Chapter 4: Neural Networks

In the previous chapters, we saw the advantages of linear and kernel learning methods. Kernel methods worked well in separating non-linearly separable data, assuming we had a good feature map ϕ. We will now turn to Neural Networks, whose main advantage will be that they can also learn this feature map. As a motivation let us look at the logistic regression in a higher dimensional space.

$$\min_{b\in\mathbb{R},\boldsymbol{w}\in\mathbb{R}^d,\phi} \frac{1}{2}\|\mathbf{w}\|^2 + C\sum_{i=1}^{n} \ln\left(1+\exp\left(-y_i\left(\mathbf{w}^\top\phi\left(\mathbf{x}_i\right)+b\right)\right)\right)$$

To find this minimum we also optimize over possible mappings ϕ. The main problem is, there are too many mappings ϕ to consider. As a motivation, we can firstly look at how our brain does this.

4.1 The brain graph

In simple terms, we can consider the brain to be a graph. We call a node **neuron** and an edge **synapse**. Another word for graph is called network, as the nodes are called neurons, we call the brain graph a **neural network**. Let us in very

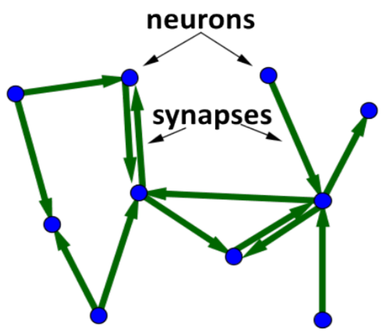

Figure 1: Visualization of the brain graph.

simple terms look at what the brain does, when it sees something.

1. Some neurons will light up (are activated).
2. Activated neuron i will shoot an electrical signal v (spike) to neuron j which is proportional to W_{ij} (the weight of the synapse).

3. If the in-going electrical signals of neuron j (called the potential of j) exceeds a threshold, it will also become activated and thus is able to shoot electrical signals v to its neighboring neurons.

McCulloch and Pitts model
Denote by u the neuron's potential and by v the emitted spike. Then:

$$v = \sigma(u)$$

where σ is the sigmoid function: $\sigma(u) = \frac{1}{1+e^{-u}}$.

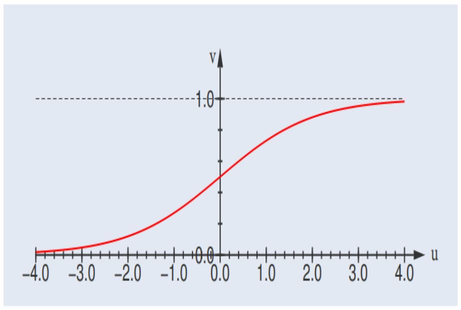

Figure 2: The sigmoid function fits the values u between 0 and 1

Artificial neural networks today use the ReLU activation function instead of the sigmoid function

$$\sigma(u) := \max(0, u)$$

4.2 Artifical neural networks (ANN)

Let us formalize, the propagation of neuron activation.
Neuron j is connected to other neurons with strength W_{ij} who emit spike v_j. The potential will be

$$u_j = \sum_i w_{ij} v_i$$

and its activation

$$v_j = \sigma\left(u_j\right) = \sigma\left(\sum_i w_{ij} v_i\right).$$

Let W be the adjacency matrix of the neural network, $\mathbf{u} = (u_1, \ldots, u_d)^\top$ and $\mathbf{v} = \sigma(\mathbf{u})$ be the activation of all neurons as a vector. Notice that by transposing W, the potential can be calculated as

$$\mathbf{u} = W^\top \mathbf{v}$$

and the activation as

$$\mathbf{v} = \sigma(\mathbf{u}) = \sigma\left(W^\top \mathbf{v}\right).$$

Notice that our equation is dependent on $\mathbf{v}$ on the left and right side, to not allow a cyclic dependency, we restrict our neural network to be an acyclic graph.

4.3 Feed-forward ANN

The acyclic construction from above will be called **feed-forward ANN** or **multi-layer perceptron**.

Example:

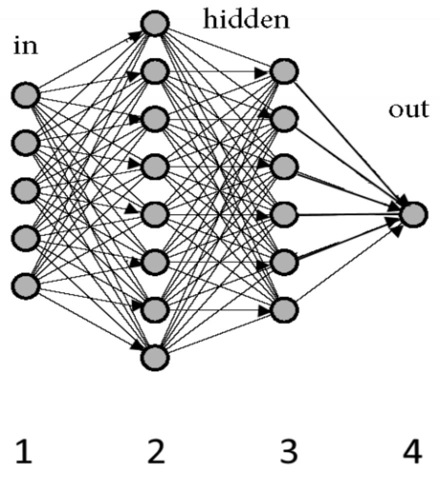

Figure 3: Example of a feed forward ANN

We use our example to define some notions. Shown above we have a feed forward network, with an input layer (1) of 5 nodes, two hidden layers (2,3) of 8 and 6 nodes and an output layer (4) with 1 node.

Let $\mathbf{v_l}$ be the vector of activation of neurons in the l-th layer and W_l be the submatrix of strengths of connections between $l-1$-th and l-th layer.
Recall that

$$\mathbf{v} = \sigma(\mathbf{u}) = \sigma\left(W^\top \mathbf{v}\right).$$

In a feed forward network, we can calculate v_{l+1} by

$$\mathbf{v}_{l+1} = \sigma(\underbrace{W_{l+1}^\top \mathbf{v}_l}_{=\mathbf{u}_{l+1}}).$$

Let us use this observation. Let 0 and $\mathbf{x} := \mathbf{v_0}$ be our starting layer and L be the number of hidden layers sorted from left to right with $L+1$ being the output layer. We can calculate $\mathbf{v_l}$ as following:

$$\phi_W(\mathbf{x}) := \mathbf{v_l} = \underbrace{\sigma\left(W_L^\top \sigma(\ldots \underbrace{\underbrace{\sigma(\underbrace{W_1^\top \underbrace{\mathbf{x}}_{=\mathbf{v}_0}}_{=\mathbf{u}_1}}_{=\mathbf{v1}}) \ldots) \right)}_{=\mathbf{v}_l}$$

and finally the potential of the nodes in the last layer

$$\mathbf{u_{l+1}} = W_{L+1}^\top \phi_w(\mathbf{x})$$

So the feature map we just created will have as output the potential u_{L+1}, which will be a scalar, if we have one output node.

To wrap it all up we will get the following optimization Problem.

$$\min_{\mathbf{w},W} \frac{1}{2}\|\mathbf{w}\|^2 + \frac{1}{2}\sum_{l=1}^{L} \|W_l\|_{\text{Fro}}^2 + C\sum_{i=1}^{n} \log\left(1 + \exp\left(-y_i \mathbf{w}^\top \phi_W(\mathbf{x}_i)\right)\right) \quad \text{(ANN)}$$

where

- $W := (W_1, \ldots, W_L)$
- $\mathbf{w} = W_{L+1}$
- $\|W_k\|_{\text{Fro}}^2 = \sum_{ij} w_{kij}^2$
- $\phi_W(\mathbf{x}_i) := \sigma\left(W_L^\top \sigma\left(\ldots \sigma\left(W_1^\top \mathbf{x}_i\right) \ldots\right)\right)$

which is just logistic regression with feature map ϕ_W based loosely on the brain, including the regularizer

$$\sum_{l=1}^{L} \|W_l\|_{\text{Fro}}^2 + C$$

which is the generalization of a vector norm to a matrix norm (Will be explained in more detail in the following chapter).
Observe that the hyperplane displacement parameter b is left out. We can do this as, if needed the machine can put the displacement inside the feature map ϕ_W.

4.4 Training ANN

Now we turn to solving the Optimization problem (ANN). Our approach will again be Stochastic Gradient Descent. We firstly need to compute some gradients.

For simplicity let us call

$$F(\mathbf{w}, W) := \frac{1}{2}\|\mathbf{w}\|^2 + \frac{1}{2}\sum_{l=1}^{L}\|W_l\|_{\text{Fro}}^2 + C\sum_{i=1}^{n}\log\left(1+\exp\left(-y_i\mathbf{w}^\top\phi_W\left(\mathbf{x}_i\right)\right)\right)$$

Let us first compute the gradient with respect to $\mathbf{w}$, by linearity we have:

$$\nabla_{\mathbf{w}}F(\mathbf{w}, W) = \underbrace{\nabla_{\mathbf{w}}\left(\frac{1}{2}\|\mathbf{w}\|^2\right)}_{=\mathbf{w}} + \underbrace{\nabla_{\mathbf{w}}\left(\frac{1}{2}\sum_{l=1}^{L}\|W_l\|_{\text{Fro}}^2\right)}_{=0} + C\sum_{i=1}^{n}\nabla_{\mathbf{w}}\left(\log\left(1+\exp\left(-y_i\mathbf{w}^\top\phi_W\left(\mathbf{x}_i\right)\right)\right)\right)$$

If we call

$$g(x) := \log(1+\exp(x)) \text{ with } g'(x) = \frac{\exp(x)}{1+\exp(x)} = \frac{1}{1+\exp(-x)}$$

and

$$f_i(\mathbf{w}) = -y_i\mathbf{w}^T\phi_W(\mathbf{x_i}) \text{ with } \nabla_{\mathbf{w}}f_i(\mathbf{w}) = -y_i\phi_W(\mathbf{x_i})$$

we get

$$\begin{aligned}
\nabla_{\mathbf{w}}\left(\log\left(1+\exp\left(-y_i\mathbf{w}^\top\phi_W\left(\mathbf{x}_i\right)\right)\right)\right) &= \nabla_{\mathbf{w}}\left(g(-y_i\mathbf{w}^\top\phi_W\left(\mathbf{x}_i\right))\right)\\
&= \nabla_{\mathbf{w}}(g(f_i(\mathbf{w}))\\
&= \nabla g\left(f_i\left(\mathbf{w}\right)\right)\nabla f_i\left(\mathbf{w}\right)\\
&= \frac{1}{1+\exp(y_i\mathbf{w}^T\phi_W(\mathbf{x_i}))}\cdot -y_i\phi_W(\mathbf{x_i})
\end{aligned}$$

In conclusion we get

$$\nabla_{\mathbf{w}}F(\mathbf{w}, W) = \mathbf{w} - C\sum_{i=1}^{n}\frac{y_i\phi_W\left(\mathbf{x}_i\right)}{1+\exp\left(y_i\mathbf{w}^\top\phi_W\left(\mathbf{x}_i\right)\right)}$$

Now it remains to find the gradient with respect to $W = (W_1, \ldots, W_L)$. Let us take the gradient with respect to each W_l. Again by linearity we have:

$$\begin{aligned}
\nabla_{W_l}F(\mathbf{w}, W) &= \underbrace{\nabla_{W_l}\left(\frac{1}{2}\left\|w\right\|^2\right)}_{=0} + \underbrace{\nabla_{W_l}\left(\frac{1}{2}\sum_{l=1}^{L}\|W_l\|_{\text{Fro}}^2\right)}_{=W_l}\\
&\quad + C\sum_{i=1}^{n}\nabla_{W_l}\left(\log\left(1+\exp\left(-y_i\mathbf{w}^\top\phi_W\left(\mathbf{x}_i\right)\right)\right)\right)\\
&= W_l - C\sum_{i=1}^{n}\frac{y_i\mathbf{w}^\top\nabla_{W_l}\phi_W\left(\mathbf{x}_l\right)}{1+\exp\left(y_i\mathbf{w}^\top\phi_W\left(\mathbf{x}_i\right)\right)}
\end{aligned}$$

It remains to calculate

$$\nabla_{w_l} \phi_W \left(\mathbf{x}_i\right).$$

For a better overview we calculate $\nabla_{W_{ijl}} \phi_W(\mathbf{x_i})$ with W_{ijl} being the i, j-th entry of the matrix W_l. By definition of $\phi_W(\mathbf{x})$ we have:

$$\nabla_{W_{ijl}} \phi_W(\mathbf{x}) := \nabla_{W_{ijl}} \underbrace{\sigma \left(W_L^\top \sigma(\dots \underbrace{\sigma(\underbrace{W_1^\top \underbrace{\mathbf{x}}_{=\mathbf{v}_0}}_{=\mathbf{u}_1})}_{=\mathbf{v1}} \dots) \right)}_{=\mathbf{v}_l}$$

We apparently need the derivative of a nested function, for which we will use the chain rule, going up to $\mathbf{u}_l$ for the matrix W_l.

$$\nabla_{w_{ijl}} \phi_W(\mathbf{x}) = \frac{\partial \mathbf{v}_L}{\partial w_{ijl}} = \frac{\partial \mathbf{v}_L}{\partial \mathbf{u}_L} \cdot \frac{\partial \mathbf{u}_L}{\partial \mathbf{v}_{L-1}} \cdot \frac{\partial \mathbf{v}_{L-1}}{\partial \mathbf{u}_{L-1}} \cdots \frac{\partial \mathbf{u}_{1+1}}{\partial \mathbf{v}_l} \cdot \frac{\partial \mathbf{v}_l}{\partial \mathbf{u}_l} \cdot \frac{\partial \mathbf{u}_l}{\partial w_{ijl}}$$

So we need to calculate three derivatives

- $\frac{\partial \mathbf{v}_l}{\partial \mathrm{u}_l}$
- $\frac{\partial \mathbf{u}_l}{\partial \mathbf{v}_{l-1}}$
- $\frac{\partial \mathbf{u}_l}{\partial w_{ijl}}$

Let us first define the following auxiliary function

$$\Theta : \mathbb{R} \to \mathbb{R}$$

$$\Theta(c) := \begin{cases} 0 & c \leq 0 \\ 1 & \text{otherwise} \end{cases}$$

where Θ of a vector is applied elementwise:

$$\Theta(\mathbf{x}) := \begin{pmatrix} \Theta\left(x_1\right) \\ \vdots \\ \Theta\left(x_d\right) \end{pmatrix}$$

Now, we go back to calculating the sought derivatives.
For the activation function we use the ReLU $\sigma(\mathbf{v}) = \max(0, u)$.
We have

$$\frac{\partial \mathbf{v}_l}{\partial \mathbf{u}_l} = \frac{\partial \sigma\left(\mathbf{u}_l\right)}{\partial \mathbf{u}_l} = \frac{\partial \max\left(0, \mathbf{u}_l\right)}{\partial \mathbf{u}_l} = \Theta\left(\mathbf{u}_l\right)$$

Moving on to the next derivative:

$$\frac{\partial \mathbf{u}_1}{\partial \mathbf{v}_{l-1}} = \frac{\partial \left(W_l^\top \mathbf{v}_{l-1} \right)}{\partial \mathbf{v}_{l-1}} = W_l^\top$$

Let us briefly, recall the coloumn-wise matrix multiplication

$$A\mathbf{x} = A_{\cdot,1}x_1 + A_{\cdot,2}x_2 + \cdots + A_{\cdot,n}x_n$$

where $A_{\cdot,i}$ stands for the i-th coloumn in the matrix A.
We will use this to calculate our final derivative. Take care that transposition interchanges indices.

$$\begin{aligned}
\frac{\partial \mathbf{u}_l}{\partial W_{ijl}} = \frac{\partial \left(W_l^\top \mathbf{v}_{l-1} \right)}{\partial W_{ijl}} &= \frac{\partial \left(W^T_{\cdot,1,l}\mathbf{v}_{1,l-1} + W^T_{\cdot,2,l}\mathbf{v}_{2,l-1} + W^T_{\cdot,n}\mathbf{v}_{n,l-1} \right)}{\partial W_{ijl}} \\
&= \frac{\partial \left(W^T_{\cdot,1,l}\mathbf{v}_{1,l-1} \right)}{\partial W_{ijl}} + \frac{\partial \left(W^T_{\cdot,2,l}\mathbf{v}_{2,l-1} \right)}{\partial W_{ijl}} + \cdots + \frac{\partial \left(W^T_{\cdot,n}\mathbf{v}_{n,l-1} \right)}{\partial W_{ijl}} \\
&= 0 + 0 + \cdots + \frac{\partial \left(W^T_{\cdot,i,l}\mathbf{v}_{i,l-1} \right)}{\partial W_{ijl}} + 0 + \cdots + 0 \\
&= \mathbf{v}_{i,l-1}\mathbf{e}_j
\end{aligned}$$

We can now look back to the chain-rule formulation of $\nabla_{w_{ijl}}\phi_W(\mathbf{x})$. With our calculations this far, it translates to

$$\begin{aligned}
\nabla_{w_{ijl}}\phi_W(\mathbf{x}) &= \frac{\partial \mathbf{v}_L}{\partial \mathbf{u}_L} \cdot \frac{\partial \mathbf{u}_L}{\partial \mathbf{v}_{L-1}} \cdot \frac{\partial \mathbf{v}_{L-1}}{\partial \mathbf{u}_{L-1}} \cdots \frac{\partial \mathbf{u}_{l+1}}{\partial \mathbf{v}_l} \cdot \frac{\partial \mathbf{v}_l}{\partial \mathbf{u}_l} \cdot \frac{\partial \mathbf{u}_l}{\partial w_{ijl}} \\
&= \Theta\left(\mathbf{u}_L\right) W_L^\top \Theta\left(\mathbf{u}_{L-1}\right) \cdots W_{l+1}^\top \Theta\left(\mathbf{u}_l\right) v_{i,l-1}\mathbf{e}_j.
\end{aligned}$$

Notice that we need to compute each $\mathbf{u}_l$, to use $\Theta(\mathbf{u}_l)$. We can achieve this with the following algorithm.

Algorithm Forward Propagation

```
1: Initialize $\mathbf{v_0} = \mathbf{x}$
2: for $l \leftarrow 1$ to $(L-1)$ do
3:     $\mathbf{u}_l := W_l^\top \mathbf{v}_{l-1}$
4:     $\mathbf{v}_l := \sigma\left(\mathbf{u}_l\right)$
5: end for
```

Now, we can finally compute our gradient $\nabla_{w_{ijl}}\phi_W(\mathbf{x})$ for all i, j, l with the following algorithm.
The algorithm will just calculate our chain-rule formulation.

Algorithm Back Propagation

```
1: Initialize $\boldsymbol{\delta}_L := \Theta(\mathbf{u}_L)$
2: $\forall i, j : \nabla_{w_{ijL}}\phi_W(\mathbf{x}) := \boldsymbol{\delta_L} v_{i,L-1}\mathbf{e}_j$
3: for $l \leftarrow (L-1)$ to 1 do
4:     $\boldsymbol{\delta}_l := \boldsymbol{\delta}_{l+1} W_{l+1}^\top \Theta(\mathbf{u}_l)$
5:     $\forall i, j : \nabla_{w_{ijl}}\phi_W(\mathbf{x}) := \boldsymbol{\delta_l}\mathbf{v_{i,l-1}}\mathbf{e}_j$
6: end for
```

4.5 Convolutional neural networks (CNN)

CNN are a type of artificial neural network. They are mostly used in image and speech recognition. In the following, we will be using the example of image classification.

In image processing, it makes sense to process portions of an image (called **patches**). An apple could be anywhere in the image and we need to make sure to identify it regardless of its position. To focus on 'interesting parts' of the image (e.g the apple stalk), we apply so called **filters**. The high level idea is to identify parts of an object and then use their position in relation to each other to determine if the object is present, however in practice it is difficult to affirm whether CNN actually follow this basic idea.

CNN incorporate spatial information and weight sharing. Weight sharing is the idea of having multiple weights be equal. To understand spatial information consider an image of an apple, now randomly permute the pixels of this image, the apple will no longer be recognizable, even though the set of pixel values is still the same. In images pixels close together have more interaction than pixels far appart, which is what we try to capture in a network.

A **convolutional filter** is a filter based on convolutions. Examples include the gaussian filter(blurring) and laplace filter(edge detection). You can think of the filter as being the weights of the edges in a neural network.
The key point of CNN is to learn the convolutional filter which helps us recognize important parts of the image.

Notice that a picture can have many pixel and thus a CNN can get quite large, so we need techniques to reduce their size.
Pooling is a common choice to reduce a CNNs size. Pooling shrinks many neighboring pixel into one. The typical pooling method is max pooling, however other pooling methods are also used in specific setting. Following we give two examples.

- **Max Pooling:** The newly formed pixel will be the maximum of all pixel values in the patch

- **Average Pooling:** The newly formed pixel will be the average of all pixel values in the patch

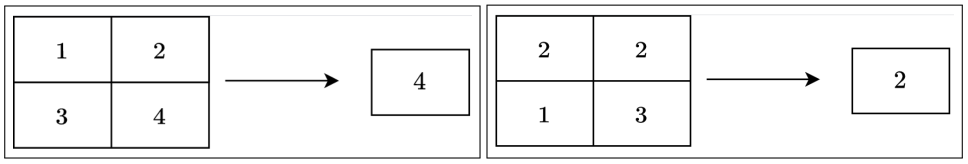

Figure 4: Examples of pooling: To the left we see max pooling, to the right average pooling.

4.6 CNN architecture

We will explain how a CNN layer is set up. First we consider an input, which has a width, height, and depth. The depth is usually referred to as the channel dimension, for typical images this dimension contains different color channels like RGB channels. We will denote width by w, height by h, and the channels by c. Then the input I is in $\mathbb{R}^{h \times w \times c}$. Now a CNN layer is made up of multiple filters, also called kernels. Let f be the number of such filters and let g be their size, i.e. one filter is of size $g \times g \times c$, typically only referred to as $g \times g$. Each filter is applied using a stride, we will call s, and a padding, we will call p. Now to formalize what the CNN layer actually does we define the size of the output of the layer and then specify how each component of the output can be computed. Afterwards, we will give intuitions into how the output can be interpreted.

The spatial dimension of the output can be computed by the following equation

$$o(i, g, p, s) = \frac{i - g + 2p}{s} + 1, \tag{1}$$

where i refers to the input size and o refers to the output size. Now we can compute the output width $ow = o(w, g, p, s)$ and the output height $oh = o(h, g, p, s)$. Thus the output O is finally in $\mathbb{R}^{oh \times ow \times f}$. We typically refer to both the input and the output as images, even though the output can typically not be visualized in the same way the input can.

Finally each component of the output can be computed as

$$O_{i,j,k} := \sigma\left(\sum_{l \in \{1,\dots,k\}} \sum_{m \in \{1,\dots,k\}} \sum_{n \in \{1,\dots,c\}} I_{si+l-p, sj+m-p, n} W^k_{g-l, g-m, c-n} \right), \tag{2}$$

where $W^k \in \mathbb{R}^{k \times k \times c}$ are the weights of the k-th filter. Note that in this equation $i, j, k, l, m,$ and n are used as indices and do not refer to the variables defined above, however s, p, and g are the variables as defined above. The indices for W^k might be confusing, this is due to the equation being a convolution. They can also be thought of as $W^k_{l,m,n}$ for simplicity's sake.

Now we can talk about the intuitions of what the CNN layer actually computes.

Each channel of the output refers to an image of where the feature that is defined by the filter is observable. If for example the first filter is an edge detector, i.e. the Laplace-filter, then the first slice of O, i.e. $O_{\cdot,\cdot,1}$ would be a black and white image of all edges, or simply the Laplace-filter applied to the original image. However, a slight complication has to be taken into account, a filter does not only have width and height, but also depth. Thus this Laplace-filter analogy only works if we either consider only one color channel of the input or if we consider an input with only 1 channel, since the Laplace-filter is only defined in two dimension, not 3.

One component of the output can be thought of as a neuron. As such it is possible to think of a CNN layer as a modified NN layer. Let $vec(M)$ be the vectorization of matrix M ad W be the weight matrix of the layer, i.e. $Wvec(I) = vec(O)$. Since each component in one channel was produced using the same filter, these components share their weights. If $vec(O)_i$ is in the same channel as $vec(O)_j$, then $W_{i,\cdot} = W_{j,\cdot}$, this is weight sharing in mathematical terms, it is effectively a constraint in a typical NN. Also W is quite sparse, as two input components in the same channel with a distance greater than g will never be present in the same sum of equation 2. One line in this matrix contains at most g^2c (the weight parameters of the filter) many non-zero values, while its actual size is $hwc >> g^2c$. A typical example for this would be the MNIST dataset, one image has size $28 \times 28 \times 1$, whereas a filter will typically have size 3×3 or 5×5, $5^2 = 25 < 784 = 28^2$. This is how the parameter count in a CNN is kept quite low.

4.7 Deep Learning

Deep Neural network An ANN having 8 or more layers is called deep neural network.
Deep CNN are state of the art in image classification.
We now have a look at the AlexNet, a CNN that competed in the ImageNet Large Scale Visual Recognition Challenge on September 30, 2012. AlexNet was trained by GPU's as they are faster in matrix and vector operations.

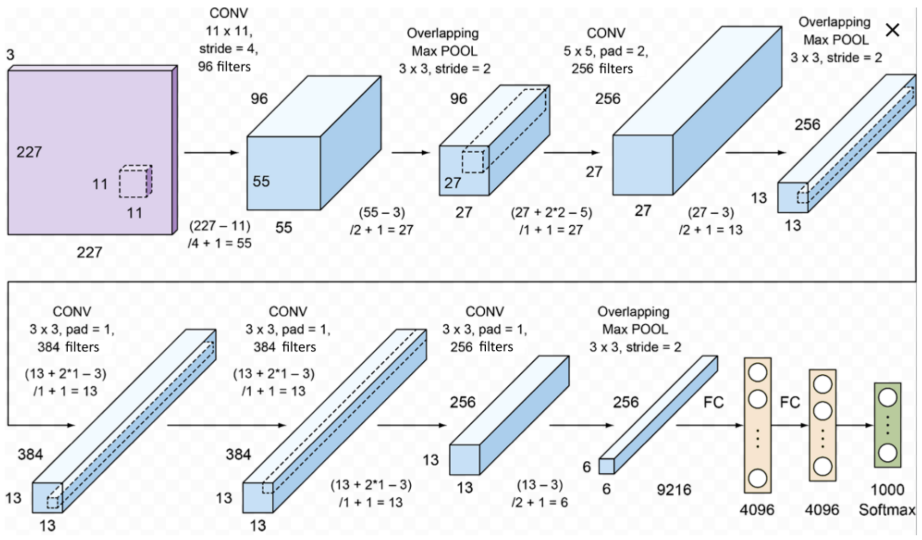

Figure 5: The AlexNet consists of 8 layers. It used max pooling and the ReLu activation function. In the image you can see the size of the patches, the number of filters used and the size of the stride. At the end we end up with fully connected layers, leading us to the output.