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, 08.11.2023

Статья 4 «Обнаружение касания лица в фреймворке TensorFlow.js, часть 1: Использование данных веб-камеры в реальном времени с глубоким обучением»

Статья **4 Обнаружение касания лица в фреймворке TensorFlow.js, часть 1: Использование данных веб-камеры в реальном времени с глубоким обучением(Face Touch Detection with TensorFlow.js Part 1: Using Real-Time Webcam Data with Deep Learning);** https://www.codeproject.com/Articles/5272773/Face-Touch-Detection-with-TensorFlow-js-Part-1-Usi) является статьей из серии статей **Обнаружение касания лица с помощью Tensorflow.js(Face Touch Detection with Tensorflow.js)**.

13 июля 2020

В этой статье мы хотим использовать все предыдущие знания и навыки с машинным зрением в **TensorFlow.js** и попытаемся создать версию этого приложения сами.

Мы хотим к нашему коду, модели распознавания объектов, добавить возможности веб-камеры и хотим посмотреть на использование **программного API-интерфейса для разработки кода по использованию HTML5-веб-камеры(HTML5 Webcam API)** с фреймворком(инфраструктурой) машинного обучения **TensorFlow.js** для **обнаружения касаний лица(detecting face touches)**.

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**!

Одной из лучших возможностей современных веб-браузеров, поддерживающих **HTML5**, является легкий доступ к **разнообразию программных API-интерфейсов для разработки кода по использованию разных компонент(variety of APIs)**, таких как веб-камера и аудио. И с недавними проблемами **COVID-19**, влияющими на здравоохранение, группа очень творческих разработчиков использовали эту ситуацию, чтобы создать приложение, названное **не касайся лица(**donottouchyourface.com**)**, которое помогает людям снижать риск заболевания, уча их прекращению касаться их лиц. В этой статье мы хотим использовать все предыдущие знания и навыки с машинным зрением в **TensorFlow.js** и попытаемся создать версию этого приложения сами.

Начальная точка

Мы хотим к нашему коду, модели распознавания объектов, добавить возможности веб-камеры и затем **захватить кадры(capture frames)** в режиме реального времени для обучения и предсказания **действий по касанию лица(face touch actions)**. Этот код покажется вам знакомым, если вы следовали за повествованием в предыдущей статье(previous article). Вот то, что сделает получающийся код:

- Импортируйте **TensorFlow.js** и модуль **tf-data.js** из **TensorFlow**
- Определите надписи категории **Касание(Touch)** по сравнению с категорией **Нет касания(Not-Touch)**

- Добавьте видео элемент для веб-камеры
- Выполните прогноз на модели каждые `200 мс` после того, как она будет обучена впервые
- Покажите результат прогноза
- Загрузите предварительно обученную модель `MobileNet` и подготовьтесь к передаче обученности
- Обучайте и классифицируйте нестандартные(пользовательские) объекты в изображениях
- Пропустите расположение изображения и целевые выборки в учебном процессе, чтобы сохранить их для многократных выполнений обучения

В этом проекте, этот код будет нашей начальной точкой, прежде чем мы добавим функциональность веб-камеры в реальном времени:

JavaScript

```
<html>
    <head>
        <meta charset="UTF-8">
        <title>Face Touch Detection with TensorFlow.js Part 1: Using
Real-Time Webcam Data with Deep Learning</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>
        <h1 id="status">Загрузка.../ Loading...</h1>
        <script>
        let touch = [];
        let notouch = [];
```

```
const labels = [
    "Касание!/Touch!",
    "Нет касания/No Touch"
];

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" ); // Это -
     // 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: 2,
        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 = [];
    for( let i = 0; i < touch.length; i++ ) {
        let result = touch[ i ];
        imageSamples.push( result );
        targetSamples.push( tf.tensor1d( [ 1, 0 ] ) );
    }
    for( let i = 0; i < notouch.length; i++ ) {
        let result = notouch[ i ];
        imageSamples.push( result );
        targetSamples.push( tf.tensor1d( [ 0, 1 ] ) );
    }
    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 () => {
            // Загрузите модель
             // Load the model
            model = await tf.loadLayersModel( mobilenet );
            model = createTransferModel( model );
            // Your Code Goes Here
            // Setup prediction every 200 ms
            setInterval( predictImage, 200 );
        })();
        </script>
    </body>
</html>
```

Использование программного API-интерфейса для разработки кода по использованию HTML5-веб-камеры(HTML5 Webcam API) с фреймворком машинного обучения TensorFlow.js

Запуск веб-камеры довольно прост в JavaScript-коде, как только у вас есть фрагмент кода для этого. Вот вспомогательная функция настройки веб-камеры, для запуска камеры и запроса от пользователя доступа к камере:

JavaScript

```
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();
        }
    });
}
```

Теперь в вашем коде, после того, как была создана модель, вызовите вспомогательную функцию настройки веб-камеры `setupWebcam()` и камера начнет работать на веб-странице. Давайте инициализируем глобальную веб-камеру, используя библиотеку `tf`-данных `tf-data`, и таким образом мы можем использовать ее функцию-помощник и легко создавать `тензоры(tensors)` из `кадра(фрейма; frame)` веб-камеры.

JavaScript

```
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 для выполнения прогноза каждые 200 мс
    // Setup prediction every 200 ms
    setInterval( predictImage, 200 );
})();
```

`Захват кадра(capturing a frame)` с использованием функции-помощника веб-камеры `TensorFlow` и нормализация пикселей могут быть сделаны в функции, как в этом коде:

JavaScript

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

Тогда давайте использовать эту функцию, чтобы **захватить изображения(capture images)** для данных тренировки в другой функции:

JavaScript

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

Наконец, давайте к странице, ниже элемента видео веб-камеры, добавим три кнопки и активируем демонстрационный **захват изображения(capture images)** и **обучение модели(model training)**:

HTML

```
<video autoplay playsinline muted id="webcam" width="224"
height="224"></video>
<button onclick="captureSample(0)">Касание/Touch</button>
<button onclick="captureSample(1)">Нет касания/No Touch</button>
<button onclick="trainModel()">Обучить(тренировать)/Train</button>
<h1 id="status">Загрузка/Loading...</h1>
```

Обнаружение касаний лица

С добавленной функциональностью веб-камеры мы готовы попробовать наше **обнаружение касания лица(face touch detection)**.

Откройте веб-страницу и в то время как вы находитесь в виде камеры, используйте кнопки **Касание/Touch** и **Нет касания/No Touch**, чтобы **захватить (capturing)** различные примеры-изображения. Оказалось, что **захват(capturing)** приблизительно **10-15** выборок(примеров) для каждого режима, **Касание/Touch** и **Нет касания/No Touch**, было достаточно хорошо, чтобы начать обнаруживать вполне хорошо.

Технические примечания

- Поскольку мы - вероятное обучаем(тренируем) нашу модель на только небольшой выборке примеров изображений, не делая фотографии большого количества различных людей, то точность обученной **AI**-модели будет низкой, когда ваше приложение попробуют другие люди
- **AI**-модель не может очень хорошо отличить глубину и может вести себя как функциональность **Обнаружение заграждения лица(Face Obstructed Detection)**, вместо функциональности **Обнаружение касания лица(Face Touch Detection)**
- Мы, возможно, назвали бы кнопки и соответствующие категории **Касание против Нет касания(Touch vs. No Touch)**, но модель не распознает импликации; она могла бы обучаться на любых двух изменениях **захваченных(captured)** фотографий, как **Собака против Кот(Dog vs Cat)** или **Круг против Прямоугольник(Circle vs Rectangle)**

Финишная черта

Вот полный код:

JavaScript

```
<html>
    <head>
```

```
        <meta charset="UTF-8">
        <title>Face Touch Detection with TensorFlow.js Part 1: Using
Real-Time Webcam Data with Deep Learning</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>
        <button onclick="captureSample(0)">Касание/Touch</button>
        <button onclick="captureSample(1)">Нет касания/No Touch</button>
        <button onclick="trainModel()">Обучить(тренировать)/Train</button>
        <h1 id="status">Загрузка.../Loading...</h1>
        <script>
        let touch = [];
        let notouch = [];

        const labels = [
            "Касание/Touch!",
            "Нет касания/No Touch"
        ];

        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( "conv_pw_13_relu" );
 // Intercepting at the convolution layer
 // might give better results
const bottleneck = model.getLayer( "dropout" ); // Это -
 // 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: 2,
        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 = [];
```

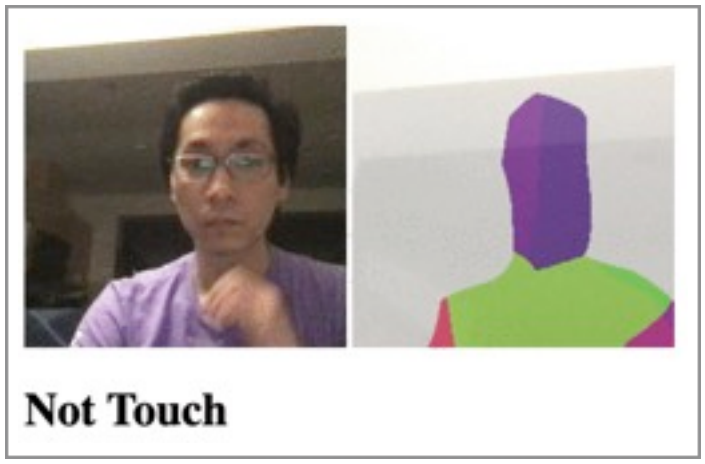
```
const targetSamples = [];
for( let i = 0; i < touch.length; i++ ) {
    let result = touch[ i ];
    imageSamples.push( result );
    targetSamples.push( tf.tensor1d( [ 1, 0 ] ) );
}
for( let i = 0; i < notouch.length; i++ ) {
    let result = notouch[ i ];
    imageSamples.push( result );
    targetSamples.push( tf.tensor1d( [ 0, 1 ] ) );
}
const xs = tf.stack( imageSamples );
const ys = tf.stack( targetSamples );

// Обучите модель на новых выборках примерах 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/model.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 ) {
        if( category === 0 ) {
            touch.push( await getWebcamImage() );
            setText( "Захвачены:/Captured: " +
                  labels[ category ] + " x" + touch.length );
        }
        else {
            notouch.push( await getWebcamImage() );
            setText( "Захвачены:/Captured: " +
                  labels[ category ] + " x" + notouch.length );
        }
    }

    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" ) );
        // Прогноз установки каждые 200 мс
         // Setup prediction every 200 ms
        setInterval( predictImage, 200 );
    })();
    </script>
  </body>
</html>
```

Что далее? Можем ли мы без обучения обнаружить касания лица?

На сей раз мы изучили, как использовать функциональность веб-камеры браузера, чтобы полностью обучить(тренировать) и распознать кадры из видео в реальном времени. Но было бы более приятнее, если бы, чтобы начать использовать это приложение, пользователь не должен был даже фактически касаться своего лица?

В серии статей, смотрите следующую статью **Обнаружение касания лица в фреймворке TensorFlow.js, часть 2: Использование BodyPix-модели(Face Touch Detection with TensorFlow.js Part 1: Using BodyPix)**, где мы будем использовать **предварительно обученную BodyPix-модель для обнаружения(pre-trained BodyPix model for detection)**.



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