserMedia ) {
                    navigator.getUserMedia( { video: true },
                        stream => {
                            webcamElement.srcObject = stream;
                            webcamElement.addEventListener( "loadeddata",
                                  resolve, false );
                        },
                    error => reject());
                }
                else {
                    reject();
                }
            });
        }

        async function getWebcamImage() {
            const img = ( await webcam.capture() ).toFloat();
            const normalized = img.div( 127 ).sub( 1 );
            return normalized;
        }

        async function captureSample( category ) {
            trainingData.push( {
                image: await getWebcamImage(),
                category: category
            });
            setText( "Захвачено:/Captured: " + labels[ category ] );
        }

        let webcam = null;
```

```
        (async () => {
            // Загрузите модель
            // Load the model
            model = await tf.loadLayersModel( mobilenet );
            model = createTransferModel( model );
            await setupWebcam();
            webcam = await tf.data.webcam(
                      document.getElementById( "webcam" ) );
            // Установка прогноза: каждые 200 мс
            // Setup prediction every 200 ms
            setInterval( predictImage, 200 );
        })();
        </script>
    </body>
</html>
```

Обнаружение жестов рук

Код начальной точки создан и готов обнаружить четыре различных категории жестов: **(Нет/None)**, **(Кулак/Rock)**, **(Ладонь/Paper)**, **(Ножницы/Scissors)**. Вы можете попробовать его, используя вашу веб-камеру, нажимая на каждую из кнопок категории, чтобы захватить некоторые фотографии (**5-6** - хорошая выборка, чтобы начать), в то время как вы удерживаете каждый жест руки, и затем нажимаете кнопку **обучения(тренировки; train)**, чтобы **передать обученность в нейронную сеть(transfer learning to the neural network)**. После этого вы можете улучшить модель, делая больше фотографий и снова нажимая на кнопку **обучения(тренировки; train)**.

Дополнительные жесты рук и жесты языка знаков

Как вероятно, вы можете вообразить, добавление большего числа категорий становится более трудным для обучения AI-системы и занимает больше времени. Однако результаты - забавны, и AI-система работает довольно хорошо даже из просто нескольких фотографий для каждой категории. Давайте попытаемся добавить некоторые жесты американского языка знаков(American Sign Language; ASL).

Чтобы добавить больше, вы можете включить больше кнопок во входной список, обновить число, передаваемое в captureSample(), и изменить массив меток labels соответственно.

Вы можете добавить, какие-нибудь желаемые вами знаки. Я добавил четыре знака, которые были частью emoji-набора:

- (Буква d/Letter D)
- (Толстый палец вверх/Thumb Up)"
- (Вулкан/Vulcan)
- (Я люблю Вас/ILY - I Love You)

Технические примечания

- Если AI-система, кажется, не распознает хорошо ваши жесты рук, то попытайтесь делать больше фотографий и затем многократно обучите модель.
- Хотя модель обучена различным жестам рук, но имейте в виду, что она видит полное изображение; она не обязательно знает, что рука отдельно отличает категории. Может быть трудно точно распознать различные жесты рук без многочисленных выборок(примеров) от различных рук.
- Иногда, модель учится дифференцировать между левой и правой руками, и иногда она не делает, что могло влиять на прогнозы после многократных раундов обучения.

Финишная черта

Вот полный код:

JavaScript

```
<html>
    <head>
        <meta charset="UTF-8">
```

```
        <title>Interpreting Hand Gestures and Sign Language in the Webcam
with AI using TensorFlow.js</title>
        <script
src="https://cdn.jsdelivr.net/npm/@tensorflow/tfjs@2.0.0/dist/tf.min.js"><
/script>
        <script
src="https://cdn.jsdelivr.net/npm/@tensorflow/tfjs-data@2.0.0/dist/tf-data
.min.js"></script>
        <style>
            img, video {
                object-fit: cover;
            }
        </style>
    </head>
    <body>
        <video autoplay playsinline muted id="webcam"
                 width="224" height="224"></video>
        <div id="buttons">
            <button onclick="captureSample(0)">(Нет/None)</button>
            <button onclick="captureSample(1)"> (Кулак/Rock)</button>
            <button onclick="captureSample(2)"> (Ладонь/Paper)</button>
          <button onclick="captureSample(3)"> (Ножницы/Scissors)</button>
          <button onclick="captureSample(4)"> (Буква d/Letter D)</button>
         <button onclick="captureSample(5)"> (Толстый палец вверх/
                                              Thumb Up)</button>
            <button onclick="captureSample(6)"> (Вулкан/Vulcan)</button>
            <button onclick="captureSample(7)"> (Я люблю Вас/
                                              ILY - I Love You)</button>
        <button onclick="trainModel()">Обучить(тренировать)/
                                              Train</button>
</div>
<h1 id="status">Loading...</h1>
<script>
let trainingData = [];

const labels = [
    "None",
    " (Кулак/Rock)",
    " (Ладонь/Paper)",
    " (Ножницы/Scissors)",
    " (Буква d/Letter D)",
    " (Толстый палец вверх/Thumb Up)",
    " (Вулкан/Vulcan)",
    " (Я люблю Вас/ILY - I Love You)"
];

function setText( text ) {
    document.getElementById( "status" ).innerText = text;
}

async function predictImage() {
    if( !hasTrained ) { return; } // Пропустите прогноз,
     // пока не обучена
     // Skip prediction until trained
```

```
    const img = await getWebcamImage();
    let result = tf.tidy( () => {
        const input = img.reshape( [ 1, 224, 224, 3 ] );
        return model.predict( input );
    });
    img.dispose();
    let prediction = await result.data();
    result.dispose();
    // Получите индекс самого высокого значения в прогнозе
     // Get the index of the highest value in the prediction
    let id = prediction.indexOf( Math.max( ...prediction ) );
    setText( labels[ id ] );
}

function createTransferModel( model ) {
    // Создайте усеченную базовую модель(удалите "верхние" слои,
     // классификация + слои узкого места)
     // Create the truncated base model (remove the "top" layers,
     // classification + bottleneck layers)
    const bottleneck = model.getLayer( "dropout" );
     // Финальный уровень, перед слоем conv_pred,
     // предварительно обученным слоем классификации
     // This is the final layer before
     // the conv_pred pre-trained classification layer
    const baseModel = tf.model({
        inputs: model.inputs,
        outputs: bottleneck.output
});
// Заморозьте сверточную базу
 // Freeze the convolutional base
for( const layer of baseModel.layers ) {
    layer.trainable = false;
}
// Добавьте «голову» классификации
 // Add a classification head
const newHead = tf.sequential();
newHead.add( tf.layers.flatten( {
     inputShape: baseModel.outputs[ 0 ].shape.slice( 1 )
                                        } ) );
newHead.add( tf.layers.dense( {
     units: 100, activation: 'relu' } ) );
newHead.add( tf.layers.dense( {
     units: 100, activation: 'relu' } ) );
newHead.add( tf.layers.dense( {
     units: 10, activation: 'relu' } ) );
newHead.add( tf.layers.dense( {
    units: labels.length,
    kernelInitializer: 'varianceScaling',
    useBias: false,
    activation: 'softmax'
} ) );
// Постройте новую модель
 // Build the new model
const newOutput = newHead.apply( baseModel.outputs[ 0 ] );
```

```
    const newModel = tf.model( {
         inputs: baseModel.inputs, outputs: newOutput } );
    return newModel;
}

async function trainModel() {
    hasTrained = false;
    setText( "Обучение.../ Training..." );

    // Установите данные обучения(тренировки)
     // Setup training data
    const imageSamples = [];
    const targetSamples = [];
    trainingData.forEach( sample => {
        imageSamples.push( sample.image );
        let cat = [];
        for( let c = 0; c < labels.length; c++ ) {
            cat.push( c === sample.category ? 1 : 0 );
        }
        targetSamples.push( tf.tensor1d( cat ) );
    });
    const xs = tf.stack( imageSamples );
    const ys = tf.stack( targetSamples );

    // Обучите модель на новых выборках изображений
    // Train the model on new image samples
    model.compile( { loss: "meanSquaredError",
                     optimizer: "adam", metrics: [ "acc" ] } );

        await model.fit( xs, ys, {
            epochs: 30,
            shuffle: true,
            callbacks: {
                onEpochEnd: ( epoch, logs ) => {
                    console.log( "Эпоха #/Epoch #", epoch, logs );
                }
            }
        });
        hasTrained = true;
    }

    // Модель Mobilenet v1 0.25 для изображений размером 224x224
     // Mobilenet v1 0.25 224x224 model
    const mobilenet =
"https://storage.googleapis.com/tfjs-models/tfjs/mobilenet_v1_0.25_224/mod
el.json";

    let model = null;
    let hasTrained = false;

    async function setupWebcam() {
        return new Promise( ( resolve, reject ) => {
            const webcamElement = document.getElementById( "webcam" );
            const navigatorAny = navigator;
```

```
            navigator.getUserMedia = navigator.getUserMedia ||
            navigatorAny.webkitGetUserMedia ||
                      navigatorAny.mozGetUserMedia ||
                      navigatorAny.msGetUserMedia;
            if( navigator.getUserMedia ) {
                navigator.getUserMedia( { video: true },
                    stream => {
                        webcamElement.srcObject = stream;
                        webcamElement.addEventListener( "loadeddata",
                              resolve, false );
                    },
                error => reject());
            }
            else {
                reject();
            }
        });
    }

    async function getWebcamImage() {
        const img = ( await webcam.capture() ).toFloat();
        const normalized = img.div( 127 ).sub( 1 );
        return normalized;
    }

    async function captureSample( category ) {
        trainingData.push( {
            image: await getWebcamImage(),
            category: category
        });
        setText( "Захвачено: /Captured: " + labels[ category ] );
    }

    let webcam = null;

    (async () => {
        // Загрузите модель
        // Load the model
        model = await tf.loadLayersModel( mobilenet );
        model = createTransferModel( model );
        await setupWebcam();
        webcam = await tf.data.webcam(
                  document.getElementById( "webcam" ) );
         // Установка прогноза: каждые 200 мс
         // Setup prediction every 200 ms
        setInterval( predictImage, 200 );
    })();
    </script>
  </body>
</html>
```

Что далее?

Этот проект показал вам, как начать обучение вашей собственной **AI**-системы машинного зрения, чтобы она распознавала потенциально неограниченные жесты, объекты, виды животных, или даже типы продуктов. Остальное дело за вами; будущее **глубокого обучения(deep learning)** и **AI**-систем могло бы начаться прямо в вашем браузере.

Я надеюсь, что вы наслаждались следуя вместе с этими примерами. И поскольку вы экспериментируете с большим количеством идей, не забывайте развлекаться!

Эта статья - статья из серии статей **Обнаружение касания лица с помощью Tensorflow.js(Face Touch Detection with Tensorflow.js)**.