**Deep Learning**

The first layer learns primitive features, like an edge in an image or the tiniest unit of speech sound. It does this by finding combinations of digitized pixels or sound waves that occur more often than they should by chance. Once that layer accurately recognizes those features, they’re fed to the next layer, which trains itself to recognize more complex features, like a corner or a combination of speech sounds. The process is repeated in successive layers until the system can reliably recognize phonemes or objects.

Problems with Deep Layers (not deep learning)

As we delve into the problem more deeply, we'll learn that the opposite phenomenon can also occur: the early layers may be learning well, but later layers can become stuck. In fact, we'll find that there's an intrinsic instability associated to learning by gradient descent in deep, many-layer neural networks. This instability tends to result in either the early or the later layers getting stuck during training.

Note: the gradient ∂C/∂b for each neuron, i.e., the rate of change of the cost with respect to the neuron's bias. Back in we saw that this gradient quantity controlled not just how rapidly the bias changes during learning, but also how rapidly the weights input to the neuron change, too.

Vanishing gradient problem (Unstable gradients): neurons in different layer learn differently. Gradients in early layers are product of terms from later layers.

Deep Learning Intro from the book

It’s strange to use fully connected layers to classify images as it does not takes into account the spatial structure of the image.

Convolutional neural networks have three basic ideas; local receptive field, shared weights and pooling.

Input: 28x28 square of neurons value corresponding to 28x28 pixel intensities we are using.

Each neuron in the hidden layer will be connected to a small region of the input image (say 5x5). The region in the input image is called the local receptive field of the hidden neuron. Every pixel connection in the receptive field of the neuron has a weight and the hidden neuron has an overall bias.

This was by sliding the receptive field by some quantity (‘stride’) we get the ZxZ dimension first hidden layer.

Each hidden neuron will have a bias and a 5x5 weight matrix. Note that *we are going to use the same weights and bias for each of the 24x24 hidden neurons.* This means that all the neurons in the first hidden layer detect exactly the same feature (think of feature as a pattern), just at different locations in the input image. To put it in more abstract terms, convolutional networks are well adapted to the translation of the image. Hence we call the map from input layer to hidden layer as *feature map*. The shared weights and bias are often said to define a *kernel or filter.* A convolutional layer consists of several feature maps. One map corresponding to one feature.

Shared weight and bias reduces the number of parameters greatly.

Pooling Layer: They are used immediately after convolutional layer. It simplifies the output from convolutional layer. We take each feature map and replace it with a condensed feature map. (Max pooling) We can think Max pooling as a way for network to ask whether a given feature is found anywhere in the image.

Fully Connected Layer: This layer connects each neuron of max pooled layer to one of the outputs (to numbers from 0 to 9 in case of MNIST dataset)

What google did!  
Like cats. Last June, Google demonstrated one of the largest neural networks yet, with more than a billion connections. A team led by Stanford computer science professor Andrew Ng and Google Fellow Jeff Dean showed the system images from 10 million randomly selected YouTube videos. One simulated neuron in the software model fixated on images of cats. Others focused on human faces, yellow flowers, and other objects. And thanks to the power of deep learning, the system identified these discrete objects even though no humans had ever defined or labeled them.

What stunned some AI experts, though, was the magnitude of improvement in image recognition. The system correctly categorized objects and themes in the ­YouTube images 16 percent of the time. That might not sound impressive, but it was 70 percent better than previous methods. And, Dean notes, there were 22,000 categories to choose from; correctly slotting objects into some of them required, for example, distinguishing between two similar varieties of skate fish. That would have been challenging even for most humans. When the system was asked to sort the images into 1,000 more general categories, the accuracy rate jumped above 50 percent.

Backpropogation /automatic differentiation (when network is sparse)

**Deep Neural Networks**

The model consists of multiple layers, each of which has a [rectified linear unit](https://en.wikipedia.org/wiki/Rectified_linear_unit) for non-linear transformation. Some layers are convolutional, while others are fully connected. Every convolutional layer has an additional max pooling. The network is trained to minimize L2 error for predicting the mask ranging over the entire training set containing bounding boxes represented as masks.

Problems with DNN

1-> Overfitting - Ivakhnenko's unit pruning; [weight decay](https://en.wikipedia.org/wiki/Weight_decay) ({\displaystyle \ell \_{2}}l2-regularization); [sparsity](https://en.wikipedia.org/wiki/Sparse_matrix) ({\displaystyle \ell \_{1}}l2-regularization); [dropout](https://en.wikipedia.org/wiki/Dropout_(neural_networks)) regularization (In dropout, some number of units are randomly omitted from the hidden layers during training. This helps to break the rare dependencies that can occur in the training data)

DNNs are prone to overfitting because of the added layers of abstraction, which allow them to model rare dependencies in the training data.

2-> Computational time - here are many training parameters to be considered with a DNN, such as the size (number of layers and number of units per layer), the learning rate and initial weights. Mini Batching is used to reduce it. Radical alternatives to backprop such as [Extreme Learning Machines](https://en.wikipedia.org/wiki/Extreme_Learning_Machines), "No-prop" networks, training without backtracking, "weightless" networks, and [non-connectionist neural networks](https://en.wikipedia.org/wiki/Holographic_associative_memory)are gaining attention.

**CONVOLUTIONAL NEURAL NETWORK**

In comparison with other deep architectures, convolutional neural networks have shown superior results in both image and speech applications. They can also be trained with standard backpropagation. CNNs are easier to train than other regular, deep, feed-forward neural networks and have many fewer parameters to estimate, making them a highly attractive architecture to use.

a convolutional neural network (CNN, or ConvNet) is a type of [feed-forward](https://en.wikipedia.org/wiki/Feedforward_neural_network) [artificial neural network](https://en.wikipedia.org/wiki/Artificial_neural_network) in which the connectivity pattern between its [neurons](https://en.wikipedia.org/wiki/Artificial_neuron) is inspired by the organization of the animal [visual cortex](https://en.wikipedia.org/wiki/Visual_cortex), whose individual neurons are arranged in such a way that they respond to overlapping regions tiling the [visual field](https://en.wikipedia.org/wiki/Visual_field). CNN are variations of [multilayer perceptrons](https://en.wikipedia.org/wiki/Multilayer_perceptron) designed to use minimal amounts of [preprocessing](https://en.wikipedia.org/wiki/Preprocessing).

Convolutional networks may include local or global pooling layers, which combine the outputs of neuron clusters.

They also consist of various combinations of [convolutional](https://en.wikipedia.org/wiki/Convolution) and fully connected layers, with [pointwise nonlinearity](https://en.wikipedia.org/w/index.php?title=Pointwise_nonlinearity&action=edit&redlink=1) applied at the end of or after each layer. One major advantage of convolutional networks is the use of shared weight in convolutional layers, which means that the same filter (weights bank) is used for each pixel in the layer; this both reduces memory footprint and improves performance.

CNN have the following distinguishing features:

1. 3D volumes of neurons. The layers of a CNN have neurons arranged in 3 dimensions: width, height and depth. The neurons inside a layer are only connected to a small region of the layer before it, called a receptive field. Distinct types of layers, both locally and completely connected, are stacked to form a CNN architecture.
2. Local connectivity: following the concept of [receptive fields](https://en.wikipedia.org/wiki/Receptive_field), CNNs exploit spatially local correlation by enforcing a local connectivity pattern between neurons of adjacent layers. The architecture thus ensures that the learnt "filters" produce the strongest response to a spatially local input pattern. Stacking many such layers leads to non-linear "filters" that become increasingly "global" (i.e. responsive to a larger region of pixel space). This allows the network to first create good representations of small parts of the input, then assemble representations of larger areas from them.
3. Shared weights: In CNNs, each filter is replicated across the entire visual field. These replicated units share the same parameterization (weight vector and bias) and form a feature map. This means that all the neurons in a given convolutional layer detect exactly the same feature. Replicating units in this way allows for features to be detected regardless of their position in the visual field, thus constituting the property of [translation invariance](https://en.wikipedia.org/wiki/Translational_symmetry).

BUILDING THE ARCHITECTURE

A CNN architecture is formed by a stack of distinct layers that transform the input volume into an output volume (e.g. holding the class scores) through a differentiable function. A few distinct types of layers are commonly used. We discuss them further below:

**Convolutional layer**

The layer's parameters consist of a set of learnable filters (or [kernels](https://en.wikipedia.org/wiki/Kernel_(image_processing))), which have a small receptive field, but extend through the full depth of the input volume. During the forward pass, each filter is convolved across the width and height of the input volume, computing the [dot product](https://en.wikipedia.org/wiki/Dot_product) between the entries of the filter and the input and producing a 2-dimensional activation map of that filter. As a result, the network learns filters that activate when they see some specific type of feature at some spatial position in the input.

Stacking the activation maps for all filters along the depth dimension forms the full output volume of the convolution layer. Every entry in the output volume can thus also be interpreted as an output of a neuron that looks at a small region in the input and shares parameters with neurons in the same activation map.

Convolutional networks exploit spatially local correlation by enforcing a local connectivity pattern between neurons of adjacent layers: each neuron is connected to only a small region of the input volume. *The extent of this connectivity is a [hyperparameter](https://en.wikipedia.org/wiki/Hyperparameter_optimization" \o "Hyperparameter optimization) called the*[*receptive field*](https://en.wikipedia.org/wiki/Receptive_field)*of the neuron.* The connections are local in space (along width and height), but always extend along the entire depth of the input volume. Such an architecture ensures that the learnt filters produce the strongest response to a spatially local input pattern.

Three [hyperparameters](https://en.wikipedia.org/wiki/Hyperparameter_optimization" \o "Hyperparameter optimization) control the size of the output volume of the convolutional layer: the depth, stride and zero-padding.

1. Depth of the output volume controls the number of neurons in the layer that connect to the same region of the input volume. All of these neurons will learn to activate for different features in the input. For example, if the first Convolutional Layer takes the raw image as input, then different neurons along the depth dimension may activate in the presence of various oriented edges, or blobs of color.
2. Stride controls how depth columns around the spatial dimensions (width and height) are allocated. When the stride is 1, a new depth column of neurons is allocated to spatial positions only 1 spatial unit apart. This leads to heavily overlapping receptive fields between the columns, and also to large output volumes. Conversely, if higher strides are used then the receptive fields will overlap less and the resulting output volume will have smaller dimensions spatially.
3. Sometimes it is convenient to pad the input with zeros on the border of the input volume. The size of this zero-padding is a third hyperparameter. Zero padding allows to control the spatial size of the output volumes. In particular, sometimes it is desirable to exactly preserve the spatial size of the input volume.

### Pooling layer

### Form of non-linear down-sampling. There are several non-linear functions to implement pooling among which [max pooling](https://en.wikipedia.org/wiki/Max_pooling) (also can use average pooling and L2 norm pooling) is the most common. It partitions the input image into a set of non-overlapping rectangles and, for each such sub-region, outputs the maximum. The intuition is that once a feature has been found, its exact location isn't as important as its rough location relative to other features. The function of the pooling layer is to progressively reduce the spatial size of the representation to reduce the amount of parameters and computation in the network, and hence to also control [overfitting](https://en.wikipedia.org/wiki/Overfitting). It is common to periodically insert a pooling layer in-between successive conv layers in a CNN architecture. The pooling operation provides a form of translation invariance.

### Due to the aggressive reduction in the size of the representation (which is helpful only for smaller datasets to control overfitting), the current trend in the literature is towards using smaller filters or discarding the pooling layer altogether.

### ReLU layer

### This is a layer of neurons that applies the non-saturating [activation function](https://en.wikipedia.org/wiki/Activation_function) {\displaystyle f(x)=\max(0,x)}f(x) = max(0,x). It increases the nonlinear properties of the decision function and of the overall network without affecting the receptive fields of the convolution layer.

### Other functions are also used to increase nonlinearity, for example the saturating [hyperbolic tangent](https://en.wikipedia.org/wiki/Hyperbolic_tangent) and the sigmoid function. Compared to other functions the usage of ReLU is preferable, because it results in the neural network training several times faster.

### Fully connected layer

Finally, after several convolutional and max pooling layers, the high-level reasoning in the neural network is done via fully connected layers. Neurons in a fully connected layer have full connections to all activations in the previous layer, as seen in regular Neural Networks. Their activations can hence be computed with a matrix multiplication followed by a bias offset.

### Loss layer

The loss layer specifies how the network training penalizes the deviation between the predicted and true labels and is normally the last layer in the network. Various loss functions appropriate for different tasks may be used there. [Softmax](https://en.wikipedia.org/wiki/Softmax_function" \o "Softmax function) loss is used for predicting a single class of K mutually exclusive classes. [Sigmoid](https://en.wikipedia.org/wiki/Sigmoid_function) [cross-entropy](https://en.wikipedia.org/wiki/Cross_entropy) loss is used for predicting K independent probability values in {\displaystyle [0,1]}[0,1]. [Euclidean](https://en.wikipedia.org/wiki/Euclidean_distance) loss is used for regressing to real-valued labels {\displaystyle [-\infty ,\infty ]}

**General:**

Network needs to explore the features and they need to be reliable. Reliable features are edges and not the colors because they are vulnerable to illumination. (A color in the sun can look different in the dark) Hence a little pre-processing is required and the white channel need to be extracted from the images. We then normalize the image as it increases the training speed. (Generally normalized with 1D Gaussian kernel)

Mathematically convolution is described as overlap of one function as it is shifted over another function. It represents the similarity of two functions.

Takes part of the input image as the size of kernel (i.e. if kernel is of size 3x3 then the 3x3 part of 7x8 image will be taken) Multiply the corresponding pixels and adds them to give answer. It decreases the size of the input image. When pixels are uniform, it gives answer 0 and when it is not then it gives non zero answer. This is how it detects the edges. Kernel Ex. Gabor filter (according to its orientation, it identifies edges at different orientation. For ex. Vertical orientation will detect vertical image)

When we train a network they themselves come out with filters. In case of Natural images Gabor filters will be created.

Feature Extraction

1. Detecting Edges : Done
2. Tanh and absolute element wise transformation: important in improving the accuracy  
   Tanh is squeezing the info between -1 to +1. Abs is making negative values positive.

We do this because Firstly, polarity of features is irrelevant to recognize objects. Secondly, rectification eliminates cancellations between neighboring filter outputs when combined with average pooling. We don’t want positive and negative values to cancel with each other.

1. Subsampling and tanh: provides with distortion invariance. To reduce the precise location of the features (for ex. the location of eyes and mouth) we subsample the feature map. In subsampling we downsample the image. We average the pixel values, multiply by weight (trainable) and add a bias (again trainable)
2. Convolutional map and tanh: We convolve the output from subsampling with filters of ‘same’ size (i.e. if output is of 7x7 then the filter is also of 7x7). Filters are created by training. Convolutional Map is basically the collection of the filters. CM does not convolve each filter with each input, instead it randomly selects some input for each filter. This is done so that during training different maps can explore different features.
3. Linear Classification layer: ouput is +1 or -1. (i.e. face or background)

Note that if the feature is not of 32x32 size then down sample its size, so that a box of 32x32 can identify it.

A network can have more number of Convolutional layers and subsampling layers. The logic is to extract the features and decrease the resolution.

The catch in convolutional neural network is that the neurons or filters are not connected to each and every input, rather only a set of inputs.(stride is the gap between the inputs)