Министерство науки и высшего образования Российской Федерации

Муромский институт

федерального государственного бюджетного образовательного учреждения
высшего образования

**«Владимирский государственный университет
имени Александра Григорьевича и Николая Григорьевича Столетовых»**

(МИВлГУ)

Факультет информационных технологий

Кафедра информационных систем

ОТЧЕТ

по НИР

тема: Обзор и исследование библиотеки TensorFlow для распознавания изображений

Руководитель

к. т. н., доц. каф. ИС
(уч.степень, звание)

Подгорнова Ю.А.
(Фамилия, инициалы)

(оценка)

(подпись) (дата)

Члены комиссии

Студент ИСм-121
(группа)

Минеев Р.Р.
(фамилия, инициалы)

(подпись) (Ф.И.О.)

(подпись) (Ф.И.О.)

(подпись) (дата)

Муром 2021

Содержание

Введение

Распознавание изображений — это совокупность методов компьютерного зрения для идентификации объектов на изображениях или видео. Эти методы применяются во многих сферах жизни для повышения производительности или улучшения качества взаимодействия человека с какими-либо прикладными программами. Самыми яркими примерами применения этих методов являются:

- распознавание лиц, которое используется производителями мобильных телефонов (как способ разблокировки смартфона), социальными сетями (распознавание людей на изображении для установления метки человека, который там изображен), и т.д.;

- системы распознавания объектов, которые выбирают и идентифицируют объекты из загруженных изображений (визуальный поиск);

- анализ изображения для получения информации о том, где и какой объект находится, какие у него в данный момент свойства и возможное поведение в будущем (используется в основном для анализа видеопотока).

Актуальность темы распознавания изображений заключается в том, что глобально все изображения со всех камер мира составляют библиотеку неструктурированных данных, которые, задействовав нейросети, машинное обучение и искусственный интеллект, необходимо структурировать и использовать для выполнения различных задач: бытовых, социальных, профессиональных и государственных, в частности, обеспечения безопасности.

В данный момент существует множество способов создать нейронную сеть или систему глубокого обучения для распознавания изображений и для многих других сфер ее применения. Выбор способа зависит от сферы применения и устройства, на котором данная система будет реализована. Наиболее простым, а самое главное производительным способом является

использование языка Python и фреймворка TensorFlow, разработанного компанией Google.

Цель данной работы – обзор методов TensorFlow для распознавания объектов на изображении и классификации изображений.

Задачи на научно–исследовательскую работу:

- разобрать методы для классификации изображений;
- разобрать методы для выделения объектов на изображении.

1. Обзор методов библиотеки TensorFlow

В настоящее время самой известной библиотекой глубокого обучения в мире является Google TensorFlow. Продукт Google использует машинное обучение во всех своих продуктах для улучшения поисковой системы, перевода, подписи к изображениям или рекомендаций.

Архитектура TensorFlow работает в трех частях:

- предварительная обработка данных;
- настройка и структурирование модели;
- обучение и оценка модели.

Название TensorFlow произошло от данных, которыми данная библиотека оперирует. Она принимает входные данные в виде многомерного массива, также известного как тензоры. Для конструирования модели необходимо создать блок-схему (последовательность) операций (называемую «График»), которую модель будет выполнять с этим вводом. Вход вводится на одном конце, а затем он проходит через систему множества операций и выходит на другом конце в качестве вывода (ответа нейронной сети).

В TensorFlow все вычисления включают тензоры [1]. Тензор — это вектор или матрица n-измерений, представляющая все типы данных. Все значения в тензоре содержат идентичный тип данных.

Каждая операция, которую выполняет TensorFlow, включает в себя манипулирование тензором. Существует четыре основных тензора:

– tf.constant – фиксированный тензор с произвольным количеством измерений, заданный пользователем;

– tf.Variable представляет узел, где значения всегда меняются;

– tf.placeholder – заполнитель, который используется для инициализации данных, передаваемых внутри тензоров;

– tf.SparseTensor позволяет при работе с тензорами, которые содержат много нулевых значений, хранить их с экономией места и времени.

Каждая нейронная сеть состоит из нейронов и связей между ними. В TensorFlow существуют способы организовать данную структуру связей без особого труда. TensorFlow включается себя подбиблиотеку Keras, которая способна организовать нейронную сеть и настроить слои (Layers).

Первое, что необходимо сделать, это создать объект-последовательность слоёв. Метод keras.Sequential собирает обую структуру в один удобный объект, который в последствии можно будет скомпилировать и обучить для последующего использования модели.

Для создания слоя необходимо вызвать метод Dense, который будет принимать несколько аргументов, такие как: количество нейронов, функцию активации и информацию о входных данных (если данный слой создаётся для приёма входящих данных).

В TensorFlow уже описаны (практически) все возможные на данный момент функции активации слоёв, так что их использование упрощено максимально.

Пример объекта Sequential для нейронной сети с двумя слоями (входным и выходным) и одним скрытым:

```
baseline_model = keras.Sequential(
[
    layers.Dense(16, activation='relu', input_shape=(8,2)),
    layers.Dense(16, activation='relu'),
    layers.Dense(1, activation='sigmoid')
]
```

Из кода выше можно понять, что будет создана модель, состоящая из трёх последовательно соединённых слоёв. В первом и последующем слое будет по 16 нейронов. Во входной слой будет поступать матрица 8 на 2. На выходе будет один нейрон, а значит модель является бинарным

классификатором, который будет выдавать либо «Да», либо «Нет» на очередной поток входных данных.

На данном этапе, когда модель собрана, ее необходимо скомпилировать для последующего обучения. Для этого существует метод compile.

```
baseline_model.compile(optimizer='adam',
                       loss='binary_crossentropy',
                       metrics=['accuracy', 'binary_crossentropy'])
```

Здесь в аргументах указываются:

1. Функция потерь — величина, минимизируемая во время обучения. Она определяет меру успеха задачи;
2. Оптимизатор — алгоритм, определяющий как обновить сеть, исходя из функции потерь. Здесь реализуется специфический вариант стохастического градиентного спуска;
3. Метрики — величины, которые необходимо отслеживать во время обучения и проверки, например точность классификации. В отличие от потерь, при обучении эти величины не оптимизируются напрямую.

В данный момент модель собрана и метод summary может вывести всё информацию о структуре модели (рис. 1).

```
Model: "sequential_5"
_________________________________________________________________
Layer (type)                 Output Shape              Param #
=================================================================
dense_15 (Dense)             (None, 16)                160016
_________________________________________________________________
dense_16 (Dense)             (None, 16)                272
_________________________________________________________________
dense_17 (Dense)             (None, 1)                 17
=================================================================
Total params: 160,305
```

Рисунок 1 – Информация о модели

Далее можно перейти непосредственно к обучению модели. Для этого есть метод fit, в котором указываются настройки процесса обучения:

– входные данные, их вид, классификация (если имеется);

– количество эпох обучения;

– данные для проверки модели (тестовая выборка);

– требуемый минимальный коэффициент ошибок, который будет говорить о том, что модель достаточно обучена;

– количество данных из обучающей выборки, которые надо обучать отдельными блоками, каждую эпоху.

```
baseline_history = baseline_model.fit(train_data,
train_labels,epochs=20, batch_size=512,
            validation_data=(test_data, test_labels),
            verbose=2)
```

Большое количество параметров, которые указываются в этих методах, может помочь при формировании моделей любой сложности. Процесс максимально упрощен для рядового пользователя и все ограничивается только его знаниями о строении нейронных сетей и задачей, которую пользователь собирается решить.

Далее будет приведено несколько примеров использования данной библиотеки для решения различного рода задач.

2. Классификация изображений

Распознавание изображения относится к задаче ввода изображения в нейронную сеть и присвоения какой-либо метки для этого изображения. Метка, которую выводит сеть, будет соответствовать заранее определенному классу. Может быть присвоено как сразу несколько классов, так и только один. Если существует всего только один класс, обычно применяется термин «распознавание», тогда как задача распознавания нескольких классов часто называется «классификацией».

Подмножество классификаций изображений является уже определением объектов, когда определенные экземпляры объектов идентифицируются как принадлежащие к определенному классу, например, животные, автомобили или люди.

Чтобы выполнить распознавание или классификацию изображений, нейронная сеть должна выполнить извлечение признаков. Признаки — это элементы данных, которые представляют максимальный интерес и которые будут передаваться по нейросети. В конкретном случае распознавания изображений такими признаками являются группы пикселей, такие как линии и точки, которые сеть будет анализировать на наличие паттерна.

Распознавание признаков (или извлечение признаков) — это процесс извлечения соответствующих признаков из входного изображения, чтобы их можно было проанализировать. Многие изображения содержат аннотации или метаданные, которые помогают нейросети находить соответствующие признаки.

При обучении нейронной сети для распознавания образов с учителем имеется выборка с истинными ответами на вопрос, что изображено на картинке – метками классов. Нейросети подаются на вход эти изображения, после чего вычисляется ошибка, сравнивающая выходные значения с истинными метками классов. В зависимости от степени и характера

несоответствия предсказания нейронной сети, её веса корректируются, ответы нейронной сети подстраиваются под истинные ответы, пока ошибка не станет минимальной.

Для того, чтобы реализовать нейронную сеть для распознавания изображений с учителем (многослойный персептрон [3]), можно воспользоваться встроенными функциями фреймворка TensorFlow.

В начале необходимо скачать набор данных (датасет), на которых будет обучаться нейронная сеть. Для этого в TensorFlow имеется модуль datasets. Скачаем датасет fashion_mnist для дальнейшего обучения нейронной сети.

```
from tensorflow.keras.datasets import fashion_mnist
(trainX, trainY), (testX, testY) = fashion_mnist.load_data()
```

Данный датасет хранит изображения вещей (рис. 2), которые можно разделить на 10 классов.

Рисунок 2 – Датасет fashion_mnist

Функция load_data выгрузит все данные из данного датасета и вернёт их в виде двух кортежей. Первый кортеж будет хранить тренировочные изображения (trainX) и названия классов (trainY), к которым эти изображения принадлежат. Второй кортеж хранит тестовые данные (testX и testY).

Обучающая (тренировочная) выборка используется собственно для «обучения» той или иной модели, т.е. для построения математических отношений между некоторой переменной-откликом и предикторами, тогда как контрольная («тестовая») выборка служит для получения оценки прогнозных свойств модели на новых данных, т.е. данных, которые не были использованы для обучения модели. Как правило, обучающая выборка составляет 75-80% от объема исходных данных. Если вывести длину trainX и testX можно увидеть, что TensorFlow сам делит выборки по этому правилу.

```
print(f'len trainX - {len(trainX)} len testX - {len(testX)}')
len trainX - 50000 len testX - 10000
```

Каждое изображение в этом датасете имеет размер 28 на 28. Если оценивать каждый пиксель как отдельный нейрон, то первый слой, обучаемый, будет состоять из 784 нейронов. Каждый пиксель хранит информацию о яркости в диапазоне от 0 до 255. Чтобы привести эти значения в более пригодный для понимания сети вид, их необходимо представить в десятичный вид от 0 до 1.

```
trainX, testX = trainX / 255.0, testX / 255.0
```

Для реализации структуры многослойного персептрона с использованием тензорного потока у TensorFlow есть встроенные методы для создания и агрегирования группой слоёв.

```
s = tf.InteractiveSession()
```

tf.InteractiveSession() — это способ напрямую запустить модель тензорного потока без создания экземпляра графа всякий раз, когда требуется запустить модель. Построим модель нейронной сети 784 (Вход) -512 (Скрытый слой 1) -256 (Скрытый слой 2) -10 (Выход).

```
num_classes = y_train.shape[1]
num_features = X_train.shape[1]
num_output = y_train.shape[1]
num_layers_0 = 512; num_layers_1 = 256
```

В тензорном потоке определим заполнитель для входных и выходных переменных и любых переменных, которые необходимо отслеживать.

```
input_X = tf.placeholder('float32', shape
=(None,num_features), name="input_X")
input_y = tf.placeholder('float32', shape =
(None,num_classes), name='input_Y')
keep_prob = tf.placeholder(tf.float32)
```

tf.placeholder – это заполнитель, который передаётся в качестве параметра для установки модели. Заполнители в основном являются держателями для различных наборов данных.

Переменная keep_prob создана для dropout-слоя. Такой подход создан для уменьшения переобучения сети. Термин «dropout» (выбивание, выбрасывание) характеризует исключение определённого процента случайных нейронов на разных эпохах обучения нейронной сети. способ усреднения моделей внутри нейронной сети. В результате более обученные нейроны получают в сети больший вес [4]. Такой приём значительно увеличивает скорость обучения, качество обучения на тренировочных

данных, а также повышает качество предсказаний модели на новых тестовых данных.

Поскольку свёрточные слои требуют весов (weights) и смещений (bias), их необходимо инициализировать случайным нормальным распределением с нулевым средним и малой дисперсией (1/квадратный корень из числа объектов) [4].

```
weights_0    =    tf.Variable(tf.random_normal([num_features,
num_layers_0], stddev=(1/tf.sqrt(float(num_features)))))
bias_0 = tf.Variable(tf.random_normal([num_layers_0]))
weights_1    =    tf.Variable(tf.random_normal([num_layers_0,
num_layers_1], stddev=(1/tf.sqrt(float(num_layers_0)))))
bias_1 = tf.Variable(tf.random_normal([num_layers_1]))
weights_2    =    tf.Variable(tf.random_normal([num_layers_1,
num_output], stddev=(1/tf.sqrt(float(num_layers_1)))))
bias_2 = tf.Variable(tf.random_normal([num_output]))
```

Теперь необходимо описать расчет графика для разработки модели. Для начала умножим входные данные каждого слоя с соответствующими весами и добавим термин смещения. После весов и смещений добавим активацию. В данном случае целесообразно использовать активацию ReLU для скрытых слоев и softmax для окончательного выходного слоя, чтобы получить оценку вероятности класса.

Функция активации ReLu возвращает значение x, если x положительно, и 0 в противном случае. Она нелинейна, из-за чего комбинация ReLu также будет нелинейна. Это означает, что появляется возможность соединять несколько слоёв без необходимости хранения большого количества малозначащих нейронов.

Softmax обычно применяется к выходному слою задач множественной классификации. Она гарантирует, что сумма всех выходных нейронов равна 1, а значение интервала [0,1], соответствующего каждому выходу, является

вероятностью выхода, а наибольшая вероятность является ответом нейронной сети на новый входной поток.

Также для предотвращения переобучения; хорошей практикой будет добавление dropout-слоя после каждого скрытого слоя.

```
hidden_output_0 = tf.nn.relu(tf.matmul(input_X,
weights_0)+bias_0)
hidden_output_0_0 = tf.nn.dropout(hidden_output_0,
keep_prob)
hidden_output_1 = tf.nn.relu(tf.matmul(hidden_output_0_0,
weights_1)+bias_1)
hidden_output_1_1 = tf.nn.dropout(hidden_output_1,
keep_prob)
predicted_y = tf.sigmoid(tf.matmul(hidden_output_1_1,
weights_2) + bias_2)
```

Теперь необходимо определить функцию потерь для оптимизации весов и смещений. Для этого используем кросс-энтропию softmax с логитами для предсказанной и правильной метки.

```
loss=
tf.reduce_mean(tf.nn.softmax_cross_entropy_with_logits_v2(
logits=predicted_y,labels=input_y)) + regularizer_rate
*(tf.reduce_sum(tf.square(bias_0)) + tf.reduce_sum(
tf.square(bias_1)))
```

Эта функция вычисляет перекрестную энтропию результата после применения функции softmax.

На этом этапе модельная конструкция построена. Для оценки производительности модели определим метрику точности, поскольку функция потерь неинтуитивна.

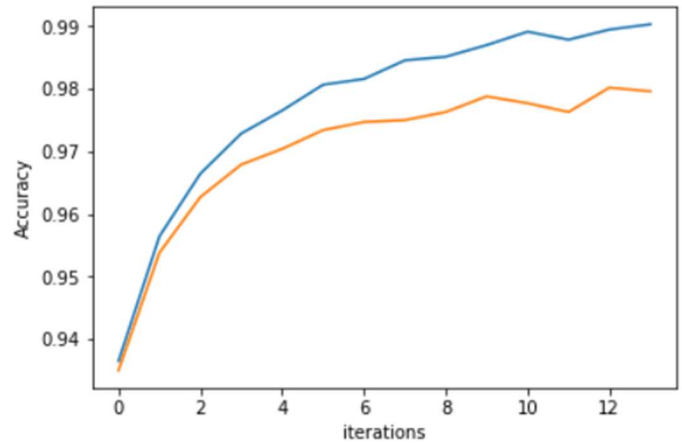
```
correct_prediction = tf.equal( tf.argmax(y_train,1),
tf.argmax(predicted_y,1))
accuracy = tf.reduce_mean( tf.cast(correct_prediction,
tf.float32))
```

Теперь, когда модель описана и есть возможность посчитать ее точность, начнём обучение модели на тестовых данных Fashion_mnist. Для этого зададим выборочное количество данных и количество эпох обучения.

```
batch_size = 128 epochs=14 dropout_prob = 0.6
training_accuracy = []
training_loss = []
testing_accuracy = []
s.run(tf.global_variables_initializer())
for epoch in range(epochs):
    arr = np.arange(X_train.shape[0])
    np.random.shuffle(arr)
    for index in range(0,X_train.shape[0],batch_size):
        s.run(optimizer, {input_X:
X_train[arr[index:index+batch_size]], input_y:
y_train[arr[index:index+batch_size]],
keep_prob:dropout_prob})
    training_accuracy.append(s.run(accuracy, feed_dict=
{input_X:X_train, input_y: y_train,keep_prob:1}))
    training_loss.append(s.run(loss, {input_X: X_train,
input_y: y_train,keep_prob:1}))
    testing_accuracy.append(accuracy_score(y_test.argmax(1),
s.run(predicted_y, {input_X: X_test,
keep_prob:1}).argmax(1)))
    print("Epoch:{0}, Train loss: {1:.2f} Train acc:
{2:.3f}, Test acc:{3:.3f}".format(epoch,
training_loss[epoch], training_accuracy[epoch],
testing_accuracy[epoch]))
```

```
Epoch:0, Train loss: 40.71 Train acc: 0.936, Test acc:0.935
Epoch:1, Train loss: 22.14 Train acc: 0.956, Test acc:0.954
Epoch:2, Train loss: 12.20 Train acc: 0.966, Test acc:0.963
Epoch:3, Train loss: 6.90 Train acc: 0.973, Test acc:0.968
Epoch:4, Train loss: 4.12 Train acc: 0.977, Test acc:0.970
Epoch:5, Train loss: 2.71 Train acc: 0.981, Test acc:0.973
Epoch:6, Train loss: 2.02 Train acc: 0.982, Test acc:0.975
Epoch:7, Train loss: 1.70 Train acc: 0.985, Test acc:0.975
Epoch:8, Train loss: 1.56 Train acc: 0.985, Test acc:0.976
Epoch:9, Train loss: 1.50 Train acc: 0.987, Test acc:0.979
Epoch:10, Train loss: 1.48 Train acc: 0.989, Test acc:0.978
Epoch:11, Train loss: 1.47 Train acc: 0.988, Test acc:0.976
Epoch:12, Train loss: 1.47 Train acc: 0.989, Test acc:0.980
Epoch:13, Train loss: 1.47 Train acc: 0.990, Test acc:0.980
```

Рисунок 3 – Результаты обучения




```
Train Accuracy: 0.99
Test Accuracy:0.98
```

Рисунок 4 – График зависимости точности модели от количества эпох

3. Распознавание объектов на изображении

Обнаружение объектов - обнаружение экземпляров определенного класса, таких как люди, автомобили или животные, на изображении или видео. Это может быть достигнуто путем изучения особенностей каждого объекта. У TensorFlow имеется множество необходимых для этого описанных функций и способов реализации. Одним из них является Tensorflow Object Detection API.

Tensorflow Object Detection API — это инфраструктура с открытым исходным кодом, которая позволяет использовать предварительно обнаруженные модели обнаружения объектов или создавать и обучать новые модели, используя переносное обучение. Это чрезвычайно полезно, потому что построение модели обнаружения объекта с нуля может быть сложным и может занять очень много времени и мощностей устройства для обучения.

Для того, чтобы создать нейросеть, способную выделять на изображениях определенные объекты, необходимо подготовить датасет для обучения. В нижеизложенном примере применения Tensorflow Object Detection API будут использоваться фотографии макарон с сыром (рис. 5).

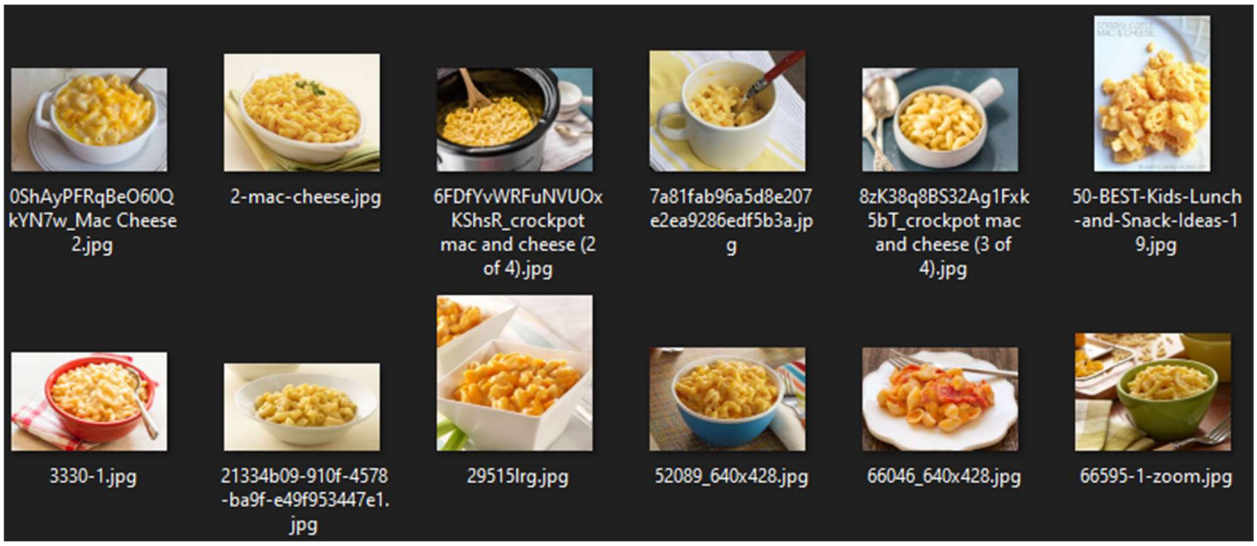


Рисунок 5 – Изображения, необходимые для обучения модели

Для Tensorflow Object Detection API необходимо показать, где именно на этих изображениях находится искомый объект для последующего обучения. Для этого нужно создать файл аннотаций, где будут описаны свойства объекта, а именно его местоположение относительно начала координат изображения.

```
<annotation>
     <folder>images</folder>
     <filename>0ShAyPFRqBeO60QkYN7w_Mac                Cheese
2.jpg</filename>
     <path>/home/paperspace/Desktop/images/0ShAyPFRqBeO60QkY
N7w_Mac Cheese 2.jpg</path>
     <source>
          <database>Unknown</database>
     </source>
     <size>
          <width>620</width>
          <height>414</height>
          <depth>3</depth>
     </size>
     <segmented>0</segmented>
     <object>
          <name>macncheese</name>
          <pose>Unspecified</pose>
          <truncated>0</truncated>
          <difficult>0</difficult>
          <bndbox>
               <xmin>155</xmin>
               <ymin>9</ymin>
               <xmax>565</xmax>
               <ymax>313</ymax>
          </bndbox>
     </object>
</annotation>
```

Это необходимо сделать со всеми изображениями, подготовленными как датасет для обучения и разделить его на два датасета: обучающий и тестовый.

Данный метод обучения нейронной сети требует создания файлов TFRecord. Это формат файлов, в которых хранится описания классов и объектов на изображениях, необходимых для обучения. Для их создания можно воспользоваться уже описанными методами, которые рекомендуются самими разработчиками TensorFlow. Эти методы описаны в python-модуле, который можно скачать с сайта TensorFlow.

После скачивания файла, можно приступить к созданию обучающих файлов TFRecord для обучающей и тестовой выборок.

```
python3 generate_tfrecord.py --
csv_input=data/train_labels.csv --
output_path=data/train.record --image_dir=images/

python3 generate_tfrecord.py --
csv_input=data/test_labels.csv --
output_path=data/test.record --image_dir=images/
```

После этого в директории появятся два файла train.record и test.record. На данном этапе необходимо настроить файл конфигурации и обучить модель для распознавания классов, которые были выделены на изображениях.

Для поиска объектов на изображениях есть возможность использовать предобученную модель COCO2017, которая содержит описание и набор слоёв нейронов для обнаружения более чем 80 классов различных объектов. Ее также необходимо скачать в директорию к проекту и видоизменить до пригодных значений и параметров.

В этот файл необходимо добавить один новый класс, созданный ранее.

```
item {
      id: 1
      name: 'macncheese'}
```

А именно макароны с сыром, которые были выделены на изображениях и сохранены в виде TFRecord файла ранее.

Далее можно приступить непосредственно к обучению модели. Для этого необходимо запустить несколько операций в командной строке.

```
python3 train.py --logtostderr --train_dir=training/ --
pipeline_config_path=training/ssd_mobilenet_v1_pets.config
```

Процесс обучения будет выглядеть следующим образом:

```
tensorflow:global step 11788: loss = 0.6717 (0.398 sec/step)
…
tensorflow:global step 11792: loss = 0.7164 (0.378 sec/step)
tensorflow:global step 11793: loss = 0.8096 (0.393 sec/step)
```

На данном этапе модель дообучена классом «Макароны с сыром». Так выглядит график уменьшения ошибки при обучении модели (рис. 6).

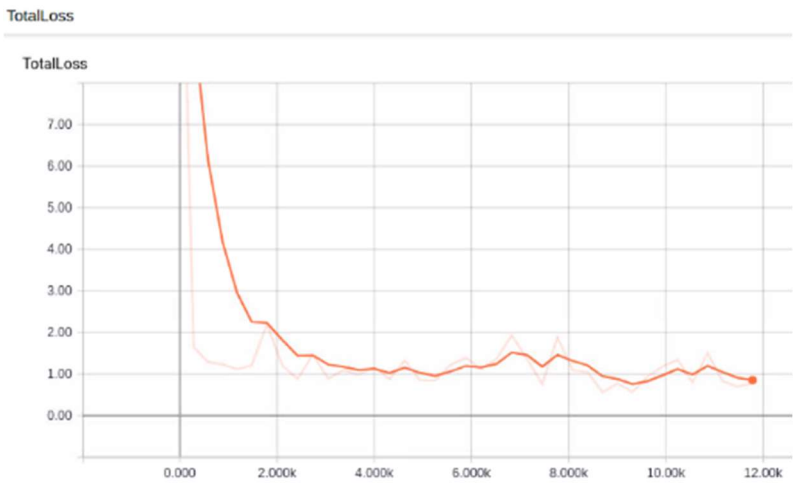


Рисунок 6 – График процесса обучения модели

Теперь можно проверить модель на новом изображении, на котором находится объект «Макароны с сыром». Для этого необходимо создать новый python-файл, в котором будет происходить загрузка обученной модели и попытка распознавания объекта.

```
MODEL_NAME = 'mac_n_cheese_inference_graph'
PATH_TO_CKPT = MODEL_NAME + '/frozen_inference_graph.pb'
PATH_TO_LABELS = os.path.join('training', 'object-
detection.pbtxt')
NUM_CLASSES = 1
TEST_IMAGE_PATHS = [ os.path.join(PATH_TO_TEST_IMAGES_DIR,
'image{}.jpg'.format(i)) for i in range(2, 8) ]
```

Рисунок 7 – Результат обучения модели на новых изображениях

Модель дообучена и готова распознавать новый для нее класс «Макароны с сыром».

Заключение

В данной работе были рассмотрены проблемы классификации и распознавания объектов на изображениях.

Библиотека-фреймворк TensorFlow, разработанная компанией Google, содержит большое количество методов, которые позволяют выполнить разработку моделей машинного обучения, нейронных сетей или компьютерного зрения любой сложности и структуры. Есть возможности описать собственную модель «вручную», сконфигурировав группы слоёв, связывающие их конструкции, функции активаций и т.д. Так же существуют более быстрые способы через встроенные конструкции и ранее описанный код другими программистами, например, через TFRecords.

Большое количество задокументированной информации по библиотеке, обучающих роликов от самой компании Google позволяют описать собственную модель, не сильно вдаваясь в подробности внутреннего устройства нейронных сетей, что значительно повышает скорость разработки и уменьшает количество возможных ошибок на этапе обучения, либо на этапе использования обученной модели.

В ходе выполнения научно-исследовательской работы были достигнуты цели, поставленные на этапе составления ТЗ:

- разобраны методы для классификации изображений;
- разобраны методы для выделения объектов на изображении.

Таким образом, можно сделать вывод о том, что цель работы выполнена, все поставленные задачи реализованы в полном виде.

Список использованных источников

1. Документация TensorFlow Python [электронный ресурс]. Режим доступа: https://www.tensorflow.org/api_docs/python/tf/all_symbols (дата обращения: 4.10.21);

2. Галушкин, А.И. Синтез многослойных систем распознавания образов. — М.: «Энергия», 1974. – 694с.

3. Dropout: A Simple Way to Prevent Neural Networks from Overfitting [электронный ресурс]. Режим доступа: https://jmlr.org/papers/v15/srivastava14a.html (дата обращения: 3.10.2021).

4. Romanuke, Vadim. Appropriate number and allocation of ReLUs in convolutional neural network (англ.) // Research Bulletin of NTUU “Kyiv Polytechnic Institute”: journal, 2017.