Convolutional Neural Network

- A convolutional neural network, or CNN, is a deep learning neural network designed for processing structured arrays of data such as images. Convolutional neural networks are widely used in computer vision and have become the state of the art for many visual applications such as image classification, and have also found success in natural language processing for text classification.
- Convolutional neural networks are very good at picking up on patterns in the input image, such as lines, gradients, circles, or even eyes and faces.
- It is this property that makes convolutional neural networks so powerful for computer vision. Unlike earlier computer vision algorithms, convolutional neural networks can operate directly on a raw image and do not need any preprocessing.
- A convolutional neural network is a feed-forward neural network, often with up to 20 or 30 layers. The power of a convolutional neural network comes from a special kind of layer called the convolutional layer.
- Convolutional neural networks contain many convolutional layers stacked on top of each other, each one capable of recognizing more sophisticated shapes. With three or four convolutional layers it is possible to recognize handwritten digits and with 25 layers it is possible to distinguish human faces.
- The usage of convolutional layers in a convolutional neural network mirrors the structure of the human visual cortex, where a series of layers process an incoming image and identify progressively more complex features.

Convolutional Neural Network Design

- The architecture of a convolutional neural network is a multi-layered feed-forward neural network, made by stacking many hidden layers on top of each other in sequence. It is this sequential design that allows convolutional neural networks to learn hierarchical features.
- The hidden layers are typically convolutional layers followed by activation layers, some of them followed by pooling layers.
- A simple convolutional neural network that aids understanding of the core design principles is the early convolutional neural network LeNet-5, published by Yann LeCun in 1998. LeNet is capable of recognizing

handwritten characters.

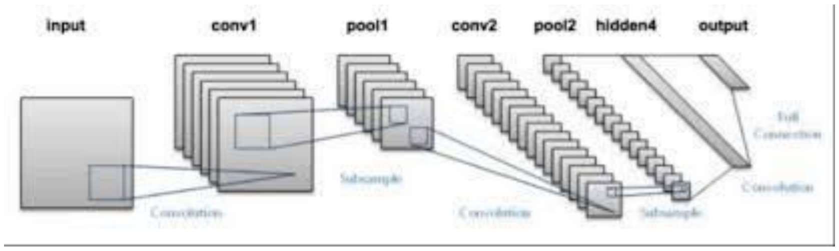
Example Convolutional Neural Network Layers Explained

LeNet takes an input image of a handwritten digit of size 32x32 pixels and passes it through a stack of the following layers. Each layer except the last is followed by a tanh activation function:

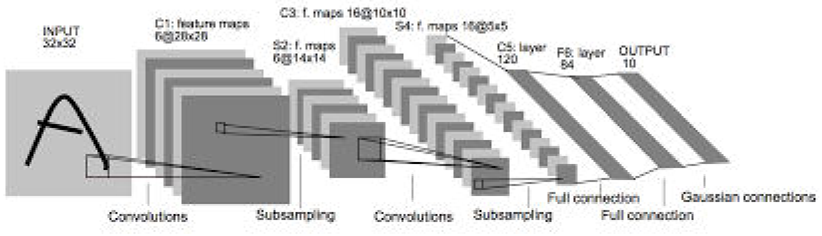
C1	The first convolutional layer. This consists of six convolutional kernels of size 5x5, which ‘walk over’ the input image. C1 outputs six images of size 28x28. The first layer of a convolutional neural network normally identifies basic features such as straight edges and corners.
S2	A subsampling layer, also known as an average pooling layer. Each square of four pixels in the C1 output is averaged to a single pixel. S2 scales down the six 28x28 images by a factor of 2, producing six output images of size 14x14.
C3	The second convolutional layer. This consists of 16 convolutional kernels, each of size 5x5, which take the six 14x14 images and walk over them again, producing 16 images of size 10x10.
S4	The second average pooling layer. S4 scales down the sixteen 10x10 images to sixteen 5x5 images.
C5	A fully connected convolutional layer with 120 outputs. Each of the 120 output nodes is connected to all of the 400 nodes (5x5x16) that came from S4. At this

	point the output is no longer an image, but a 1D array of length 120.
F6	A fully connected layer mapping the 120-array to a new array of length 10. Each element of the array now corresponds to a handwritten digit 0-9.
Output Layer	A softmax function which transforms the output of F6 into a probability distribution of 10 values which sum to 1.

LeNet-5 is one of the simplest convolutional neural networks, with six layers. This gives it enough power to distinguish small handwritten digits but not, for example, the 26 letters of the alphabet, and especially not faces or objects. Today the most sophisticated networks may have more than 30 layers and millions of parameters, and also involve branching, however the basic building blocks of convolutional kernels remain the same.



- It is **Originally trained to classify hand written digits from 0–9, of the MNIST Dataset**. It comprises of 7 — layers, all made of trainable parameters. It takes in a 32 X 32 pixel image, which was comparatively large in size w.r.t the images present in the data sets on which the network was trained. The activation function applied is **RELU** function. The layers are arranged in the following manner:



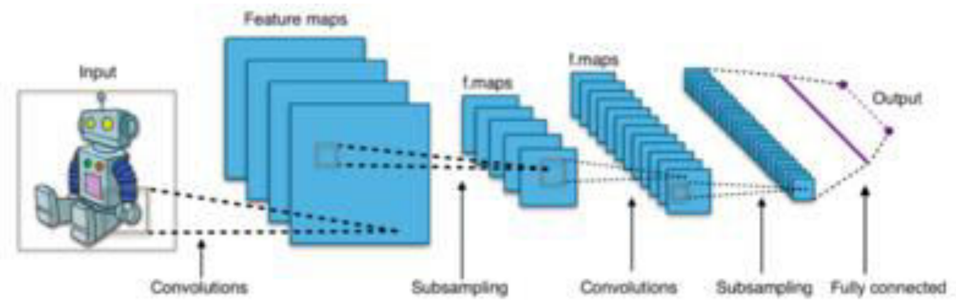
1. The **First Convolutional Layer** consist of **6 filters** of **size 5 X 5** and a **stride of 1**.
2. The **Second Layer** is a "**sub-sampling**" or **average-pooling** layer of **size 2 X 2** and a **stride of 2**.
3. The **Third Layer** is also a **Convolutional layer** consisting of **16 filters of size 5 X 5** and **stride of 1.**
4. The **Fourth Layer** is again an **average-pooling layer** of **size 2 X 2** and **stride of 2.**
5. The **Fifth Layer** is connecting the output of the *fourth layer* (**400 parameters**) to a **fully connected layer** of **120 nodes.**
6. The **Sixth Layer** is a similarly **fully-connected layer consisting** of **84 nodes**, deriving from the outputs of the 120 nodes of the *fifth-layer*.

7. The **Seventh Layer (, or the last layer)** consist of **classifying the output of the last layer into 10 classes related to the 10-digits** that it was primarily trained to classify.

It was one of the successful digit-recognition algorithm of its time implemented to classify hand written digits. Present day implementation of this architecture, on the data sets, using various libraries, would earn you an **accuracy of around 98.9 %.** However, when it came to processing large size image and classifying among a large no of classes of object, this network failed to be effective in terms of computation cost or accuracy.

Convolutional Layer

The key building block in a convolutional neural network is the convolutional layer. We can visualize a convolutional layer as many small square templates, called convolutional kernels, which slide over the image and look for patterns. Where that part of the image matches the kernel's pattern, the kernel returns a large positive value, and when there is no match, the kernel returns zero or a smaller value.



In a CNN, the input is a tensor with a shape: (number of inputs) x (input height) x (input width) x (input channels). After passing through a convolutional layer, the image becomes abstracted to a feature map, also called an activation map, with shape: (number of inputs) x (feature map height) x (feature map width) x (feature map channels).

Example Calculation of a Convolution on a Matrix

Mathematically, the kernel is a matrix of weights. For example, the following 3x3 kernel detects vertical lines.

$$\begin{bmatrix} 0 & 1 & 0 \\ 0 & 1 & 0 \\ 0 & 1 & 0 \end{bmatrix}$$

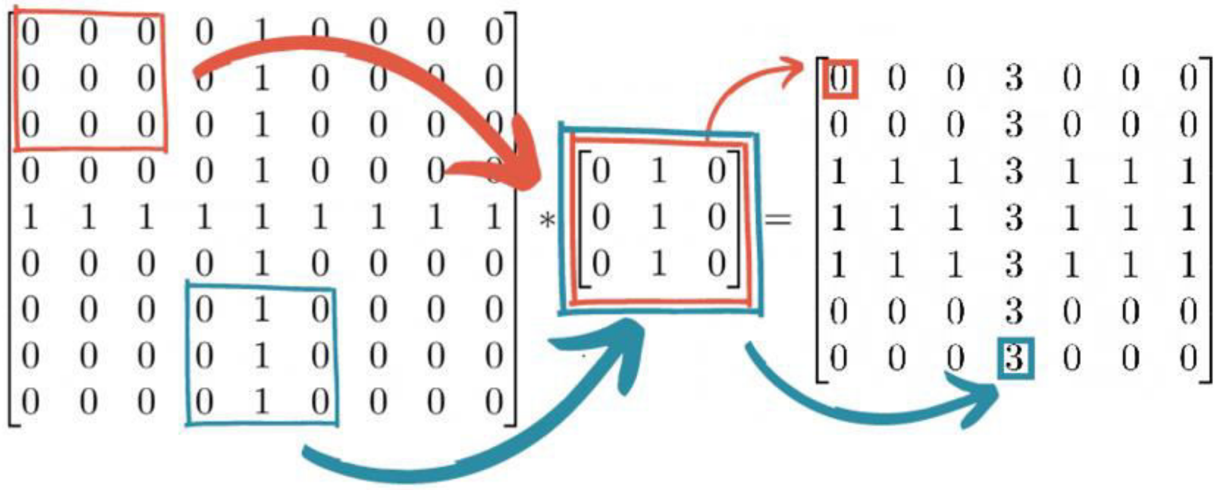
Let us imagine an 9x9 input image of a plus sign. This has two kinds of lines, horizontal and vertical, and a crossover. In matrix format the image would look as follows:

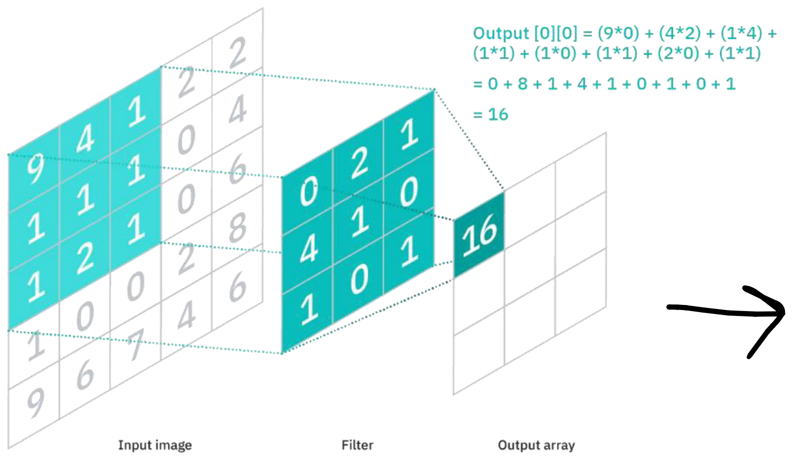
$$\begin{bmatrix} 0 & 0 & 0 & 0 & 1 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 1 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 1 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 1 & 0 & 0 & 0 & 0 \\ 1 & 1 & 1 & 1 & 1 & 1 & 1 & 1 & 1 \\ 0 & 0 & 0 & 0 & 1 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 1 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 1 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 1 & 0 & 0 & 0 & 0 \end{bmatrix}$$

Imagine we want to test the vertical line detector kernel on the plus sign image. To perform the convolution, we slide the convolution kernel over the image. At each position, we multiply each element of the convolution kernel by the element of the image that it covers, and sum the results.

Since the kernel has width 3, it can only be positioned at 7 different positions horizontally in an image of width 9. So the end result of the convolution operation on an image of size 9x9 with a 3x3 convolution kernel is a new image

of size 7x7.





just an eg

not- related to the above one

So in the above example, first the kernel is placed in the top left corner and each element of the kernel is multiplied by each element in the red box in the top left of the original image. Since these values are all 0, the result for that cell is 0 in the top left of the output matrix.

Now let us consider the position of the blue box in the above example. It contains part of a vertical line. When the kernel is placed over this vertical line, it matches and returns 3.

Recall that this convolution kernel is a vertical line detector. For the parts of the original image which contained a vertical line, the kernel has returned a value 3, whereas it has returned a value of 1 for the horizontal line, and 0 for the empty areas of the image.

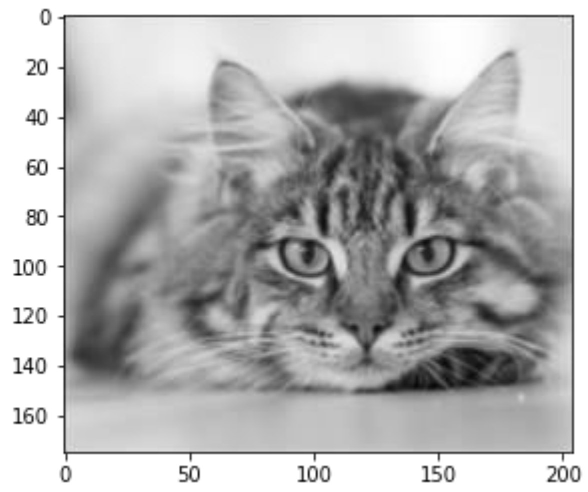
In practice, a convolution kernel contains both weights and biases, similar to the formula for linear regression. So an input pixel is multiplied by the weight and then the bias is added.

Example of Convolution on a Image

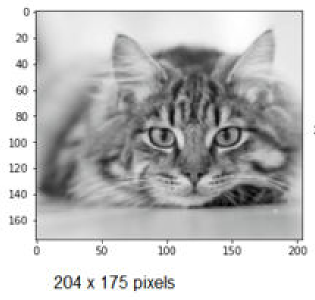
Let us consider the following 9x9 convolution kernel, which is a slightly more sophisticated vertical line detector than the kernel used in the last example:

$$\begin{bmatrix} 0 & 0 & 0 & -1 & 2 & -1 & 0 & 0 & 0 \\ 0 & 0 & 0 & -1 & 2 & -1 & 0 & 0 & 0 \\ 0 & 0 & 0 & -1 & 2 & -1 & 0 & 0 & 0 \\ 0 & 0 & 0 & -1 & 2 & -1 & 0 & 0 & 0 \\ 0 & 0 & 0 & -1 & 2 & -1 & 0 & 0 & 0 \\ 0 & 0 & 0 & -1 & 2 & -1 & 0 & 0 & 0 \\ 0 & 0 & 0 & -1 & 2 & -1 & 0 & 0 & 0 \\ 0 & 0 & 0 & -1 & 2 & -1 & 0 & 0 & 0 \\ 0 & 0 & 0 & -1 & 2 & -1 & 0 & 0 & 0 \end{bmatrix}$$

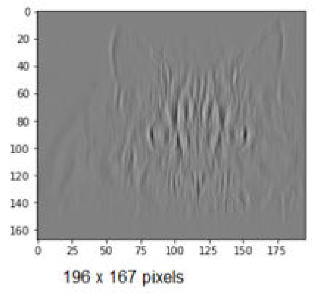
And we can take the following image of a tabby cat with dimensions 204x175, which we can represent as a matrix with values in the range from 0 to 1, where 1 is white and 0 is black.



Applying the convolution, we find that the filter has performed a kind of vertical line detection. The vertical stripes on the tabby cat's head are highlighted in the output. The output image is 8 pixels smaller in both dimensions due to the size of the kernel (9x9).



$$* \begin{bmatrix} 0 & 0 & 0 & -1 & 2 & -1 & 0 & 0 & 0 \\ 0 & 0 & 0 & -1 & 2 & -1 & 0 & 0 & 0 \\ 0 & 0 & 0 & -1 & 2 & -1 & 0 & 0 & 0 \\ 0 & 0 & 0 & -1 & 2 & -1 & 0 & 0 & 0 \\ 0 & 0 & 0 & -1 & 2 & -1 & 0 & 0 & 0 \\ 0 & 0 & 0 & -1 & 2 & -1 & 0 & 0 & 0 \\ 0 & 0 & 0 & -1 & 2 & -1 & 0 & 0 & 0 \\ 0 & 0 & 0 & -1 & 2 & -1 & 0 & 0 & 0 \\ 0 & 0 & 0 & -1 & 2 & -1 & 0 & 0 & 0 \end{bmatrix} =$$



Despite its simplicity, the ability to detect vertical or horizontal lines, corners, curves, and other simple features, is an extremely powerful property of the convolution kernel. We recall that a convolutional layer is made up of a series of convolution kernels. Typically, the first layer of a convolutional neural network contains a vertical line detector, a horizontal line detector, and various diagonal, curve and corner detectors. These feature detector kernels are not programmed by a human but in fact are learned by the neural network during training, and serve as the first stage of the image recognition process.

Later layers in the neural network are able to build on the features detected by earlier layers and identify ever more complex shapes.

Activation functions in a Convolutional Neural Network

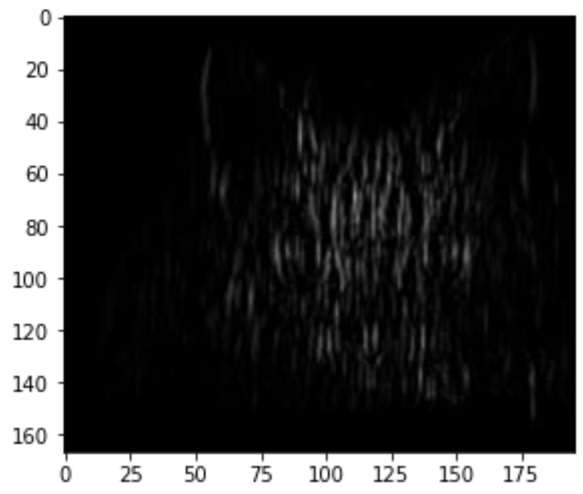
After passing an image through a convolutional layer, the output is normally passed through an activation function. Common activation functions include the sigmoid function:

$$S(x) = \frac{1}{1 + e^{-x}}$$

and the ReLU function, also known as the rectified linear unit, which is the same as taking the positive component of the input:

$$f(x) = \max(0, x)$$

The activation function has the effect of adding non-linearity into the convolutional neural network. If the activation function was not present, all the layers of the neural network could be condensed down to a single matrix multiplication. In the case of the cat image above, applying a ReLU function to the first layer output results in a stronger contrast highlighting the vertical lines, and removes the noise originating from other non-vertical features.



Repeating Structure of a Convolutional Neural Network

A basic convolutional neural network can be viewed as a series of convolutional layers, followed by an activation function, followed by a pooling (downscaling) layer, repeated many times.

With the repeated combination of these operations, the first layer detects simple features such as edges in an image, and the second layer begins to detect higher-level features. By the tenth layer, a convolutional neural network is able to detect more complex shapes such as eyes. By the twentieth layer, it is often able to differentiate human faces from one another.

This power comes from the repeated layering of operations, each of which can detect slightly higher-order features than its predecessor.

Convolutional Neural Networks vs Fully-Connected Feedforward Neural Networks

A convolutional neural network is a special kind of feedforward neural network with fewer weights than a fully-connected network.

In a fully-connected feedforward neural network, every node in the input is tied to every node in the first layer, and so on. There is no convolution kernel.

So in the example above of a 9x9 image in the input and a 7x7 image as the first layer output, if this were implemented as a fully-connected feedforward neural network, there would be

$$9 \times 9 \times 7 \times 7 = 3969 \text{ connections}$$

However, when this is implemented as a convolutional layer with a single 3x3 convolutional kernel, there are

$$3 \times 3 = 9 \text{ connections}$$

It is clear that a convolutional neural network uses far fewer parameters than the equivalent fully connected feedforward neural network with the same layer dimensions. This is because the network parameters are reused as the convolution kernel slides across the image. Intuitively, this is because a convolutional neural network should be able to detect features in an image no matter where they are located. This resilience of convolutional neural networks is called ‘translation invariance’.

Prior to the invention of convolutional neural networks, one early technique for face recognition systems, called ‘eigenfaces’, involved a direct comparison of pixels in an input image. This was not successful because it was not translation invariant. For the same reason, a fully connected network would not be appropriate for image recognition due to the lack of translation invariance, as well as the inconvenience of having to train a network with so many weights.

Note that the final layer of a convolutional neural network is normally fully connected. For example, the last layer of LeNet translates an array of length 84 to an array of length 10, by means of 840 connections.

Training a Convolutional Neural Network for Image Classification

The process of training a convolutional neural network is fundamentally the same as training any other feedforward neural network, and uses the backpropagation algorithm.

Initially, the network is created with random values in all of its weights and biases. The training examples consist of a set of tuples of images and classes.

Let us imagine the case of training a convolutional neural network to categorize images as 'cat' or 'dog'.

Each training image is passed through the entire network and the final softmax layer outputs a vector containing a probability estimate. For example the output

$$\begin{bmatrix} 0.71 \\ 0.29 \end{bmatrix}$$

can be interpreted as 71% confidence that the image is a cat and 29% confidence that it is a dog.

We can use the cross-entropy loss function, which is a measure of the accuracy of the network. When the network is initialized with random values, the loss function will be high, and the aim of training the network is to reduce the loss function as low as possible. A neural network with a low loss function classifies the training set with higher accuracy. The formula for the cross-entropy loss is as follows

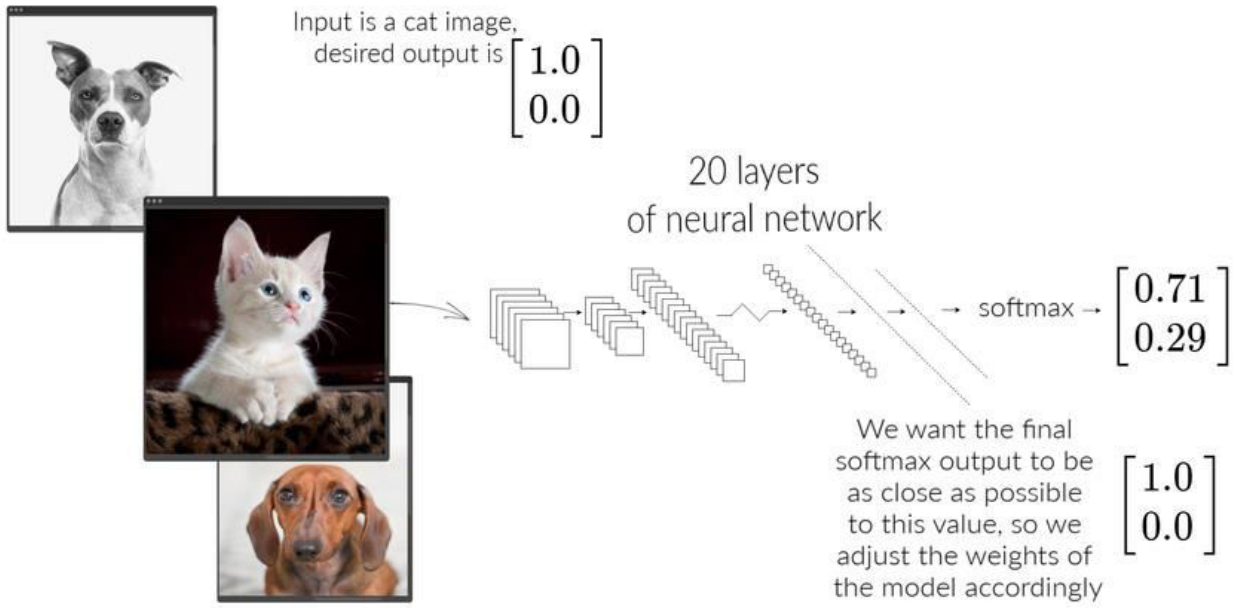
$$L = -\sum_{c=1}^{M} y_0 \log(p_{o,c})$$

Cross-entropy loss equation symbols explained

M	The number of classes that the classifier should learn. In the case of the cat vs dog classifier, M is 2.
y_o	A binary indicator, whether the label c is the correct classification for the object o
$p_{o,c}$	The neural network's predicted probability that o is of class c

We pass every training image through the network and calculate the cross-entropy loss of the network on the training set using the above formula.

Using calculus, we are then able to calculate how the weights and biases of the network must be adjusted, in order to reduce the loss further.


Input is a cat image, desired output is $\begin{bmatrix}1.0\\0.0\end{bmatrix}$
20 layers of neural network
softmax
$\begin{bmatrix}0.71\\0.29\end{bmatrix}$
We want the final softmax output to be as close as possible to this value, so we adjust the weights of the model accordingly
$\begin{bmatrix}1.0\\0.0\end{bmatrix}$

In practice, we will pass an entire batch, for example 32 images, through the network, and then calculate the loss and adjust the network parameters, and repeat for the next 32 images.

This process proceeds until we determine that the network has reached the required level of accuracy, or that it is no longer improving.

Note that this process results in the first layer of a convolutional neural network learning to detect horizontal and vertical edges without human intervention. As the training proceeds, the various layers further down the network learn to pick up useful features from the images, depending on which domain the images come from. For example, a network trained on face images learns to pick up relevant facial features.

Applications of Convolutional Neural Networks

Convolutional neural networks are most widely known for image analysis but they have also been adapted for several applications in other areas of machine learning, such as natural language processing.

Convolutional Neural Networks for Self-Driving Cars

Several companies, such as Tesla and Uber, are using convolutional neural networks as the computer vision component of a self-driving car.

A self-driving car’s computer vision system must be capable of localization, obstacle avoidance, and path planning.

Let us consider the case of pedestrian detection. A pedestrian is a kind of obstacle which moves. A convolutional neural network must be able to identify the location of the pedestrian and extrapolate their current motion in order to calculate if a collision is imminent.

A convolutional neural network for object detection is slightly more complex than a classification model, in that it must not only classify an object, but also return the four coordinates of its bounding box.

Furthermore, the convolutional neural network designer must avoid unnecessary false alarms for irrelevant objects, such as litter, but also take into account the high cost of miscategorizing a true pedestrian and causing a fatal accident.

A major challenge for this kind of use is collecting labeled training data. Google’s Captcha system is used for authenticating on websites, where a user is asked to categorize images as fire hydrants, traffic lights, cars, etc. This is actually a useful way to collect labeled training images for purposes such as self-driving cars and Google StreetView.

Convolutional Neural Networks for Text Classification

Although convolutional neural networks were initially conceived as a computer vision tool, they have been adapted for the field of natural language processing with great success.

The principle behind their use on text is very similar to the process for images, with the exception of a preprocessing stage. To use a convolutional neural network for text classification, the input sentence is tokenized and then converted into an array of word vector embeddings using a lookup such as word2vec. It is then passed through a convolutional neural network with a final softmax layer in the usual way, as if it were an image.

Consider a model which is to classify the sentence “Supreme Court to Consider Release of Mueller Grand Jury Materials to Congress” into one of two categories, ‘politics’ or ‘sport’.

Each of the 12 words in the sentence is converted to a vector, and these vectors are joined together into a matrix. Here we are using a word vector size of 5 but in practice, large numbers such as 300 are often used.

$$\begin{bmatrix} 0.417 & 0.97 & 0.484 & 0.471 & 0.327 & 0.954 & 0.037 & 0.208 & 0.916 & 0.41 & 0.357 & 0.573 \\ 0.007 & 0.729 & 0.393 & 0.436 & 0.346 & 0.032 & 0.808 & 0.457 & 0.839 & 0.825 & 0.286 & 0.552 \\ 0.541 & 0.764 & 0.398 & 0.637 & 0.798 & 0.74 & 0.086 & 0.405 & 0.831 & 0.637 & 0.999 & 0.871 \\ 0.752 & 0.529 & 0.849 & 0.665 & 0.557 & 0.463 & 0.998 & 0.529 & 0.789 & 0.373 & 0.716 & 0.85 \\ 0.274 & 0.545 & 0.007 & 0.572 & 0.922 & 0.365 & 0.598 & 0.979 & 0.071 & 0.11 & 0.751 & 0.653 \end{bmatrix}$$

This 2D matrix can be treated as an image and passed through a regular convolutional neural network, which outputs a probability for each class and which can be trained using backpropagation as in the ‘cat’ vs ‘dog’ example.

Because sentence lengths can vary, but the size of the input image to a network must be fixed, if a sentence is shorter than the maximum size then the unused values of the matrix can be padded with an appropriate value such as zeroes.

This approach to text classification also has the limitation that it cannot process sentences longer than the width of the input matrix. One workaround to this problem involves splitting sentences up into segments, passing each segment through the network individually, and averaging the output of the network over all sentences.

Convolutional Neural Networks for Drug Discovery

The first stage of a drug development program is drug discovery, where a pharmaceutical company identifies candidate compounds which are more likely to interact with the body in a certain way. Testing candidate molecules in pre-clinical or clinical trials is expensive, and so it is advantageous to be able to screen molecules as early as possible.

Proteins which play an important role in a disease are known as ‘targets’. There are targets that can cause inflammation or help tumors grow. The goal of drug discovery is to identify molecules that will interact with the target for a particular disease. The drug molecule must have the appropriate shape to interact with the target and bind to it, like a key fitting in a lock.

The San Francisco based startup Atomwise developed an algorithm called AtomNet, based on a convolutional neural network, which was able to analyze and predict interactions between molecules. Without being taught the rules of chemistry, AtomNet was able to learn essential organic chemical interactions.

Atomwise was able to use AtomNet to identify lead candidates for drug research programs. AtomNet successfully identified a candidate treatment for the Ebola virus, which had previously not been known to have any antiviral activity. The molecule later went on to pre-clinical trials.

Convolutional Neural Network History

In the 1950s and 1960s, the American David Hubel and the Swede Torsten Wiesel began to research the visual system of cats and monkeys at the Johns Hopkins School of Medicine. They published a series of papers presenting the theory that the neurons in the visual cortex are each limited to particular parts of the visual field. They further postulated that visual processing proceeds in a cascade, from neurons dedicated to simple shapes towards neurons that pick up more complex patterns. Their discoveries won them the 1981 Nobel Prize in Physiology or Medicine.

In 1980, the Japanese computer scientist Kunihiko Fukushima invented the “neocognitron”, a kind of neural network consisting of convolutional layers and downsampling layers, taking inspiration from the discoveries of Hubel and Wiesel. The neocognitron could perform some basic image processing tasks such as character recognition.

From 1987 to 1990, many researchers including Alex Waibel and Kouichi Yamaguchi further adapted the neocognitron, introducing innovations such as backpropagation which made it easier to train. Then in 1998, Yann LeCun developed LeNet, a convolutional neural network with five convolutional layers which was capable of recognizing handwritten zipcode digits with great accuracy. Another major milestone was the Ukrainian-Canadian PhD student Alex Krizhevsky’s convolutional neural network AlexNet, published in 2012. AlexNet beat all other candidates on the ImageNet image recognition competition by more than 10 percentage points, signaling the beginning of a new era in computer vision.

Until around 2015, image tasks such as face recognition were typically done by means of laborious hand coded programs that picked up facial features such as eyebrows and noses. Many OCR or face recognition applications were not using machine learning at all. The rapid acceleration in computing power, and the wide availability of large datasets, GPUs, and deep learning software, meant that around the mid 2010s, convolutional neural networks were able to deliver much better accuracy than the traditional methods and suddenly became the standard for nearly all computer vision related tasks in academia and industry. Now the average smartphone user probably has one or two apps running convolutional neural networks in their pocket, a concept that would have been unthinkable in 2010.