Дата поступления рукописи в редакцию: 25.04.2026 г.
Дата принятия рукописи в печать: 14.05.2026 г.
DOI: 10.24412/2224-9133-2026-4-140-148

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

ОПТИМИЗАЦИЯ ОБУЧЕНИЯ НЕЙРОННЫХ СЕТЕЙ: ФУНКЦИИ ПОТЕРЬ, ГРАДИЕНТНЫЙ СПУСК И ОБРАТНОЕ РАСПРОСТРАНЕНИЕ ОШИБКИ

Аннотация. *Работа посвящена математическим основам обучения нейронных сетей через минимизацию функции потерь как меры качества модели. Обучение формулируется как задача оптимизации, решаемая с помощью градиентного спуска с учетом его особенностей в многомерных пространствах параметров. Описываются роль размеченных данных, влияние размера батча и организации эпох на процесс обучения. Поясняется применение метода обратного распространения ошибки для эффективного вычисления градиента в сложных архитектурах нейросетей.*

Ключевые слова: *нейронные сети, функция потерь, лосс, градиентный спуск, оптимизация, машинное обучение, обучение с учителем, обучение без учителя, разметка данных, батч, эпоха, градиент, частная производная, обратное распространение ошибки, слои нейросети, прямой и обратный ход.*

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

OPTIMIZING NEURAL NETWORK TRAINING: LOSS FUNCTIONS, GRADIENT DESCENT, AND BACKPROPAGATION

Annotation. *This paper examines the mathematical foundations of neural network training through minimizing the loss function as a measure of model quality. Training is formulated as an optimization problem solved using gradient descent, taking into account its specific features in multidimensional parameter spaces. The role of labeled data, the influence of batch size, and epoch organization on the training process are described. The application of the backpropagation method for efficient gradient computation in complex neural network architectures is explained.*

Key words: *neural networks, loss function, loss, gradient descent, optimization, machine learning, supervised learning, unsupervised learning, data labeling, batch, epoch, gradient, partial derivative, backpropagation, neural network layers, forward and backward progression.*

Изучим теперь, как для нейросети *f* подбираются значения весов *w*. В математике существует масса численных методов для решения самых разных задач, один из таких методов позволяет находить минимум функции. Если мы сконструируем функцию, значение которой будет тем меньше, чем лучше веса *w* подходят для решение некоторой практической задачи, то обучение нейросети как раз и будет заключаться в поиске минимума указанной функции.

Сформулируем для начала, что же означает «лучше всего» для одной единицы входных данных. Для этого создадим функцию *L*, которая будет принимать на вход результат работы нейросети *f* и оценивать, насколько результат работы плох. При этом вне зависимости от того, насколько сложны аргументы функции *L*, ее результат должен быть одним числом *z* – чем меньше это число, тем лучше результат. Для оценки качества нейросети в целом будем вычислять сумму *S* значений *L*, полученных на некотором наборе входных данных, пронумеруем эти входные данные и соответствующие им результаты вычислений индексом *i*. Формульное выражение *S*, в таком случае, будет зависеть только от результатов y_i вычислений нейросети. Однако, если воспринимать нейросеть *f* как часть функции *L*, *S* можно изучать именно как функцию, зависящую от весов нейросети *w* (*формула на рисунок 1*).

$$S = \sum_i z_i \, , z_i = L(y_i), y_i = f(w, x_i) \; \rightarrow S = \sum_i L\big(f(w, x_i)\big) \rightarrow S(w)$$

Рисунок 1. Формульное выражение S.

Зафиксируем несколько наборов входных данных x_i и будем считать сумму *S* значений функции *L* для каждого из этих наборов мерой того, насколько хорошо работает нейросеть. При наборе весов w_1 мы получим одно значение суммы, при наборе весов w_2 – другое, и т. д. Если мы теперь попробуем найти минимум функции *S*, а точнее точку w_{min}, в которой достигается минимум функции *S*, то мы, таким образом, как раз и найдем оптимальный (в каком-то смысле) набор весов *w* для нейросети *f*.

Лосс /loss/функция потерь – функция, оценивающая качество работы нейросети, которую искусственно подбирают таким образом, чтобы лучшее качество нейросети соответствовало меньшему значению функции. Таким путем задачу обучения нейросети сводят к поиску минимума лосса. Как правило, лосс либо придумывают сообразно поставленной задаче, либо заимствуют функцию, которую традиционно используют в задачах, аналогичных решаемой.

В приведенном примере лоссом является функция *L*, но на практике часто лоссом называют также и сумму *S* значений *L*. Если от нейросети требуется достижение заданных значений сразу по нескольким измеряемым отдельно друг от друга параметрам, то для каждого параметра определяется своя функция, а лосс составляют в виде суммы этих функций с заданными коэффициентами, выражающими относительную важность этих параметров друг относительно друга (подобно тому, как на соревнованиях по фигурному катанию судьи отдельно оценивают артистизм и исполнение акробатических приемов).

Обратим внимание также на то, что для того чтобы лосс мог адекватно отразить, насколько хорошо отработала нейросеть на входных данных, у него должна быть информация о том, какой результат мы считаем правильным. В практике машинного обучения применяются некоторые алгоритмы, которые не требуют наличия размеченных данных, т. е. не требуют от человека уметь решать задачу, которую он ставит нейросети. Вместе они собирательно называются «обучением без учителя». К ним относится, например, генетический алгоритм. Методы и обходы, требующие наличия размеченных данных, называют, соответственно, «обучением с учителем».

Наиболее часто возникающая задача из тех, которые решаются методами обучения без учителя, это задача кластеризации, т. е. задача, в которой от машины требуется выдумать критерий для разделения данных на несколько групп, в каждой из которых данные будут обладать схожими свойствами. В целом тема обучения без учителя гораздо беднее, многие из методов обучения без учителя относятся скорее к классической математике, а почти все достижения в области искусственного интеллекта, напротив, относятся к обучению с учителем. Другими словами, для каждой единицы входных данных *x*, используемой при обучении, мы должны предоставить ожидаемый нами идеальный результат работы нейросети $\hat{y}$, чтобы сравнить с ним реально полученное значение *y*. Такая информация называется разметкой.

Разметка – дополнительная информация в датасете (помимо входных данных), представляющая собой идеальный результат работы нейросети для каждой единицы входных данных в датасете и используемая при вычислении лосса.

С учетом того, что значение лосса зависит все же не только от результата работы нейросети, но также и от разметки и тех данных, на которых ее запускали, более правильные выражения для лосса *L* и суммы *S* будут выглядеть следующим образом (*рисунок 2*).

$$S(w, \hat{y}_1, \hat{y}_2, \dots, x_1, x_2, \dots) = \sum_i L(\hat{y}_i, f(w, x_i))$$

Рисунок 2. Формула сумм S.

Поскольку сами нейросети могут быть очень сложны, и набор весов *w* может состоять из миллиардов чисел, то минимизация функции *S* на практике не может быть достигнута путем упражнений с формулами на бумаге, эту задачу можно решить только численно. Для численного решения задачи поиска минимума функции существует метод, называемый градиентным спуском, и, как и любой численный метод, он работает только для аналитичных функций. Огромная доля всего машинного обучения основана именно на градиентном спуске.

Градиентный спуск – численный метод нахождения минимума функций, широко применяемый в машинном обучении.

Разберемся с тем, как он устроен. Допустим, мы ищем минимум функции *f* одной переменной, для которой у нас нет формульной записи, т. е. единственное, что мы с ней можем делать, это узнавать ее значение в выбранной точке. Если предположить, что функция аналитична, то можно и предполагать, что недалеко от любой точки она в целом ведет себя похоже на касательную к ее графику в этой точке. Следовательно, по углу наклона касательной (т. е. по производной) можно определить направление (влево или вправо на графике), в котором функция уменьшается. Постепенно переходя от точки к точке, в какой-то момент можно оказаться около точки минимума. Этот процесс можно представлять себе так: мы «встаем» на график функции в произвольной точке и ставим на него шарик, куда шарик покатится – в том направлении шагнем, снова поставим шарик и т. д. Процесс отражен на *рисунке 3*.

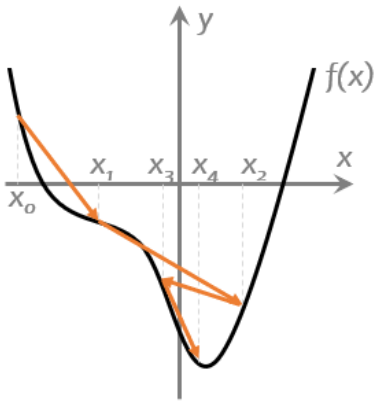


Рисунок 3. Шарик на функции.

Нетрудно заметить: для того, чтобы процесс действительно приближал нас к точке минимума, по крайней мере нужно, чтобы шаг, с которым мы двигаемся вдоль оси *x*, не был одинаковый, иначе в какой-то момент мы зациклимся и будем прыгать между двумя точками, находящимися по разные стороны минимума (на рисунке 3 это точки x_2 и x_4). Как же выбрать величину шага?

На *рисунке 4* изображены графики трех функций. Если допустить, что мы начали градиентный спуск из верхней точки графика f_1, то мы можем случайно перепрыгнуть узкую «яму» с минимумом и оказаться в соседней (слева), более широкой яме. Чтобы не перепрыгнуть яму, можно выбрать размер шага поменьше. Однако такая логика вряд ли приведет нас к результату, если мы применим градиентный спуск к функции f_2, поскольку у ее графика много ям, и если мы будем шагать с маленьким шагом, то мы окажемся в первой попавшейся яме, которая совершенно не обязательно будет содержать нужный нам минимум функции. Чтобы решить эту проблему, можно попробовать запускать градиентный спуск из многих разных точек и сравнивать результат, но для такой функции, как f_3, данный подход займет слишком много времени для изучения одного только изображенного на рисунке участка функции, на котором и вовсе минимумов нет.

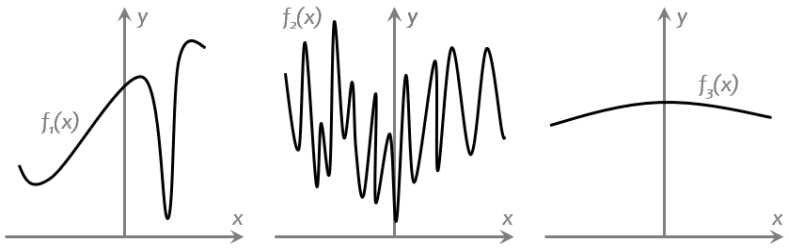


Рисунок 4. Графики трех функций.

Как видно, вопрос логики применения градиентного спуска достаточно сложен. На самом деле можно придумать массу самых различных ухищрений для определения величины шага градиентного спуска на каждой итерации или числа начальных точек, но ни одно из них не будет работать во всех случаях. Другими словами, в общем случае нет идеальной стратегии применения градиентного спуска. Существует масса различных именованных модификаций градиентного спуска, еще больше неименованных и бесчисленное множество комбинаций и тех, и других. Все их объединяет то, что они до какой-то степени рассчитывают на аналитичность функции. Задача специалиста по машинному обучению как раз и заключается в том, чтобы подобрать по возможности более аналитичные нейросеть *f* (это помимо того, что она еще должна каким-то образом решать целевую задачу) и лосс *L* и наиболее эффективный вариант градиентного спуска. При этом по мере того, как человечество решает очередной класс задач с помощью машинного обучения, у мирового сообщества накапливается опыт, из которого постепенно складываются шаблоны, примеры и рекомендации для выбора всего перечисленного. Тем не менее, этот выбор все равно бывает неоднозначным, и специалисту приходится время от времени упражняться в своей математической интуиции, и тем больше, чем необычнее задача.

Внесем ясность в то, как применяется градиентный спуск для функции *f*, у которой аргумент *x* состоит, как минимум, из нескольких значений. Для этого необходимо освежить в памяти понятие частной производной: это на самом деле обыкновенная производная функции, посчитанная в условиях, когда все аргументы, кроме одного, на время вычислений считаются как бы не переменными, а просто числами, в таком случае говорят «частная производная по такому-то аргументу», вводят соответствующее обозначение. Например, на *рисунке 3.5* приведена функция от двух переменных и две ее производные по каждому из аргументов. Вектор, составленный из производных функции по каждому из ее аргументов, и называется градиентом, мы будем обозначать его просто штрихом над функцией, как обозначаем у функций одной переменной обычную производную.

$$f(x,y) = sin(2x)cos(3y)$$
$$f'_x = 2\,cos(2x)cos(3y)\,, f'_y = -3sin\,(2x)\,sin(3y)$$
$$f' = (\,2\,cos(2x)cos(3y)\,, -3sin\,(2x)\,sin(3y)\,)$$

Рисунок 5. Функция от двух переменных и две ее производные по каждому из аргументов.

Градиент обладает тем свойством, что в многомерном пространстве аргументов функции он указывает направление скорейшего роста функции (т. е. ровно противоположеное тому, которое нужно для поиска минимума). Отсюда можно вывести формулу (*рисунок 6*) для простейшего варианта градиентного спуска применительно к сумме *S*, в ней в качестве нового значения весов *w* принимается старое значение *w*, из которого вычли градиент, умноженный на некоторую величину *a*, определяемую каким-то образом на каждом шаге или фиксированную, задающую размер шага градиентного спуска. Процесс может начинаться с произвольной точки w_0.

$$w \leftarrow w - a \cdot S'(w)$$

Рисунок 6. Формула для простейшего варианта градиентного спуска применительно к сумме S.

Заметим, что у нейросети, имеющей миллиарды весов, число локальных минимумов может быть гигантским. Искать среди них «самый минимальный» минимум нецелесообразно, достаточно выбрать какой-то, в котором значение *S* уже достаточно мало, т. е., другими словами, остановиться на значениях весов, при которых нейросеть *f* работает уже вполне удовлетворительно. Тем не менее, даже нахождение такого достаточно эффективного набора весов, как правило, требует многих тысяч, миллионов или даже миллиардов итераций градиентного спуска. Как правило, итерации градиентного спуска группируют так, чтобы в процессе обучения использовалась каждая единица данных из датасета до того, как любая единица данных будет использована в следующий раз.

Эпоха – несколько итераций градиентного спуска, в течение которых все данные из датасета были использованы ровно по одному разу. Как правило, число итераций градиентного спуска исчисляют в эпохах.

Вернемся к формуле (*рисунок 2*) и обратим внимание, что значение суммы *S* все же зависит не только от весов, но также и от значений входных данных. Если в нашем случае мы будем обучать нейросеть, применяя всегда один и тот же набор значений $x_1, x_2, \dots x_N$, то получится, что подберем хорошие веса конкретно для этих значений входных данных, но не гарантируем, что на других входных данных нейросеть сработает хорошо. Для решения этой проблемы при обучении нейросетей на разных шагах градиентного спуска используют разные входные данных из датасета. При этом на одном шаге градиентного спуска, как правило, используются несколько единиц входных данных (иначе бы сумма *S* просто равнялась *L*).

Батч – набор из нескольких единиц входных данных, взятых для выполнения одного шага градиентного спуска.

Батч-сайз/размер батча – количество единиц данных в батче.

Если выбрать слишком большой батч, то влияние разных слагаемых суммы *S* на конечный результат может слишком усреднить значение градиента. Если, наоборот, выбрать батч слишком маленький, разница между данными в разных батчах может быть настолько велика, что направление шагов градиентного спуска будет от итерации к итерации меняться в произвольном порядке.

В обоих этих крайних случаях процесс градиентного спуска получается бесполезным. Здесь уместна следующая аналогия: если учащемуся преподавать некоторую математическую дисциплину на занятиях длиной по 10 часов, то к концу занятия он уже не будет помнить, что ему объяснили в начале. Напротив, если занятия по нескольким дисциплинам сделать длиной пять минут и постоянно их чередовать, то постоянные переключения с одной дисциплины на другую превратят весь преподанный материал в кашу в памяти обучающегося, от которой также не будет толку. По этой причине на практике выбор правильного размера батча является одной из важнейших настроек процесса машинного обучения.

Вспомним снова, что нейросеть может содержать многие миллионы весов. Чтобы сделать один шаг градиентного спуска, нужно посчитать градиент, т. е. производные по каждому из значений весов. Вычисление каждой производной численно потребует посчитать функцию *f* как минимум столько же раз, сколько у нее весов, а записать формульно выражение производной *L* при очень сложной нейросети *f* может быть весьма затруднительно. Поэтому производные *L* вычисляют формульно, но по частям. Для этого пользуются правилом взятия производной сложной функции (*рисунок 7*).

$$f(x), g(x), h(x) = f\big(g(x)\big) \to h'(x) = f'(g(x)) \cdot g'(x)$$

Рисунок 7. Правило взятия производной сложной функции.

Другими словами, для того чтобы найти производную какой-то сложной функции *h*, нужно действовать так:

1) представить сложную функцию *h* в виде комбинации применения двух функций *f* и *g*: *h(x)=f(g(x))*;
2) определить производную *g'(x)* функции *g(x)*;
3) определить производную *f '(x)* функции *f(x)* и подставить в нее в качестве аргумента *g(x)*;
4) перемножить два выражения, полученных на двух предыдущих шагах.

Заметим также, что формульно выражение *f '(g(x))* никак не зависит от функции *g*, в него нужно только подставить значение функции *g* в точке *x*. Можно выполнять вычисления в таком порядке:

сначала посчитать значение *g(x)*, потом значение *f (g(x))*, потом *f '(g(x))* (это можно, ведь значение *g(x)* уже посчитано), потом *g'(x)* и, в конце концов, *h'(x)*. При этом порядке получается, что вычисление значения *h(x)* выполняется в порядке от внутренней функции к внешней, а значения *h'(x)* – от внешней функции к внутренней (подобно тому, как матрешка разбирается от большей куклы к меньшей и собирается от меньшей к большей). Это позволяет применять правило вычисления производной сложной функции «в глубину». Представим себе некоторую сложную нейросеть *f* в виде последовательности применения нескольких функций f_i и значений весов w_i. (*рисунок 8*).

$$L(\hat{y}, f(x, w)) = L(\hat{y}, f_n(w_n, f_{n-1}(w_{n-1}, f_{n-2}(\dots f_1(w_1, x) \dots))$$

Рисунок 8. Сложная нейросеть.

Воспользуемся правилом вычисления производной сложной функции по *x*, для этого мы последовательно посчитаем значения f_1, f_2, f_3 и т. д., а потом последовательно, но в обратном порядке, значения производных *L*', f'_n, f'_{n-1} и т. д. Представим себе, что происходит в момент, когда мы вычисляем производную для некоторой функции f_i, получающей на вход несколько значений x_{ij} и весов w_{iz} и выдающей несколько выходных значений y_{ik}. Согласно правилу вычисления производной сложной функции, производные лосса *L* по x_{ij}, y_{ik} и w_{iz} связаны следующим образом (*рисунок 9*).

$$L'_{w_{iz}} = \sum_k L'_{y_{ik}} \cdot y'_{ik_{w_{iz}}}, L'_{x_{ij}} = \sum_k L'_{y_{ik}} \cdot y'_{ik_{x_{ij}}}$$

Рисунок 9. Правила вычисления производной сложной функции.

В первой формуле последнее выражение означает «производная y_{ik} по весу w_{iz}», во второй формуле последнее выражение означает «производная y_{ik} по весу x_{ij}». эти производные вычисляются из формулы функции f_i, поскольку именно с ее помощью значения y_{ik} вычисляются из значений x_{ij}. Вспомним теперь, что при вычислении производной *L* по *x* можно сначала в прямом порядке посчитать значения функций f_i, а потом в обратном – их производные. Тогда получается, что к моменту вычислений по формуле (рисунок 9) значения производных *L* по y_{ik} уже вычислены. Другими словами, при известном выражении f_i для вычисления производной *L* по x_{ij} и w_{iz} достаточно знать значения производной *L* по y_{ik}. Обратим внимание, что вычисленные значения производных *L* по входным значениям x_{ij} будут для функции f_{i-1} являться значениями производных *L* по ее выходным значениям. Это позволяет после вычислений по формуле (рисунок 9) для функции f_i перейти к аналогичным вычислениям для функции f_{i-1}, потом для f_{i-2}, и так далее. Именно этим способом и вычисляют градиент для очень сложных функций *f*.

Метод обратного распространения ошибки/ back propagation – способ вычисления градиента, заключающийся в том, что нейросеть рассматривают как сложную функцию, состоящую из нескольких простых, и вычисляют производные лосса по весам последовательно, от простой функции к простой функции в порядке от самой внешней к самой внутренней. Каждая такая простая функция называется слоем.

Слой /layer – составной элемент нейросети. Каждая нейросеть состоит из нескольких слоев, чаще всего выстроенных последовательно, а иногда параллельно. При использовании метода обратного распространения ошибки вычисления градиента выполняются послойно.

Невязки слоя – значения производной лосса по выходным значениям слоя. Иногда для удобства значения невязок самого слоя называют выходными невязками (так как они соответствуют выходным значениям слоя), а значения невязок предыдущего слоя – входным невязками (так как они соответствуют входным значениям).

Если выходные значения слоя используются в нескольких разных слоях, то производные лосса по соответствующим выходным значениям слоя будут содержать слагаемые, вычисляемые на основе каждого из зависимых слоев. Для того, чтобы научиться применять метод обратного распространения ошибки, не нужно обладать абсолютно никакими знаниями, достаточно как следует вдуматься в правило вычисления производной сложной функции и поупражняться в нем.

Вспомним снова, что для вычисления значений производных все же сначала нужно посчи-

тать значения результатов вычислений для всех слоев. Эти вычисления выполняются в порядке от первого (вложенного, принимающего входные данные нейросети) к последнему (передающему значения в лосс), а вычисления производных – в обратном порядке (*рисунок 10*). При желании вычисления невязок r_i (зеленые стрелки) могут быть полностью выполнены до вычислений непосредственно значений градиента (оранжевые стрелки).

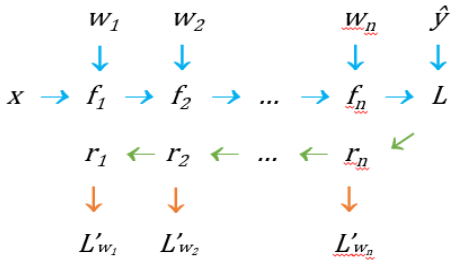


Рисунок 10. Вычисления производных в обратном порядке.

Прямой ход – последовательность вычислений, начинающаяся подстановкой в нейросеть входного значения и заканчивающаяся вычислением значения функции L.

Обратный ход – последовательность вычислений невязок. Иногда к обратному ходу относят также вычисления непосредственно значений градиента.

На практике входные и выходные значения функций чаще всего многомерны, соответственно, многомерны и значения невязок. Практическое составление формулы вычислений обратного хода для некоторого слоя может быть затруднительно, особенно если слой записан не в виде математической формулы, а в виде программного кода. Для самопроверки в этих случаях следует воспользоваться следующим правилом: если некоторое выходное значение на слое зависит от некоторого входного, то между соответствующими им значениями входных и выходных невязок также есть зависимость, но в обратном порядке.

Следует помнить, что для вычислений обратного хода требуются выходные значения всех слоев. Если нейросеть f состоит из большого числа слоев, для ее обучения градиентным спуском потребуется хранить большой объем промежуточных данных. Если, как это чаще всего и бывает, лосс вычисляется по целому батчу входных данных, то промежуточные значения придется хранить одновременно.

Нейросети, применяемые на практике, редко состоят из 2–3 слоев, а гораздо чаще из 10–100. Каждый слой fi может быть реализован в виде вызова функции программы, при чем одинаковые или похожие слои могут реализовываться одной функцией, что делает программную реализацию обратного распространения ошибки особенно удобной. Деление на слои, как правило, выбирается исходя из удобства их программной реализации, однако теоретически оно может быть каким угодно. На практике в мировом сообществе для каждого типа нейросетей со временем устаканивается общее представление о типовых слоях, но это однозначно происходит только для наиболее простых конструкций, в связи с чем разные разработчики могут называть слоями разные вещи и по-разному делить одну и ту же нейросеть на слои.

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: _ 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/ (access date: 01/16/2026).

[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 (accessed: 01/15/2026).

[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: 01/11/2026).

[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 (access date: 01/10/2026).

[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: 01/13/2026).

[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: 01/11/2026).

[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: 01/11/2026).

[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 (access date: 01/12/2026).

[9] Lavin A., Scott Gray Fast Algorithms for Convolutional Neural Networks. 2015. URL: https://arxiv.org/abs/1509.09308 (access date: 01/11/2026).

[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 (accessed: 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 (accessed: 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/ (accessed: 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: 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 (accessed: 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 (accessed: 11.01.2026).

[16] Official website of ONNX. URL: https://onnx.ai/.

[17] Fundamentals of Artificial Intelligence Technologies: a tutorial / A.A. Semenov, V.V. Gonchar, A.S. Erdniev. - Moscow University of the Ministry of Internal Affairs of Russia named after V. Ya, Kikot, 2025. - 190 p.

[18] Financial Monitoring: a tutorial / V.I. Glotov, A.U. Albekov, O.N. Tisen et al. - KnoRus, 2022. - 198 p.

[19] Tisen O. N. Counteracting modern techniques and methods of recruitment into international terrorist organizations // Russian Justice. 2018, No. 9. pp. 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. No. 1. pp. 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. No. 4. pp. 25-29.

