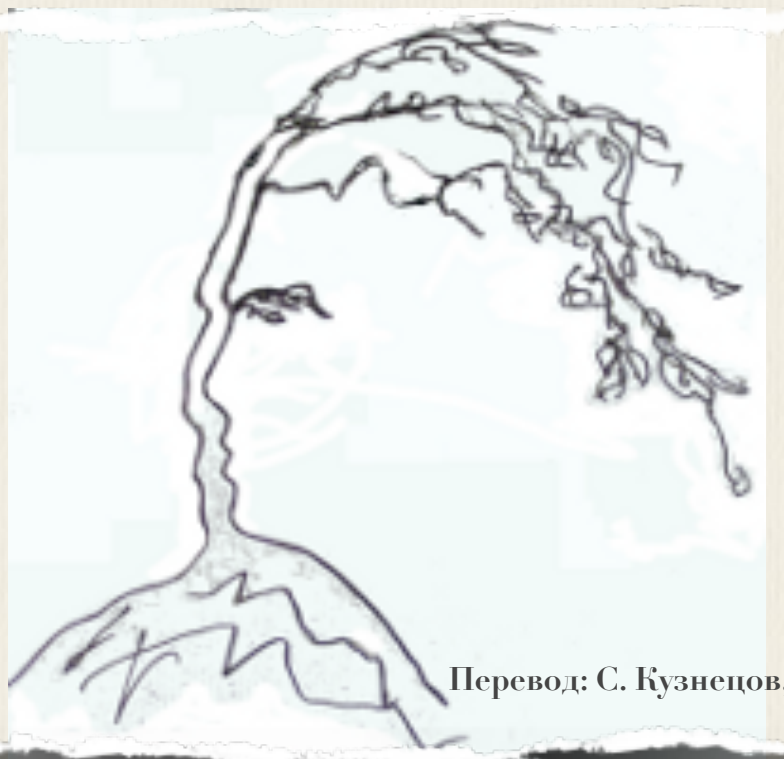
РАФАЭЛЬ МУН(RAPHAEL MUN)

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, 03.11.2023

Статья 3 «Детектор пушистых животных: распознавание нестандартных объектов в браузере, путем передачи обученности в TensorFlow.js»

Статья **3 «Детектор пушистых животных: распознавание нестандартных объектов в браузере, путем передачи обученности в TensorFlow.js»(«Fluffy Animal Detector: Recognizing Custom Objects in the Browser by Transfer Learning in TensorFlow.js»);** https://www.codeproject.com/Articles/5272772/Fluffy-Animal-Detector-Recognizing-Custom-Objects) является статьей из серии статей **Обнаружение касания лица с помощью Tensorflow.js(Face Touch Detection with Tensorflow.js)**.

10 июля 2020

В этой статье мы построим **Детектор пушистых животных(Fluffy Animal Detector)**, где я покажу вам способ усилить предварительно обученную модель **сверточной нейронной сети(Convolutional Neural Network; CNN)**, подобную модели **MobileNet**.

Здесь мы рассмотрим **передачу обученности(transfer learning)** на архитектуре **MobileNet v1**, **модификацию модели(modifying the model)**, и **обучение(тренировку) новой модели(training the new model)**.

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**!

Это - третья статья в нашей серии из шести статей:

1. Начало работы с глубоким изучением в браузере с использованием фреймворка TensorFlow.js(Getting Started With Deep Learning in Your Browser Using TensorFlow.js)

2. Собаки и пицца: машинное зрение в браузере с использованием TensorFlow.js(Dogs and Pizza: Computer Vision in the Browser With TensorFlow.js)

3. Детектор пушистых животных: распознавание нестандартных объектов в браузере, путем передачи обученности в TensorFlow.js(Fluffy Animal Detector: RecognizingCustom Objects in the Browser by Transfer Learning in TensorFlow.js).

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)

5. Обнаружение касания лица в фреймворке TensorFlow.js, часть 2: Использование BodyPix(Face Touch Detection With TensorFlow.js Part 2: Using BodyPix)

6. Интерпретация жестов рук и языка жестов в веб-камере с AI, используя TensorFlow.js(Interpreting Hand Gestures and Sign Language in the Webcam With AI Using TensorFlow.js)

Как насчет еще более крутого машинного зрения(computer vision) прямо в веб-браузере? На этот раз мы построим Детектор пушистых животных(Fluffy Animal Detector), где я покажу вам способ усилить предварительно обученную модель сверточной нейронной сети(Convolutional Neural Network; CNN), подобную модели MobileNet. Модель поступит в виде модели, обученной на миллионах изображений на ЭВМ с огромной вычислительной мощностью; мы загрузим ее, чтобы быстро обучить(натренировать; quickly learn) для распознавания(recognize) других типов объектов для вашего специфичного сценария, путем передачи обученности с использованием TensorFlow.js(by Transfer Learning using TensorFlow.js).

So Cute & Fluffy!

Начальная точка

Чтобы запустить обучение `распознаванию нестандартных объектов(custom object recognition)` на `базе предварительно обученной модели MobileNet(based on the pre-trained MobileNet model)`, мы должны сделать следующее:

- Собрать демонстрационные изображения, разбитые на категории `"пушистый"("fluffy")` и `"не - пушистый"("not-fluffy")`, включая некоторые изображения, которые не являются частью `MobileNet`-предварительно обученных категорий(изображения, которые я использовал в этом проекте, от pexels.com)
- Импортировать фреймворк машинного обучения `TensorFlow.js`
- Определить метки категорий `Пушистые(Fluffy)` и `Не-Пушистые(Not-Fluffy)`
- В произвольном порядке выберите и загрузите одно из изображений
- Покажите результат прогноза в тексте
- Загрузите предварительно обученную модель `MobileNet` и классифицируйте изображения

Для этого проекта нашей начальной точкой будет эта страница:

JavaScript

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

```
        <title>Fluffy Animal Detector: Recognizing Custom Objects in
             the Browser by Transfer Learning in TensorFlow.js</title>
         <script
src="https://cdn.jsdelivr.net/npm/@tensorflow/tfjs@2.0.0/dist/tf.min.js"><
/script>
        <style>
            img {
                object-fit: cover;
            }
        </style>
    </head>
    <body>
        <img id="image" src="" width="224" height="224" />
        <h1 id="status">Загрузка/Loading...</h1>
        <script>
        const fluffy = [
            "web/dalmation.jpg", //
https://www.pexels.com/photo/selective-focus-photography-of-woman-holding-
adult-dalmatian-dog-1852225/
            "web/maltese.jpg", //
https://www.pexels.com/photo/white-long-cot-puppy-on-lap-167085/
            "web/pug.jpg", //
https://www.pexels.com/photo/a-wrinkly-pug-sitting-in-a-wooden-table-34756
80/
            "web/pomeranians.jpg", //
https://www.pexels.com/photo/photo-of-pomeranian-puppies-4065609/
            "web/kitty.jpg", //
https://www.pexels.com/photo/eyes-cat-coach-sofa-96938/
            "web/upsidedowncat.jpg", //
https://www.pexels.com/photo/silver-tabby-cat-1276553/
            "web/babychick.jpg", //
https://www.pexels.com/photo/animal-easter-chick-chicken-5145/
            "web/chickcute.jpg", //
https://www.pexels.com/photo/animal-bird-chick-cute-583677/
            "web/beakchick.jpg", //
https://www.pexels.com/photo/animal-beak-blur-chick-583675/
            "web/easterchick.jpg", //
https://www.pexels.com/photo/cute-animals-easter-chicken-5143/
            "web/closeupchick.jpg", //
https://www.pexels.com/photo/close-up-photo-of-chick-2695703/
            "web/yellowcute.jpg", //
https://www.pexels.com/photo/nature-bird-yellow-cute-55834/
            "web/chickbaby.jpg", //
https://www.pexels.com/photo/baby-chick-58906/
        ];

        const notfluffy = [
            "web/pizzaslice.jpg", //
https://www.pexels.com/photo/pizza-on-brown-wooden-board-825661/
            "web/pizzaboard.jpg", //
https://www.pexels.com/photo/pizza-on-brown-wooden-board-825661/
            "web/squarepizza.jpg", //
https://www.pexels.com/photo/pizza-with-bacon-toppings-1435900/
```

```
            "web/pizza.jpg", //
https://www.pexels.com/photo/pizza-on-plate-with-slicer-and-fork-2260200/
            "web/salad.jpg", //
https://www.pexels.com/photo/vegetable-salad-on-plate-1059905/
            "web/salad2.jpg", //
https://www.pexels.com/photo/vegetable-salad-with-wheat-bread-on-the-side-
1213710/
        ];

        // Создайте окончательный, объединенный список изображений
         // Create the ultimate, combined list of images
        const images = fluffy.concat( notfluffy );

        // Недавно определенные Метки
         // Newly defined Labels
        const labels = [
            "So Cute & Fluffy!",
            "Not Fluffy"
        ];

        function pickImage() {
            document.getElementById( "image" ).src =
              images[ Math.floor( Math.random() * images.length ) ];
        }

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

        async function predictImage() {
            let result = tf.tidy( () => {
                const img = tf.browser.fromPixels(
                      document.getElementById( "image" ) ).toFloat();
                const normalized = img.div( 127 ).sub( 1 );
                 // Нормализуем из диапазона [0,255] в диапазон [-1,1]
                 // Normalize from [0,255] to [-1,1]
                const input = normalized.reshape( [ 1, 224, 224, 3 ] );
                return model.predict( input );
            });
            let prediction = await result.data();
            result.dispose();
            // Get the index of the highest value in the prediction
            let id = prediction.indexOf( Math.max( ...prediction ) );
            setText( labels[ id ] );
        }

        // Модель Mobilenet v1 0.25 для изображения 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;
```

```
        (async () => {
            // Загрузите модель
            // Load the model
            model = await tf.loadLayersModel( mobilenet );
            setInterval( pickImage, 5000 );
            document.getElementById( "image" ).onload = predictImage;
        })();
        </script>
    </body>
</html>
```

Вы можете изменить массив изображений, чтобы имена соответствовали именам файлов ваших тестовых изображений. Как только откроете в браузере, то эта страница будет показывать вам различное, в произвольном порядке выбранное изображение каждые пять секунд.

Прежде чем мы пойдем далее, я хочу отметить, чтобы этот проект выполнялся правильно, веб-страница и изображения должны быть размещены на веб-сервере (из-за `canvas`-ограничений языка `HTML5`).

Полное объяснение смотрите в предыдущей статье.

Передача обученности на архитектуре MobileNet v1

Прежде чем мы сможем применить передачу любой обученности, важно понять архитектуру нейронной сети модели `MobileNet`.

Модели MobileNets были разработаны с прицелом на применение передачи обученности; они работают через прямые, последовательные `сверточные слои(convolutional layers)` прежде, чем их вывод передать финальному набору слоев классификации, которые определяют вывод через `1000` категорий.

Проверьте это напечатанный вид архитектуры, когда вы выполните функцию модели `model.summary()`:

```
_________________________________________________________________
Слой/Layer (тип/type) Выход/Output форма(shape) Параметр/Param #
=================================================================
input_1 (InputLayer) [null,224,224,3] 0
_________________________________________________________________
```

```
conv1 (Conv2D) [null,112,112,8] 216
_________________________________________________________________
conv1_bn (BatchNormalization [null,112,112,8] 32
_________________________________________________________________
conv1_relu (Activation) [null,112,112,8] 0
_________________________________________________________________
....
_________________________________________________________________
conv_pw_13_bn (BatchNormaliz [null,7,7,256] 1024
_________________________________________________________________
conv_pw_13_relu (Activation) [null,7,7,256] 0
_________________________________________________________________
global_average_pooling2d_1 ( [null,256] 0
_________________________________________________________________
reshape_1 (Reshape) [null,1,1,256] 0
_________________________________________________________________
dropout (Dropout) [null,1,1,256] 0
_________________________________________________________________
conv_preds (Conv2D) [null,1,1,1000] 257000
_________________________________________________________________
act_softmax (Activation) [null,1,1,1000] 0
_________________________________________________________________
reshape_2 (Reshape) [null,1000] 0
=================================================================
Всего параметров(Total params): 475544
Обучаемые параметры(Trainable params): 470072
Не-Обучаемые параметры(Non-trainable params): 5472
```

Все верхние слои, которые начинаются с приставки **conv**, являются **сетевыми слоями(network layers)**, **поиска в пространственной информации пикселей(looking at spatial information of the pixels)**, которые в конечном счете компилируют в **запуск классификации(start of the classification)** с **global_average_pooling2d_1**, и затем **в финале проходят(finally pass)** через **conv_preds** слой, который выводит **1000** исходных категорий, на которые **MobileNet** был **обучен предсказывать(trained to predict)**.

Мы собираемся отрезать эту модель прямо перед слоем **conv_preds**(т.е., в слое **"dropout"("недоучка")**), к **"top"("верх")** присоединить новые слои для классификации и только эти слои обучить предсказывать всего две категории - категория **Пушистый(Fluffy)** по сравнению с категорией **Не-Пушистый(Not-Fluffy)** - при **сохранении предварительно обученных пространственных слоев в целости(keeping the pre-trained spatial layers intact)**.

Давайте сделаем это!

Изменение модели

После загрузки предварительно обученной MobileNet-модели мы можем найти наш слой "bottleneck"("узкое место") и создать новый слой, усеченная базовая модель(truncated base model):

Python

```
const bottleneck = model.getLayer( "dropout" ); // Это - финальный слой,
перед слоем conv_pred предварительно обученный слой классификации
// This is the final layer before the conv_pred pre-trained classification
layer
const baseModel = tf.model({
    inputs: model.inputs,
    outputs: bottleneck.output
});
```

Затем, давайте «заморозим» все слои перед слоем "bottleneck"("узкое место"), чтобы сохранить обученность модели, так, чтобы мы могли усилить всю вычислительную мощность модели, которая уже была помещена в этот блок модели.

python

```
// Заморозьте сверточную базу
// Freeze the convolutional base
for( const layer of baseModel.layers ) {
    layer.trainable = false;
}
```

Затем мы можем к выводу базовой модели присоединить голову/верхушку нестандартной классификации, состоящую из множества плотных dense-слоев, для новой модели TensorFlow, которая готова к обучению(тренировке).

Финальный плотный dense-слой содержит только два модуля, соответствуя категории Пушистый(Fluffy) по сравнению с категорией Не-Пушистый(Not-Fluffy), и использует softmax-активацию, которая нормализует сумму выводов, чтобы она стала равна 1.0, означая, что мы можем использовать каждую предсказанную категорию(predicted category) в

качестве значения уверенности прогноза модели(model's prediction confidence value).

Python

```
// 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'
} ) );
// Создайте новую модель
// Build the new model
const newOutput = newHead.apply( baseModel.outputs[ 0 ] );
const newModel = tf.model( { inputs: baseModel.inputs, outputs: newOutput
} );
```

Чтобы содержать код в чистоте, мы можем поместить его в функцию и выполнить его прямо после того, как мы загрузим модель MobileNet:

Python

```
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 основу
    // 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'
    } ) );
    // Создайте 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 () => {
    // Загрузите модель
    // Load the model
    model = await tf.loadLayersModel( mobilenet );
    model = createTransferModel( model );
    setInterval( pickImage, 2000 );
    document.getElementById( "image" ).onload = predictImage;
})();
```

Обучение новой модели

Мы сделали почти все. Еще осталось пройти только один шаг, который должен обучить нашу новую **TensorFlow**-модель на наших нестандартных(пользовательских) данных обучения(тренировки).

Чтобы генерировать тензоры данных обучения(тренировки) из нестандартных(пользовательских) изображений, давайте создадим функцию, которая загружает изображение в элемент изображения веб-страницы и получает **нормализованный тензор(normalized tensor)**:

JavaScript

```
async function getTrainingImage( url ) {
    return new Promise( ( resolve, reject ) => {
        document.getElementById( "image" ).src = url;
        document.getElementById( "image" ).onload = () => {
            const img = tf.browser.fromPixels(
                    document.getElementById( "image" ) ).toFloat();
            const normalized = img.div( 127 ).sub( 1 );
            resolve( normalized );
        };
    });
}
```

И теперь, мы можем использовать эту функцию, чтобы создать нашу **стек(стопку) ввода и целевых тензоров(stack of input and target tensors)**. Вы могли бы помнить их как **xs**- и **ys**-, который мы использовали для обучения в первой статье серии статей. Мы будем использовать только половину изображений из каждой категории для обучения, чтобы проверить, что наша **новая модель делает прогнозы(new model makes predictions)** для новых(неизвестных) изображений.

Python

```
// Установите данные обучени(тренировки)
// Setup training data
const imageSamples = [];
const targetSamples = [];
for( let i = 0; i < fluffy.length / 2; i++ ) {
    let result = await getTrainingImage( fluffy[ i ] );
    imageSamples.push( result );
    targetSamples.push( tf.tensor1d( [ 1, 0 ] ) );
}
for( let i = 0; i < notfluffy.length / 2; i++ ) {
    let result = await getTrainingImage( notfluffy[ i ] );
    imageSamples.push( result );
    targetSamples.push( tf.tensor1d( [ 0, 1 ] ) );
}
const xs = tf.stack( imageSamples );
const ys = tf.stack( targetSamples );
tf.dispose( [ imageSamples, targetSamples ] );
```

Наконец, мы **компилируем и соответствуем модели данным(compile and fit the model to the data)**. Благодаря всему предварительному обучению в

модели `MobileNet`, на этот раз нам потребуются только приблизительно `30` эпох (вместо `100`), чтобы надежно различать категории.

Python

```
model.compile( { loss: "meanSquaredError", optimizer: "adam", metrics: [
"acc" ] } );

// Обучите модель на новых выборках изображения
// Train the model on new image samples
await model.fit( xs, ys, {
    epochs: 30,
    shuffle: true,
    callbacks: {
        onEpochEnd: ( epoch, logs ) => {
            console.log( "Epoch #", epoch, logs );
        }
    }
});
```

При применении метода кодирования по методике сайта KonMari method to code, давайте зажжем некоторые радостные элементы, помещая весь вышеупомянутый код в функцию прежде, чем вызвать его:

pyhton

```
async function trainModel() {
    setText( "Training..." );

    // Установите данные обучения(тренировки)
    // Setup training data
    const imageSamples = [];
    const targetSamples = [];
    for( let i = 0; i < fluffy.length / 2; i++ ) {
        let result = await getTrainingImage( fluffy[ i ] );
        imageSamples.push( result );
        targetSamples.push( tf.tensor1d( [ 1, 0 ] ) );
    }
    for( let i = 0; i < notfluffy.length / 2; i++ ) {
        let result = await getTrainingImage( notfluffy[ i ] );
        imageSamples.push( result );
        targetSamples.push( tf.tensor1d( [ 0, 1 ] ) );
    }
    const xs = tf.stack( imageSamples );
    const ys = tf.stack( targetSamples );
    tf.dispose( [ imageSamples, targetSamples ] );
```

```
    model.compile( { loss: "meanSquaredError", optimizer: "adam", metrics:
[ "acc" ] } );

    // Обучите модель на новых выборках изображения
    // Train the model on new image samples
    await model.fit( xs, ys, {
        epochs: 30,
        shuffle: true,
        callbacks: {
            onEpochEnd: ( epoch, logs ) => {
                console.log( "Epoch #", epoch, logs );
            }
        }
    });
}
...
(async () => {
    // Загрузите модель
    // Load the model
    model = await tf.loadLayersModel( mobilenet );
    model = createTransferModel( model );
    await trainModel();
    setInterval( pickImage, 2000 );
    document.getElementById( "image" ).onload = predictImage;
})();
```

Выполнение распознавания объектов

Со всеми объединенными частями кода, нам нужно выполнить наш **«Детектор пушистых животных(«Fluffy Animal Detector»)** и увидеть, что он **учится распознавать пушистость(learn to recognize the fluffiness)**!:

So Cute & Fluffy!

Not Fluffy

Финишная черта

Чтобы обернуть наш проект, вот, заключительный код:

JavaScript

```
<html>
    <head>
        <title>Fluffy Animal Detector: Recognizing Custom Objects in the
Browser by Transfer Learning in TensorFlow.js</title>
        <script
src="https://cdn.jsdelivr.net/npm/@tensorflow/tfjs@2.0.0/dist/tf.min.js"><
/script>
        <style>
            img {
                object-fit: cover;
            }
        </style>
    </head>
    <body>
        <img id="image" src="" width="224" height="224" />
        <h1 id="status">Загрузка/Loading...</h1>
        <script>
        const fluffy = [
            "web/dalmation.jpg", //
https://www.pexels.com/photo/selective-focus-photography-of-woman-holding-
adult-dalmatian-dog-1852225/
            "web/maltese.jpg", //
https://www.pexels.com/photo/white-long-cot-puppy-on-lap-167085/
            "web/pug.jpg", //
https://www.pexels.com/photo/a-wrinkly-pug-sitting-in-a-wooden-table-34756
80/
```

```
            "web/pomeranians.jpg", //
https://www.pexels.com/photo/photo-of-pomeranian-puppies-4065609/
            "web/kitty.jpg", //
https://www.pexels.com/photo/eyes-cat-coach-sofa-96938/
            "web/upsidedowncat.jpg", //
https://www.pexels.com/photo/silver-tabby-cat-1276553/
            "web/babychick.jpg", //
https://www.pexels.com/photo/animal-easter-chick-chicken-5145/
            "web/chickcute.jpg", //
https://www.pexels.com/photo/animal-bird-chick-cute-583677/
            "web/beakchick.jpg", //
https://www.pexels.com/photo/animal-beak-blur-chick-583675/
            "web/easterchick.jpg", //
https://www.pexels.com/photo/cute-animals-easter-chicken-5143/
            "web/closeupchick.jpg", //
https://www.pexels.com/photo/close-up-photo-of-chick-2695703/
            "web/yellowcute.jpg", //
https://www.pexels.com/photo/nature-bird-yellow-cute-55834/
            "web/chickbaby.jpg", //
https://www.pexels.com/photo/baby-chick-58906/
        ];

        const notfluffy = [
            "web/pizzaslice.jpg", //
https://www.pexels.com/photo/pizza-on-brown-wooden-board-825661/
            "web/pizzaboard.jpg", //
https://www.pexels.com/photo/pizza-on-brown-wooden-board-825661/
            "web/squarepizza.jpg", //
https://www.pexels.com/photo/pizza-with-bacon-toppings-1435900/
            "web/pizza.jpg", //
https://www.pexels.com/photo/pizza-on-plate-with-slicer-and-fork-2260200/
            "web/salad.jpg", //
https://www.pexels.com/photo/vegetable-salad-on-plate-1059905/
            "web/salad2.jpg", //
https://www.pexels.com/photo/vegetable-salad-with-wheat-bread-on-the-side-
1213710/
        ];

        // Создайте окончательный, объединенный список изображений
        // Create the ultimate, combined list of images
        const images = fluffy.concat( notfluffy );

        // Новые определенные метки
        // Newly defined Labels
        const labels = [
            "Настолько Милый & Пушистый!/So Cute & Fluffy!",
            "Не пушистый/ NotFluffy"
        ];

        function pickImage() {
            document.getElementById( "image" ).src =
                images[ Math.floor( Math.random() * images.length ) ];
        }
```

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

async function predictImage() {
    let result = tf.tidy( () => {
        const img = tf.browser.fromPixels(
              document.getElementById( "image" ) ).toFloat();
        const normalized = img.div( 127 ).sub( 1 );
              // Нормализуем из диапазона [0,255] в диапазон [-1,1]
              // Normalize from [0,255] to [-1,1]
        const input = normalized.reshape( [ 1, 224, 224, 3 ] );
        return model.predict( input );
    });
    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 - заключительный
     // уровень, прежде чем conv_pred
     // предварительно обучил уровень
     //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 основу
 // 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'
    } ) );
    // Создайте 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 getTrainingImage( url ) {
    return new Promise( ( resolve, reject ) => {
        document.getElementById( "image" ).src = url;
        document.getElementById( "image" ).onload = () => {
            const img = tf.browser.fromPixels(
                  document.getElementById( "image" ) ).toFloat();
            const normalized = img.div( 127 ).sub( 1 );
            resolve( normalized );
        };
    });
}

async function trainModel() {
    setText( "Training..." );

    // Установите данные обучения(тренировки)
    // Setup training data
    const imageSamples = [];
    const targetSamples = [];
    for( let i = 0; i < fluffy.length / 2; i++ ) {
        let result = await getTrainingImage( fluffy[ i ] );
        imageSamples.push( result );
        targetSamples.push( tf.tensor1d( [ 1, 0 ] ) );
    }
    for( let i = 0; i < notfluffy.length / 2; i++ ) {
        let result = await getTrainingImage( notfluffy[ i ] );
        imageSamples.push( result );
        targetSamples.push( tf.tensor1d( [ 0, 1 ] ) );
    }
    const xs = tf.stack( imageSamples );
    const ys = tf.stack( targetSamples );
    tf.dispose( [ imageSamples, targetSamples ] );

    model.compile( { loss: "meanSquaredError",
```

```
                    optimizer: "adam", metrics: [ "acc" ] } );

            // Обучите модель на новых
            // выборках изображения
            // Train the model on new image samples
            await model.fit( xs, ys, {
                epochs: 30,
                shuffle: true,
                callbacks: {
                    onEpochEnd: ( epoch, logs ) => {
                        console.log( "Epoch #", epoch, logs );
                    }
                }
            });
        }

        // 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;

        (async () => {
            // Загрузите модель
            // Load the model
            model = await tf.loadLayersModel( mobilenet );
            model = createTransferModel( model );
            await trainModel();
            setInterval( pickImage, 2000 );
            document.getElementById( "image" ).onload = predictImage;
        })();
        </script>
    </body>
</html>
```

Что далее? Можем мы обнаружить лица?

Вы уже поражены тем, что возможно с глубоким изучением в веб-странице, или насколько быстро и просто это? Далее мы используем простой в использовании **API**-интерфейс для разработки для веб-камеры браузера в языке **HTML5**, чтобы **обучить и выполнить прогнозы на изображениях в реальном времени(train and run predictions on real-time images)**.

В серии статей, смотрите следующую статью **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).