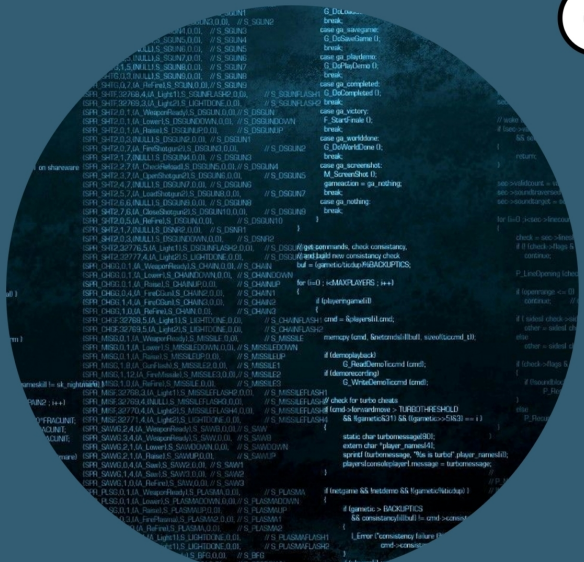
Shtoltc Eugeny

0+



# Machine Learning practice

from PyTorch model to Kubeflow in the cloud for BigData

**Eugeny Shtoltc**

# Machine learning in practice – from PyTorch model to Kubeflow in the cloud for BigData

*http://www.litres.ru/pages/biblio_book/?art=63595623*
*SelfPub; 2020*

## Аннотация

In this book, the Chief Architect of the Department of Architecture and Management of Technical Architecture (Cloud Native Competence Center and the Corporate University of Architects) of Sberbank shares his knowledge and experience with readers in the field of ML. received in the work of the School of Architects and. Author: * guides the reader through the process of creating, learning and developing a neural network, showing in detail with examples * increases horizons, showing how it can take place in BigData from the point of view of the Architect * introduces real models of use in the product environment

# Содержание

# Eugeny Shtoltc
# Machine learning in practice – from PyTorch model to Kubeflow in the cloud for BigData

## About the book

The book is structured like a textbook – from simple to complex. The reader will be able to:

* in the first three chapters, create the simplest neural network for image recognition and classification,

* in the following – to delve into the device and architecture for optimization,

* further expand the understanding of the company's ecosystem as a whole, in which neural networks operate, as its integral part,

and she interacts with and uses surrounding technologies,

* finish the study by deploying a full-scale production system in the full-cycle cloud.

Almost every chapter begins with the general information

needed for the practical part that follows. In the practical part:

* demonstrates the process of preparing the environment, but more often free ready-made cloud services are used,

* demonstrates the writing process when with a parsing of the written and an overview of alternative solutions,

* analysis of the result and the formation of a vision of options for further development.

The book consists of sections:

* Introduction to Machine Learning. This is the only chapter without a practical part to get you started.

understanding the limits of their applicability, advantages over other methods and their general structure for beginners. Also produced

classification of neural networks according to the principles laid down in them, and the selection of a group, which will be discussed in the book.

* Basics for writing networks. It provides the basic knowledge necessary to write the first network in PyTorch, familiarity with the development environment

Jupyter in the Google Colab cloud service, which is a simplified version of the Google ML cloud platform, running the code in it and using the PyTorch framework for writing neural networks.

* We create the first network. The author demonstrates for the reader's practice how to create a simple neural network on PyTorch in

Colab with a detailed analysis of the written code, training it on the MNIST image dataset and launch it.

* Improving the recognition of the neural network on complex images. Here the author demonstrates to the reader not practice

training and prediction of neural networks for color pictures, methods to improve the quality of network predictions. Understands in detail

device, pitfalls in writing and training effective neural networks.

* Modern architectures of neural networks. The architectural principles used in modern neural networks for

improving the quality of predictions. An analysis of various neural network architectures that have made a breakthrough in the quality of training is given.

and brought approaches. Various architectural universal quality enhancement patterns such as ensemble are discussed.

* Using pre-trained networks. The use of already trained layers in their networks is demonstrated.

* ML scaling. Examples of preparing the environment for launching them in a cloud infrastructure are given.

* Receiving data from BigData. It tells how you can connect to various sources from Jupyter

data, including BigData, for training models.

* Big data preparation. This section describes BigData technologies such as Hadoop and Spark, which

are data sources for training models.

* ML in an industrial environment. This section covers systems such as Kubeflow and MLflow. The reader can try to expand

platform, set up the learning process and run the model in the cloud, as is done in companies.

# Introduction Machine Learning

Artificial intelligence is a field at the intersection of many sciences. One of the ways to achieve it is machine learning, when we supply it with data and, on its basis, we learn how to find solutions and identify patterns and data that were not there before. For training, statistical algorithms can be used, for example, in the R language, depth-first search in Prolog or breadth-first search in Refal, as well as adaptive structures – neural networks. Neural networks, depending on the tasks, will hide according to different principles, have a structure and learn in different ways. Recently, the greatest breakthrough has been received by neural networks of data representation (Representation learning), which are engaged in identifying patterns in information, since they cannot remember the information itself due to its size. Deep neural networks with many levels of features give great effects, features at subsequent levels are built on the basis of features from previous levels, which will be discussed in this chapter.

Machine learning (ML) refers to the adaptation of the transformation of input data into outputs, depending on the history of decisions. This class of problems is solved either in an algorithmic way, or with the help of neural networks, and we will talk about where and in what situation it is better to apply what solution. For example, let's take uppercase numbers from zero to

nine, which we will compare with printed ones. If the uppercase letters exactly fall into the contour, then everything is simple, we just need to go through the contours of the printed ones and get a suitable option. Such a task is not relevant to machine learning tasks. Now let's complicate the task – our numbers don't exactly match the pattern. If the uppercase numbers do not fit a little into the contour, we just find some kind of deviation. And this is where the difficulty arises when categorizing an uppercase number into zero and nine, when the size of the tail separates the careless spelling of zero from nine. Another point in the categorization of eight and nine. So, if the tip is bent, this is ten, and if it is bent and touches, then it is eight. To solve such a situation, you need to divide the figure into areas and depending on and assign them different coefficients. So, the connection of the tail of the lower part has a very high value than the shape of the circles themselves in the classification into eights and nines. Statistical data on a pre-given sample of the correspondence of figures to eights and nines will help to determine, where the researcher will be able to determine when it is already possible to calculate the lower ring closed and talk about the correspondence of the figure eight, and when not, talk about the correspondence of the nine. But we can programmatically divide the numbers into sectors and assign coefficients to them.

Another difficulty may be that the digit may not be in the observed area, but in an arbitrary one, for example, in a corner. To analyze the digit itself, we need to move the analyzing window

to the place where the digit is. For simplicity, for now, we will assume that the dimensions of the analyzing window are equal to the dimensions of the figure under study. To solve this problem, an analyzing layer is put in front of the network, which forms a map of finding the numbers. The task of this layer is to determine the location of the number in the picture. Let's take a black image on a white sheet for simplicity. We need to go through the digit analyzer line by line throughout the sheet and determine the locations. Let us take the black area on the indicator as an indicator. After passing over a piece of paper and determining the area, we get a matrix with the numbers of the areas in black. Where there are more areas of black color, in that place the figure fits into the indicator as much as possible. Converting an image into a matrix of areas is called a convolution operation, and if it is performed by a neural layer, it is called a convolution layer. Neural networks that have a Conv Layers are called Convolutional Neural Networks (CNNs). Such networks are used in image recognition, now they have been adapted for speech recognition. The principle of operation was borrowed from the biological optic nerve.

If the image is not only in the derived place, but there are other images, then to determine and it will take several layers of the neural network to perform the determination, the result of which will also be a map of the location of the digit, but making a decision about its location needs to be identified. Thus, the first layer will have the number of neurons displaying maps,

which horizontally and vertically will correspond to the width and height of the minute leaf, corresponding to the width and height of the analyzing screen, divided by the step of shifting the analyzing window. The dimension of the second layer in neurons is equal to the dimension of the analyzed window in order to be able to identify the digit. If we make connections from all the neurons of the search layer to the analysis window layer, then the network at the output will get a set of images poured together. The next layer will be measured by the number of analyzed digit elements. For example, a figure can be represented as an incompletely filled eight, then there will be seven segments to be painted. All neurons in the convolutional layer will be connected to all neurons in the digit segment analysis layer. The task of the neuron of this layer is to be connected with the neurons of the previous one, responsible for this segment and to give the result of the presence or absence of this segment in the digital. The next layer has ten neurons, corresponding to numbers from zero to nine. In total, its neurons are connected to the previous layer and are activated when receiving signals from them. So, the neuron branching for the number one will be activated if it receives information that the two extreme right sectors will be active and all the others are not active.

At the output, we will receive the activation of that output neuron that corresponds to a certain number. It does this on the basis of data received from neurons from the previous layer, which are responsible for the digit sectors, namely from which

neurons the signals came and which ones did not. Let's take that the incoming signals from neurons through the connections are zero, that is, the sector is not filled, if one, then the sector is painted. Then, the weights of the links from the right sectors are half, which will give one, while the rest have negative weights, which will not give one at the output if some other sector is activated. At the output of the neuron, there is a normalizer that decides for the decision. He needs to decide, based on the input data and weights, to give one or zero. To do this, it multiplies the input data by the weights, adds them, and outputs one or zero based on the threshold value. This normalizer is needed so that, after summing up the information coming from neurons, it will transfer logical information to the next layer of neurons, the degree of importance of which will be determined by the weights on the receiving neuron, and not by them. For this, functions are used that convert the entire range of input signal levels to the range from zero to one. Such a function is called activation functions and is selected for the entire neural network. There are many functions that consider everything less than one to be zero. The weights themselves are not encoded, but are selected during training. Teaching is either supervised or ansupervised and suitable for different classes of tasks. When teaching without a teacher (auto-encoders and generating networks), we give data to the input of the neurons of the network about the expectation when it itself finds some patterns, while the data is not labeled (does not have any labels by classification), which will allow

us to identify previously unknown features, similarities and differences, and classifies according to not yet found signs, but how this will happen is difficult to predict. For most tasks, we need to get a classification according to the given groups, for which we input a training set with marked data containing labels about the correct solution (for example, classification, and try to achieve a match with this test set. It can also be reinforced), in which the network tries to find the best solution based on incentives, for example, when playing to achieve superiority over an opponent, but for now, we will postpone consideration of this learning strategy for later. When teaching with a teacher, much less attempts to pick up the weight are required, but still it is from several hundred to tens of thousands, while the network itself contains a huge number of connections. In order to find the weights, we select them by selection and directed refinement. With each pass, we reduce the error, and when the accuracy suits us, we can submit a test sample to validate the quality of the cut ( the network could learn badly or retrain), then you can use the network. However, the numbers may be slightly skewed, but because we are highlighting the areas, this does not greatly affect the accuracy.

When a neuron is trained with a teacher, we send training signals to it and get results at the output. For each input and output signal, we receive a result about the degree of error in prediction. When we ran all the training signals, we got a set (vector) of errors that can be represented as a function of errors.

This error function depends on the input parameters (weights) and we need to find the weights at which this error function becomes minimal. To determine these weights, the Gradient Descent algorithm is used, the essence of which is to gradually move to the local target, and the direction of movement is determined by the derivative of this function and the activation function. The activation function is usually sigmoid for regular networks or truncated ReLU for deep networks. Sigmoid outputs a range from zero to one at all times. The truncated ReLU still allows for very large numbers (information is very important) at the input to transfer more than one to the output, there they themselves affect the layers that follow immediately after. For example, the dot above the dash separates the letter L from the letter i, and the information of one pixel affects the decision making at the output, so it is important not to lose this feature and transfer it to the last level. There are not so many varieties of activation functions – they are limited by the requirements for ease of training when it is required to take a derivative. So the sigmoid f after arbitrarily turns into f (1-f), which is effective. With Leaky ReLu (truncated ReLu with leakage) it is even simpler, since it takes the value 0 at x <0, then its wired in this section is also 0, and at x> = 0 it takes 0.01 * x, which with the derivative will be will be 0.01, and for x> 1 it takes 1 + 0.01 * x, which gives 0.01 for the derivative. Calculation is not required here at all, so learning is much faster.

Since we send the sum of the products of signals by their

weights to the input of the activation function, then conceived, we need a different threshold level than from 0.5. We can shift it by a constant, adding it to the sum at the input to the activation function using the bias neuron to remember it. It has no inputs and always outputs one, and the offset itself is set by the weight of the connection with it. But, for multi-neural networks, it is not required, since the weights themselves by the previous layers are adjusted to such a size (smaller or negative) in order to use the standard threshold level – this gives standardization, but requires a larger number of neurons.

When training a neuron, we know the error of the network itself, that is, on the input neurons. Based on them, you can calculate the error in the previous layer and so on up to the input – which is called the Backpropagation method.

The learning process itself can be divided into stages: initialization, learning itself, and prediction.

If our figure can be of different sizes, then pooling layers are applied, which scale the image down. Which algorithm will calculate what will be written when merging depends on the algorithm, usually this is the "max" function for the "max pooling" or "avg" algorithm (mean-square value of neighboring matrix cells) – average pooling.

We already have a few layers. But, in neural networks used in practice, there can be a lot of them. Networks with more than four layers are commonly called deep neural networks (DML, Deep ML). But, there can be a lot of them, so there are 152 of

them in ResNet and this is far from the deepest network. But, as you have already noticed, the number of layers is not taken, according to the principle, the more the better, but prototyped. An excessive amount degrades the quality due to attenuation, unless certain solutions are used for this, such as data forwarding with subsequent summation. Examples of neural network architectures include ResNeXt, SENet, DenseNet, Inception-Res Net-V2, Inception-V4, Xception, NASNet, MobileNet V2, Shuffle Net, and Squeeze Net. Most of these networks are designed for image analysis and it is the images that often contain the greatest amount of detail and the largest number of operations is assigned to these networks, which determines their depth. We will consider one of such architectures when creating a number classification network – LeNet-5, created in 1998.

If we need not just to recognize a number or a letter, but their sequence, the meaning inherent in them, we need a connection between them. For this, the neural network, after analyzing the first letter, poisons the result of the analysis of the current one along with the next letter to its input. This can be compared to dynamic memory, and a network that implements this principle is called recurrent (RNN). Examples of such networks (with feedbacks): Kohonen's network, Hopfield's network, ART-model. The recurrent network analyzes text, speech, video information, translates from one language into another, generates text descriptions for images, generates speech (WaveNet MoL, Tacotron 2), categorizes texts by content (belonging to spam).

The main direction in which researchers are working in an attempt to improve in such networks is to determine the principle by which the network will decide which, for how long and how much the network will take into account the previous information in the future. Networks adopting specialized tools for storing information are called LSTM (Long-short term memory).

Not all combinations are successful, some only allow solving narrow problems. As the complexity increases, a smaller percentage of possible architectures are successful and bear their own names.

In general, there are networks that are fundamentally different in structure and principles:

* direct distribution networks
* convolutional neural networks
* recurrent neural networks
* autoencoder (classic, thin, variational, noise canceling)
* networks of trust ("deep belief")
* generative adversarial networks – opposition of two networks: generator and evaluator
* neural Turing machines – a neural network with a block of memory
* Kohonen neural networks – for unsupervised learning
* various architectures of circular neural networks: Hopfield neural network, Markov chain, Boltzmann machine

Let us consider in more detail the most commonly used, namely, feedforward, convolutional and recurrent networks:

Direct distribution networks:

* two entrances and one exit – Percetron (P)

* two inputs, two fully connected neurons with an output and one output – Feed Forward (FF) or Redial Basics Network (RBN)

* three inputs, two layers of four fully connected neurons and two Deep Feed Forward (DFF) outputs

* deep neural networks

* extreme propagation network – a network with random connections (neural echo network)

Convolutional neural networks:

* traditional convolutional neural networks (CNN) – image classification * unfolding neural networks – image generation by type * deep convolutional inverse graphic networks (DCEGC) – connecting convolutional and unrolling neural networks to transform or combine images

Recurrent neural networks:

* recurrent neural networks – networks with memory in neurons for sequence analysis, in which the sequence

matters such as text, sound and video

* Long Short Term Memory (LSTM) networks – the development of recurrent neural networks in which neurons can

classify data that are worth remembering into long-lived memory from those that are worth forgetting and delete information

from their memory

* deep residual networks – networks with connections between layers (similar in work to LSTM)

* recruited recute neurons (GRU)

# Basics for writing networks.

Until 2015, scikit-learn was leading by a wide margin, which Caffe was catching up with, but with the release of TensorFlow, it immediately became the leader. Over time, only gaining a gap from two to three times by 2020, when there were more than 140 thousand projects on GitHub, and the closest competitor had just over 45 thousand. In 2020, Keras, scikit-learn, PyTorch (FaceBook), Caffe, MXNet, XGBoost, Fastai, Microsoft CNTK (CogNiive ToolKit), DarkNet and some other lesser known libraries are located in descending order. The most popular are the Pytorch and TenserFlow libraries. Pytorch is good for prototyping, learning and trying out new models. TenserFlow is popular in production environments and the low-level issue is addressed by Keras.

* FaceBook Pytorch is a good option for learning and prototyping due to the high level and support of various

environments, a dynamic graph, can give advantages in learning. Used by Twitter, Salesforce.

* Google TenserFlow – originally had a static solution graph, now dynamic is also supported. Used in

Gmail, Google Translate, Uber, Airbnb, Dropbox. To attract use in the Google cloud for it

Google TPU (Google Tensor Processing Unit) hardware processor is being implemented.

* Keras is a high-level tweak providing more abstraction for TensorFlow, Theano

or CNTK. A good option for learning. For example, he

allows you not to specify the dimension of layers, calculating it yourself, allowing the developer to focus on the layers

architecture. Usually used on top of TenserFlow. The code on it is maintained by Microsoft CNTK.

There are also more specialized frameworks:

* Apache MXNet (Amazon) and a high-level add-on for it Gluon. MXNet is a framework with an emphasis on

scaling, supports integration with Hadoop and Cassandra. Supported

C ++, Python, R, Julia, JavaScript, Scala, Go and Perl.

* Microsoft CNTK has integrations with Python, R, C # due to the fact that most of the code is written in C ++. That all sonova

written in C ++, this does not mean that CNTK will train the model in C ++, and TenserFlow in Python (which is slow),

since TenserFlow builds graphs and its execution is already carried out in C ++. Features CNTK

from Google TenserFlow and the fact that it was originally designed to run on Azure clusters with multiple graphical

processors, but now the situation is leveled and TenserFlow supports the cluster.

* Caffe2 is a framework for mobile environments.

* Sonnet – DeepMind add-on on top of TensorFlow for training super-deep neural networks.

* DL4J (Deep Learning for Java) is a framework with an emphasis on Java Enterprise Edition. High support for BigData in Java: Hadoop and Spark.

With the speed of availability of new pre-trained models, the situation is different and, so far, Pytorch is leading. In terms of support for environments, in particular public clouds, it is better for the farms promoted by the vendors of these clouds, so TensorFlow support is better in Google Cloud, MXNet in AWS, CNTK in Microsoft Azure, D4LJ in Android, Core ML in iOS. By languages, almost everyone has common support in Python, in particular, TensorFlow supports JavaScript, C ++, Java, Go, C # and Julia.

Many frameworks support TeserBodrd rendering. It is a complex Web interface for multi-level visualization of the state and the learning process and its debugging. To connect, you need to specify the path to the "tenserboard –logdir = $ PATH_MODEL" model and open localhost: 6006. Interface control is based on navigating through the graph of logical blocks and opening blocks of interest for subsequent repetition of the process.

For experiments, we need a programming language and a library. Often the language used is a simple language with a low entry threshold, such as Python. There may be other general-purpose languages like JavaScript or specialized languages like R. I'll take Python. In order not to install the language and libraries, we will use the free service colab.research.google.com/

notebooks/intro.ipynb containing Jupiter Notebook. Notebook contains the ability not only to write code with comments in the console form, but to format it as a document. You can try Notebook features in the educational playbook https://colab.research.google.com/notebooks/welcome.ipynb, such as formatting text in the MD markup language with formulas in the TEX markup language, running scripts in Python, displaying the results of their work in text form and in the form of graphs using the standard Python library: NumPy (NamPay), matplotlib.pyplot. Colab itself provides a Tesla K80 graphics card for 12 hours at a time (per session) for free. It supports a variety of deep machine learning frameworks, including Keras, TenserFlow, and Pytorch. The price of a GPU instance in Google Cloud:

* Tesla T4: 1GPU 16GB GDDR6 0.35 $ / hour
* Tesla P4: 1GPU 8GB GDDR5 0.60 $ / hour
* Tesla V100: 1GPU 16GB HBM2 2.48 $ / hour
* Tesla P100: 1GPU 16GB HBM2 $ 1.46 / hour

Let's try. Let's follow the link colab.research.google.com and press the button "create a notepad". We will have a blank Notebook. You can enter an expression:

```
10 ** 3/2 + 3
```

and clicking on play – we get the result 503.0. You can display the graph of the parabola by clicking the "+ Code" button in the new cell in the code:

```
def F (x):
```

```
return x * x
import numpy as np
import matplotlib.pyplot as plt
x = np.linspace (-5, 5, 100)
y = list (map (F, x))
plt.plot (x, y)
plt.ylabel ("Y")
plt.xlabel ("X")
```

Or displaying an image as well:

```
import os
! wget https://www.python.org/static/img/python-logo.png
import PIL
img = PIL.Image.open ("python-logo.png")
img
```

Popular frameworks:

* Caffe, Caffe2, CNTK, Kaldi, DL4J, Keras – a set of modules for design;

* TensorFlow, Theano, MXNet – graph programming;

* Torch and PyTorch – register the main parameters, and the graph will be built automatically.

Consider the PyTorch library (NumPy + CUDA + Autograd) because of its simplicity. Let's look at operations with tensors – multidimensional arrays. Let's connect the library and declare two tensors: press + Code, enter the code into the cell and press execute:

```
import torch
```

a = torch.FloatTensor ([[1, 2, 3], [5, 6, 7], [8, 9, 10]])

b = torch.FloatTensor ([[– 1, -2, -3], [-10, -20, -30], [-100, -200, -300]])

Element-wise operations such as "+", "-", "*", "/" on two matrices of the same dimensions perform operations with their corresponding elements:

a + b

tensor ([[0., 0., 0.],

[-5., -14., -23.],

[-92., -191., -290.]])

Another option for the elementwise operation is to apply one operation to all elements one by one, for example, multiply by -1 or apply a function:

a

tensor ([[1., 2., 3.],

[5., 6., 7.],

[8., 9., 10.]])

a * -1

tensor ([[-1., -2., -3.],

[-5., -6., -7.],

[-8., -9., -10.]])

a.abs ()

tensor ([[1., 2., 3.],

[5., 6., 7.],

[8., 9., 10.]])

There are also convolution operations, such as sum, min, max,

which, as input, give the sum of all elements, the smallest or largest element of the matrix:

```
a.sum ()
tensor (51.)
a.min ()
tensor (1.)
a.max ()
tensor (10.)
```

But, we will be more interested in post-column operations (the operation will be performed on each column):

```
a.sum (0)
tensor ([14., 17., 20.])
a.min (0)
torch.return_types.min (values = tensor ([1., 2., 3.]), indices = tensor ([0, 0, 0]))
a.max (0)
torch.return_types.max (values = tensor ([8., 9., 10.]), indices = tensor ([2, 2, 2]))
```

As we remember, a neural network consists of three layers, a layer of neurons, and a neuron contains connections at the input with weights in the form of prime numbers. The weight is set by an ordinary number, then the incoming connections to the neuron can be described by a sequence of numbers – a vector (one-dimensional array or list), the length of which is the number of connections. Since the network is fully connected, all the neurons of this layer are connected to the previous one,

and therefore the vectors demonstrating them also have the same length, creating a list of vectors of equal length – a matrix. It is a convenient and compact layer representation optimized for use on a computer. At the output of the neuron, there is an activation function (sigmoid or, ReLU for deep and ultra-deep networks), which determines whether the neuron outputs a value or not. To do this, it is necessary to apply it to each neuron, that is, to each column: we have already seen the operation on columns.

# Accelerating learning

These operations are used for convolutions, which take over 99% of the time and therefore there are specialized tools for their optimization. The calculations themselves are performed not in Python, but in C – Python only calls the API of low-level math libraries. Since such computations are easily parallelized, processors designed for parallel image processing (GPU) are used instead of general-purpose processors (CPUs). So, if a PC has from 2 to 8 cores in a processor, and a server has from 10 to 20 cores, then in a GPU there are hundreds or thousands of highly specialized for processing matrices and vectors. The most popular standard for the group of drivers providing access to the NVidia GPU is called CUDA (Computed Unified Device Architecture), which you can check for support with "lspci | grep-i Nvidia". The alternate is OpenCL promoted by AMD for its GPUs, but development and support in frameworks is rudimentary. For more optimization in processors for ML, special instructions are used that are used in special libraries. For example, Intel Xeon SCalate processors in eight-bit numbers and special pipelines that are activated when using OpenVINO, which gives an increase in speed up to 3.7 times for PyTorch. To speed up the classic ML (classification) XGboost giving an increase of up to 15 times. For now, a low-power CPU is enough for us.

Another type of specialized processor is the reprogrammable processor. So in 2018, Intel introduced a processor with an embedded FPGA (field-programmable gate array) module, developed by the purchased Altera company, in its Intel Xeon SP-6138P. Another major FPGA manufacturer is Xilinx, which created Altera. The idea of programmable logic blocks (field programmable gate arrays) is not new and dates back to long before general purpose processors. The point is not in executing the program on a universal processor, which each time executes an algorithm to solve the task, but in creating a logical architecture of the processor for this task, which is much faster. In order not to order the development and production of an individual microcircuit every time, universal boards are used in which the necessary architecture is created by software. At the time of its creation, ana became a replacement for micro-assemblies, when workers in the production manually placed its elements into a chip. The architecture is achieved by destroying unnecessary links during "sewing", which are built on the principle of a grid, in the nodes of which the necessary elements are located. A popular example is Static RAM, which is used in the BIOS of a computer, prototyping ASICs before mass production, or building the desired controller, such as building an Enthernet controller at home. For programming the controller architecture with a neural network, FPGA controllers are provided by the same Intel and Xilinx using the Caffe and TensorFlow frameworks. You can experiment in the Amazon

cloud. A promising area is the use of edge computing neural networks, that is, on end devices such as modules for unmanned vehicles, robots, sensors and video cameras.

# Конец ознакомительного фрагмента.