Deep Learning and Convolution

Jakub Binda

May 19, 2024

1 We Need To Go Deeper

Figure 1: Internet meme cited in a classic paper introducing GoogLeNet deep convolutional model. [3]

In general, machine learning (statistical learning) was introduced to solve problems without explicitly specifying how to do that. There are plenty of machine learning tasks, like:

- **Classification** - e.g. classifying if patient is healthy or ill based on markers.
- **Regression** - e.g. predicting a house prices based on a house and a location's features.
- **Machine translation** - e.g. translating Polish to ancient Latin.
- **Anomaly detection** - e.g. estimation probability of a transaction to be a fraud.
- **Synthesis and sampling** - e.g. image generation.
- **Denoising** - e.g. sharpening filters in image editors.
- ... and so on

In a classic attempt to a supervised learning, models are given with examples, which consists of features and desired label. Then, to solve a given task model needs to **estimate mapping from the feature space to the output space**. So, our goal is to achieve function estimating the ground truth relation between features and labels. There is a lot of well studied models, which empower us to deal with such problems, with few listed below.

Table 1: Examples of classic (shallow) models.

Name	Type	Task
Linear Regression	linear	Regression
Elastic Net	linear	Regression
Linear Discriminant Analysis	linear	Classification
Quadratic Discriminant Analysis	non-linear	Classification
Logistic Regression	linear	Classification
Decision Trees	non-linear	Classification/Regression

1.1 So, why we need to go deeper?

When we are so lucky to have structured data, with records consisting of rows of experts derived, abstract features, shallow classifiers are off the shelf solution. Breast Cancer Wisconsin Data Set, could be an example of such a dataset. It consist of expert derived cancer features and benign/malignant label.
But what if we don't have predefined features?
Firstly, data annotation by experts is a costly and time consuming process. Secondly, sometimes we don't have an idea how to create features based on raw data. The latter is in particular obvious in a Computer Vision tasks. Assuming we want to develop bird classifier based on bird pictures. How to define/compute "birdish" features based on raw pixel data? It lead us to a paradigm change. **What if we learn not only mapping from the input space to the output space, but also abstract representations of data?** And the representation learning is where deep neural networks gave tremendous improvements.[2][4]

2 Deep Neural Networks

2.1 Multilayer Perceptron (MLP)

The MLP, also called fully connected neural network is the simplest neural network (NN) architecture, where each neuron is connected to all neurons/inputs from the previous layer. MLP example is presented at Figure 2.
In such an architecture, each neuron is a quite simple non-linear transformation, which is applied to the whole layer input. Diving into math, output $H_{l,i}$ of the i-th neuron in the l-th layer is defined as follow:

$$H_{l,i} = \sigma(x^T w_{l,i} + b_{l,i}) \tag{1}$$

where x is a layer input, $w_{l,i}$ and $b_{l,i}$ are neuron parameters, $\sigma()$ is a non-linear function.

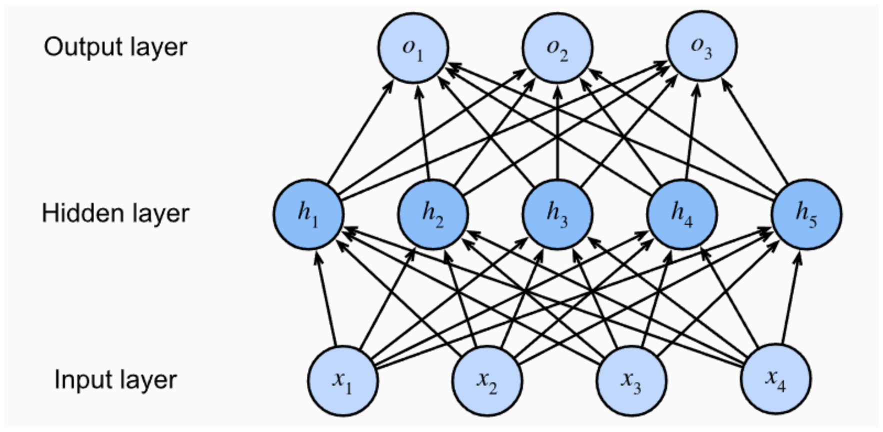

Figure 2: MLP with three layers (input, hidden and output layer). The scheme borrowed from D2L.

Equation 1 clearly indicates that each neuron performs firstly affine transformation parameterized by weights vector and bias, then non-linear transformation with a non-linear function. So, we can interpret the whole NN as an complex composition of quite simple neural units. Deep Learning is built on an intuitive idea, that complex problem may be solved by sequentially solving simpler problems. In that case, problem is solved layer by layer, where each layer is expected to solve more and more complex sub-problem of the major problem. Layer takes the input and transform it to a new representation (hidden state), which is a complex, non-linear transformation of previous state.
In the context of Computer Vision, first layers may learn to extract simple features like edges from pixel tensors. Then, following layers build more complex features on top of the simpler ones from the previous layer.
It is important to note, that process of emerging features from raw data is purely spontaneous! NN by itself learns to extract useful representations. It also worth to note, that it is an intuitive way of NN interpretation and quite naive. It is not always true that each layer learns perfectly separate, and readily interpretable features, as NN is optimized as a whole.

2.1.1 Why we need a non-linearity?

Is it possible to build neural network only with affine transformations? Technically yes, but such a NN has capability/flexibility similar to simple linear regression. Explanation for that arises from linear algebra, where it is known fact that composition of affine transformations still has the same complexity as one affine transformation. Below simple example:
Taking a model $F(X)$ with two layers parameterized by weight matrices: $W^{(1)}$, $W^{(2)}$ and bias vectors: $b^{(1)}$, $b^{(2)}$.

$$F(X) = (XW^{(1)} + b^{(1)})W^{(2)} + b^{(2)} = XW^{(1)}W^{(2)} + b^{(1)}W^{(2)} + b^{(1)} = XW + b$$

where $W = W^{(1)}W^{(2)}$ and $b = b^{(1)}W^{(2)} + b^{(2)}$
To overcome that limitation, affine transformations must be separated by a non-linearity. It prevents collapse of NN and increase drastically model capability. Typical non-linearities are ReLU(), Sigmoid(), Tanh(), GeLU().

2.1.2 How it learns?

Similarly to other machine learning models, NN parameters are optimized according to the Loss function. Loss function is a mathematical formula which measures how bad (in some sense) model performs. It may be different for different tasks. For example Mean Squared Error (MSE) for a Regression or CrossEntropy for a Classification. Optimization of NN parameters wasn't a trivial problem. There is not exact solution and it is characterised by a tremendous amount of parameters in a multidimensional space. From that reason NN parameters have to be optimized with gradient based algorithms like Stochastic Gradient Descent, ADAM or RMSProp. Thanks to modern frameworks like PyTorch or Tensorflow computing partial derivatives of Loss, with respect to parameters is currently very easy.

2.2 Convolutional Neural Networks (CNNs)

Convolutional Neural Networks are more sophisticated architectures and they appeared as a state of the art models in a Computer Vision (CV). Actually, development of CNNs was strictly connected to CV and drove innovations in that field. After success in a CV, CNNs were adapted to other areas. Like for example speech recognition. Nowadays CNNs still achieve very good results on image data, however recently Vision Transformers beats CNNs in a Computer Vision tasks. [1]

2.2.1 Why Computer Vision needed novel architecture?

In a previous section I have described simple MLP architecture, which turned out to be not very effective when it comes to image data. Why?
MLP accepts vector as input, so before feeding image to MLP there is a need to flatten image to tensor of an order 1, i.e. vector. This operation destroys internal structure of an image and results in an highly dimensional input. Classical image dataset: ImageNet consist of images of size 224x224x3 (Figure 3). It means, at the beginning images are reshaped into $x \in \mathbb{R}^{224*224*3} = \mathbb{R}^{150528}$, so each neuron in a first layer should have more than 150k parameters! Including only 100 neurons in the first layer, results in more than 15M parameters. In only the first layer! What is more, output of that first layer will be condensed to 100 dimensional hidden state. It means, that layer performs very aggressive reduction of data dimensionality, simultaneously losing a lot of information. In practical examples, in such a setting, there should be much more neurons in the first layer, to prevent aggressive dimensionality reduction, but it will quickly increase model complexity and number of parameters. Summing up, such a naive attempt to images lead to very complex and hard to train models. It is why novel architecture was desired.

2.2.2 Convolution (or precisely Cross-Correlation)

Before introducing convolution operation, let's have a look on what MLP really does and what should change in a novel architecture. Figure 3 presents image as 3-rd order tensor representation.

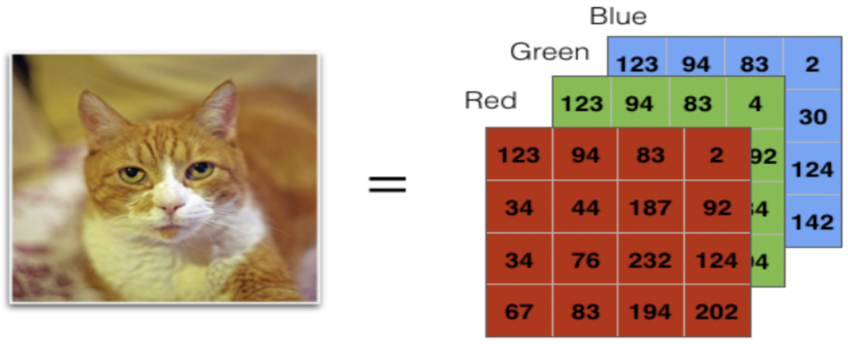

Figure 3: Natural images are represented in a computer as tensors of order 3 with pixel intensity values across color channels. Scheme borrowed from Andrew Ng CNNs course materials.

Naive MLP by taking flattened tensor as input to the first layer, process all pixel intensities values by each neuron separately. Without flattening to vector, similar operation may be described with the equation below. To keep things simple, for now assume image with only one color channel.

$$H_{i,j} = B_{i,j} + \sum_k \sum_l W_{i,j,k,l} X_{k,l} \tag{2}$$

Now we got as a result hidden state in a form of 2nd order tensor H, where each element is computed as affine transformation of all pixel values in an input tensor. We can imagine that each (i-th, j-th) element of the tensor H is a result of separate neuron, with separate parameters: weights and bias.
Such an attempt quickly makes model too complex, but we can reduce complexity thanks to few heuristics, called also an **inductive bias**. Namely, let's make following assumptions:

- **Translational invariance** - If there is any feature on an image, it doesn't matter where it is precisely. e.g. If we have a helmet on an image, it should be equally detected if it is located in a center of image, or in the right, upper corner.
- **Locality** - Detection of features is conducted locally. It means, if we detect feature on (i,j) position, we look only on a close proximity to that point. To identify a helmet in a upper, right corner we don't have to look on the entire image.

Now, we can insert assumptions into Eq.2 to get Eq.3.

$$H_{i,j} = B + \sum_{a=-\Delta}^{\Delta} \sum_{b=-\Delta}^{\Delta} W_{a,b} X_{i+a,j+b} \tag{3}$$

Achieved Eq.3 describes so called Convolution. Now, each element of the tensor H is computed by an affine transformation (neuron) of pixel values in the proximity parameterized by Δ (kernel size). This states for **Locality assumption**.
It is important, that neuron weights are no longer position dependent (note: $W_{a,b}$ in the Eq.3 and $W_{i,j,k,l}$ in the Eq.2). Now neurons share the same weights in each location. It states for **Translational Invariance assumption**.
Visually this operation is presented at Figure 4:

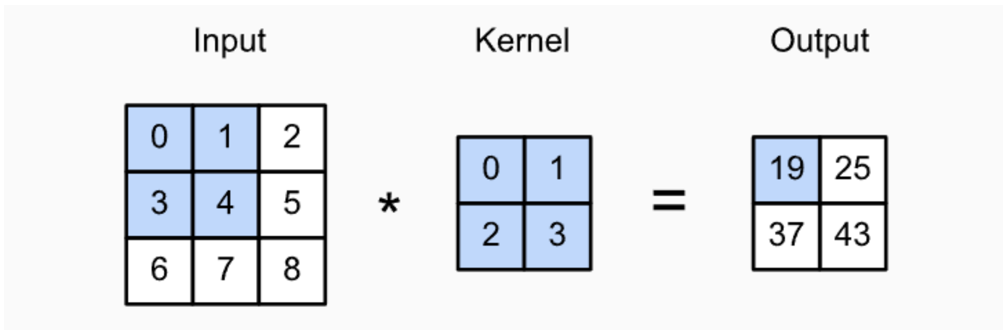

Figure 4: Schematic visualization of a convolution operation. A convolution kernel slides through an input tensor and there is performed element-wise multiplication between the input tensor and the kernel. The scheme borrowed from D2L.

This operation generalize easily to multiple channel input and visually it is depicted at Figure 5

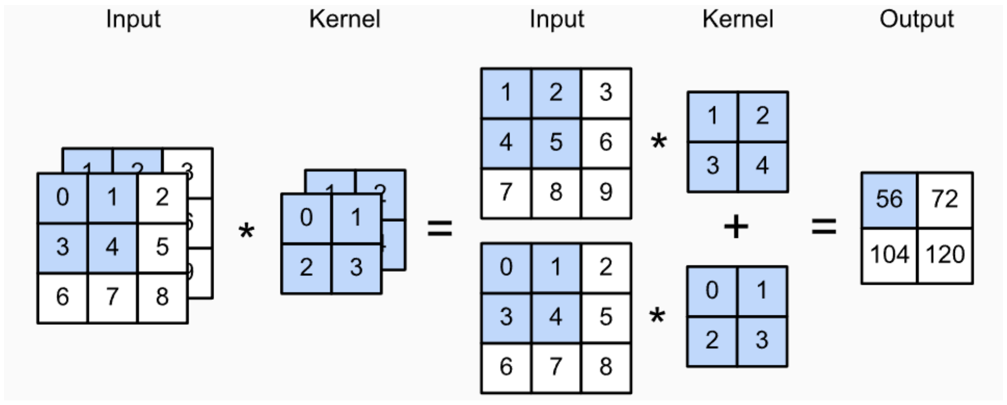

Figure 5: Schematic visualization of a convolution operation in the multiple channel setting. The scheme borrowed from D2L.

As a side note: From the mathematical point of view, this operation should be rather referred to **Cross-Correlation**, but DL literature call it Convolution.

2.2.3 Shared weights matrix interpretation

Intuitively, shared weights may be interpreted as filters. Sliding such a filter on input tensor, detects learned features in different parts of an image. After this, we end up with a feature map, which represents where a given filter detected a learned feature.
This approach reduced significantly number of trainable parameters in a Convolutional Layer. For example, taking an natural image with 3 color channels as input and assuming kernel of size 3x3x3, layer with one filter has only $3 * 3 * 3 + 1 = 28$ parameters. It is an astonishing reduction of a complexity in comparison to the first layer of the MLP. From the other hand, it may be too big reduction, but nothing prevents us to add more filters. We can readily increase layer complexity by adding more filters. With naive interpretation, adding more filters, makes layer able to learn more features. Example of how filters may differentiate their role/feature specificity is well presented at Figure 6.
Also I strongly recommend to play with this web application: adamharley.com, which visualize feature maps.

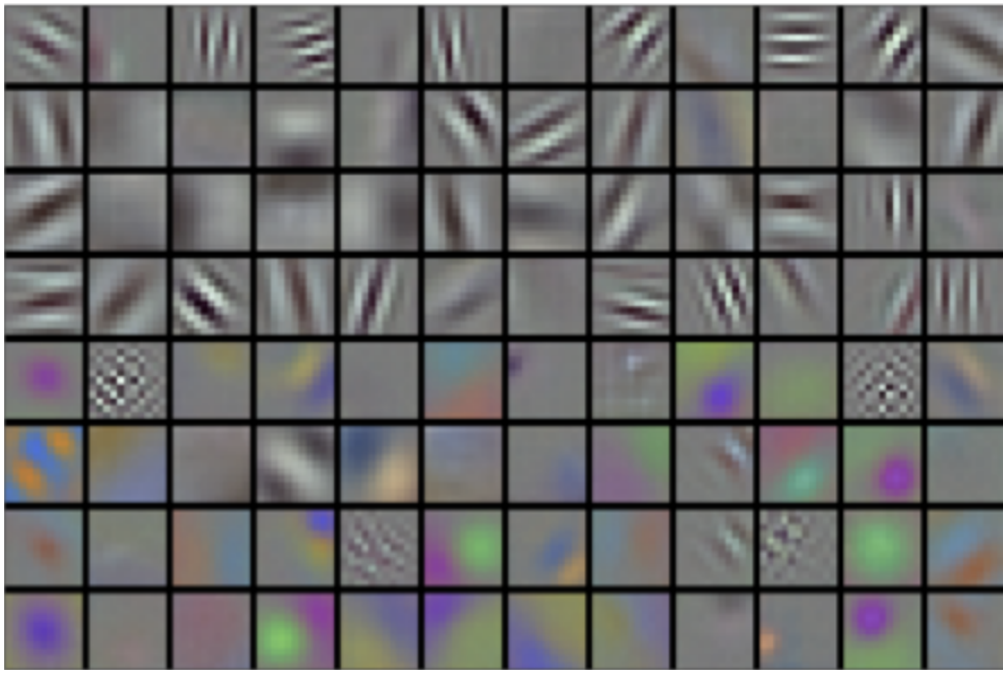

Figure 6: Image filters learned by the first layer of AlexNet CNN. The scheme borrowed from D2L.

2.2.4 Technical details

When working with Convolutional layers there are three important layer hyperparameters:

- **kernel size**: It has been already mentioned. This parameter control size of layer's filters.
- **stride**: This parameter controls sliding of a kernel. With stride=1, kernel moves one by one. With stride=2, kernel moves every two positions, etc.
- **padding**: By default, convolution because of it's nature produces feature maps with smaller size than original input. To prevent reduction of hidden state size, input is padded with additional zeros on border. Padding size define number of rows and columns of zeros which will be added on an image border.

2.2.5 Exemplary CNNs models

- LeNet (1995)
- AlexNet (2012)
- Network in Network (2013)
- VGG (2014)
- GoogLeNet (2015)
- ResNet (2016)
- DenseNet (2017)

References

[1] Alexey Dosovitskiy, Lucas Beyer, Alexander Kolesnikov, Dirk Weissenborn, Xiaohua Zhai, Thomas Unterthiner, Mostafa Dehghani, Matthias Minderer, Georg Heigold, Sylvain Gelly, Jakob Uszkoreit, and Neil Houlsby. An Image is Worth 16x16 Words: Transformers for Image Recognition at Scale, June 2021. arXiv:2010.11929 [cs].

[2] Ian Goodfellow, Yoshua Bengio, and Aaron Courville. *Deep Learning*. MIT Press, 2016. http://www.deeplearningbook.org.

[3] Christian Szegedy, Wei Liu, Yangqing Jia, Pierre Sermanet, Scott Reed, Dragomir Anguelov, Dumitru Erhan, Vincent Vanhoucke, and Andrew Rabinovich. Going Deeper with Convolutions, September 2014. arXiv:1409.4842 [cs].

[4] Aston Zhang, Zachary C. Lipton, Mu Li, and Alexander J. Smola. Dive into deep learning. *arXiv preprint arXiv:2106.11342*, 2021.