УДК 004.032.26:004.8

РАЗРАБОТКА СВЕРТОЧНОЙ НЕЙРОННОЙ СЕТИ ДЛЯ РАСПОЗНАВАНИЯ РУКОПИСНЫХ СИМВОЛОВ

Маркин Е.И., Подопригора И.А., Бершадская Е.Г.

ФГБОУ ВО «Пензенский государственный технологический университет», Пенза, e-mail: evgeniymarkin1@gmail.com

В данной статье рассказывается о применении сверточной нейронной сети при решении задачи распознавания изображений. Приводится пример реализации модели сверточной нейронной сети, на технологии TensorFlow, для обучения на наборе данных MNIST с целью распознавания рукописных цифр. MNIST набор данных является свободно распространяемым и содержит в себе 60 000 учебных примеров и 10 000 тестовых примеров рукописных цифр от 0 до 9, которые представлены в виде монохромных изображений размером 28x28 пикселей. Предложенная модель после обучения имеет точность равную 97,3 %, что говорит о высокой эффективности данной модели в поставленной задаче. Такая точно была достигнута за счет использования двух уровней сверточных слоев, с использованием функции активации ReLU, и двух слоев объединения. Данная сверточная нейронная сеть является очень простой в реализации, за счет использования высокоуровневого API библиотеки TensorFlow. TensorFlow содержит в себе методы tf.layers, conv2d() и dense(), которые позволяют реализовывать сверточные слои для нейронных сетей, проводить их объединение по заданным методам и создавать плотные (логические) слои. Полученная модель может успешно использоваться для решения задач классификации изображений и распознавания образов.

Ключевые слова: сверточные нейронные сети, машинное обучение, TensorFlow, MNIST, распознавание образов, компьютерное зрение

DEVELOPMENT OF A CONVENTIONAL NEURAL NETWORK FOR RECOGNITION OF MANUSCRIPT SYMBOLS

Markin E.I., Podoprigora I.A., Bershadskaya E.G.

Federal State Budget Educational Institution of Higher Education Penza State Technological University, Penza, e-mail: evgeniymarkin1@gmail.com

This article describes the use of a convolutional neural network in solving the problem of image recognition. An example of the implementation of the convolutional neural network model, on TensorFlow technology, for learning on the MNIST data set for the purpose of recognition of handwritten figures is given. MNIST data set is freely distributed and contains 60 000 training examples and 10 000 test examples of handwritten figures from 0 to 9, which are presented in the form of monochrome images with a size of 28x28 pixels. The proposed model after training has an accuracy of 97.3 %, which indicates the high efficiency of this model, in the task. This was accurately achieved by using two levels of convolutional layers, using the ReLU activation function, and two layers of union. This convolutional neural network is very simple to implement, due to the use of the high-level API of the TensorFlow library. TensorFlow contains the methods tf.layers, conv2d() and dense (), which allow you to implement convolutional layers for neural networks, combine them with specified methods, and create dense (logical) layers. The resulting model can be successfully used to solve problems of image classification and pattern recognition.

Keywords: convolutional neural networks, machine learning, TensorFlow, MNIST, pattern recognition, computer vision

Человеческий мозг получает большое количество информации при помощи различных образов, для нас нет никакой сложности в том, чтобы узнавать людей по лицам, описывать различных животных и явления. Данная задача может казаться очень легкой для человеческого мозга, но является очень сложной в реализации при помощи компьютера.

За последние несколько лет машинное обучение добилось огромного прогресса в решении данной весьма сложной проблемы. В частности, было обнаружено, что нейронная модель, называемая глубокой сверточной нейронной сетью (*CNN*), может достичь разумной производительности по задачам жесткого визуального распознавания изображения и их классификации [1].

Сверточные нейронные сети (CNN) представляют собой современную модельную архитектуру для задач классификации изображений. CNN применяют серию фильтров к необработанным пиксельным данным изображения для извлечения и изучения функций более высокого уровня, модель которой затем может использоваться для классификации. CNN содержит три компонента [2]:

- Сверточные слои, которые применяют к изображению определенное количество фильтров свертки. Для каждой подобласти слой выполняет набор математических операций для создания единственного значения на карте характеристик вывода. Затем сверточные слои обычно применяют функцию активации к выходу для введения нелинейности в модель.

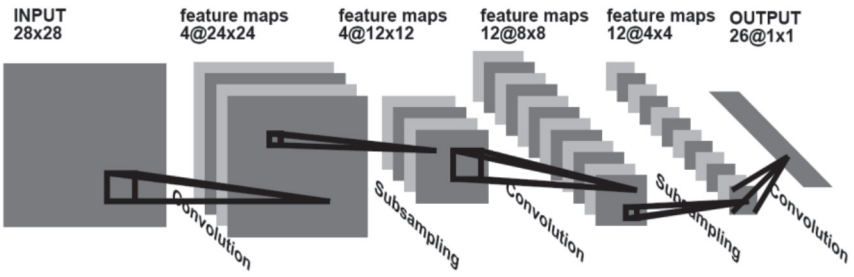

Рис. 1. Схема сверточной нейронной сети

● Объединение слоев, которые уменьшают данные изображения, извлеченные сверточными слоями, для уменьшения размерности карты признаков, чтобы уменьшить время обработки. Обычно используемым алгоритмом объединения является максимальное объединение, которое извлекает субрегионы карты функций (например, 2x2-пиксельные плитки), сохраняет их максимальное значение и отбрасывает все другие значения.

● Плотные (полностью связанные) слои, которые выполняют классификацию на свойствах, извлеченных сверточными слоями и уменьшающихся по слоям объединения. В плотном слое каждый узел в слое связан с каждым узлом предыдущего слоя.

Как правило, CNN состоит из стека сверточных модулей, которые выполняют извлечение признаков. Каждый модуль состоит из сверточного слоя, за которым следует слой объединения. За последним сверточным модулем следует один или несколько плотных слоев, которые выполняют классификацию (рис. 1). Последний плотный слой в CNN содержит единственный узел для каждого целевого класса в модели с функцией активации для создания значения между 0–1 для каждого узла. Можно интерпретировать значения функции активации для данного изображения как относительные измерения того, насколько вероятно, что изображение попадает в каждый целевой класс.

TensorFlow, библиотека машинного обучения, разработанная компанией Google, предоставляет модуль *layers*, обеспечивающий высокий уровень API, который позволяет легко построить нейронную сеть. Он предоставляет методы, облегчающие создание плотных слоев и сверточных слоев, добавление функций активации и т.д. [3].

Для построения модели классификации изображений в наборе данных MNIST [4], набор данных содержащий 60 000 учебных примеров и 10 000 тестовых примеров рукописных цифр 0–9, представленные как монохромные изображения 28x28 пикселей, с использованием технологии TensorFlow, воспроизведем следующую архитектуру CNN [5]:

● Сверточный слой № 1: применяет 32 фильтра 5x5 (извлекает 5x5-пиксельные субрегионы), с функцией активации ReLU.

● Объединение слоев № 1: выполняет максимальное объединение с фильтром 2x2 и шагом 2.

● Сверточный слой № 2: применяет 64 фильтра 5x5, с функцией активации ReLU.

● Объединение слоев № 2: опять же выполняется максимальное объединение с фильтром 2x2 и шагом 2.

● Плотный слой № 1: 1024 нейронов, с коэффициентом регуляризации отсева 0,4.

● Плотный слой № 2 (Логический слой): 10 нейронов, по одному для каждого целевого класса цифр.

Для реализации данных слоев TensorFlow предоставляет следующие модули:

tf.layers – модуль содержит методы для создания каждого из трех типов слоев выше:

conv2d() – создает двумерный сверточный слой. Принимает количество фильтров, размер ядра фильтра, дополнение и функцию активации в качестве аргументов.

max_pooling2d() – создает двумерный пул, используя алгоритм максимального объединения. Принимает размер фильтра фильтрации и шагает в качестве аргументов.

dense() – создает плотный слой. Принимает количество нейронов и функцию активации в качестве аргументов.

Каждый из этих методов принимает тензор в качестве входных данных и возвращает преобразованный тензор в качестве выхода. Это упрощает подключение одного слоя к другому.

Методы в *layers* модуле для создания сверточных и объединенных слоев данных двумерного изображения предполагают, что входные тензоры имеют форму, определяемую следующим образом: *[batch_size, image_width, image_height, channels*], где

– *batch_size*, размер подмножества примеров для использования при выполнении градиентного спуска во время обучения;

– *image_width*, ширина примерных изображений;

– *image_height*, высота примерных изображений;

– *channels*, количество цветных каналов в примерах изображений. Для цветных изображений количество каналов равно 3 (красный, зеленый, синий). Для монохромных изображений имеется всего 1 канал (черный).

Набор данных MNIST состоит из монохромных изображений размером 28x28 пикселей, поэтому желаемая форма для входного слоя: *[batch_size, 28, 28, 1]*.

Чтобы преобразовать карту функций ввода в эту форму, достаточно выполнить следующую *reshape* операцию:

input_layer = tf.reshape(features["x"], [–1, 28, 28, 1]).

В качестве размера пакета указывается –1, это указывает, что измерение должно динамически вычисляться на основе количества входных значений *features["x"]*. Это позволяет рассматривать *batch_size* как параметр, который можно настроить.

В первом сверточном слое нужно применить 32 фильтра 5x5 к входному слою с функцией активации ReLU. Можно использовать *conv2d()* метод в *layers* модуле для создания этого слоя:

conv1 = tf.layers.conv2d(inputs = input_layer, filters = 32, kernel_size = [5, 5], padding = «same», activation = tf.nn.relu)

Аргумент *filters* определяет тензор входного сигнала, который должен иметь форму *[batch_size, image_width, image_height, channels]*. Здесь объединяется первый сверточный слой с *input_layer*, который имеет вид *[batch_size, 28, 28, 1]*.

Аргумент *filters* задает количество применяемых фильтров, и *kernel_size* определяет размеры фильтров *[width, height]*.

Аргумент *padding* задает один из двух перечисленных значений *valid* или *same*. Чтобы указать, что выходной тензор должен иметь те же самые значения ширины и высоты, что и входной тензор, указывается *padding = same*, который сообщает TensorFlow добавлять 0 значений к краям входного тензора для сохранения ширины и высоты 28.

Аргумент activation определяет функцию активации, чтобы применить к выходу свертки. Здесь указывается активация ReLU с *tf.nn.relu*.

Полученный тензор *conv2d()* имеет вид *[batch_size, 28, 28, 32]*, те же размеры по ширине и высоте, что и входной, но теперь с 32 каналами, содержащий выходной сигнал от каждого из фильтров.

Далее соединяются первый слой объединения со сверточным слоем, который только что был создан. Для этого можно использовать *max_pooling2d()* метод *layers* для построения слоя, который выполняет максимальное объединение с фильтром 2x2 и шагом 2:

pool1 = tf.layers.max_pooling2d(inputs = = conv1, pool_size = [2, 2], strides = 2).

Опять же в качестве аргумента inputs задается входной тензор с формой *[batch_size, image_width, image_height, channels]*. Здесь входной тензор – это выход из первого сверточного слоя, который имеет форму *conv1[batch_size, 28, 28, 32]*.

Аргумент *pool_size* задает размер максимального объединяющего фильтра *[width, height]*. Если оба измерения имеют одинаковое значение, можно указать одно целое число.

Аргумент *strides* задает размер шага. Здесь устанавливается шаг 2, что указывает на то, что субрегионы, выделенные фильтром, должны быть разделены на 2 пикселя как по ширине, так и по высоте. Если необходимо установить разные значения шага для ширины и высоты, тогда нужно вместо этого указать кортеж или список (например, *stride = [3, 6]*).

Выходной тензор, создаваемый *max_pooling2d()*, имеет вид: *[batch_size, 14, 14, 32]* фильтр 2x2 уменьшает ширину и высоту на 50%.

Можно подключить второй сверточный и объединяющий слои к CNN, используя *conv2d()* и *max_pooling2d()* как ранее. Для сверточного слоя № 2 настраивается 64 фильтра 5x5 с активацией ReLU, а для слоя объединения 2 используются те же параметры, что и для уровня объединения № 1.

Сверточный слой № 2 будет принимать выходной тензор первого объединенного слоя в качестве входного сигнала и выдает тензор в *conv2* качестве выхода. *conv2* имеет вид *[batch_size, 14, 14, 64]* и ту же ширину и высоту, что и *pool1*, и 64 канала для применяемых 64 фильтров.

Слой объединения № 2 берет conv2 в качестве входных данных, создавая *pool2* как выход. *pool2* имеет форму *[batch_size, 7, 7, 64]*.

После нужно добавить плотный слой в CNN для выполнения классификации по свойствам, выделенным слоями свертки/объединения. Однако, перед тем как подключить слой, сначала нужно привести карту свойств к виду *[batch_size, features]*, чтобы тензор имел только два измерения:

*pool2_flat = tf.reshape(pool2, [–1, 7 * 7 * 64]).*

В *reshape()* функции «–1» означает, что размеры *batch_size* будут динамически вычисляться на основе количества примеров входных данных. Каждый пример имеет 7 (ширина *pool2*) * 7 (высота *pool2*) * 64 (каналы *pool2*), поэтому, чтобы *features* размер имел значение 7 * 7 * 64 (всего 3136). Выходной тензор *pool2_flat*, будет иметь форму [*batch_size, 3136]*.

Теперь можно использовать метод *dense()* для соединения плотного слоя:

dense = tf.layers.dense(inputs = pool2_flat, units = 1024, activation = tf.nn.relu).

Аргумент *inputs* определяет тензор входа: сглаженную карту функции, *pool2_flat*. Аргумент *units* указывает количество нейронов в плотном слое. Аргумент *activation* принимает функцию активации; можно снова использовать *tf.nn.relu* для добавления активации ReLU.

Чтобы улучшить результаты модели, можно также воспользоваться нормализацией выпадения плотного слоя, используя *dropout* метод в *layers*:

dropout = tf.layers.dropout(inputs = = dense, rate = 0.4, training = mode = = tf.estimator.ModeKeys.TRAIN).

Опять же *inputs* определяет входной тензор, являющийся выходным тензором из плотного слоя.

Аргумент *rate* определяет отсев; здесь используется 0,4, что означает, что 40 % элементов будут случайно выпадать во время обучения.

Аргумент *training* принимает логическое значение, определяющее, работает ли модель в настоящее время в режиме обучения; выпадение будет выполняться только, если *training* равен *True*. Здесь проверяется, является ли переданная функция *mode* модели *cnn_model_fn* в *TRAIN* режиме.

Итоговый тензор *dropout* имеет форму *[batch_size, 1024]*.

Последний слой в данной нейронной сети – это логический уровень, который возвращает исходные значения для прогнозов. Создается плотный слой с 10 нейронами (по одному для каждого целевого класса 0–9) с линейной активацией:

logits = tf.layers.dense(inputs = = dropout, units = 10).

Конечный выходной тензор CNN *logits* имеет форму *[batch_size, 10]*.

Логический уровень модели возвращает прогнозы как необработанные значения в *[batch_size, 10]*-мерном тензоре

Для обучения и оценки необходимо определить функцию потерь, которая измеряет, насколько точно предсказания модели соответствуют целевым классам. Для задач многоклассовой классификации, таких как MNIST, кросс-энтропия обычно используется как метрика потерь:

```
onehot_labels = tf.one_hot(indices = tf.cast(labels, tf.int32), depth = 10)
loss = tf.losses.softmax_cross_entropy(onehot_labels = onehot_labels, logits = logits)
```

Затем создается оценщик *Estimator* для модели:

```
mnist_classifier = tf.estimator.Estimator(
   model_fn = cnn_model_fn, model_dir = "/tmp/mnist_convnet_model")
```

Аргумент *model_fn* определяет функцию модели, использующую для обучения, оценки и прогнозирования; передается модель *cnn_model_fn*, которая была создана ранее. Аргумент *model_dir* указывает каталог, в котором будет сохранена модель данных.

Обучить модель можно путем создания *train_input_fn* вызова *train()* на *mnist_classifier*:

```
train_input_fn = tf.estimator.inputs.numpy_input_fn(x = {"x": train_data}, y = train_labels, batch_size = 100, num_epochs = None, shuffle = True)
mnist_classifier.train(input_fn = train_input_fn, steps = 20000, hooks = [logging_hook])
```

В *numpy_input_fn* вызове передаются данные и метки функций обучения *x* и *y* соответственно. *batch_size* указывается 100, что означает, что модель будет обучаться на *minibatches* из 100 примеров на каждом шаге. *num_epochs = None* означает, что модель будет тренироваться до тех пор, пока не будет достигнуто указанное количество шагов. В *train* вызове устанавливается *steps = 20000*, что означает, что модель будет тренироваться на 20 000 шагов.

По завершении обучения можно оценить модель, чтобы определить ее точность в тестовом наборе MNIST. Для этого вызывается *evaluate* метод, который оценивает показатели, указанные в *eval_metric_ops* аргументе в аргументе *model_fn*:

```
eval_input_fn = tf.estimator.inputs.numpy_input_fn(x = {"x":  eval_data},  y = eval_labels, num_epochs = 1, shuffle = False)
eval_results = mnist_classifier.evaluate(input_fn = eval_input_fn)
print(eval_results)
```

```
INFO:tensorflow:loss = 2.36026, step = 1
INFO:tensorflow:probabilities = [[ 0.07722801  0.08618255  0.09256398, ...]]
...
INFO:tensorflow:loss = 2.13119, step = 101
INFO:tensorflow:global_step/sec: 5.44132
...
INFO:tensorflow:Loss for final step: 0.553216.

INFO:tensorflow:Restored model from /tmp/mnist_convnet_model
INFO:tensorflow:Eval steps [0,inf) for training step 20000.
INFO:tensorflow:Input iterator is exhausted.
INFO:tensorflow:Saving evaluation summary for step 20000: accuracy = 0.9733, loss = 0.0902271
{'loss': 0.090227105, 'global_step': 20000, 'accuracy': 0.97329998}
```

Рис. 2. Вывод содержания журнала обучения

Для создания *eval_input_fn* устанавливается *num_epochs = 1* так, чтобы модель оценивала метрику за одно поколение данных и возвращала результат. Также устанавливается *shuffle = False* последовательно итерацию данных.

Полученную функцию модели CNN *Estimator* и логику обучения/оценки можно проверить в действии.

По мере создания модели будет виден вывод журнала, который выглядит так, как представлено на рис. 2.

Данная модель достигает точности 97,3 % в данном тестовом наборе данных.

Список литературы

1. Ясницкий Л.Н. Введение в искусственный интеллект / Л.Н. Ясницкий. – М.: Издательский центр «Академия», 2010. – 176 с.

2. Goodfellow I. Deep Learning (Adaptive Computation and Machine Learning series) / I. Goodfellow, Y. Bengio, A. Courville. – Cambridge: The MIT Press, 2016. – 775 с.

3. McClure N. TensorFlow Machine Learning Cookbook / N. McClure – Birmingham: Packt Publishing, 2017. – 370 с.

4. Образцы рукописных символов MNIST [Электронный ресурс]. – Режим доступа: http://yann.lecun.com/exdb/mnist (дата обращения: 13.02.2018).

5. Hope T. Learning TensorFlow: A Guide to Building Deep Learning Systems / T. Hope, I. Lieder – Sebastopol: O'Reilly Media, 2017. – 242 с.