Дата поступления рукописи в редакцию: 07.06.2026 г.
Дата принятия рукописи в печать: 29.06.2026 г.
DOI: 10.24412/2076-1503-2026-6-453-468

СЕМЕНОВ Александр Александрович,
научный сотрудник ФКУ НПО «СТиС» МВД России,
e-mail: mail@law-books.ru

АВЕТИСЯН Карэн Рафаелович,
Старший преподаватель кафедры информационных технологий и организации расследования киберпреступлений Московская академия Следственного комитета Российской Федерации имени А.Я. Сухарева,
e-mail: Karen-Avetisyan-1989@bk.ru

МАТЕМАТИЧЕСКИЕ ОСНОВЫ И АРХИТЕКТУРЫ СВЕРТОЧНЫХ НЕЙРОСЕТЕЙ ДЛЯ ОБРАБОТКИ ИЗОБРАЖЕНИЙ

Аннотация. *В работе рассматриваются принципы функционирования сверточных нейросетей для обработки изображений, основанные на интерпретации скалярного произведения как меры сходства векторов. Описываются ключевые компоненты архитектуры: сверточные слои, фильтры, карты признаков, функции активации, пулинг и механизмы обучения. Анализируются современные подходы к построению архитектур нейросетей, включая оптимизацию вычислений и глубины моделей. Также рассматриваются основные задачи компьютерного зрения, такие как классификация, детекция и семантическая сегментация, и способы их решения с использованием сверточных нейросетей.*

Ключевые слова: *сверточные нейросети, обработка изображений, скалярное произведение, карта признаков, фильтр, свертка, функция активации, ReLU, пулинг, архитектура нейросети, машинное обучение, компьютерное зрение, детекция объектов, семантическая сегментация, генеративные нейросети, глубинное обучение.*

SEMENOV Aleksandr Aleksandrovich,
Chief Researcher, Federal State Budgetary Institution "STIS" of the Ministry of Internal Affairs of the Russian Federation

AVETISYAN Karen Rafaelovich,
Senior Lecturer, Department of Information Technology and Cybercrime Investigation Organization Moscow Academy of the Investigative Committee of the Russian Federation named after A. Ya. Sukharev

MATHEMATICAL FOUNDATIONS AND ARCHITECTURES OF CONVOLUTIONAL NEURAL NETWORKS FOR IMAGE PROCESSING

Annotation. *This paper examines the operating principles of convolutional neural networks for image processing, based on the interpretation of the dot product as a measure of vector similarity. Key components of the architecture are described: convolutional layers, filters, feature maps, activation functions, pooling, and learning mechanisms. Modern approaches to constructing neural network architectures are analyzed, including optimization of computation and model depth. Key computer vision tasks, such as classification, detection, and semantic segmentation, and methods for solving them using convolutional neural networks are also discussed.*

Key words: *convolutional neural networks, image processing, dot product, feature map, filter, convolution, activation function, ReLU, pooling, neural network architecture, machine learning, computer vision, object detection, semantic segmentation, generative neural networks, deep learning.*

Разберем, как устроены нейросети, применяемые для обработки изображений. Вспомним понятие скалярного произведения векторов и длины вектора (*рисунок 1*).

$$x = (x_1, x_2), y = (y_1, y_2),$$
$$x \cdot y = \sum x_i \cdot y_i = x_1 \cdot y_1 + x_2 \cdot y_2, |x| = \sqrt{x_1^2 + x_2^2}$$

Рисунок 1. Скалярное произведение векторов и длины вектора.

Обратим внимание, что длина вектора вычисляется как корень из скалярного произведения вектора с самим собой. Вспомним теперь, что для двух векторов и угла α между ними со школы известно следующее соотношение (*рисунок 2*).

$$x \cdot y = |x| \cdot |y| \cdot cos(\alpha)$$

Рисунок 2. Соотношение для двух векторов и угла α между ними.

Забудем на некоторое время про любые векторы, кроме тех, длина которых равна 1. Из формулы, представленной на рисунке 2 вытекает, что для таких векторов косинус угла между ними в точности равен скалярному произведению. Задумаемся теперь над тем, что сонаправленность двух векторов можно интерпретировать как похожесть. Получается, что скалярное произведение векторов с единичной длиной является мерой их похожести: для почти совпадающих векторов оно близко к 1, для противоположно направленных – к −1, а для перпендикулярных близко к 0 (*рисунок 3*). Посчитав скалярное произведение двух векторов, мы узнаем степень их похожести.

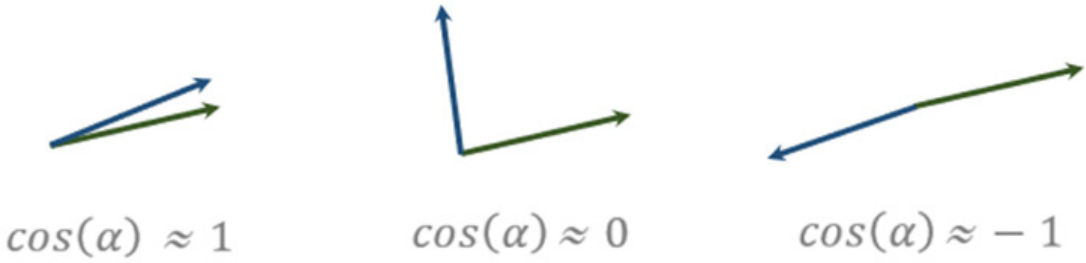

Рисунок 3. Скалярное произведение векторов с единичной длиной.

Заметим, что если применить указанные рассуждения к трехмерным, а не двумерным векторам, то они останутся в силе. Осознаем теперь, что вектор – это просто массив чисел, ему необязательно придавать геометрическую интерпретацию. Определение скалярного произведения (*рисунок 1*) не зависит от числа элементов вектора. Многомерное пространство может быть сложно себе представить, но в нем скалярное произведение векторов все также будет оставаться мерой похожести двух векторов вне зависимости от того, какая логическая сущность выражается этими векторами.

Представим себе теперь, что мы рассматриваем некоторое изображение на мониторе, и введем вектор, который будет содержать значение для красного, зеленого и синего цветов каждого из девяти пикселей в произвольном квадрате 3×3. Такой вектор будет состоять из 27 значений (*рисунок 4*).

$$x = (x_{11R}, x_{11G}, x_{11B}, x_{12R}, \ldots, x_{33B})$$

Рисунок 4. Вектор изображения.

Возьмем два таких вектора в произвольных точках монитора и посчитаем их скалярное произведение. Поскольку оно является мерой похожести, то по значению скалярного произведения можно будет судить о том, насколько похожи фрагменты изображения в двух выбранных точках. Любой такой вектор можно сохранить и использовать в качестве некоторого шаблона для сравнения с ним любых точек на изображении. Даже если мы теперь разрешим векторам иметь длину, отличную от 1, то значение скалярного произведение произвольного вектора с шаблоном все еще можно будет интерпретировать, как степень выраженности признака, заложенного в шаблоне, в произвольном векторе: чем больше значение скалярного произведения по модулю, тем ярче выражен признак, при этом знак скалярного произведения будет указывать на то, выражен ли сам признак или его противоположность.

Положим теперь, что нам нужно создать алгоритм, который будет находить на изображении красные, зеленые и синие круги. На *рисунке 5* слева приведен пример такого изображения. Возьмем теперь шесть векторов, изображенных на *рисунке 5* справа, каждый из них содержит фрагмент, характерный для изображения красного кружка. Если взять каждый из этих векторов и посчитать его скалярное произведение в точках изображения, то значения скалярного произведения будут больше всего именно в тех местах, в которых изображение на мониторе совпадает с шаблоном.

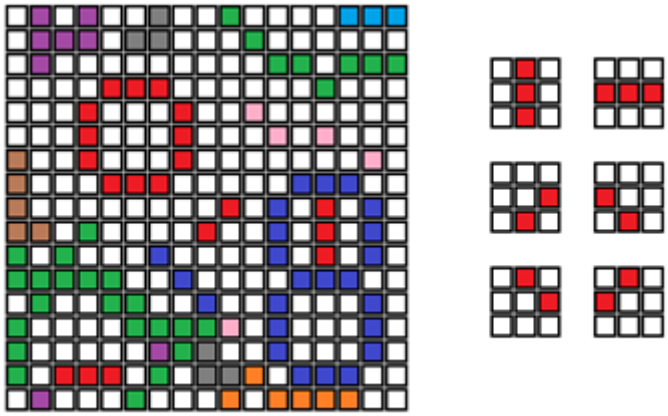

Рисунок 5. Пример изображения для алгоритма

Поступив аналогично с остальными шаблонами, можно собрать информацию о сходстве исходного изображения с разными шаблонами и собрать из этой информации новое изображение меньшего размера, в котором вместе цветов будут использоваться значения выраженности признаков каждого из четырех шаблонов. На *рисунке 6* слева изображен получающийся результат в случае, если векторы-шаблоны прикладывались не в каждом возможном пикселе, а с шагом в два пикселя. При этом значения выраженности признаков проиллюстрированы соответствующими значками.

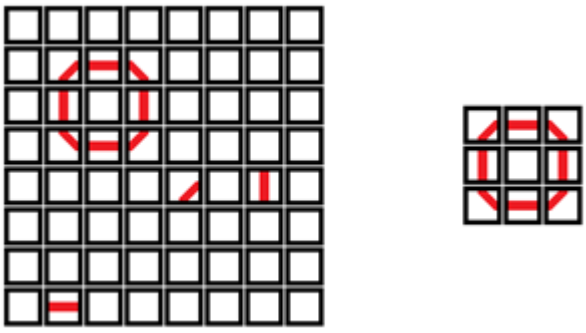

Рисунок 6. Векторы-шаблоны прикладывались не в каждом возможном пикселе, а с шагом в два пикселя

Представим себе теперь, что полученный результат мы подвергнем аналогичной процедуре, но с шаблоном, изображенным на рисунке 6 справа. При вычислении скалярного произведения этого шаблона с фрагментами изображения на *рисунке 6* слева будет обнаружена точка на этом изображении и, следовательно, на исходном, в которой искомый признак проявляется больше всего, что и даст решение исходной задачи для красных кружков. Для синих и зеленых она решается аналогично путем добавления соответствующих шаблонов.

На самом деле именно таким образом и работают сверточные нейросети, обрабатывающие изображения. Выше была описана нейросеть из двух сверточных слоев.

Сверточный слой/свертка/convolution – преобразование изображения, при котором выполняется скалярное произведение фрагментов изображения с набором шаблонов, при этом из результатов вычисления составляется новое изображение. В математике у термина «свертка» есть сразу два значения, от которых одновременно произошло название сверточного слоя, хотя в строгом смысле оно ни одному из них не соответствует.

Сверточная нейросеть – нейросеть, среди слоев которой преобладают сверточные.

Карта признаков/feature map – промежуточное изображение, возникающее в результате вычислений сверточного слоя, геометрически ассоциированное с исходным изображением, но содержащее вместо значений цветов значения выраженности некоторых признаков. Для общности исходное изображение иногда называют входной картой признаков.

Фильтр – вектор-шаблон, с которым в сверточном слое сверяются фрагменты изображения.

Канал /channel – совокупность значений одного признака в карте признаков.

Глубина карты признаков – число каналов карты признаков.

В приведенном выше примере распознавание красного кружка удалось сделать с помощью двух слоев, но более сложные объекты на больших изображениях потребуют определение гораздо большего числа слоев и различных признаков на них. Придумать в уме эти признаки для каждого слоя, которые бы подошли для решения серьезной задачи, весьма затруднительно. Именно поэтому сверточные нейросети стали применять именно с развитием машинного обучения.

При машинном обучении сверточных нейросетей весами являются значения фильтров.

Практика показала, что для устойчивости машинного обучения целесообразно к фильтрам добавить еще дополнительные веса, по одному на каждый канал.

Биас /bias – дополнительные веса сверточного слоя. Каждому фильтру соответствует один биас, который прибавляется к результату любого скалярного произведения с этим фильтром.

Фильтры необязательно имеют форму 3×3, они могут иметь также и другие размеры, одинаковые для всех фильтров одного слоя.

Ядро свертки/kernel – умозрительная рамка, передвигаемая по изображению при выполнении вычислений сверточного слояс выбранным фильтром.

Ширина/высота ядра свертки – ширина/высота фильтров в количестве пикселей. Ширина и высота в общем случае могут быть разные.

В современных сверточных нейросетях используются почти исключительно ядра 1×1 и 3×3, а также в качестве исключения 7×7 для первого сверточного слоя. Ядро свертки можно применять к исходному изображению в каждой точке, а можно, как в примере 5.5–5.6 не к каждой.

Шаг свертки/страйд/stride – шаг (в количестве пикселей), с которым ядро свертки передвигается по входному изображению. Горизонтальный и вертикальный страйды в общем случае могут отличаться.

В современной практике почти всегда используются страйд, равный 1, для всех сверток, кроме входной, для которой используется равный 2. Иногда страйд, равный 2, используется в некоторых промежуточных сверточных слоях. Видно, что при использовании страйда, отличного от 1, ширина и высота выходной карты признаков будут почти кратно отличаться от ширины и высоты входной карты признаков слоя. При этом, даже если использовать единичный страйд, то для сверток, отличных от 1×1, ширина и высота выходной карты признаков все равно будут получаться на несколько пикселей меньше, чем для входной. Такое явление приводит к неудобствам, во избежание которых принято для сверток с ядром, большим 1×1, и единичным шагом мысленно дорисовывать по краям входной карты признаков нулевые пиксели так, чтобы ширины и высоты входной и выходной карт признаков соответственно совпадали.

Паддинг /padding – мысленное дорисовывание к входной карте признаков нулей для подгона размеров выходной карты признаков слоя. Паддинг может быть разный с четырех сторон.

Паддинг является наиболее «подлым» параметром сверточного слоя, поскольку в некоторых ситуациях дорисовывать нулевые пиксели можно с разных сторон по разным соображениям, которые почти всегда опускаются в научных статьях как несущественные. Вследствие этого различные проблемы совместимости, связанные со сверточными нейросетями, очень часто возникают именно из-за разных трактовок этого параметра.

Все традиционные параметры сверточного слоя мы разобрали. На *рисунке 7* приведен псевдокод, реализующий вычисления сверточного слоя с шириной, высотой и глубиной входной карты признаков равными 99, 99 и 40 соответственно, 20 фильтрами с ядром 3×3, двойным страйдом по высоте и ширине и единичным паддингом со всех краев. Можно вычислить, что ширина и высота выходной карты признаков для такого слоя будут равны 50.

```
for                                                    (y=0;y<50;y++)
 for                                                   (x=0;x<50;x++)
 for                                                   (f=0;f<20;f++){
 OUT[y][x][f] = BIAS[f];
 for (ry=-1;ry<2;ry++) if (2*y+ry>-1 && 2*y+ry<99)
 for (rx=-1;rx<2;rx++) if (2*x+rx>-1 && 2*x+rx<99)
 for (l=0;l<40;l++)
 OUT[y][x][f]+=IN[2*y+ry][2*x+rx][l]*FILTERS[ry][rx][f][l]}
```

Рисунок 7. Псевдокод реализующий вычисления сверточного слоя

Сверточный слой реализует линейное преобразование, поэтому для определения того, как выглядят вычисления обратного хода сверточного слоя, необходимо проследить, какие значения зависят от каких в прямом ходе. Это не всегда просто, особенно с экзотическими значениями паддингов, страйдов и т. д. Например, при использовании страйда, равного 2, некоторые пиксели входной карты признаков используются при вычислении четырех пикселей выходной карты признаков, некоторые – при вычислении двух, а некоторые – при вычислении только одного (в соответствии с тем, сколько раз ядро свертки попадает на пиксель, как показано на рисунок 8). Чтобы не ошибиться, нужно всегда держать в голове принцип: если некоторое выходное значение функции зависит от некоторого входного, то соответствующие этим значениям невязки зависят в обратном порядке. Для примера, приведенного на рисунке 7, псевдокод вычислений обратного хода (т. е. псевдокод, реализующий вычисление частных производных лосса по входным значениям слоя, используя значения производных лосса по выходном значениям слоя) приведен на *рисунке 9.*

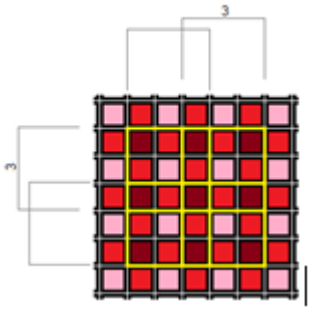

Рисунок 8. Ядро свертки попадает на пиксель

```
    for                                                    (y=0;y<99;y++)
for                                                        (x=0;x<99;x++)
for (l=0;l<40;l++)
     N_IN[y][x][l] = 0;
    for                                                    (y=0;y<50;y++)
for                                                        (x=0;x<50;x++)
for                                                        (l=0;l<40;l++){
for (ry=-1;ry<2;ry++) if (2*y-ry>-1 && 2*y-ry<99)
     for (rx=-1;rx<2;rx++) if (2*x-rx>-1 && 2*x-rx<99)
     for (f=0;f<20;f++)
     N_IN[2*y-ry][2*x-rx][l]+=N_OUT[y][x][f]*FILTERS[ry][rx][f][l]; }
    for                                                    (f=0;f<20;f++)
for                                                        (l=0;l<40;l++)
for (ry=-1;ry<2;ry++)
     for (rx=-1;rx<2;rx++) {
     GR_FILTERS[ry][rx][f][l] = 0;
     for (y=0;y<50;y++) if (2*y+ry>-1 && 2*y+ry<99)
     for (x=0;x<50;x++) if (2*x+rx>-1 && 2*x+rx<99)
     GR_FILTERS[ry][rx][f][l]+=IN[2*y+ry][2*x+rx][l]*N_OUT[y][x][f];}
    for (f=0;f<20;f++) {
     GR_BIAS[f]=0;
     for (y=0;y<50;y++)
     for (x=0;x<50;x++)
     GR_BIAS[f]+=N_OUT[y][x][f];}
```

Рисунок 9. Псевдокод вычислений обратного хода

Поскольку сверточный слой реализует линейное преобразование, то применение двух сверточных слоев одного за другим эквивалентно применению некоторого другого сверточного слоя. Соответственно, сверточные слои должны быть разделены какой-нибудь нелинейной функцией. Представим себе теперь, что перед фигурной скобкой в конце рисунка 7 была бы добавлена операция, обнуляющая значение OUTyxf, если оно оказалось отрицательным. Такая функция является нелинейной (*рисунок 10*), и, следовательно, вполне подходит в качестве функции активации.

ReLU/Rectified linear unit – нелинейная функция, обнуляющая отрицательные значения и пропускающая положительные.

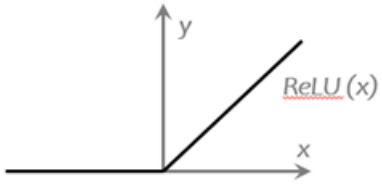

Рисунок 10. Нелинейная функция

На практике в сверточных нейросетях в качестве функции активации используется почти исключительно ReLU, реже – модифицированные варианты ReLU, и почти никогда – сигмоида и другие сложные нелинейные функции. Вычисления обратного хода для фунции ReLU легко представить – там, где входное значение прямого хода было положительным, значение невязки передается без изменений, а где входное значение было обнулено, значение невязки также обнуляется. Использование такой функции активации также упрощает логику понимания применимости сверточного слоя, поскольку приравнивает интуитивно непонятный случай «выраженности признака со знаком минус» (когда, как мы разбирались в начале главы, две вектора направлены противоположно) к случаю, когда признак просто не выражен (когда два вектора перпендикулярны). Представим теперь сверточную нейросеть из нескольких десятков сверточных слоев. Допустим, на всех слоях используется свертка с ядром 3×3 и единичным страйдом, тогда один пиксель выходной карты признаков последнего слоя характеризует фрагмент 3×3 входной карты признаков этого слоя, который, в свою очередь, характеризует фрагмент 5×5 входной карты признаков предыдущего слоя, а он – 7×7 предыдущего слоя, 9×9, 11×11 и т. д. При таких обстоятельствах для того, чтобы сопоставить на определенном слое признаки, расположенные на краях фрагмента 100×100 входного изображения, в нейросети должно быть хотя бы 50 слоев. С другой стороны, каждый последующий слой должен содержать признаки более высокого порядка, и нет никакой причины, почему должно быть именно 50 уровней таких признаков. Для того чтобы число слоев в нейросети не приходилось ставить в такую жесткую зависимость от размера объектов на изображениях, вставляют специальный слой, который прореживает карты признаков, уменьшая их ширину и высоту.

По аналогии со сверткой, пулинг применяется небольшими окнами, 3×3 или 2×2, другие варианты на практике не встречаются.

Пулинг /pooling – слой нейросети, задача которого заключается в кратном уменьшении размеров карты признаков с сохранением содержательной информации. Не содержит весов. Не влияет на глубину карты признаков.

Макс-пулинг /max pooling – тип пулинга, при котором из нескольких пикселей входной карты признаков в каждом канале выбирается максимальное значение.

Средний пулинг/average pooling – тип пулинга, при котором значения из нескольких пикселей входной карты признаков в каждом канале усредняются.

У пулинга есть страйд, однако он почти без исключений равен 2, а также падding. Примеры работы двух вариантов – максимального и среднего – слоя пулинга с ядром 2x2 и страйдом 2 (*рисунок 11*).

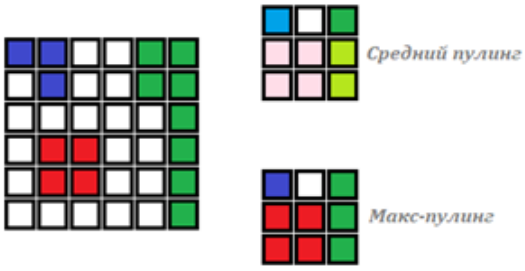

Рисунок 11. Примеры работы двух вариантов максимального и среднего – слоя пулинга с ядром 2x2 и страйдом 2

В противовес свертке для пулинга характерно, что паддинг выбирается разный с разных сторон. Вычисления обратного хода для среднего пулинга выглядят так же, как прямого хода – значения невязок, передаваемые на предыдущий слой, вычисляются как среднее значение невязок, переданных на обратном ходе на слой пулинга, в соответствующем окне. Вычисления обратного хода для макс-пулинга не во всех ситуациях определены однозначно, так как в некоторых ситуациях несколько пикселей входной карты признаков, попавшие в рамку, могут содержать максимальное значение.

В этом случае математика не дает четкого ответа на вопрос, как поступить, есть два варианта: можно сделать ненулевыми невязки для каждого пикселя с максимальным значением, или только для одного случайного. Первый вариант в каком-то смысле более правильный, но второй вариант позволяет реализовать вычисления обратного хода более эффективно. Псевдокод для среднего пулинга с размерами входной карты признаков, равными 99, 99 и 40, ядром 3×3 и двойным страйдом, а также его обратного хода, приведен на *рисунке 12*, а аналогичный для макс-пулинга – на *рисунке 13.*

```
for (y=0;y<49;y++)
 for (x=0;x<49;x++)
 for (l=0;l<40;l++) {
 OUT[y][x][l] = 0;
 for (ry=-1;ry<2;ry++)
 for (rx=-1;rx<2;rx++)
 OUT[y][x][l]+=(IN[2*y+ry+1][2*x+rx+1][l] / 9); }
for (y=0;y<99;y++)
for (x=0;x<99;x++)
for (l=0;l<40;l++)
 N_IN[y][x][l]=0;
for (y=0;y<49;y++)
for (x=0;x<49;x++)
for (l=0;l<40;l++)
 for (ry=-1;ry<2;ry++)
 for (rx=-1;rx<2;rx++)
 N_IN[2*y+ry+1][2*x+rx+1][l]+=(N_OUT[y][x][l] / 9);
```

Рисунок 12. Псевдокод для обратного хода

```
    for                                                    (y=0;y<49;y++)
for                                                        (x=0;x<49;x++)
for (l=0;l<40;l++) {
     MAX[y][x][l] = -99999999999999999999;
     for (ry=-1;ry<2;ry++)
     for (rx=-1;rx<2;rx++)
     if (IN[2*y+ry+1][2*x+rx+1][l]> MAX[y][x][l])
     MAX[y][x][l] = OUT[y][x][l];
     OUT[y][x][l] = MAX[y][x][l];  }
    for                                                    (y=0;y<99;y++)
for                                                        (x=0;x<99;x++)
for (l=0;l<40;l++)
     N_IN[y][x][l]=0;
    for                                                    (y=0;y<49;y++)
for                                                        (x=0;x<49;x++)
for (l=0;l<40;l++)
     for (ry=-1;ry<2;ry++)
     for (rx=-1;rx<2;rx++)
     if   (IN[2*y+ry+1][2*x+rx+1][l]   ==   MAX[y][x][l])   N_IN[2*y+ry+1][2*x+rx+1][l]   +=
N_OUT[y][x][l];
```

Рисунок 13. Псевдокод для макс-пулинга

Представим себе теперь, что мы решаем задачу классификации объекта на изображении, например, определяем марку автомобиля. Будем строить нейросеть из сверточных слоев, слоев ReLU и слоев пулинга, постепенно уменьшая высоту и ширину карт признаков. По идее, начиная с некоторого слоя, на котором высота и ширина карты признаков будут невелики, можно приделать один или несколько полносвязных слоев со слоями сигмоиды, софтмакс и лосс, приведенный на рисунке 7. Приблизительно так и поступили создатели сверточной нейросети AlexNet0F [8], с которой и начался бум использования сверточных нейросетей в компьютерном зрении. Иллюстрация нейросети AlexNet приведена на *рисунке 14.*

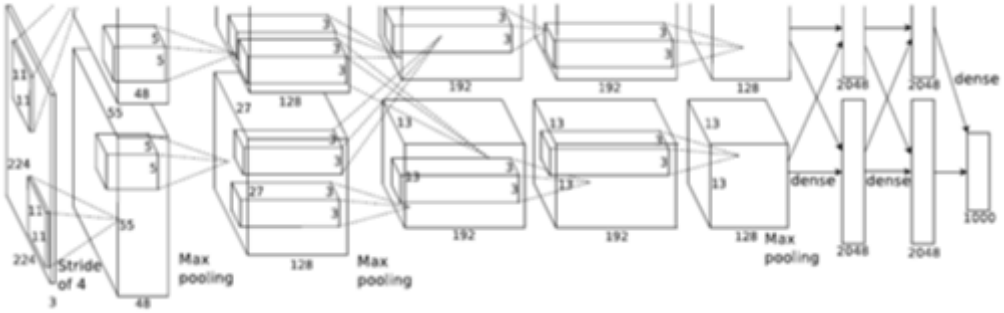

Рисунок 14. Нейросеть AlexNet

Практически сразу стало ясно, что подавать в полносвязный слой карту признаков с большой шириной и высотой неудобно, поскольку на этом слое возникает огромное количество весов. К тому же такая схема означает, что количество весов нейросети строго привязано к размеру обрабатываемого изображения. Чтобы избавиться от этой зависимости, стали применять так называемый глобальный пулинг – слой пулинга, на котором нет паддинга и страйда, а ядро по размеру совпадает с шириной и высотой входной карты признаков. Так, например, сделали разработчики нейросети VGG1F [4].

На рисунке 14 можно обнаружить некоторую странность: на промежуточных слоях существует как будто два набора карт признаков. Это действительно так, в наших терминах это означает, что попросту внутри нейросети есть две ветки слоев, которые растут из входной карты признаков и сходятся там, где выходные карты признаков двух веток поступают в последний полносвязный слой. Изначально эта схема была применена авторами по той причине, что у них попросту было две видеокарты, и они таким образом добились эффективного распараллеливания вычислений. При этом обе ветви были симметричны структурно, но отличались выученными значениями весов. В более общем случае нейросеть не обязана состоять из строго последовательных слоев, она может содержать параллельные ветви, в том числе отличающиеся. Этот принцип был применен и развит авторами нейросети SqueezeNet2F [4], которые использовали его следующим образом. Они задумались над тем, что сверточный слой 3×3 выполняет две задачи: строит признаки следующего уровня на основе признаков текущего уровня и сопоставляет значения соседних пикселей. Авторы отметили, что ядро сверточного слоя должно быть 3×3 или более только для второй задачи, для первой же достаточно ядра 1×1, при этом такой сверточный слой будет содержать в девять раз меньше весов и вычислительных операций. В связи с этим они предложили строить нейросеть из специальных блоков, содержащих три сверточных слоя, из которых два слоя параллельны, но обладают разными ядрами – 1×1 и 3×3 (*рис. 15*). Для того чтобы подать выходные карты признаков с параллельных веток в следующий слой или блок не делалось специальных вычислений, их выходные карты признаков просто слеплялись, соответствующая операция была названа слоем склейки (concat, от англ. concatenation). Получившаяся в результате нейросеть не уступала предыдущим по качеству, зато была значительно меньше по объему и быстрее училась.

Архитектура нейросети – описание структуры нейросети, т. е. из каких слоев с какими параметрами в какой последовательности состоит нейросеть.

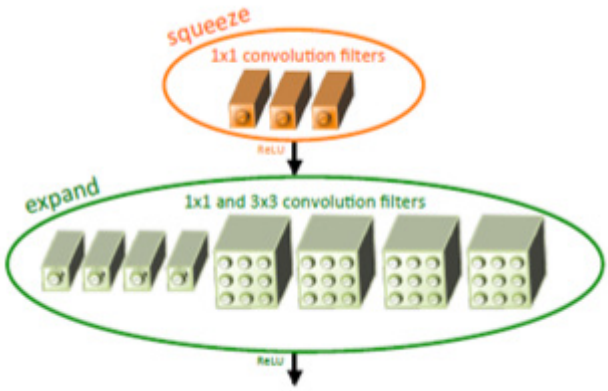

Рисунок 15. Нейросеть из специальных блоков, содержащих три сверточных слоя

Идея выстраивания сверточной нейросети с параллельными ветвями одно время породила много фантазий на тему того, из каких слоев должна быть составлена нейросеть, примерно тогда же закрепился следующий термин.

Одно время казалось, что поиски более продвинутых нейросетей нужно вести именно в направлении сложных архитектур. Так появилось целое семейство нейросетей Inception3F [13] от компании Google. Все нейросети семейства состояли из сложных блоков нескольких типов, каждый из которых содержал несколько параллельных ветвей сверточных слоев и слоев пулинга. Схема построения одного из типов блоков, используемых в нейросети семейства второй версии, приведена на *рисунке 16.*

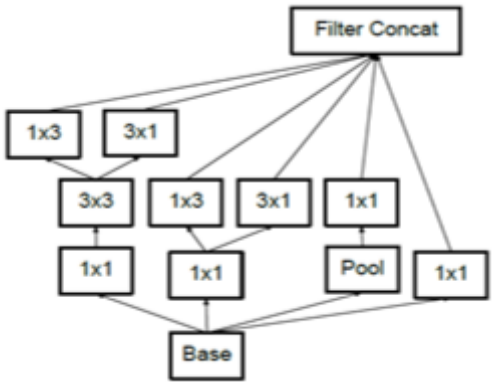

Рисунок 16. Схема построения
одного из типов блоков, используемых в нейросети семейства второй версии.

Несмотря на то, что развитием этого семейства нейросетей занималась компания Google, человечеству оно ничего не дало. Этот пример поучителен, так как он хорошо иллюстрирует экспериментальную природу машинного обучения: если даже за простыми нейросетями, за исключением однослойного перцептрона, не стоит никакого детального теоретического исследования, то такие монструозные конструкции, как на рисунке 16, никак не могли бы являться продуктом научной мысли, а порождены исключительно фантазией.

Вернемся снова к мысли, которая была озвучена в контексте нейросети SqueezeNet: свертки с ядрами 3×3 нужны только для сопоставления признаков с соседних пикселей, для построения более глубоких признаков достаточно сверток с ядрами 1×1. Разработчики нейросети MobileNet4F [3] развили эту мысль следующим образом: сопоставлять значения в соседних пикселях можно отдельно от сопоставления значений в соседних каналах. В качестве реализации этого принципа они предложили концепцию поканальной свертки (авторский перевод англ. depthwise), т. е. замены сверточного слоя 3×3 на последовательность из двух слоев, один из них – сверточный с ядром 1×1, а другой представляет собой сверточный слой 3×3, но у фильтров которого единичная глубина, при этом каждый фильтр применяется только к одному каналу входной карты признаков и порождает, соответственно, один канал выходной карты признаков (*рисунок 17*).

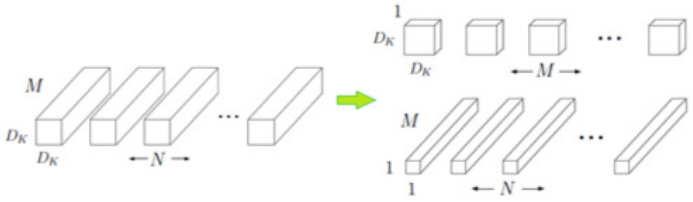

Рисунок 17. Фильтры и каналы карты признаков.

Предложенная замена, конечно, алгоритмически не эквивалентная применению сверточного слоя 3×3, однако тоже оказалась работоспособна. При этом число весов и вычислений при ее использовании сокращается значительно, чуть менее чем в девять раз.

Еще одно значительное изобретение в области архитектур сверточных нейросетей сделали разработчики нейросети ResNet5F [6,7]. Ее авторы обратили внимание на то, что при обучении нейросетей гораздо быстрее меняются значения весов слоев, ближе к концу нейросети, нежели слоев, близких к началу. Это можно понять интуитивно, если вспомнить, что при выполнении итераций градиентного спуска вычисляется градиент сразу по всем весам. Закономерно, что вес последнего слоя может гораздо больше повлиять на результат работы нейросети, чем вес любого из предыдущих, поскольку последнее воздействие именно за ним, а оно может преобразовать значения входной карты признаков до неузнаваемости. Таким образом, чем больше слоев у нейросети, тем хуже учатся ее более ранние слои. Эта закономерность, по идее, наметила неявный предел применения сверточных нейросетей, однако авторы нейросети ResNet придумали, как ее обойти. Для этого они стали просто прибавлять к результатам некоторых сверточных слоев соответствующие им по размерам карты признаков с одного из предыдущих слоев (*рисунок 18*). Слой, выполняющий сложение двух карт признаков, назвали поэлементными (англ. eltwise).

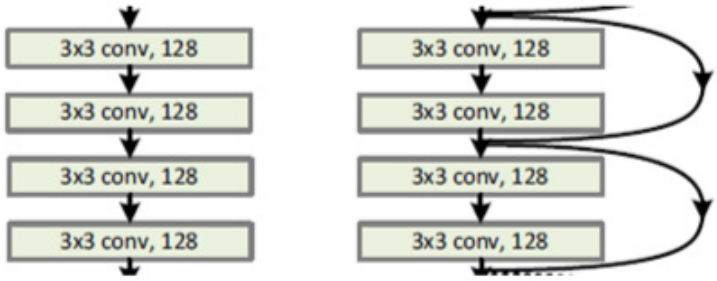

Рисунок 18. Слои карты признаков.

В результате выстраивания указанных связей даже карты признаков, возникающие в самом начале нейросети, имели прямое влияние на близкие к концу нейросети слои через последовательность сложений. Как следствие, это учитывалось при обратном ходе, слои в начале нейросети в целом стали учиться лучше, что позволило учить нейросети, состоящие из 100 и более слоев. С того момента этот прием широко применяется как в сверточных нейросетях, так и в других.

Приблизительно к 2020 г. большая часть того разумного, что можно было придумать в области сверточных нейросетей, уже была придумана, внимание мирового сообщества постепенно переключилось со сверточных нейросетей на другие. Набор базовых слоев, из которых собираются сверточные нейросети, и приемов их обучения устаканился, специалист по машинному обучению на практике редко выходит за их границы.

Типовая используемая на практике сверточная нейросеть содержит два-четыре десятков слоев, преимущественно сверточных 1×1, сверточных 3×3 или поканальных сверточных 3×3, пулинга, поэлементных, склейки, активаций типа ReLU, на входе в качестве исключения сверточный слой 7×7. Ширина и высота карт признаков в нейросети постепенно уменьшаются с помощью слоев пулинга от типовых 227×227 до 15×15 или 7×7. Глубины карт признаков, наоборот, посте-

пенно растут от 3 до 256–512 к концу нейросети, иногда большие и малые значения глубин чередуются. Следует обратить внимание на то, что в связи с особенностями взаимосвязей размеров карт признаков и размеров ядер, паддингов и страйдов, нейросети почти всегда описывают в базовом варианте с входными размерами изображения от 222×222 до 227×227, подразумевая, что для применения на изображениях других размеров могут быть незначительно поправлены паддинги и/или параметры последних слоев.

Гораздо разнообразнее масса всевозможных модификаций слоев и приемов, которые либо используются редко, либо вообще оказались не востребованы. Среди них:

1) свертка с «раздвинутым» (англ. dilated) ядром, которое берет не соседние пиксели, а взятые с некоторым шагом;
2) разнообразные варианты функций активации;
3) слои, перемешивающие между собой каналы карты признаков;
4) слои активаций с обучаемыми весами;
5) сложные архитектуры нейросетей со склейкой многих карт признаков и т. д.

Существует также множество специфический слоев, применяемых не во всех сверточных нейросетях, а только при решении конкретных задач технического зрения. Рассмотрим, например, задачу детекции.

Задача детекции/detection/задача обнаружения и распознавания – задача, при которой заранее известны несколько классов объектов, а от нейросети требуется найти все объекты каждого из этих классов на изображении и указать для каждого найденного объекта его положение, размер и класс.

Баундин-бокс /bounding box – прямоугольник в координатах изображения, обводящий один объект.

Для задачи детекции сверточные нейросети адаптируют приблизительно одним и тем же образом. Входное изображение мысленно делится на несколько небольших фрагментов, каждому соответствует набор значений выходной карты признаков последнего сверточного слоя, состоящий из значений конфиденсов (по одному для каждого из известных классов), а также значений баундин-бокса, т. е. двух координат прямоугольника и двух его размеров (*рисунок 19*).

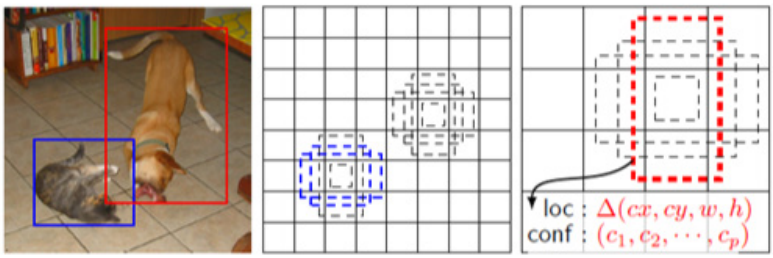

Рисунок 19. Значения баундин-бокса

Этот подход одним из первых был описан авторами нейросети SSD6F [15]. Разумеется, что для обучения такой нейросети должна использоваться соответствующая разметка, включающая для каждой картинки для каждого объекта на ней его класс и размер и координаты прямоугольника, его ограничивающего. Заметим, что в предложенной схеме максимальное число объектов может быть значительно больше, чем реальное число объектов на картинке, поэтому нейросеть может найти один и тот же объект несколько раз.

В связи с этим результаты работы нейросети нужно подвергнуть своего рода фильтрации, чтобы определить, какие из них описывают один объект, а какие – разные.

Более современные вариации нейросетей, используемых для решения задачи детекции, используют более совершенные механизмы, а также некоторые другие вариации слоев, например, пулинг с настраиваемыми параметрами 7F [6].

Рассмотрим теперь задачу семантической сегментации, пример ее решения приведен на *рисунке 20*.

Рисунок 20. Пример решения задачи семантической сегментации

Задача семантической сегментации/semantic segmentation – задача, при которой заранее известны несколько типов зон на изображении, а от нейросети требуется отнести каждый пиксель изображения к одной или нескольким зонам.

Сегментационная маска – карта признаков, содержащая информацию об отнесении каждого пикселя к одной или другой зоне.

Решение задачи семантической сегментации с помощью нейросетей требует построения сегментационной маски. Чтобы сегментационная маска была достаточно точной, слой нейросети, ее порождающий, должен иметь высоту и ширину карт признаков, равные высоте и ширине исходного изображения. Однако это противоречит тому, что мы выяснили ранее: для эффективного выявления признаков сложных объектов ширина и высота карты признаков должны от входа к выходу постепенно уменьшаться. Чтобы решить это противоречие, в сверточных нейросетях, применяемых для решения задачи семантической сегментации, используются дополнительные слои.

Деконволюция /deconvolution/обратная свертка – слой, похожий на сверточный, отличающийся тем, что ширина и высота выходной карты признаков больше ширины и высоты входной карты признаков. Вычисления обратного хода сверточного слоя зачастую в точности являются вычислениями прямого хода слоя деконволюции, и наоборот.

Апсэмпл /upsample – слой, похожий на пулинг, отличающийся тем, что ширина и высота выходной карты признаков больше ширины и высоты входной карты признаков.

Применение этих слов позволяет выстроить сверточную нейросеть по образу песочных часов. В первой половине слоев выполняется основной анализ изображения и выявление признаков, при этом ширина и высота карт признаков постепенно уменьшаются, а во второй половине результаты как бы разворачиваются обратно в масштаб изображения с учетом границ объектов, соответственно, ширины и высоты карт признаков увеличиваются. Характерным примером такой нейросети является U-net8F [12].

Обратим внимание, что нейросеть, выполняющая сегментацию, генерирует некоторую информацию, по объему сравнимую с исходной. В более общем случае нейросеть не обязательно учить делать лаконичные выводы по большому объему информации, можно и наоборот – учить нейросеть генерировать нужную информацию по некоторому объему входных данных.

Генеративная нейросеть – нейросеть, предназначенная для генерации данных, а не их анализа. Строго формально любая нейросеть генерирует какие-то данные, поэтому генеративными нейросетями называют те, содержательная часть выходных данных которых превосходит содержательную часть входных данных.

Генеративный искусственный интеллект – то же, что генеративные нейросети.

Задача идентификации – задача, в которой по входным данным, содержащим информацию об объекте, требуется определить конкретный объект среди всего их множества. Является своего рода предельной постановкой задачи классификации.

Задача верификации – задача, в которой требуется по двум входным данным определить, изображен ли на них один и тот же объект или нет.

В других задачах обработки и генерации изображений принцип остается тот же: в основе решения лежит сверточная нейросеть, модифицированная одним или несколькими специальными математическими операциями или нейросетевыми слоями.

Бэкбон – нейросеть, взятая за основу другой нейросети или алгоритма на основе нейросети. Никакого алгоритмического смысла данное понятие не несет, применяется для удобства.

В некоторых случаях не нейросетевые операции, выполняемые вместе с вычислениями нейросети, могут быть достаточно интеллектуальным, например, в задачах идентификации и верификации.

Наиболее популярным примером задачи идентификации является задача распознавания человека по лицу. Для распознавания системой нового лица не выполняют заново обучение нейросети, а делают 1−2 фотографии. Это возможно благодаря тому, что нейросети не ставится задача определения конкретного объекта, вместо это нейросеть обучают таким образом, чтобы она генерировала большой набор (около 1000) разнообразных признаков по изображению лица так, чтобы для лица одного и того же человека набор оставался приблизительно одним и тем же, а для двух разных людей – существенно отличался. Нейросеть при этом структурно состоит из тех же слоев, что и выполняющая задачу классификации. Полученный нейросетью набор признаков обрабатывается уже традиционными математическими алгоритмами: для сохранения нового лица этот вектор сохранятся в базу данных, а для распознавания – сравнивается с другими векторами из базы, в результате чего находится наиболее похожий. Аналогично решается задача верификации – нейросеть выдает для двух изображений два вектора признаков, и если они оказываются достаточно похожими, то считается, что на двух изображениях изображен один и тот же объект.

Список литературы:

[1] Chen Zhang, Joohee Kim Video object detection with two-path convolutional LSTM pyramid. 2016. – URL: https://www.researchgate.net/publication /343702579_Video_Object_Detection_ With_ Two-Path_ Convolutional_LSTM_ Pyramid/fulltext/5f3b2658928 51cd302013482/Video-Object-Detection-With-Two-Path-Convolutional-LSTM-Pyramid.pdf/ (дата обращения: 16.01.2026).

[2] Hanson A., Koutilya PNVR, Sanjukta Krishnagopal, Larry Davis Bidirectional Convolutional LSTM for the Detection of Violence in Videos. 2018. – URL: https://openaccess.thecvf.com/content_eccv_2018_workshops/w10 /html/Hanson_Bidirectional_Convolutional_LSTM_or_the_ Detection_of_ Violence_in_ Videos_ ECCVW_2018_paper.html (дата обращения: 15.01.2026).

[3] Howard A.G., Menglong Zhu, Bo Chen, Kalenichenko D. MobileNets: Efficient Convolutional Neural Networks for Mobile Vision Applications. 2017. – URL: https://arxiv.org/pdf/1704.04861/ (дата обращения: 11.01.2026).

[4] Iandola F.N., Han S., Moskewicz M.W., Ashraf K., Dally W.J., Keutzer K. SqueezeNet: AlexNet-level accuracy with 50x fewer parameters and <0.5MB model size Recognition. 2016. Published as a conference paper at ICLR. 2015. – URL: https://www. researchgate.net/publication/ 301878495_SqueezeNet_AlexNet-level_accuracy_ with_50x_ fewer_ parameters_ and_05MB_model_size (дата обращения: 10.01.2026).

[5] Ioffe S., Szegedy C. Batch Normalization: Accelerating Deep Network Training by Reducing Internal Covariate Shift. 2015. – URL: https://arxiv.org/pdf/1502.03167/ (дата обращения: 13.01.2026).

[6] Kaiming He, Xiangyu Zhang, Shaoqing Ren, Jian Sun Deep Residual Learning for Image Recognition. 2015. – URL: https://arxiv.org/pdf/1512.03385 (дата обращения: 11.01.2026).

[7] Kaiming He, Xiangyu Zhang, Shaoqing Ren, Jian Sun Spatial pyramid pooling in deep convolutional networks for visual recognition. 2015. – URL: https://arxiv.org/pdf/1406.4729/ (дата обращения: 11.01.2026).

[8] Krizhevsky I., Sutskever I., Hinton G.E. Imagenet classification with deep convolutional neural networks. 2012. – URL: https://proceedings.neurips.cc/paper_files/paper/2012/file/c399862d3b9d6b76c8436e 924a68c45b-Paper.pdf (дата обращения: 12.01.2026).

[9] Lavin A., Scott Gray Fast Algorithms for Convolutional Neural Networks. 2015. – URL: https://arxiv.org/abs/1509.09308 (дата обращения: 11.01.2026).

[10] Mikolov T., Kai Chen, Corrado G., Jeffrey Dean Efficient Estimation of Word Representations in Vector Space. 2013. – URL: https://arxiv.org/abs/1301.3781 (дата обращения: 15.01.2026).

[11] Norman P. Jouppi, Cliff Young, Nishant Patil and others In-Datacenter Performance Analysis of a Tensor Processing Unit. 2017. – URL: https://arxiv.org/pdf/1704.04760 (дата обращения: 11.01.2026).

[12] Ronneberger O., Fischer Ph., Brox Th. U-Net: Convolutional Networks for Biomedical Image Segmentation. 2015. – URL: https://arxiv.org/pdf/1505.04597/ (дата обращения: 13.01.2026).

[13] Szegedy C., Wei Liu, Chapel Hill, Yangqing Jia, Pierre Sermanet, Scott Reed, Dragomir Anguelov, Dumitru Erhan, Vincent Vanhoucke, Andrew Rabinovich Going Deeper with Convolutions. 2014. – URL: https://arxiv.org/pdf/1409.4842 (дата обращения: 12.01.2026)

[14] Simonyan K., Zisserman A. Very Deep Convolutional Networks for Large-Scale Image Recognition. 2015. Published as a conference paper at ICLR 2015. – URL: https://arxiv.org/pdf/1409.1556 (дата обращения: 10.01.2026).

[15] Wei Liu, Anguelov D., Erhan D., Szegedy C., Reed S., Cheng-Yang Fu, Berg A.C. SSD: Single Shot MultiBox Detector. 2016. – URL: https://arxiv.org/pdf/1512.02325 (дата обращения: 11.01.2026).

[16] Официальный сайт ONNX. – URL: https://onnx.ai/.

[17] Основы технологий искусственного интеллекта: учебное пособие/ А.А. Семенов, В.В. Гончар, А.С. Эрдниев. – Московский университет МВД России имени В.Я, Кикотя, 2025. – 190с.

[18] Финансовый мониторинг: учебное пособие / В.И. Глотов, А.У. Альбеков, О.Н. Тисен и др. – КноРус, 2022. – 198 с.

[19] Тисен О.Н. Противодействие современным приемам и способам вербовки в международные террористические организации // Российская юстиция. 2018, №9. С. 51-54.

[20] Гончар В.В. О важности формирования единообразного понятийного аппарата необходимого для расследования преступлений в сфере компьютерной информации// Вестник Экономической безопасности. 2018. №1. С. 225-230.

[21] Орлова А.А., Гончар В.В. Особенности расследования преступлений, совершаемых в сфере информационных технологий // Безопасность бизнеса. 2021. № 4. С. 25-29.

References:

[1] Chen Zhang, Joohee Kim Video object detection with two-path convolutional LSTM pyramid. 2016. – URL: https://www.researchgate.net/publication /343702579_Video_Object_Detection_With_ Two-Path_ Convolutional_LSTM_ Pyramid/fulltext/5f3b2658928 51cd302013482/Video-Object-Detection-With-Two-Path-Convolutional-LSTM-Pyramid.pdf/ (дата обращения: 16.01.2026).

[2] Hanson A., Koutilya PNVR, Sanjukta Krishnagopal, Larry Davis Bidirectional Convolutional LSTM for the Detection of Violence in Videos. 2018. – URL: https://openaccess.thecvf.com/content_eccv_2018_workshops/w10 /html/Hanson_Bidirectional_Convolutional_LSTM_or_the_ Detection_of_ Violence_in_ Videos_ ECCVW_2018_paper.html (дата обращения: 15.01.2026).

[3] Howard A.G., Menglong Zhu, Bo Chen, Kalenichenko D. MobileNets: Efficient Convolutional Neural Networks for Mobile Vision Applications. 2017. - URL: https://arxiv.org/pdf/1704.04861/ (access date: 11.01.2026).

[4] Iandola F.N., Han S., Moskewicz M.W., Ashraf K., Dally W.J., Keutzer K. SqueezeNet: AlexNet-level accuracy with 50x fewer parameters and <0.5MB model size Recognition. 2016. Published as a conference paper at ICLR. 2015. – URL: https://www. researchgate.net/publication/ 301878495_ SqueezeNet_AlexNet-level_accuracy_ with_50x_ fewer_ parameters_ and_05MB_model_size (дата обращения: 10.01.2026).

[5] Ioffe S., Szegedy C. Batch Normalization: Accelerating Deep Network Training by Reducing Internal Covariate Shift. 2015. – URL: https://arxiv.org/pdf/1502.03167/ (access date: 13.01.2026).

[6] Kaiming He, Xiangyu Zhang, Shaoqing Ren, Jian Sun Deep Residual Learning for Image Recognition. 2015. – URL: https://arxiv.org/pdf/1512.03385 (access date: 11.01.2026).

[7] Kaiming He, Xiangyu Zhang, Shaoqing Ren, Jian Sun Spatial pyramid pooling in deep convolutional networks for visual recognition. 2015. - URL: https://arxiv.org/pdf/1406.4729/ (access date: 11.01.2026).

[8] Krizhevsky I., Sutskever I., Hinton G.E. Imagenet classification with deep convolutional neural networks. 2012. - URL: https://proceedings.neurips.cc/paper_files/paper/2012/file/c399862d3b9d-6b76c8436e 924a68c45b-Paper.pdf (accessed on: 12.01.2026).

[9] Lavin A., Scott Gray Fast Algorithms for Convolutional Neural Networks. 2015. - URL: https://arxiv.org/abs/1509.09308 (access date: 11.01.2026).

[10] Mikolov T., Kai Chen, Corrado G., Jeffrey Dean Efficient Estimation of Word Representations in Vector Space. 2013. – URL: https://arxiv.org/abs/1301.3781 (access date: 15.01.2026).

[11] Norman P. Jouppi, Cliff Young, Nishant Patil and others In-Datacenter Performance Analysis of a Tensor Processing Unit. 2017. – URL: https://arxiv.org/pdf/1704.04760 (access date: 11.01.2026).

[12] Ronneberger O., Fischer Ph., Brox Th. U-Net: Convolutional Networks for Biomedical Image Segmentation. 2015. – URL: https://arxiv.org/pdf/1505.04597/ (access date: 13.01.2026).

[13] Szegedy C., Wei Liu, Chapel Hill, Yangqing Jia, Pierre Sermanet, Scott Reed, Dragomir Anguelov, Dumitru Erhan, Vincent Vanhoucke, Andrew Rabinovich Going Deeper with Convolutions. 2014. – URL: https://arxiv.org/pdf/1409.4842 (accessed on: 12.01.2026)

[14] Simonyan K., Zisserman A. Very Deep Convolutional Networks for Large-Scale Image Recognition. 2015. Published as a conference paper at ICLR 2015. - URL: https://arxiv.org/pdf/1409.1556 (access date: 10.01.2026).

[15] Wei Liu, Anguelov D., Erhan D., Szegedy C., Reed S., Cheng-Yang Fu, Berg A.C. SSD: Single Shot MultiBox Detector. 2016. – URL: https://arxiv.org/pdf/1512.02325 (access date: 11.01.2026).

[16] ONNX official website. – URL: https://onnx.ai/.

[17] Fundamentals of artificial intelligence technologies: a textbook/A.A. Semenov, V.V. Gonchar, A.S. Erdniev. - Moscow University of the Ministry of Internal Affairs of Russia named after V.Ya, Kikotya, 2025. - 190s.

[18] Financial monitoring: training manual/V.I. Glotov, A.U. Albekov, O.N. Tisen et al. - KnoRus, 2022. - 198 s.

[19] Tisen O.N. Countering modern methods and methods of recruitment to international terrorist organizations//Russian Justice. 2018, №9. S. 51-54.

[20] Gonchar V.V. On the importance of forming a uniform conceptual apparatus necessary for investigating crimes in the field of computer information//Bulletin of Economic Security. 2018. №1. S. 225-230.

[21] Orlova A.A., Gonchar V.V. Features of the investigation of crimes committed in the field of information technology//Business security. 2021. № 4. S. 25-29.