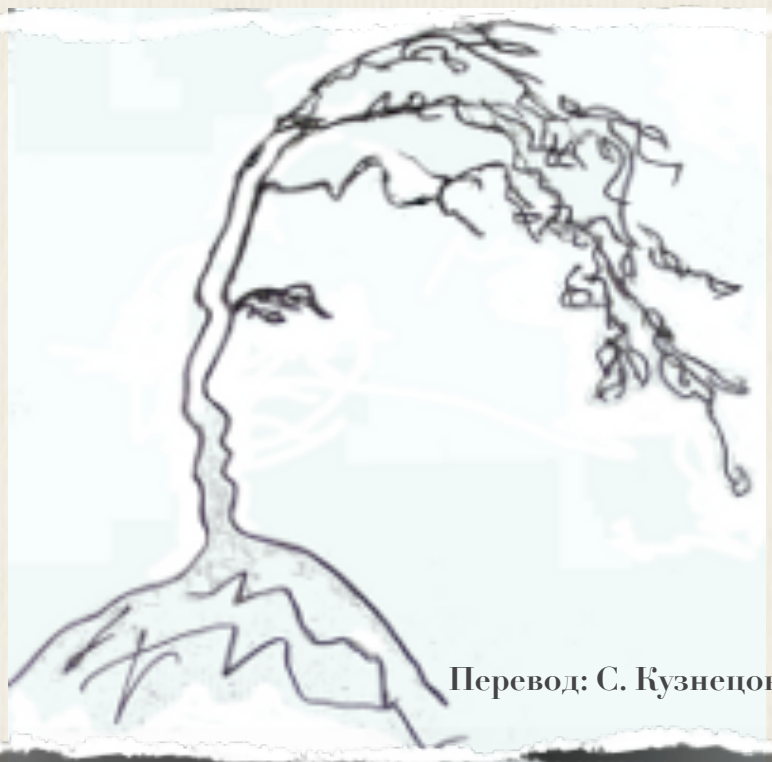
РАФАЭЛЬ МУН(RAPHAEL MUN)

Серия статей «Фильтры искусственного интеллекта лица в браузере»

УЧЕБНЫЕ РУКОВОДСТВА



Перевод: С. Кузнецов, 2024 г.

Articles: AI Face Filters in the Browser

Raphael Mun

2021

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

Серия статей «Фильтры искусственного интеллекта лица в браузере»

Рафаэль Мун

2021

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

Перевод: С. Кузнецов, 11.02.2024

Статья 2 «Обнаружение эмоций на лице в браузере с помощью глубокого обучения с использованием библиотеки TensorFlow.js»

Статья 2 Обнаружение эмоций на лице в браузере с помощью глубокого обучения с использованием библиотеки TensorFlow.js(Detecting Facial Emotions in the Browser with Deep Learning Using TensorFlow.js); https://www.codeproject.com/Articles/5293492/Detecting-Facial-Emotions-in-the-Browser-with-Deep) является статьей из серии статей Фильтры искусственного интеллекта лица в браузере(AI Face Filters in the Browser).

3 февраля 2021

В этой статье мы будем использовать ключевые признаки лица(ключевые точки на лице; key facial landmarks), чтобы вывести больше информации о лице из изображений.

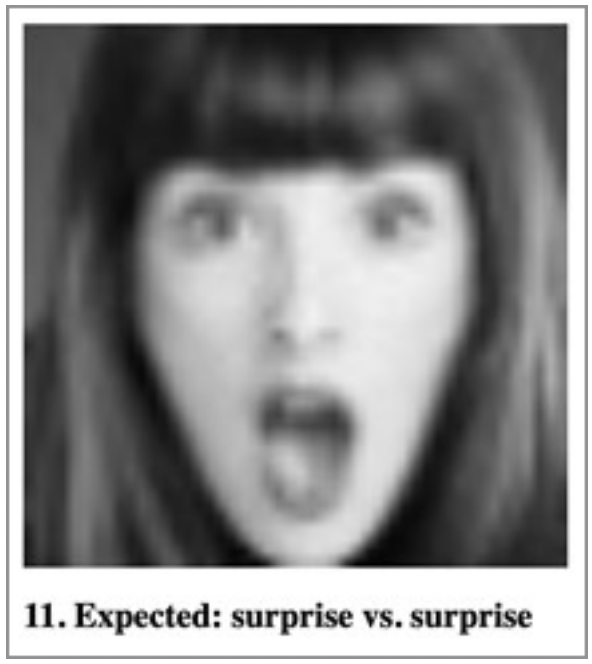
Здесь мы будем использовать глубокое обучение(Deep Learning) на отслеженных лицах из FER-набора данных об эмоциях на лицах(FER+dataset) и попытаемся точно предсказать эмоцию человека(accurately predict a person's emotion) на основе точек на лице(facial points) в браузере с библиотекой TensorFlow.js.

- **Загрузка кода и файлов - 565.6 KB**

Введение

Приложения, подобные приложению Snapchat, предлагают удивительное разнообразие фильтров лиц и линз, которые позволяют Вам накладывать интересные эффекты на фотографии и видео. Если когда-либо вы «приделывали» себе виртуальные уши собаки или маскарадную шляпу, то знаете, это может быть забавно!

Задавались ли вы вопросом, как создать эти виды фильтров с нуля? Ну, теперь есть шанс научиться делать все в веб-браузере! В этой серии статей мы собираемся показать, как в браузере создать фильтры в стиле **Snapchat**, используя отслеживание лица и библиотеку **Tensorflow.js**.



11. Expected: surprise vs. surprise

(надпись под снимком экрана: **11.Ожидаемо: удивление по сравнению с удивлением(11. Expected: surprise vs. surprise)**)

Вы можете загрузить демонстрационный пример этого проекта. Возможно, для обеспечения производительности, вы будете должны в своем веб-браузере включить поддержку **Web**-графики **WebGL**. Также можно загрузить the код и файлы для этой серии статей.

Предполагается, что вы знакомы с языками **JavaScript** и **HTML** и имеете, по крайней мере, базовое понимание нейронных сетей. Если вы плохо знакомы с фреймворком **TensorFlow.js**, то рекомендуем прочитать статью «Начало работы с глубоким изучением в браузере с использованием фреймворка TensorFlow.js»**(«Getting Started With Deep Learning in Your B r o w s e r U s i n g T e n s o r F l o w . j s » ; https://www.codeproject.com/Articles/5272760/Getting-Started-With-Deep-Lea rning-in-Your-Browser)**, которая является статьей из серии статей **Обнаружение касания лица с помощью Tensorflow.js(Face Touch Detection with Tensorflow.js)**

Если хотели бы увидеть больше того, что возможно в веб-браузере с помощью фреймворка **TensorFlow.js**, прочтите статьи из серии по

искусственному интеллекту(AI): «Собаки и пицца: машинное зрение в браузере с использованием TensorFlow.js»(«Dogs and Pizza: Computer Vision in the Browser With Tensorflow.js»; https://www.codeproject.com/Articles/5272771/Dogs-and-Pizza-Computer-Vision-in-the-Browser-With) и «Роботы чатов с помощью фреймворка TensorFlow.js»(Chatbots using TensorFlow.js.)

В предыдущей статье мы изучили, как использовать модели искусственного интеллекта(AI models), чтобы обнаружить форму лиц(shape of faces). В этой статье мы будем использовать ключевые признаки лица(ключевые точки на лице; key facial landmarks), чтобы вывести больше информации о лице из изображений.

Соединяя наш код отслеживание лица(face tracking code) с FER-набором данных выражения лица(FER facial emotion dataset; Facial Expression Recognition; Распознавание выражения лица) мы будем обучать(тренировать) модель второй нейронной сети для предсказания эмоции человека на основе нескольких 3D-ключевых точек(3D key points).

Установка и настройка FER2013-набора данных выражения лица

Мы используем код отслеживания лица, из предыдущей статьи Отслеживание лица в реальном времени в браузере с помощью библиотеки TensorFlow.js, для создания двух веб-страниц. Одна страница будет использоваться для обучения(тренировки) AI-модели с отслеженными точками лица(tracked facial points) на FER-наборе данных эмоций на лице(FER facial emotion dataset), и другая веб-страница загрузит и выполнит обученную(натренированную) AI-модели на тестовом наборе данных(test dataset).

Давайте изменим финальный код из предыдущего проекта отслеживания лица, чтобы на FER2013-наборе данных выражения лица обучать(тренировать) AI-модель нейронной сети и выполнить ее с тестовыми данными на лице. FER2013-набор данных выражения лица(FER2013 dataset; Facial Expression Recognition; Распознавание выражения лица) состоит из более чем 28 тысяч

маркированных изображений лиц; этот набор доступен на Kaggle. Мы загрузили эту эту версию, в которой есть набор данных, уже преобразованный в файлы изображений и помещенный его в каталог(папку) **web/fer2013**. Затем мы обновили код **NodeJS**-сервера в файле **index.js**, чтобы он возвратил список ссылок на изображения по **URL**-адресу http://localhost:8080/data/, так, чтобы вы могли получить полный **JSON**-объект, если выполняете сервер локально.

Чтобы сделать процедуру немного проще, мы для вас сохранили этот **JSON**-объект в файле **web/fer2013.js**, чтобы использовать его напрямую без необходимости выполнить сервер локально. Вы можете включить его с другими файлами скриптов в верхней части страницы:

JavaScript

```
<script src="web/fer2013.js"></script>
```

Мы собираемся работать с изображениями, а не с видео из веб-камеры (не волнуйтесь, мы вернемся к видео в следующей статье [11]!), и поэтому мы должны элемент **<video>** заменить на элемент **<img>** и можем переименовать его **ID**-идентификатор на **«image»**. Мы можем также удалить функцию установки и настройки веб-камеры **setupWebcam**, потому она не нужна нам для этого проекта.

HTML

```
<img id="image" style="
    visibility: hidden;
    width: auto;
    height: auto;
    "/>
```

Затем, давайте добавим служебную функцию для установки изображения в элементе **setImage** и другую служебную функцию для перемешивания массива данных **shuffleArray**. Поскольку исходные изображения имеют размер всего **48x48** пикселей, то давайте определим больший выходной

размер в 500 пикселей, чтобы получить более детализированное отслеживание лиц и быть в состоянии видеть результат в более крупном HTML-элементе canvas(холст), и давайте обновим служебные функции черчения линий и многоугольников, чтобы масштабировать в соответствии с выходом.

JavaScript

```
async function setImage( url ) {
    return new Promise( res => {
        let image = document.getElementById( "image" );
        image.src = url;
        image.onload = () => {
            res();
        };
    });
}

function shuffleArray( array ) {
    for( let i = array.length - 1; i > 0; i-- ) {
        const j = Math.floor( Math.random() * ( i + 1 ) );
        [ array[ i ], array[ j ] ] = [ array[ j ], array[ i ] ];
    }
}

const OUTPUT_SIZE = 500;
```

Нам потребуются некоторые глобальные переменные, а именно для списка категорий эмоций(list of emotion categories), списка агрегированных массивов FER-данных(aggregated array list of FER data) и индекса массива(index for the array):

JavaScript

```
//            [   "сердитый",  "отвращение", "страх", "счастливый",
//              "нейтральный", "печальный",  "удивление"];
const emotions = [ "angry", "disgust", "fear", "happy", "neutral", "sad",
"surprise" ];
let ferData = [];
let setIndex = 0;
```

Внутри блока асинхронности **async** мы можем подготовить и перемешать **FER**-данные и изменить размеры **HTML**-элемента **canvas(холст)** на **500x500** пикселей:

JavaScript

```
const minSamples = Math.min( ...Object.keys( fer2013 ).map( em => fer2013[
em ].length ) );
Object.keys( fer2013 ).forEach( em => {
    shuffleArray( fer2013[ em ] );
    for( let i = 0; i < minSamples; i++ ) {
        ferData.push({
            emotion: em,
            file: fer2013[ em ][ i ]
        });
    }
});
shuffleArray( ferData );

let canvas = document.getElementById( "output" );
canvas.width = OUTPUT_SIZE;
canvas.height = OUTPUT_SIZE;
```

Мы должны в шаблоне кода сделать одно обновление на первой странице перед обучением(тренировкой) **AI**-модели и на второй странице перед применением обученной модели. Также мы должны обновить функцию отслеживания лица **trackFace**, чтобы работать с элементом изображения вместо элемента видео. Кроме этого масштабируем ограничивающий прямоугольник и выходные данные меша(каркаса) лица, чтобы соответствовать размеру **HTML**-элемента **canvas(холст)**. В конце функции мы прирастим индекс **setIndex** для перехода к следующему изображению.

JavaScript

```
async function trackFace() {
    // Установить следующее изображение для обучения(тренировки)
    // Set to the next training image
    await setImage( ferData[ setIndex ].file );
    const image = document.getElementById( "image" );
    const faces = await model.estimateFaces( {
        input: image,
```

```
        returnTensors: false,
        flipHorizontal: false,
    });
    output.drawImage(
        image,
        0, 0, image.width, image.height,
        0, 0, OUTPUT_SIZE, OUTPUT_SIZE
    );

    const scale = OUTPUT_SIZE / image.width;

    faces.forEach( face => {
        // Рисуем ограничивающий прямоугольник вокруг лица
        // Draw the bounding box
        const x1 = face.boundingBox.topLeft[ 0 ];
        const y1 = face.boundingBox.topLeft[ 1 ];
        const x2 = face.boundingBox.bottomRight[ 0 ];
        const y2 = face.boundingBox.bottomRight[ 1 ];
        const bWidth = x2 - x1;
        const bHeight = y2 - y1;
        drawLine( output, x1, y1, x2, y1, scale );
        drawLine( output, x2, y1, x2, y2, scale );
        drawLine( output, x1, y2, x2, y2, scale );
        drawLine( output, x1, y1, x1, y2, scale );

        // Рисуем мешь лица
        // Draw the face mesh
        const keypoints = face.scaledMesh;
        for( let i = 0; i < FaceTriangles.length / 3; i++ ) {
            let pointA = keypoints[ FaceTriangles[ i * 3 ] ];
            let pointB = keypoints[ FaceTriangles[ i * 3 + 1 ] ];
            let pointC = keypoints[ FaceTriangles[ i * 3 + 2 ] ];
            drawTriangle( output, pointA[ 0 ], pointA[ 1 ], pointB[ 0 ],
                          pointB[ 1 ], pointC[ 0 ], pointC[ 1 ], scale );
        }
    });

    setText( `${setIndex + 1}. Face Tracking Confidence:
${face.faceInViewConfidence.toFixed( 3 )} - ${ferData[ setIndex
].emotion}` );
    setIndex++;
    requestAnimationFrame( trackFace );
}
```

Теперь наш измененный шаблон кода готов. Сделайте две копии этого кода для того, чтобы мы могли одну копию установить в странице для **глубокого обучения(Deep Learning)** модели и другую копию установить в странице для для **тестирования(testing)**.

Часть1: Глубокое обучение модели эмоциям на лице

В этом первом файле веб-страницы мы собираемся установить данные для обучения(тренировки), создать модель нейронной сети, и затем обучать ее и сохранить веса в файле. В поставленный архив кода включена предварительно обученная модель(смотрите каталог **web/model**) и поэтому при желании вы можете пропустить чтение **Части 1** об обучении и перейти к чтению **Части 2**. о выполнении обученной модели.

Добавьте глобальную переменную для хранения данных обучения(тренировки) и служебную функцию преобразования меток эмоций в единичный вектор и таким образом, мы можем использовать его для данных обучения(тренировки):

JavaScript

```
let trainingData = [];

function emotionToArray( emotion ) {
    let array = [];
    for( let i = 0; i < emotions.length; i++ ) {
        array.push( emotion === emotions[ i ] ? 1 : 0 );
    }
    return array;
}
```

В функции отслеживания лица **trackFace** мы возьмем разные **ключевые признаки лица(key facial features)**, масштабируем их относительно **размера ограничивающего прямоугольника(size of the bounding box)** и добавим их в набор данных обучения(тренировки), если значение уверенности отслеживания лица достаточно велико. Чтобы упростить данные, мы закомментировали некоторые **дополнительные признаки лица(additional facial features)**, но можно добавить их обратно(раскомментировать), если захотите поэкспериментировать. Если вы это делаете так, то не забудьте соответствовать эти те же признаки при выполнении модели.

JavaScript

```
// Добавьте только нос, щеки, глаза, брови & рот
// Add just the nose, cheeks, eyes, eyebrows & mouth
const features = [
    "noseTip",
    "leftCheek",
    "rightCheek",
    "leftEyeLower1", "leftEyeUpper1",
    "rightEyeLower1", "rightEyeUpper1",
    "leftEyebrowLower", //"leftEyebrowUpper",
    "rightEyebrowLower", //"rightEyebrowUpper",
    "lipsLowerInner", //"lipsLowerOuter",
    "lipsUpperInner", //"lipsUpperOuter",
];
let points = [];
features.forEach( feature => {
    face.annotations[ feature ].forEach( x => {
        points.push( ( x[ 0 ] - x1 ) / bWidth );
        points.push( ( x[ 1 ] - y1 ) / bHeight );
    });
});
// Только захватите лица, в которых уверены
// Only grab the faces that are confident
if( face.faceInViewConfidence > 0.9 ) {
    trainingData.push({
        input: points,
        output: ferData[ setIndex ].emotion,
    });
}
```

Как только мы скомпилировали достаточно данных обучения(тренировки), мы можем передать их в функцию обучения(тренировки) сети `trainNet`. В верхней части функции отслеживания лица `trackFace` давайте прервем цикл отслеживания лиц после получения `200` изображений и вызовем функцию обучения(тренировки) сети `trainNet`:

JavaScript

```
async function trackFace() {
     // Отслеживание для быстрого обучения только на 200 изображениях
     // Fast train on just 200 of the images
    if( setIndex >= 200 ) {
        setText( "Завершено!/Finished!" );
        trainNet();
        return;
```

```
    }
    ...
}
```

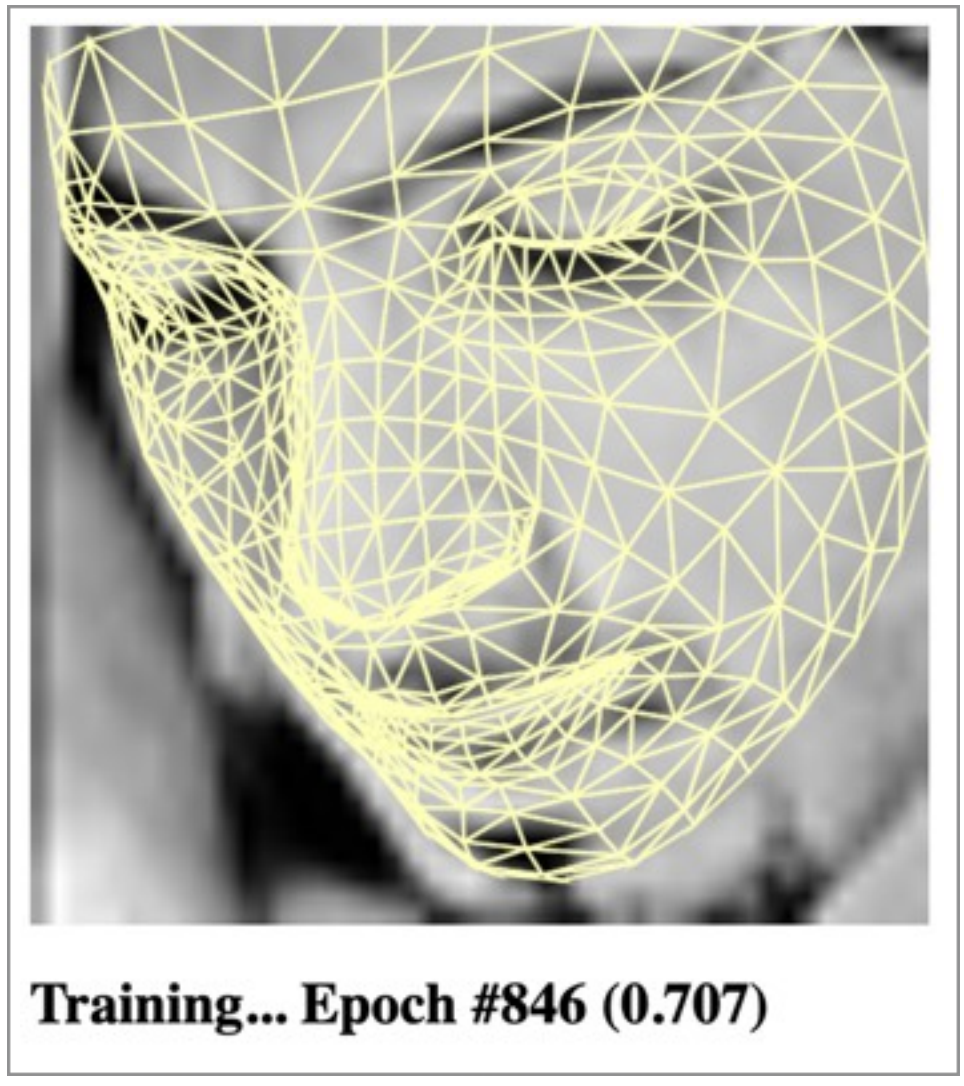
Наконец долгожданная часть: давайте создадим функцию обучения(тренировки) сети **trainNet** и давайте обучим нашу **AI**-модель!

Эта функция разделит данные обучения(тренировки) на входной массив **ключевых точек(key points)** и выходной массив **единичных векторов эмоций (emotion one-hot vectors)**, создаст по-категорийную **TensorFlow**-модель с несколькими скрытыми слоями, выполнит обучение(тренировку) за **1,000** итераций(эпох), и загрузит обученную(натренированную) модель. Число итераций(эпох) можно увеличить, если захотите обучать модель больше.

JavaScript

```
async function trainNet() {
    let inputs = trainingData.map( x => x.input );
    let outputs = trainingData.map( x => emotionToArray( x.output ) );

    // Определите нашу модель с несколькими скрытыми слоями
    // Define our model with several hidden layers
    const model = tf.sequential();
    model.add(tf.layers.dense( { units: 100, activation: "relu",
                                 inputShape: [ inputs[ 0 ].length ] } ) );
    model.add(tf.layers.dense( { units: 100, activation: "relu" } ) );
    model.add(tf.layers.dense( { units: 100, activation: "relu" } ) );
    model.add(tf.layers.dense( {
        units: emotions.length,
        kernelInitializer: 'varianceScaling',
        useBias: false,
        activation: "softmax"
    } ) );

    model.compile({
        optimizer: "adam",
        loss: "categoricalCrossentropy",
        metrics: "acc"
    });

    const xs = tf.stack( inputs.map( x => tf.tensor1d( x ) ) );
    const ys = tf.stack( outputs.map( x => tf.tensor1d( x ) ) );
    await model.fit( xs, ys, {
        epochs: 1000,
        shuffle: true,
        callbacks: {
            onEpochEnd: ( epoch, logs ) => {
```

```
                    setText( `Обучение(тренировка)/Training...
                              Итерация(эпоха)/Epoch #${epoch}
                              (${logs.acc.toFixed( 3 )})` );
                    console.log( "Итерация(эпоха)/Epoch #", epoch, logs );
                }
            }
        } );

        // Загрузите обученную(натренированную) модель
        // Download the trained model
        const saveResult = await model.save( "downloads://facemo" );
}
```



(надпись под снимком экрана: **Обучение(тренировка)... Итерация(эпоха) номер 846(0.707) (Training...Epoch #846(0.707)**)

И вот именно так! Эта веб-страница будет обучать(тренировать) нашу **AI**-модель **распознавать выражения лица(recognize facial expressions)** в различных категориях и в результате выдаст модель, для последующей ее загрузки и выполнения.

Часть 1: Финишная черта

Вот полный код для обучения(тренировки) модели на **FER**-наборе данных:

HTML

```
<html>
    <head>
        <title>Обучение модели - Обнаружение эмоций на лице в браузере с помощью глубокого обучения с использованием библиотеки TensorFlow.js</title>
        <script src="https://cdn.jsdelivr.net/npm/@tensorflow/tfjs@2.4.0/dist/tf.min.js"></script>
        <script src="https://cdn.jsdelivr.net/npm/@tensorflow-models/face-landmarks-detection@0.0.1/dist/face-landmarks-detection.js"></script>
        <script src="web/triangles.js"></script>
        <script src="web/fer2013.js"></script>
    </head>
    <body>
<canvas id="output"></canvas>
<img id="image" style="
    visibility: hidden;
    width: auto;
    height: auto;
    "/>
<h1 id="status">Загрузка/Loading...</h1>
<script>
function setText( text ) {
    document.getElementById( "status" ).innerText = text;
}

async function setImage( url ) {
    return new Promise( res => {
        let image = document.getElementById( "image" );
        image.src = url;
        image.onload = () => {
            res();
        };
    });
}

function shuffleArray( array ) {
    for( let i = array.length - 1; i > 0; i-- ) {
        const j = Math.floor( Math.random() * ( i + 1 ) );
        [ array[ i ], array[ j ] ] = [ array[ j ], array[ i ] ];
```

```
    }
}

function drawLine( ctx, x1, y1, x2, y2, scale = 1 ) {
    ctx.beginPath();
    ctx.moveTo( x1 * scale, y1 * scale );
    ctx.lineTo( x2 * scale, y2 * scale );
    ctx.stroke();
}

function drawTriangle( ctx, x1, y1, x2, y2, x3, y3, scale = 1 ) {
    ctx.beginPath();
    ctx.moveTo( x1 * scale, y1 * scale );
    ctx.lineTo( x2 * scale, y2 * scale );
    ctx.lineTo( x3 * scale, y3 * scale );
    ctx.lineTo( x1 * scale, y1 * scale );
    ctx.stroke();
}

const OUTPUT_SIZE = 500;

//            ["сердитый",  "отвращение", "страх", "счастливый",
//            "нейтральный", "печальный",  "удивление"];
// const emotions = [ "angry", "disgust", "fear",
//       "happy", "neutral", "sad", "surprise" ];
 const emotions = [ "сердитый(angry)", "отвращение(disgust)",
 "страх(fear)", "счастливый(happy)", "нейтральный(neutral)",
 "печальный(sad)", "удивление(surprise)" ];

let ferData = [];
let setIndex = 0;
let trainingData = [];

let output = null;
let model = null;

function emotionToArray( emotion ) {
    let array = [];
    for( let i = 0; i < emotions.length; i++ ) {
        array.push( emotion === emotions[ i ] ? 1 : 0 );
    }
    return array;
}

async function trainNet() {
    let inputs = trainingData.map( x => x.input );
    let outputs =
         trainingData.map( x => emotionToArray( x.output ) );

    // Определите модель с несколькими скрытыми слоями
     // Define our model with several hidden layers
```

```
const model = tf.sequential();
model.add(tf.layers.dense( { units: 100, activation: "relu",
                  inputShape: [ inputs[ 0 ].length ] } ) );
model.add(tf.layers.dense(
                  { units: 100, activation: "relu" } ) );
model.add(tf.layers.dense(
           { units: 100, activation: "relu" } ) );
model.add(tf.layers.dense( {
    units: emotions.length,
    kernelInitializer: 'varianceScaling',
    useBias: false,
    activation: "softmax"
} ) );

model.compile({
    optimizer: "adam",
    loss: "categoricalCrossentropy",
    metrics: "acc"
});

const xs = tf.stack( inputs.map( x => tf.tensor1d( x ) ) );
const ys = tf.stack( outputs.map( x => tf.tensor1d( x ) ) );
await model.fit( xs, ys, {
    epochs: 1000,
    shuffle: true,
    callbacks: {
                onEpochEnd: ( epoch, logs ) => {
                    setText( `Обучение(тренировка)/Training...
                                Итерация(эпоха)/Epoch #${epoch}
                                (${logs.acc.toFixed( 3 )})` );
            console.log( "Итерация(эпоха)/Epoch #", epoch, logs );
                }
            }
    } );

    // Загрузите обученную(натренированную) модель
     // Download the trained model
    const saveResult = await model.save( "downloads://facemo" );
}

async function trackFace() {
    // Отслеживание для быстрого обучения
    // только на 200 изображениях
    // Fast train on just 200 of the images
    if( setIndex >= 200 ) {//ferData.length ) {
        setText( "Завершено!/Finished!" );
        trainNet();
        return;
    }
    // Установить следующее изображение для обучения(тренировки)
    // Set to the next training image
```

```
await setImage( ferData[ setIndex ].file );
const image = document.getElementById( "image" );
const faces = await model.estimateFaces( {
    input: image,
    returnTensors: false,
    flipHorizontal: false,
});
output.drawImage(
    image,
    0, 0, image.width, image.height,
    0, 0, OUTPUT_SIZE, OUTPUT_SIZE
);

const scale = OUTPUT_SIZE / image.width;

faces.forEach( face => {
    // Рисуем ограничивающий прямоугольник вокруг лица
    // Draw the bounding box
    const x1 = face.boundingBox.topLeft[ 0 ];
    const y1 = face.boundingBox.topLeft[ 1 ];
    const x2 = face.boundingBox.bottomRight[ 0 ];
    const y2 = face.boundingBox.bottomRight[ 1 ];
    const bWidth = x2 - x1;
    const bHeight = y2 - y1;
    drawLine( output, x1, y1, x2, y1, scale );
    drawLine( output, x2, y1, x2, y2, scale );
    drawLine( output, x1, y2, x2, y2, scale );
    drawLine( output, x1, y1, x1, y2, scale );

    // Рисуем мешь лица
    // Draw the face mesh
    const keypoints = face.scaledMesh;
    for( let i = 0; i < FaceTriangles.length / 3; i++ ) {
        let pointA = keypoints[ FaceTriangles[ i * 3 ] ];
        let pointB = keypoints[ FaceTriangles[ i * 3 + 1 ] ];
        let pointC = keypoints[ FaceTriangles[ i * 3 + 2 ] ];
        drawTriangle( output, pointA[ 0 ], pointA[ 1 ],
                      pointB[ 0 ], pointB[ 1 ], pointC[ 0 ],
                      pointC[ 1 ], scale );
    }

    // Добавьте только нос, щеки, глаза, брови и рот
    // Add just the nose, cheeks, eyes, eyebrows & mouth
    const features = [
        "noseTip",
        "leftCheek",
        "rightCheek",
        "leftEyeLower1", "leftEyeUpper1",
        "rightEyeLower1", "rightEyeUpper1",
        "leftEyebrowLower", //"leftEyebrowUpper",
        "rightEyebrowLower", //"rightEyebrowUpper",
```

```
            "lipsLowerInner", //"lipsLowerOuter",
            "lipsUpperInner", //"lipsUpperOuter",
        ];
        let points = [];
        features.forEach( feature => {
            face.annotations[ feature ].forEach( x => {
                points.push( ( x[ 0 ] - x1 ) / bWidth );
                points.push( ( x[ 1 ] - y1 ) / bHeight );
            });
        });
        // Только захватите лица, в которых уверены
        // Only grab the faces that are confident
        if( face.faceInViewConfidence > 0.9 ) {
            trainingData.push({
                input: points,
                output: ferData[ setIndex ].emotion,
            });
        }
});

setText( `${setIndex + 1}. Уверенность отслеживания лица/
          Face Tracking Confidence:
          ${face.faceInViewConfidence.toFixed( 3 )} -
          ${ferData[ setIndex ].emotion}` );
setIndex++;
requestAnimationFrame( trackFace );
}

(async () => {
    // Получите FER-2013-данные с локального веб-сервера
     // https://www.kaggle.com/msambare/fer2013
     // Данные могут быть загружены с Kaggle и
     // помещены в каталог "web/fer2013"
     // Получите самое малое число примеров
     // всех категорий эмоций
     // Get FER-2013 data from the local web server
      // https://www.kaggle.com/msambare/fer2013
      // The data can be downloaded from Kaggle and
     // placed inside the "web/fer2013" folder
      // Get the lowest number of samples out
     // of all emotion categories
    const minSamples = Math.min( ...Object.keys( fer2013 ).map(
                em => fer2013[ em ].length ) );
    Object.keys( fer2013 ).forEach( em => {
        shuffleArray( fer2013[ em ] );
        for( let i = 0; i < minSamples; i++ ) {
            ferData.push({
                emotion: em,
                file: fer2013[ em ][ i ]
            });
        }
```

```
        });
        shuffleArray( ferData );

        let canvas = document.getElementById( "output" );
        canvas.width = OUTPUT_SIZE;
        canvas.height = OUTPUT_SIZE;

        output = canvas.getContext( "2d" );
        output.translate( canvas.width, 0 );
        output.scale( -1, 1 ); // Зеркалируем
        output.fillStyle = "#fdffb6";
        output.strokeStyle = "#fdffb6";
        output.lineWidth = 2;

        // Загрузка модели обнаружения признаков лица
        // Load Face Landmarks Detection
        model = await faceLandmarksDetection.load(
            faceLandmarksDetection.SupportedPackages.mediapipeFacemesh
        );

        setText( "Загружено!/Loaded!" );

        trackFace();
    })();
    </script>
</body>
</html>
```

Часть 2: Выполнение модели «Обнаружение эмоций на лице»(«Facial Emotion Detection»)

Мы сделали почти все. Выполнение модели детектора эмоции на лице более просто, чем ее обучение. На этой веб-странице мы собираемся загрузить обученную(натренированную) **TensorFlow**-модель и протестировать ее на случайных лицах из **FER**-набора данных.

Мы можем модель обнаружения эмоции на лице загрузить в глобальную переменную прямо под кодом загрузки модели **«Обнаружение признаков лица»(«Face Landmarks Detection»)**. Если в **Части 1** статьи вы обучили(натренировали) свою собственную модель, то можете обновить путь, чтобы он соответствовал месту, где вы сохранили свою модель.

JavaScript

```
let emotionModel = null;

(async () => {
    ...
    // Загрузка модели обнаружения признаков лица
    // Load Face Landmarks Detection
    model = await faceLandmarksDetection.load(
        faceLandmarksDetection.SupportedPackages.mediapipeFacemesh
    );
    // Загрузка модели обнаружения эмоций на лице
    // Load Emotion Detection
    emotionModel = await tf.loadLayersModel( 'web/model/facemo.json' );
    ...
})();
```

После этого мы можем написать функцию выполнения модели на основе введенных **ключевых лицевых точек(key facial points)** и возвращающую название **обнаруженной эмоции(detected emotion)**:

JavaScript

```
async function predictEmotion( points ) {
    let result = tf.tidy( () => {
        const xs = tf.stack( [ tf.tensor1d( points ) ] );
        return emotionModel.predict( xs );
    });
    let prediction = await result.data();
    result.dispose();
    // Получите индекс максимального значения
    // Get the index of the maximum value
    let id = prediction.indexOf( Math.max( ...prediction ) );
    return emotions[ id ];
}
```

Чтобы было приятно просматривать тесты, мы можем сделать паузу с ожиданием в несколько секунд между тестовыми изображениями, и для этого создадим функцию утилиты ожидания:

JavaScript

```
function wait( ms ) {
    return new Promise( res => setTimeout( res, ms ) );
```

```
}
```

Теперь, чтобы задействовать утилиту, мы можем взять **ключевые точки отслеженного лица(key points of the tracked face)**, масштабировать их относительно ограничивающего прямоугольника, для подготовки в качестве входных данных, и выполнить **прогноз эмоции(emotion prediction)** и показать **ожидаемый результат(expected result)** по сравнению с **обнаруженным результатом(detected result)**, с паузой в **2** секунды между изображениями.

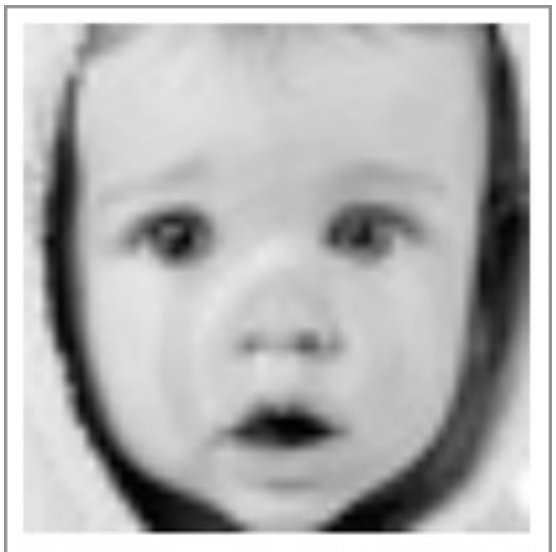
JavaScript

```
async function trackFace() {
    ...

    let points = null;
    faces.forEach( face => {
        ...
        // Добавьте только нос, щеки, глаза, брови и рот
        // Add just the nose, cheeks, eyes, eyebrows & mouth
        const features = [
            "noseTip",
            "leftCheek",
            "rightCheek",
            "leftEyeLower1", "leftEyeUpper1",
            "rightEyeLower1", "rightEyeUpper1",
            "leftEyebrowLower", //"leftEyebrowUpper",
            "rightEyebrowLower", //"rightEyebrowUpper",
            "lipsLowerInner", //"lipsLowerOuter",
            "lipsUpperInner", //"lipsUpperOuter",
        ];
        points = [];
        features.forEach( feature => {
            face.annotations[ feature ].forEach( x => {
                points.push( ( x[ 0 ] - x1 ) / bWidth );
                points.push( ( x[ 1 ] - y1 ) / bHeight );
            });
        });
});

if( points ) {
    let emotion = await predictEmotion( points );
```

```
      // Перевод, далее: закомментирован исходный код и
      // добавлен код с руссификацией названий эмоций
      // setText( `${setIndex + 1}.
      //       Expected: ${ferData[ setIndex ].emotion} vs.
      //            ${emotion}` );
      let cur_emotion = ferData[ setIndex ].emotion
      let rus_eng_cur_emotion = cur_emotion
if ( cur_emotion == "angry" ){ rus_eng_cur_emotion =
                               "сердитый(angry)" }
else if ( cur_emotion == "disgust" ){ rus_eng_cur_emotion =
                               "отвращение(disgust)" }
else if ( cur_emotion == "fear" ) { rus_eng_cur_emotion =
                               "страх(fear)" }
else if ( cur_emotion == "happy" ) { rus_eng_cur_emotion =
                               "счастливый(happy)" }
else if ( cur_emotion == "neutral" ) { rus_eng_cur_emotion =
                               "нейтральный(neutral)" }
else if ( cur_emotion == "sad" ) { rus_eng_cur_emotion =
                               "печальный(sad)" }
else if ( cur_emotion == "surprise" ) {rus_eng_cur_emotion =
                               "удивление(surprise)" }

setText( `${setIndex + 1}. Ожидаемо: ${rus_eng_cur_emotion} vs.
                                        ${emotion}` );
}
else {
      setText( "No Face" );
    }

    setIndex++;
    await wait( 2000 );
    requestAnimationFrame( trackFace );
}
```

11. Expected: surprise vs. surprise

16. Expected: disgust vs. sad



38. Expected: surprise vs. surprise

(надпись под левым снимком экрана **11.Ожидаемо: удивление по сравнению с удивлением(11.Expected: surprise vs. surprise)**

(надпись под средним снимком экрана **16.Ожидаемо: отвращение по сравнению с печалью(16.Expected: disgust vs. sad)**

(надпись под правым снимком экрана **38.Ожидаемо: удивление по сравнению с удивлением(38.Expected: surprise vs. surprise)**

Все готово! Наш код должен начать **предсказывать эмоции(predicting the emotions)** на **FER**-изображениях, чтобы соответствовать **ожидаемой эмоции(expected emotion)**. Попробуйте код и посмотрите, как он выполняется.

Часть 2: Финишная черта

Посмотрите на полный код, для выполнения обученной(натренированной) модели на изображениях **FER**-набора данных:

HTML

```
<html>
    <head>
        <title>Выполнение модели - Обнаружение эмоций на лице в браузере с помощью глубокого обучения с использованием библиотеки TensorFlow.js</title>
        <script src="https://cdn.jsdelivr.net/npm/@tensorflow/tfjs@2.4.0/dist/tf.min.js"></script>
        <script src="https://cdn.jsdelivr.net/npm/@tensorflow-models/face-landmarks-detection@0.0.1/dist/face-landmarks-detection.js"></script>
        <script src="web/fer2013.js"></script>
    </head>
    <body>
        <canvas id="output"></canvas>
        <img id="image" style="
            visibility: hidden;
            width: auto;
            height: auto;
            "/>
        <h1 id="status">Загрузка/Loading...</h1>
        <script>
        function setText( text ) {
            document.getElementById( "status" ).innerText = text;
        }
```

```
async function setImage( url ) {
    return new Promise( res => {
        let image = document.getElementById( "image" );
        image.src = url;
        image.onload = () => {
            res();
        };
    });
}

function shuffleArray( array ) {
    for( let i = array.length - 1; i > 0; i-- ) {
        const j = Math.floor( Math.random() * ( i + 1 ) );
        [ array[ i ], array[ j ] ] = [ array[ j ], array[ i ] ];
    }
}

function drawLine( ctx, x1, y1, x2, y2, scale = 1 ) {
    ctx.beginPath();
    ctx.moveTo( x1 * scale, y1 * scale );
    ctx.lineTo( x2 * scale, y2 * scale );
    ctx.stroke();
}

function drawTriangle( ctx, x1, y1, x2, y2, x3, y3, scale = 1 ) {
    ctx.beginPath();
    ctx.moveTo( x1 * scale, y1 * scale );
    ctx.lineTo( x2 * scale, y2 * scale );
    ctx.lineTo( x3 * scale, y3 * scale );
    ctx.lineTo( x1 * scale, y1 * scale );
    ctx.stroke();
}

function wait( ms ) {
    return new Promise( res => setTimeout( res, ms ) );
}

const OUTPUT_SIZE = 500;

//             ["сердитый",  "отвращение", "страх", "счастливый",
//             "нейтральный", "печальный",  "удивление"];
// const emotions = [ "angry", "disgust", "fear",
//       "happy", "neutral", "sad", "surprise" ];
const emotions = [ "сердитый(angry)", "отвращение(disgust)",
"страх(fear)", "счастливый(happy)", "нейтральный(neutral)",
"печальный(sad)", "удивление(surprise)" ];

let ferData = [];
let setIndex = 0;
let emotionModel = null;
```

```
let output = null;
let model = null;

async function predictEmotion( points ) {
    let result = tf.tidy( () => {
        const xs = tf.stack( [ tf.tensor1d( points ) ] );
        return emotionModel.predict( xs );
    });
    let prediction = await result.data();
    result.dispose();
    // Получите индекс максимального значения
    // Get the index of the maximum value
    let id = prediction.indexOf( Math.max( ...prediction ) );
    return emotions[ id ];
}

async function trackFace() {
    // Установить следующее изображение для обучения(тренировки)
    // Set to the next training image
    await setImage( ferData[ setIndex ].file );
    const image = document.getElementById( "image" );
    const faces = await model.estimateFaces( {
        input: image,
        returnTensors: false,
        flipHorizontal: false,
    });
output.drawImage(
    image,
    0, 0, image.width, image.height,
    0, 0, OUTPUT_SIZE, OUTPUT_SIZE
);

const scale = OUTPUT_SIZE / image.width;

let points = null;
faces.forEach( face => {
    // Рисуем мешь лица
    // Draw the face mesh
    const x1 = face.boundingBox.topLeft[ 0 ];
    const y1 = face.boundingBox.topLeft[ 1 ];
    const x2 = face.boundingBox.bottomRight[ 0 ];
    const y2 = face.boundingBox.bottomRight[ 1 ];
    const bWidth = x2 - x1;
    const bHeight = y2 - y1;
    drawLine( output, x1, y1, x2, y1, scale );
    drawLine( output, x2, y1, x2, y2, scale );
    drawLine( output, x1, y2, x2, y2, scale );
    drawLine( output, x1, y1, x1, y2, scale );

    // Добавьте только нос, щеки, глаза, брови и рот
    // Add just the nose, cheeks, eyes, eyebrows & mouth
```

```
            const features = [
                "noseTip",
                "leftCheek",
                "rightCheek",
                "leftEyeLower1", "leftEyeUpper1",
                "rightEyeLower1", "rightEyeUpper1",
                "leftEyebrowLower", //"leftEyebrowUpper",
                "rightEyebrowLower", //"rightEyebrowUpper",
                "lipsLowerInner", //"lipsLowerOuter",
                "lipsUpperInner", //"lipsUpperOuter",
            ];
            points = [];
            features.forEach( feature => {
                face.annotations[ feature ].forEach( x => {
                    points.push( ( x[ 0 ] - x1 ) / bWidth );
                    points.push( ( x[ 1 ] - y1 ) / bHeight );
                });
            });
        });

        if( points ) {
            let emotion = await predictEmotion( points );
// Перевод, далее: закомментирован исходный код и
// добавлен код с руссификацией названий эмоций
// setText( `${setIndex + 1}.
//       Expected: ${ferData[ setIndex ].emotion} vs.
//            ${emotion}` );
     let cur_emotion = ferData[ setIndex ].emotion
     let rus_eng_cur_emotion = cur_emotion
if ( cur_emotion == "angry" ){ rus_eng_cur_emotion =
                            "сердитый(angry)" }
else if ( cur_emotion == "disgust" ){ rus_eng_cur_emotion =
                            "отвращение(disgust)" }
else if ( cur_emotion == "fear" ) { rus_eng_cur_emotion =
                            "страх(fear)" }
else if ( cur_emotion == "happy" ) { rus_eng_cur_emotion =
                            "счастливый(happy)" }
else if ( cur_emotion == "neutral" ) { rus_eng_cur_emotion =
                            "нейтральный(neutral)" }
else if ( cur_emotion == "sad" ) { rus_eng_cur_emotion =
                            "печальный(sad)" }
else if ( cur_emotion == "surprise" ) {rus_eng_cur_emotion =
                            "удивление(surprise)" }

setText( `${setIndex + 1}. Ожидаемо: ${rus_eng_cur_emotion} vs.
                                        ${emotion}` );
        }
        else {
            setText( "Не лицо/No Face" );
        }

        setIndex++;
```

```
    await wait( 2000 );
    requestAnimationFrame( trackFace );
}

(async () => {
     // Получите FER-2013-данные с локального веб-сервера
     // https://www.kaggle.com/msambare/fer2013
     // Данные могут быть загружены с Kaggle и
     // помещены в каталог "web/fer2013"
     // Получите самое малое число примеров
     // всех категорий эмоций
     // Get FER-2013 data from the local web server
      // https://www.kaggle.com/msambare/fer2013
      // The data can be downloaded from Kaggle and
     // placed inside the "web/fer2013" folder
      // Get the lowest number of samples out
     // of all emotion categories
    const minSamples = Math.min( ...Object.keys( fer2013 ).map(
              em => fer2013[ em ].length ) );
    Object.keys( fer2013 ).forEach( em => {
        shuffleArray( fer2013[ em ] );
        for( let i = 0; i < minSamples; i++ ) {
            ferData.push({
                emotion: em,
                file: fer2013[ em ][ i ]
            });
        }
    });
    shuffleArray( ferData );

    let canvas = document.getElementById( "output" );
    canvas.width = OUTPUT_SIZE;
    canvas.height = OUTPUT_SIZE;

    output = canvas.getContext( "2d" );
    output.translate( canvas.width, 0 );
    output.scale( -1, 1 ); // Зеркалируем
    output.fillStyle = "#fdffb6";
    output.strokeStyle = "#fdffb6";
    output.lineWidth = 2;

    // Загрузка модели обнаружения признаков лица
    // Load Face Landmarks Detection
    model = await faceLandmarksDetection.load(
        faceLandmarksDetection.SupportedPackages.mediapipeFacemesh
    );
    // Загрузка модели обнаружения эмоций на лице
    // Load Emotion Detection
    emotionModel = await tf.loadLayersModel(
                        'web/model/facemo.json' );

    setText( "Загружено/Loaded!" );
```

```
            trackFace();
        })();
        </script>
    </body>
</html>
```

Что далее? Можно ли с помощью этой модели обнаружить эмоции на нашем лице?

В этой статье мы выходные данные **TensorFlow**-модели **«Обнаружение признаков лица»(«Face Landmarks Detection»)** скомбинировали с независимым набором данных, чтобы сгенерировать новую модель, которая может извлечь и предсказать больше информации из изображения, чем прежде. Реальным тестированием этой модели должно было бы быть предсказание эмоции на любом лице.

Давайте перейдем к следующей статье из этой серии статей, в которой мы будем использовать живое видео из веб-камеры нашего лица и посмотрим , сможет ли модель реагировать на наши выражения лица в режиме реального времени.

Эта статья является статьей из серии статей **Фильтры искусственного интеллекта лица в браузере(AI Face Filters in the Browser)**.