UNIT-IV

Convolutional Neural Networks: CNN architecture Overview, Input Layers, Convolutional Layers, Filters, Convolutional Layer Hyper parameters, Pooling Layers, CNN architectures-LeNet, ResNet, AlexNet, GoogLeNet. Transfer Learning: Transfer learning Techniques, Variants of CNN: DenseNet, PixelNet.

Basic architecture of CNN

Convolution Neural Network is a kind of deep neural network.

"convolutional neural network" indicates that the network employs a mathematical operation called convolution.

Convolutional networks are a specialized type of neural networks that use convolution in place of general matrix multiplication in at least one of their layers.

"A convolutional neural network (CNN) is a type of artificial neural network used in image recognition and processing that is specifically designed to process pixel data."

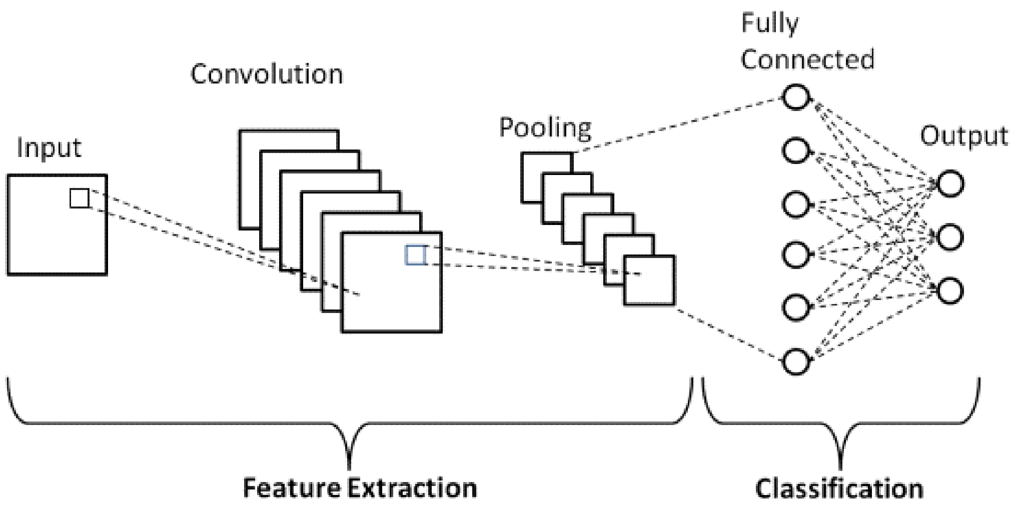
CNNs compare the images piece by piece, the pieces that it looks for all called features.

Applications of CNN

- **Self-Driving Cars**
- **Face recognition**
- **Image classification**
- **Object Detection / Object Classification / Object recognition**
- **Video Analysis**
- **Natural language processing**
- **Anomaly detection**
- **Time series forecasting etc**

There are two main parts in a CNN architecture

- A convolution tool (Convolution Layers) that separates and identifies the various features of the image for analysis in a process called as Feature Extraction
- A fully connected layer that utilizes the output from the convolution process and predicts the class of the image based on the features extracted in previous stages.

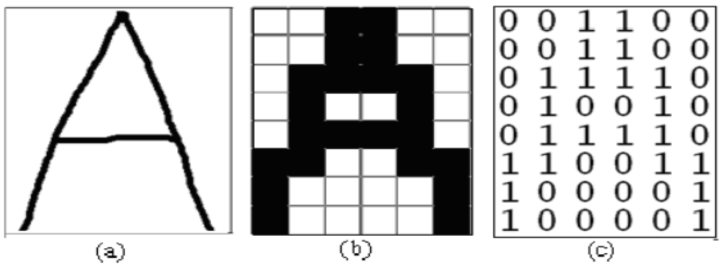


Building Block of CNN:

A convolutional neural network consists of an input layer, hidden layers and an output layer.

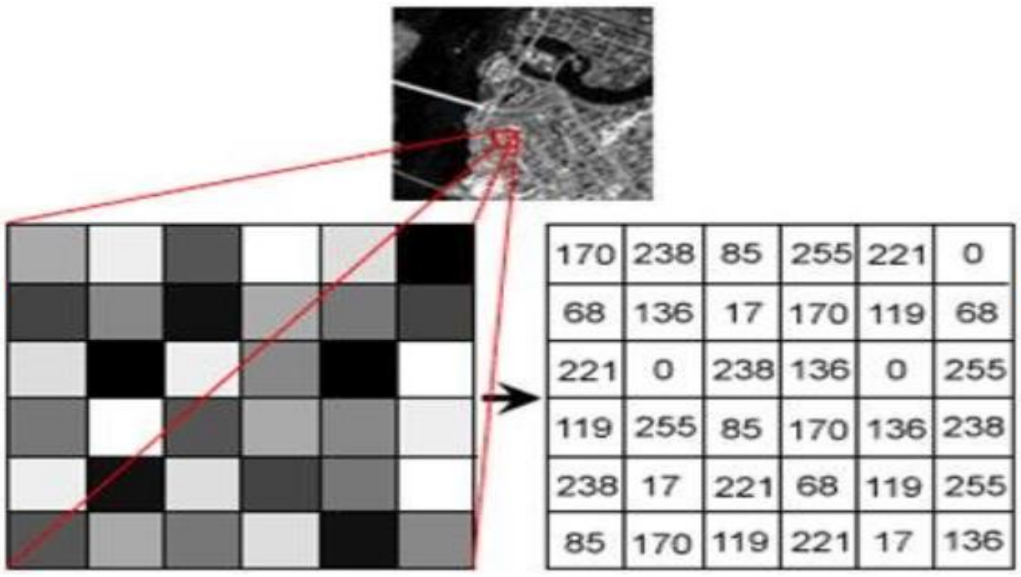
Input Layer

The input layer represents an image or video frame in pixels. (Picture Element) For gray scale image 1-bit represents 1 or 0 for black or white, 8-bit value 0-255 represents intensity of the image in a matrix. These values are represented in a 2-D vector. For colored images the RGB channels are used to represent 3 colors from values 0-255. These values can be normalized during the neural network training.



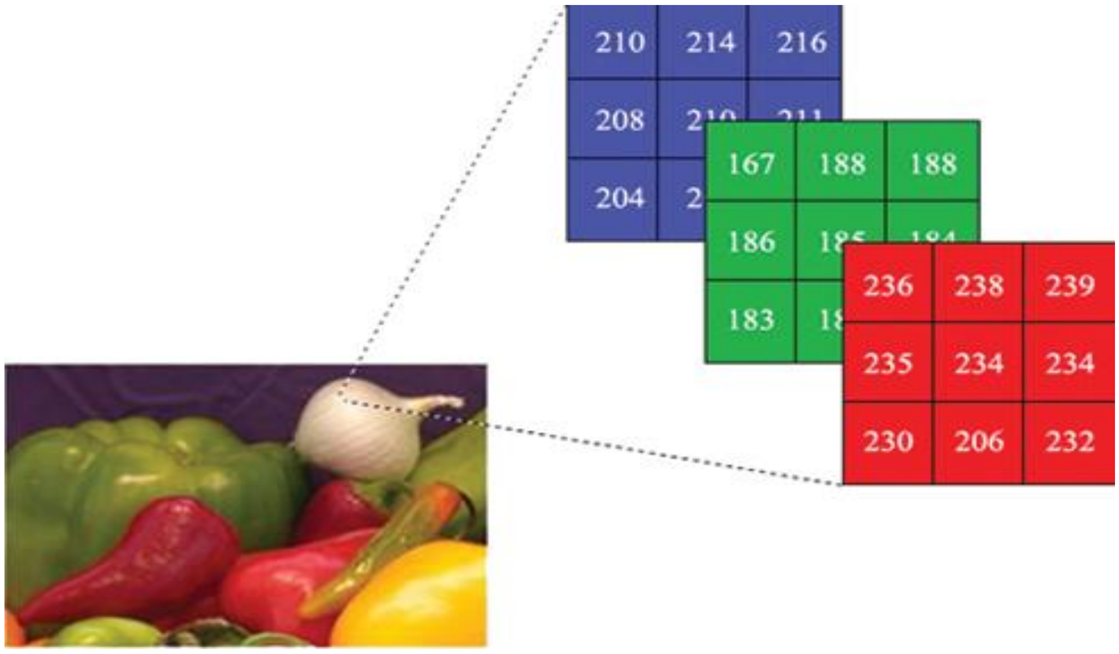
0	0	1	1	0	0
0	0	1	1	0	0
0	1	1	1	1	0
0	1	0	0	1	0
0	1	1	1	1	0
1	1	0	0	1	1
1	0	0	0	0	1
1	0	0	0	0	1

(a) Grayscale image of character 'A' (b) Binary representation of character 'A'; (c) Binary matrix representation



170	238	85	255	221	0
68	136	17	170	119	68
221	0	238	136	0	255
119	255	85	170	136	238
238	17	221	68	119	255
85	170	119	221	17	136

Matrix for certain area of a grayscale image [17].



Convolutional Layers

The hidden layers / Convolutional Layers include layers that perform convolutions. Typically, this includes a layer that performs a dot product of the convolution kernel with the layer's input matrix. This product is usually the Frobenius inner product, and its activation function is commonly ReLU.

As the convolution kernel slides along the input matrix for the layer, the convolution operation generates a feature map, which in turn contributes to the input of the next layer. This is followed by other layers such as pooling layers, fully connected layers

This layer is the first layer that is used to extract the various features from the input images. In this layer, the mathematical operation of convolution is performed between the input image and a filter of a particular size MxM. By sliding the filter over the input image, the dot product is taken between the filter and the parts of the input image with respect to the size of the filter (MxM).

The output is termed as the Feature map which gives us information about the image such as the corners, color and edges. Later, this feature map is fed to other layers to learn several other features of the input image.

The convolutional layer is the core building block of a CNN. The layer's parameters consist of a set of learnable filters (or kernels), 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 between the filter entries and the input, producing a 2-dimensional activation map of that filter. As a result, the network learns filters that activate when it detects 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.

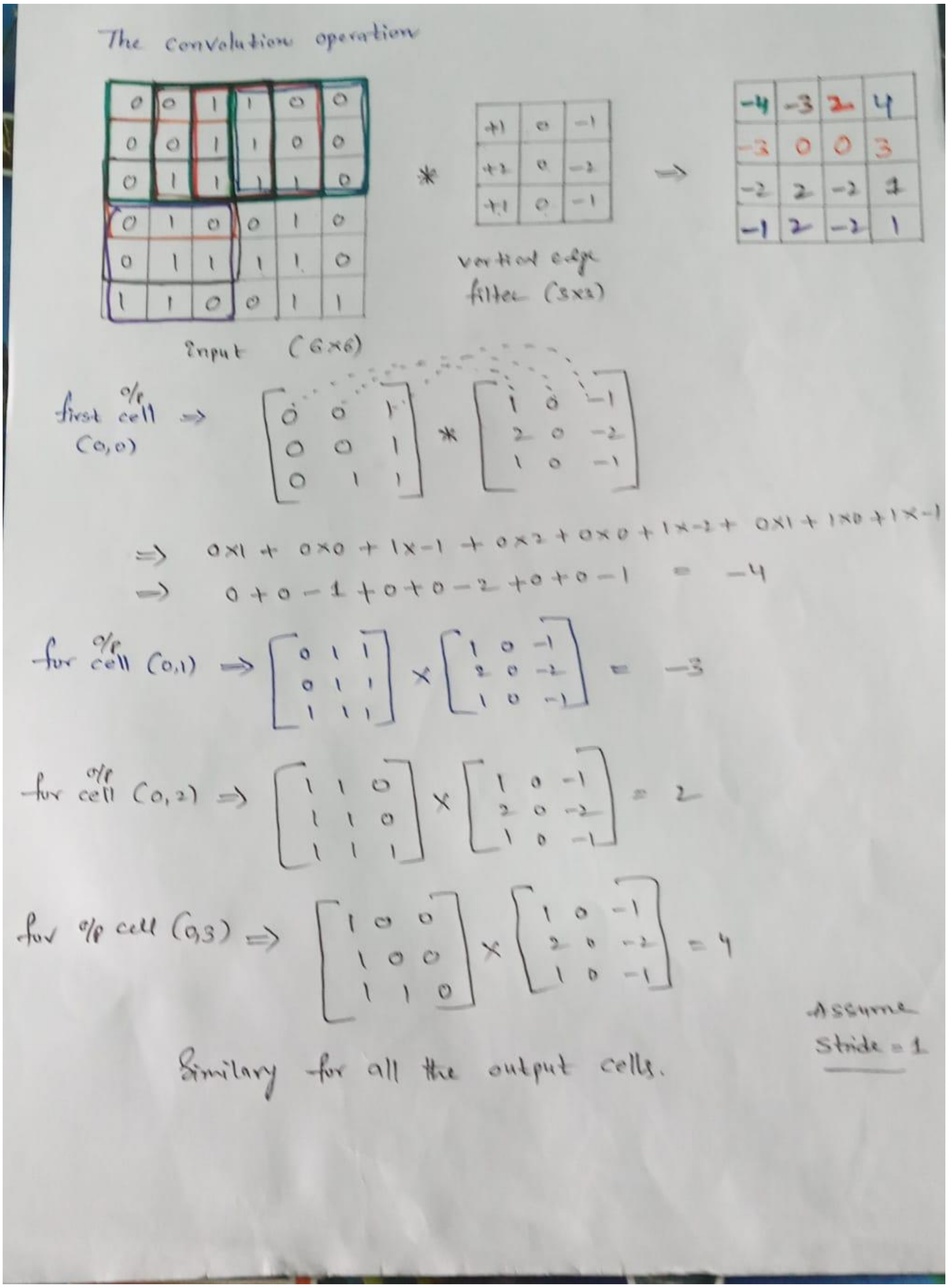
Three hyperparameters control the size of the output volume of the convolutional layer: the depth, stride, and padding size:

The depth of the output volume controls the number of neurons in a layer that connect to the same region of the input volume. These neurons learn to activate for different features in the input.

Stride controls how depth columns around the width and height are allocated. If the stride is 1, then we move the filters one pixel at a time. This leads to heavily overlapping receptive fields between the columns, and to large output volumes. For any integer S>0, a stride S means that the filter is translated S units at a time per output. In practice, S=3 is rare. A greater stride means smaller overlap of receptive fields and smaller spatial dimensions of the output volume. Striding will reduce output and loose some data when filter is applied.

Sometimes, it is convenient to pad the input with zeros (or other values, such as the average of the region or near value padding) on the border of the input volume. Padding provides control of the output volume's spatial size. In particular, sometimes it is desirable to exactly preserve the spatial size of the input volume, this is commonly referred to as "same" padding.

The convolution operation with stride=1 is



The Convolution operation

$$\begin{bmatrix} 0 & 0 & 1 & 1 & 0 & 0 \\ 0 & 0 & 1 & 1 & 0 & 0 \\ 0 & 1 & 1 & 1 & 1 & 0 \\ 0 & 1 & 0 & 0 & 1 & 0 \\ 0 & 1 & 1 & 1 & 1 & 0 \\ 1 & 1 & 0 & 0 & 1 & 1 \end{bmatrix} * \begin{bmatrix} +1 & 0 & -1 \\ +2 & 0 & -2 \\ +1 & 0 & -1 \end{bmatrix} \Rightarrow \begin{bmatrix} -4 & -3 & 2 & 4 \\ -3 & 0 & 0 & 3 \\ -2 & 2 & -2 & 1 \\ -1 & 2 & -2 & 1 \end{bmatrix}$$

Input (6×6)

vertical edge filter (3×3)

first o/p cell (0,0) $\Rightarrow \begin{bmatrix} 0 & 0 & 1 \\ 0 & 0 & 1 \\ 0 & 1 & 1 \end{bmatrix} * \begin{bmatrix} 1 & 0 & -1 \\ 2 & 0 & -2 \\ 1 & 0 & -1 \end{bmatrix}$

$\Rightarrow 0\times1 + 0\times0 + 1\times-1 + 0\times2 + 0\times0 + 1\times-2 + 0\times1 + 1\times0 + 1\times-1$

$\Rightarrow 0+0-1+0+0-2+0+0-1 = -4$

for o/p cell (0,1) $\Rightarrow \begin{bmatrix} 0 & 1 & 1 \\ 0 & 1 & 1 \\ 1 & 1 & 1 \end{bmatrix} \times \begin{bmatrix} 1 & 0 & -1 \\ 2 & 0 & -2 \\ 1 & 0 & -1 \end{bmatrix} = -3$

for o/p cell (0,2) $\Rightarrow \begin{bmatrix} 1 & 1 & 0 \\ 1 & 1 & 0 \\ 1 & 1 & 1 \end{bmatrix} \times \begin{bmatrix} 1 & 0 & -1 \\ 2 & 0 & -2 \\ 1 & 0 & -1 \end{bmatrix} = 2$

for o/p cell (0,3) $\Rightarrow \begin{bmatrix} 1 & 0 & 0 \\ 1 & 0 & 0 \\ 1 & 1 & 0 \end{bmatrix} \times \begin{bmatrix} 1 & 0 & -1 \\ 2 & 0 & -2 \\ 1 & 0 & -1 \end{bmatrix} = 4$

Assume Stride = 1

Similary for all the output cells.

Let n => Input vector of size n*n [eg. 6]

f => filter of size f*f [eg.3]

s=> stride =1

The output vector => n-f+1

⇨ 6-3+1 = 4. [4*4]

To keep the input volume and output volume same, padding is used.

If padding (p) =1, Zero Padding

0	0	0	0	0	0	0	0
0	0	0	1	1	0	0	0
0	0	0	1	1	0	0	0
0	0	1	1	1	1	0	0
0	0	1	0	0	1	0	0
0	0	1	1	1	1	0	0
0	1	1	0	0	1	1	0
0	0	0	0	0	0	0	0

The output vector => n+2p-f+1

⇨ 6+2-3+1

⇨ 6 [6*6]

Note: Padding provides to get more useful insights from the image. But computing overhead is more if convolutional layers' increases.

The generalized form for any stride and padding is

The output vector => ((n-f+2p)/s)+ 1.

Eg: if n=6, f=3, p=1 and s=1 => 6

If n=6, f=3, p=1 and s=2 => 3

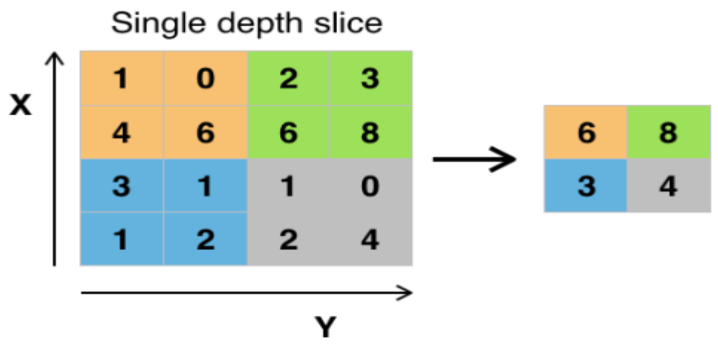
Pooling Layers

Pooling Layer is a form of non-linear down-sampling. There are several non-linear functions to implement pooling, where max pooling is the most common. It partitions the input image into a set of rectangles and, for each such sub-region, outputs the maximum.

Intuitively, the exact location of a feature is less important than its rough location relative to other features. This is the idea behind the use of pooling in convolutional neural networks. The pooling layer serves to progressively reduce the spatial size of the representation, to reduce the number of parameters, memory footprint and amount of computation in the network, and hence to also control overfitting. This is known as down-sampling.

It is common to periodically insert a pooling layer between successive convolutional layers in a CNN architecture.

There are two common types of pooling in popular use: max and average. Max pooling uses the maximum value of each local cluster of neurons in the feature map, while average pooling takes the average value. The total sum of the elements in the predefined section is computed in Sum Pooling.



In our example, after max pooling the output is

-4	-3	2	4
-3	0	0	3
-2	2	-2	1
-1	2	-2	1

After max pooling

0	4
2	1

The maximum intensity of the image is obtained using max pooling.

After Pooling, an activation function, such as a ReLU layer is used for non-linearity in the network. i.e. ReLU activation function is applied to each cell of the output vector.

ReLU applies the non-saturating activation function f(x)= max(0,x), It effectively removes negative values from an activation map by setting them to zero. It introduces nonlinearities to the decision function and in the overall network without affecting the receptive fields of the convolution layers. ReLU is often preferred to other functions because it trains the neural network several times faster. The other activation functions tanh and sigmoid also used.

The Pooling Layer usually serves as a bridge between the Convolutional Layer and the FC Layer.

During back propagation the filter/kernel values are updated, so that the output volume also changes. The process is continued to detect the images accurately.

Fully Connected Layers

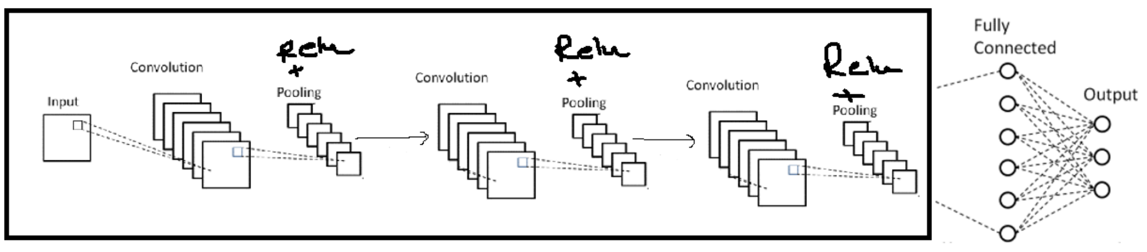
After several convolutional and max pooling layers, the final classification is done via fully connected layers. Neurons in a fully connected layer have connections to all activations in the previous layer.

The Fully Connected (FC) layer consists of the weights and biases along with the neurons and is used to connect the neurons between two different layers. These layers are usually placed before the output layer and form the last few layers of a CNN Architecture.

In this, the input image from the previous layers are flattened and fed to the FC layer. The flattened vector then undergoes few more FC layers where the mathematical functions operations usually take place. In this stage, the classification process begins to take place.

Usually, when all the features are connected to the FC layer, it can cause overfitting in the training dataset. To overcome this problem, a dropout layer is utilized wherein a few neurons are dropped from the neural network during training process resulting in reduced size of the model.

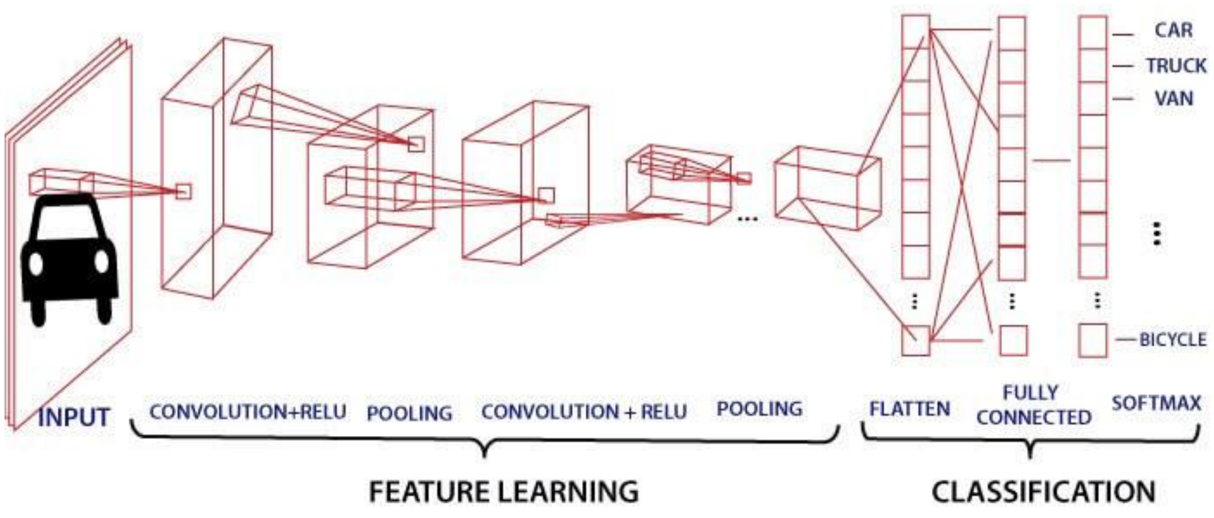
The "loss layer", or "loss function", specifies how training penalizes the deviation between the predicted output of the network and the true data labels. Various loss functions can be used, depending on the specific task like soft-max, cross-entropy etc.



CNN LAYERS - FEATURE MAPS

CLASSIFICATION

In CNN, each input image will pass through a sequence of convolution layers along with pooling, fully connected layers, filters (Also known as kernels). After that, we will apply the Soft-max function to classify an object with probabilistic values 0 and 1.



Convolutional Layer Hyper parameters

Hyper parameters are various settings that are used to control the learning process. CNNs use more hyper parameters than a standard multilayer perceptron (MLP).

Kernel / filter size: **The kernel is the number of pixels processed together. It is typically expressed as the kernel's dimensions, e.g., 2x2, or 3x3.**

Padding: **Padding is the addition of (typically) 0-valued pixels on the borders of an image. This is done so that the border pixels are not undervalued (lost) from the output because**

they would ordinarily participate in only a single receptive field instance. The padding applied is typically one less than the corresponding kernel dimension. For example, a convolutional layer using 3x3 kernels would receive a 2-pixel pad on all sides of the image.

Stride: The stride is the number of pixels that the analysis window moves on each iteration. A stride of 2 means that each kernel is offset by 2 pixels from its predecessor.

Number of filters: Since feature map size decreases with depth, layers near the input layer tend to have fewer filters while higher layers can have more. To equalize computation at each layer, the product of feature values with pixel position is kept roughly constant across layers. Preserving more information about the input would require keeping the total number of activations (number of feature maps times number of pixel positions) non-decreasing from one layer to the next.

Filter size: Common filter sizes found in the literature vary greatly, and are usually chosen based on the data set. A "Kernel" refers to a 2D array of weights. The term "filter" is for 3D structures of multiple kernels stacked together. For a 2D filter, filter is same as kernel. But for a 3D filter and most convolutions in deep learning, a filter is a collection of kernels.

Pooling type and size: Max pooling is typically used, often with a 2x2 dimension. This implies that the input is drastically down sampled, reducing processing cost. Large input volumes may warrant 4×4 pooling in the lower layers.

Greater pooling reduces the dimension of the signal, and may result in unacceptable information loss.

Dilation: Dilation involves ignoring pixels within a kernel. This reduces processing/memory potentially without significant signal loss. A dilation of 2 on a 3x3 kernel expands the kernel to 7x7, while still processing 9 (evenly spaced) pixels.

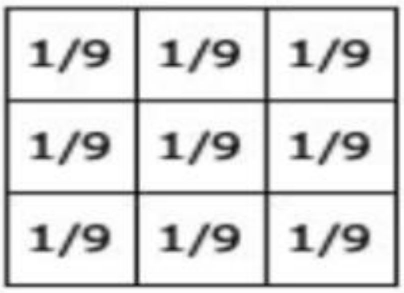
Convolutional Filters

There are multiple convolutional filters available for us to use in Convolutional Neural Networks (CNNs) to extract features from images.

Each filter will extract the specific feature of an image. Any number of filters can be applied to obtain more insights.

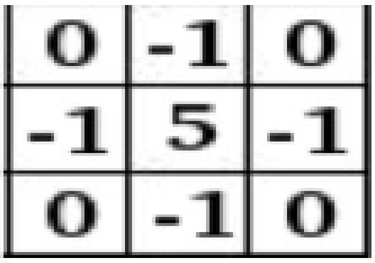
The specific type of filter is applied depends on what action is to be done on the image.

A filter used to blur an image



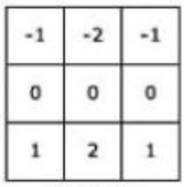
1/9	1/9	1/9
1/9	1/9	1/9
1/9	1/9	1/9

A filter used to sharpen an image



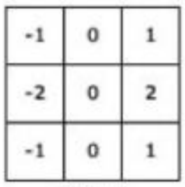
0	-1	0
-1	5	-1
0	-1	0

A filter used for edge detection



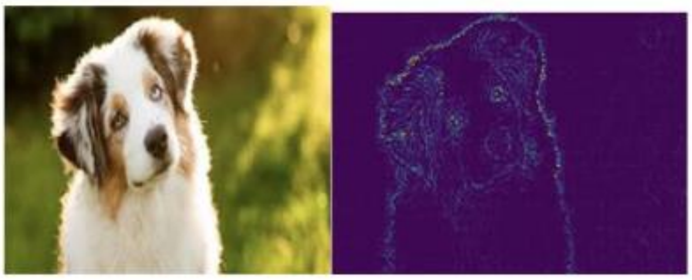
-1	-2	-1
0	0	0
1	2	1

Horizontal



-1	0	1
-2	0	2
-1	0	1

Vertical



In CNNs, filters are not defined. The value of each filter is learned during the training process.

By being able to learn the values of different filters, CNNs can find more meaning from images that humans and human designed filters might not be able to find.

More often than not, we see the filters in a convolutional layer learn to detect abstract concepts, like the boundary of a face or the shoulders of a person. By stacking layers of convolutions on top of each other, we can get more abstract and in-depth information from a CNN.

A second layer of convolution might be able to detect the shapes of eyes or the edges of a shoulder and so on. This also allows CNNs to perform hierarchical feature learning; which is how our brains are thought to identify objects.

Each layer will identify a specific feature of an image.

Transfer Learning Techniques

Transfer learning is a method where a model developed for a task is reused as the starting point for a model on a second task.

It is an approach in deep learning where pre-trained models are used as the starting point on computer vision and natural language processing tasks given the vast compute and time resources required to develop neural network models on these problems and from the huge jumps in skill that they provide on related problems.

Using transfer learning we can speed up training and improve the performance of deep learning model.

"Transfer learning and domain adaptation refer to the situation where what has been learned in one setting … is exploited to improve generalization in another setting".

"Transfer learning is the improvement of learning in a new task through the transfer of knowledge from a related task that has already been learned."

The transfer learning process in deep learning model uses Pre-Trained Model Approach as

1. Select Source Model. A pre-trained source model is chosen from available models. Many research institutions release models on large and challenging datasets that may be included in the pool of candidate models from which to choose from.

2. **Reuse Model.** The model pre-trained model can then be used as the starting point for a model on the second task of interest. This may involve using all or parts of the model, depending on the modeling technique used.
3. **Tune Model.** Optionally, the model may need to be adapted or refined on the input-output pair data available for the task of interest.

Examples of Transfer Learning with Deep Learning:

1. **Transfer Learning with Image Data:** Use image data as input. This may be a prediction task that takes photographs or video data as input. The examples are
   - Oxford VGG Model (VGGNet)
   - Google Inception Model (GoogleNet)
   - Microsoft ResNet Model (ResNet)
2. **Transfer Learning with Language Data:** Use text as input or output. For these types of problems, a word embedding is used that is a mapping of words to a high-dimensional continuous vector space where different words with a similar meaning have a similar vector representation. The examples are
   - Google's word2vec Model (Word2Vec)
   - Stanford's GloVe Model (Glove)

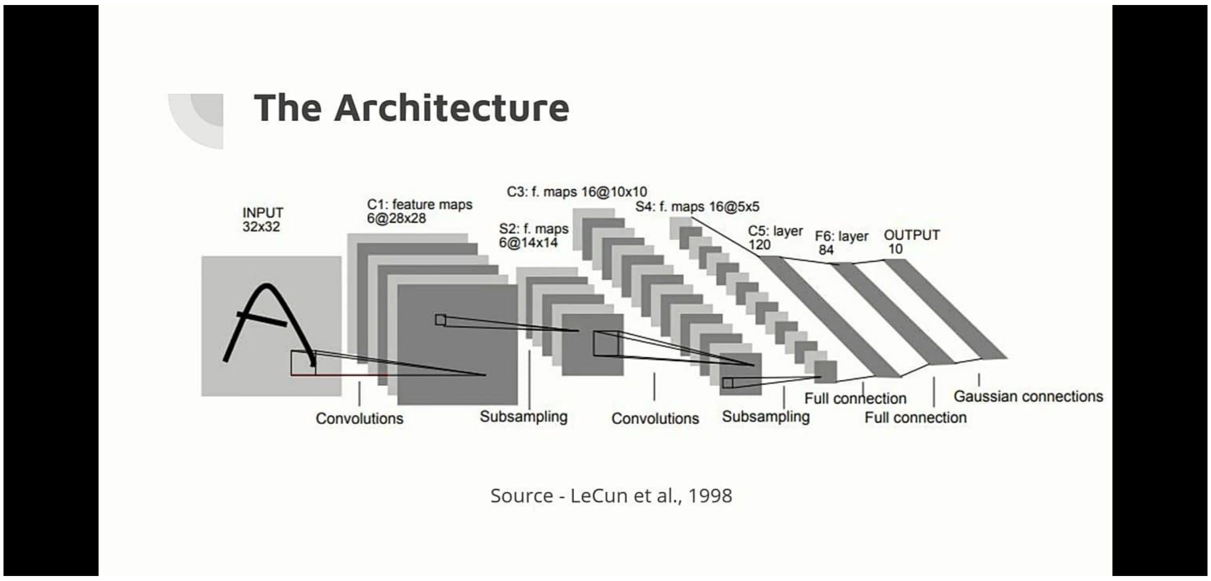
1. CNN architectures- LeNet 5

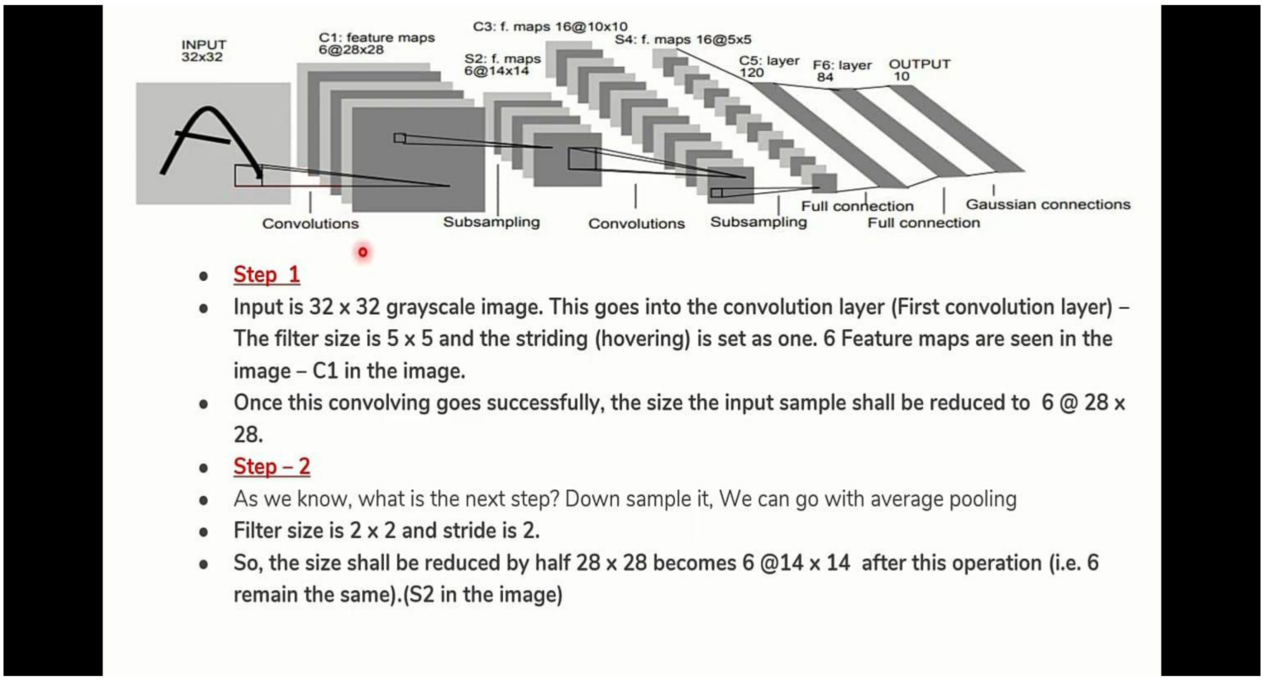
Lenet-5 is one of the earliest pre-trained models proposed by Yann LeCun

Used this architecture for recognizing the handwritten and machine-printed characters.

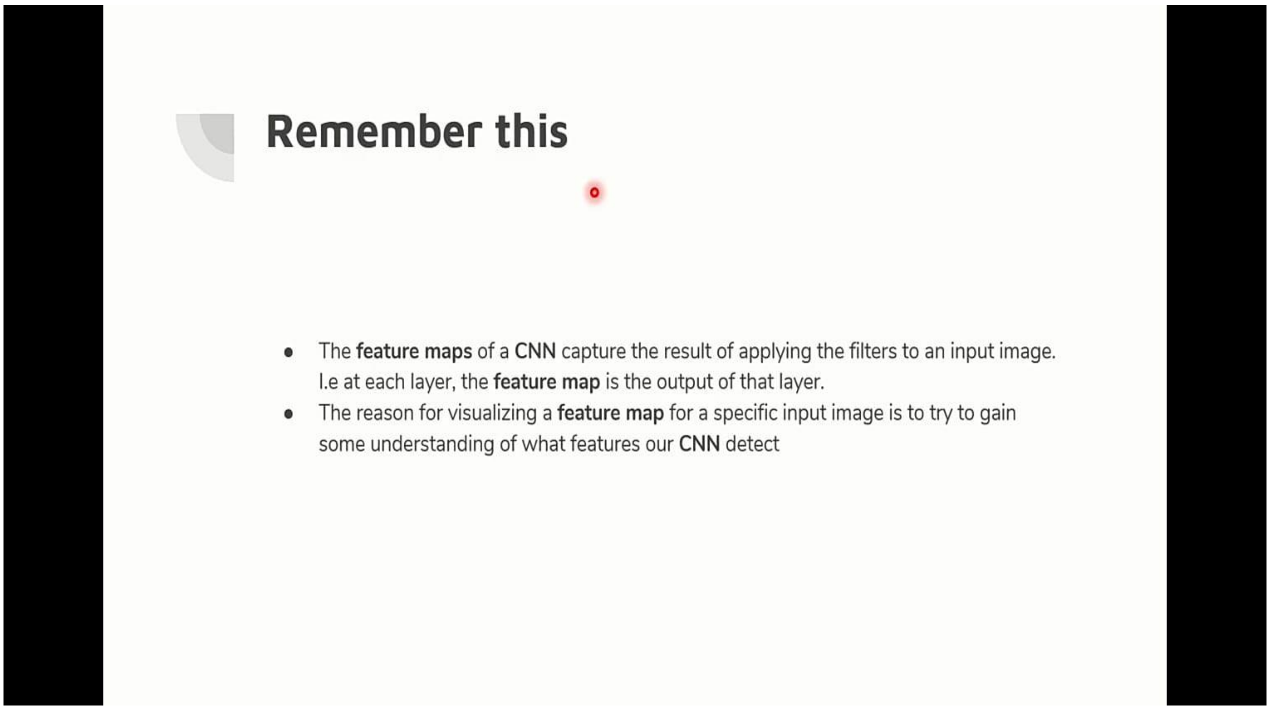
The main reason behind the popularity of this model was its simple and straightforward architecture. It is a multi-layer convolution neural network for image classification.

The network has 5 layers with learnable parameters and hence named Lenet-5. It has three sets of convolution layers with a combination of average pooling. After the convolution and average pooling layers, we have two fully connected layers. At last, a Softmax classifier which classifies the images into respective class.



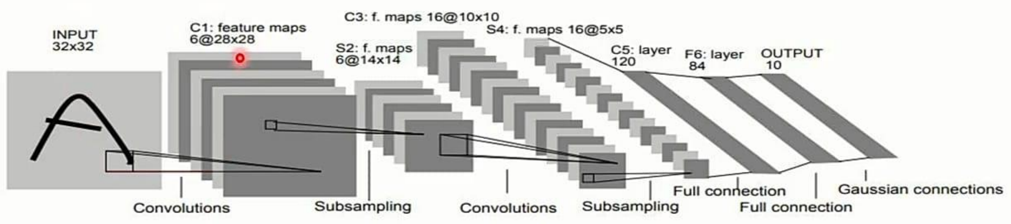


- <u>Step 1</u>
- Input is 32 x 32 grayscale image. This goes into the convolution layer (First convolution layer) – The filter size is 5 x 5 and the striding (hovering) is set as one. 6 Feature maps are seen in the image – C1 in the image.
- Once this convolving goes successfully, the size the input sample shall be reduced to 6 @ 28 x 28.
- <u>Step – 2</u>
- As we know, what is the next step? Down sample it, We can go with average pooling
- Filter size is 2 x 2 and stride is 2.
- So, the size shall be reduced by half 28 x 28 becomes 6 @14 x 14 after this operation (i.e. 6 remain the same).(S2 in the image)



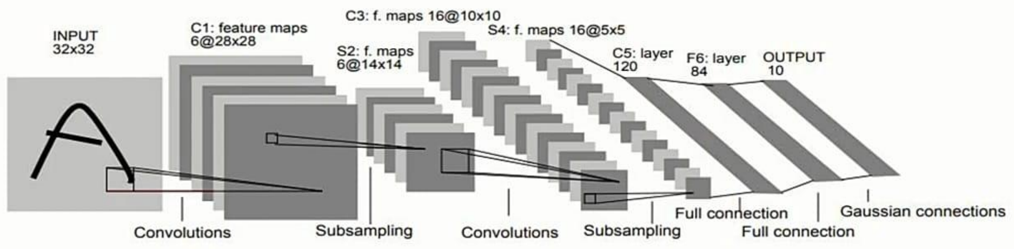
Remember this

- The **feature maps** of a **CNN** capture the result of applying the filters to an input image. I.e at each layer, the **feature map** is the output of that layer.
- The reason for visualizing a **feature map** for a specific input image is to try to gain some understanding of what features our **CNN** detect



- <u>Step 3</u>
- Convolution happens again here and 6 @14x 14 gets restructured as 16 @10x 10 with having 16 feature maps available.

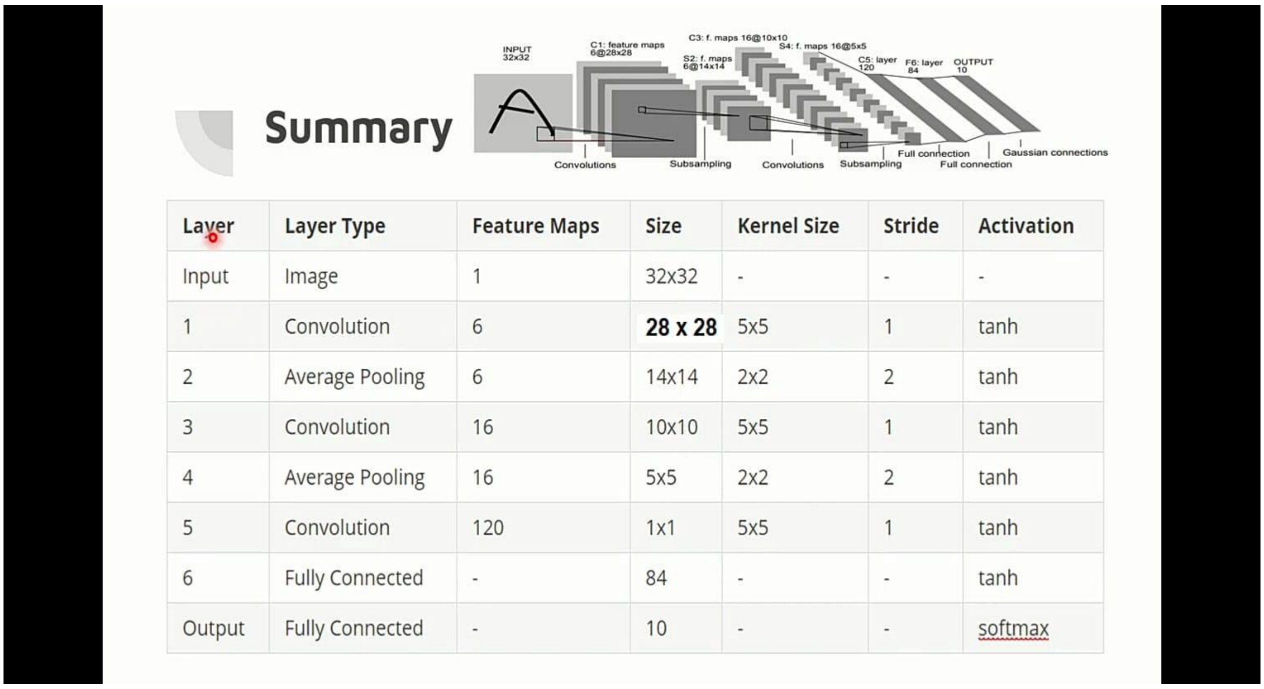
- <u>Step 4</u>
- This layer is again down sampling or subsampling layer.
- Filter size is 2 x 2 and the stride is 2.
- So, the reduction of the pixels shall happen as expended and we get 10 x 10 reduced to 16 @ 5 x 5.



- <u>Step 5</u>
- The fifth layer (C5) is a fully connected convolutional layer with 120 feature maps each of size 1×1.
- Each of the 120 units in C5 is connected to all the 400 nodes (5x5x16) in the fourth layer S4.

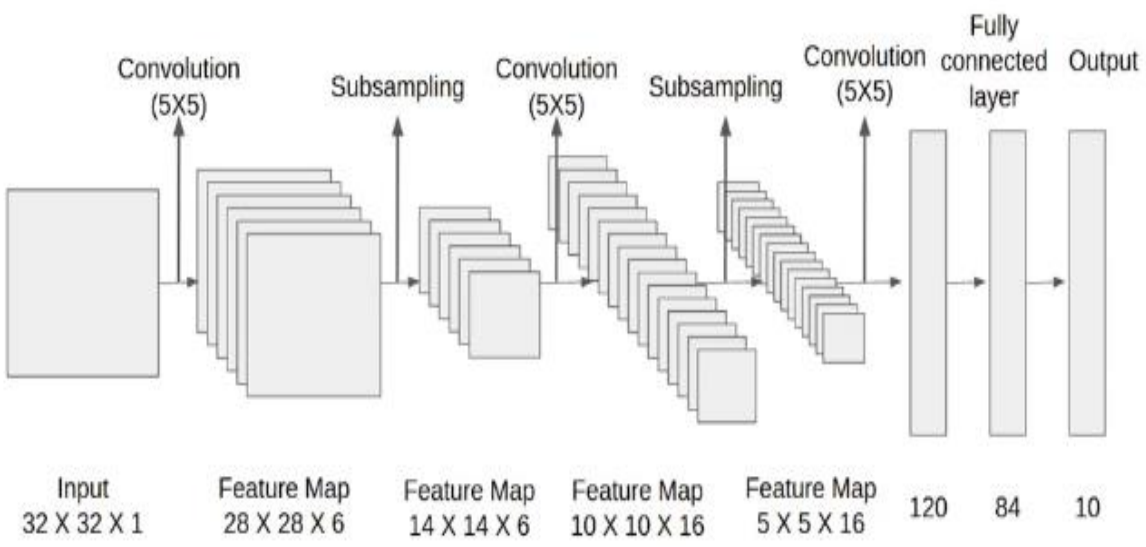
- <u>Step 6</u>
- The sixth layer is a fully connected layer (F6) with 84 units.

- <u>Step 7</u>
- Fully connected softmax output layer with 10 possible values range from 0 to 9



Summary

Layer	Layer Type	Feature Maps	Size	Kernel Size	Stride	Activation
Input	Image	1	32x32	-	-	-
1	Convolution	6	**28 x 28**	5x5	1	tanh
2	Average Pooling	6	14x14	2x2	2	tanh
3	Convolution	16	10x10	5x5	1	tanh
4	Average Pooling	16	5x5	2x2	2	tanh
5	Convolution	120	1x1	5x5	1	tanh
6	Fully Connected	-	84	-	-	tanh
Output	Fully Connected	-	10	-	-	softmax



2. CNN architectures- AlexNet

The model was proposed in 2012 in the research paper named Imagenet Classification with Deep Convolution Neural Network by Alex Krizhevsky and his colleagues.

In this model, the depth of the network was increased in comparison to Lenet-5.

The Alexnet has eight layers with learnable parameters. The model consists of five layers with a combination of max pooling followed by 3 fully connected layers and they use Relu activation in each of these layers except the output layer.

Using the relu as an activation function accelerated the speed of the training process by almost six times. The network also used the dropout layers, that prevented their model from overfitting.

The model is trained on the Imagenet dataset. The Imagenet dataset has almost 14 million images across a thousand classes.

Since Alexnet is a deep architecture, padding is introduced to prevent the size of the feature maps from reducing drastically.

The input to this model is the images of size 227X227X3.

- **It has 8 layers with learnable parameters.**
- **The input to the Model is RGB images.**
- **It has 5 convolution layers with a combination of max-pooling layers.**

- **Then it has 3 fully connected layers.**
- **The activation function used in all layers is Relu.**
- **It used two Dropout layers.**
- **The activation function used in the output layer is Softmax.**
- **The total number of learnable parameters in this architecture is 62.3 million.**

The first convolution layer with 96 filters of size 11X11 with stride 4. The activation function used in this layer is relu. The output feature map is 55X55X96. The first Maxpooling layer, of size 3X3 and stride 2. Then we get the resulting feature map with the size 27X27X96.

The second convolution operation use the filter size of 5X5 and have 256 such filters. The stride is 1 and padding 2. The activation function used is relu. Now the output size we get is 27X27X256. Again we applied a second max-pooling layer of size 3X3 with stride 2. The resulting feature map is of shape 13X13X256.

The third convolution operation with 384 filters of size 3X3 stride 1 and also padding 1. Again the activation function used is relu. The output feature map is of shape 13X13X384.

The fourth convolution operation with 384 filters of size 3X3. The stride along with the padding is 1. On top of that activation function used is relu. Now the output size remains unchanged i.e 13X13X384.

The final (fifth) convolution layer of size 3X3 with 256 such filters. The stride and padding are set to one also the activation function is relu. The resulting feature map is of shape 13X13X256. We apply the third max-pooling layer of size 3X3 and stride 2. Resulting in the feature map of the shape 6X6X256.

In this architecture, the number of filters is increasing as we are going deeper. Hence it is extracting more features as we move deeper into the architecture.

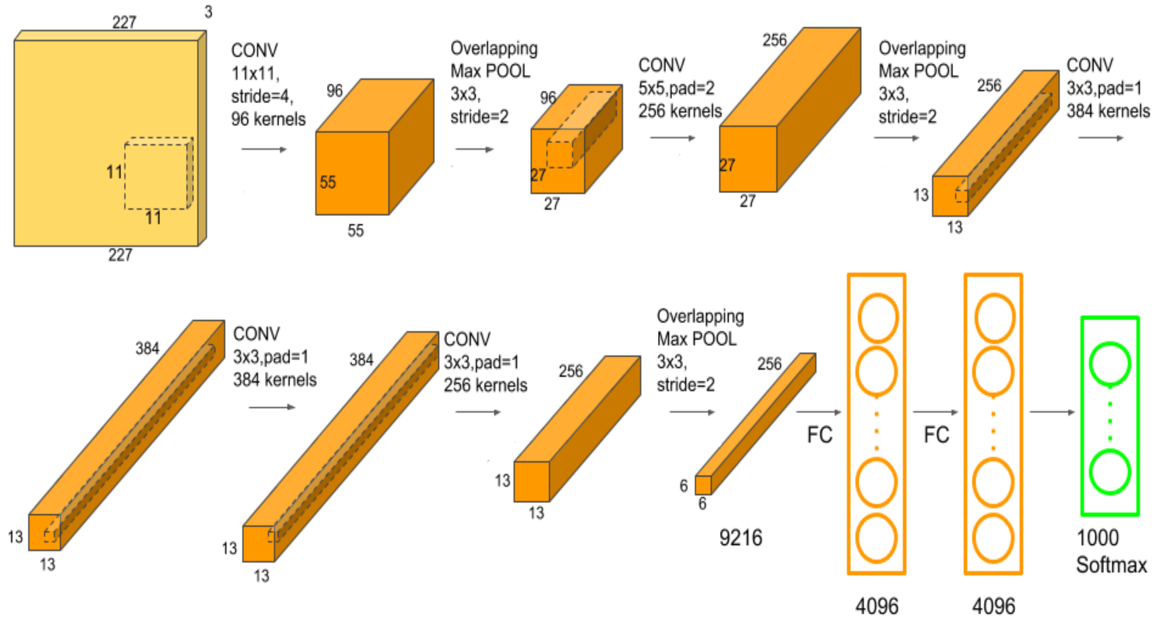
Also, the filter size is reducing, which means the initial filter was larger and as we go ahead the filter size is decreasing, resulting in a decrease in the feature map shape.

After this, we have our first dropout layer. The drop-out rate is set to be 0.5.

Then we have the first fully connected layer with a relu activation function. The size of the output is 4096.

Next comes another dropout layer with the dropout rate fixed at 0.5. This is followed by a second fully connected layer with 4096 neurons and relu activation.

Finally, we have the last fully connected layer or output layer with 1000 neurons as we have 1000 classes in the data set. The activation function used at this layer is Softmax.


227
3
CONV 11x11, stride=4, 96 kernels
96
55
55
11
11
227
Overlapping Max POOL 3x3, stride=2
96
27
27
CONV 5x5,pad=2 256 kernels
256
27
27
Overlapping Max POOL 3x3, stride=2
256
13
13
CONV 3x3,pad=1 384 kernels
384
13
13
CONV 3x3,pad=1 384 kernels
384
13
13
CONV 3x3,pad=1 256 kernels
256
13
13
Overlapping Max POOL 3x3, stride=2
256
6
6
9216
FC
4096
FC
4096
1000 Softmax

Layer	# filters / neurons	Filter size	Stride	Padding	Size of feature map	Activation function
Input	-	-	-	-	227 x 227 x 3	-
Conv 1	96	11 x 11	4	-	55 x 55 x 96	ReLU
Max Pool 1	-	3 x 3	2	-	27 x 27 x 96	-
Conv 2	256	5 x 5	1	2	27 x 27 x 256	ReLU
Max Pool 2	-	3 x 3	2	-	13 x 13 x 256	-
Conv 3	384	3 x 3	1	1	13 x 13 x 384	ReLU
Conv 4	384	3 x 3	1	1	13 x 13 x 384	ReLU
Conv 5	256	3 x 3	1	1	13 x 13 x 256	ReLU
Max Pool 3	-	3 x 3	2	-	6 x 6 x 256	-
Dropout 1	rate = 0.5	-	-	-	6 x 6 x 256	-
Fully Connected 1	-	-	-	-	4096	ReLU
Dropout 2	rate = 0.5	-	-	-	4096	-
Fully Connected 2	-	-	-	-	4096	ReLU
Fully Connected 3	-	-	-	-	1000	Softmax

Convolution Operation Key Points

Convolutions is an element wise multiplication and summation of the input and kernel/filter elements.

1. **Input matrix will have only one channel for gray scale images. Hence represented in height and width.**
   Eg: 64 * 64 * 1 is same as 64*64 pixels.

2. **Input in most cases, will have more than one channel. This is sometimes referred to as depth.**
   Eg: 64X64 pixel RGB input from an image will have 3 channels so the input is 64X64X3.

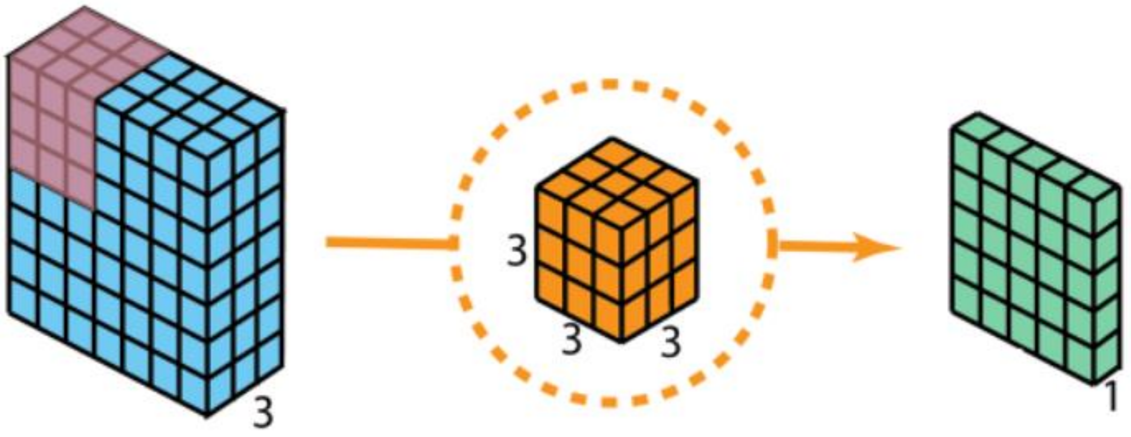
3. **The filter is also same depth as input depth.**
   Eg: Filter of 3X3 will have 3 channels as well, hence the filter should be represented as 3X3X3.

4. **The output of Convolution step will have the depth equal to number of filters we choose.**

5. **Output of Convolution step of the 3D input (64X64X3) and the filter we chose (3X3X3) will have the depth of 1 (Because we have only one filter).**

   The Convolution step on the 3D input 64X64X3 with filter size of 3X3X3 will have the filter 'sliding' along the width and height of the input.

So, when we convolve the 3D filter with the 3D image, the operation moves the filter on the input in 2 directions (Along the width and height) and we do the element wise multiplication and addition at each position to end up with an output with a depth of 1.



1X1 Convolution

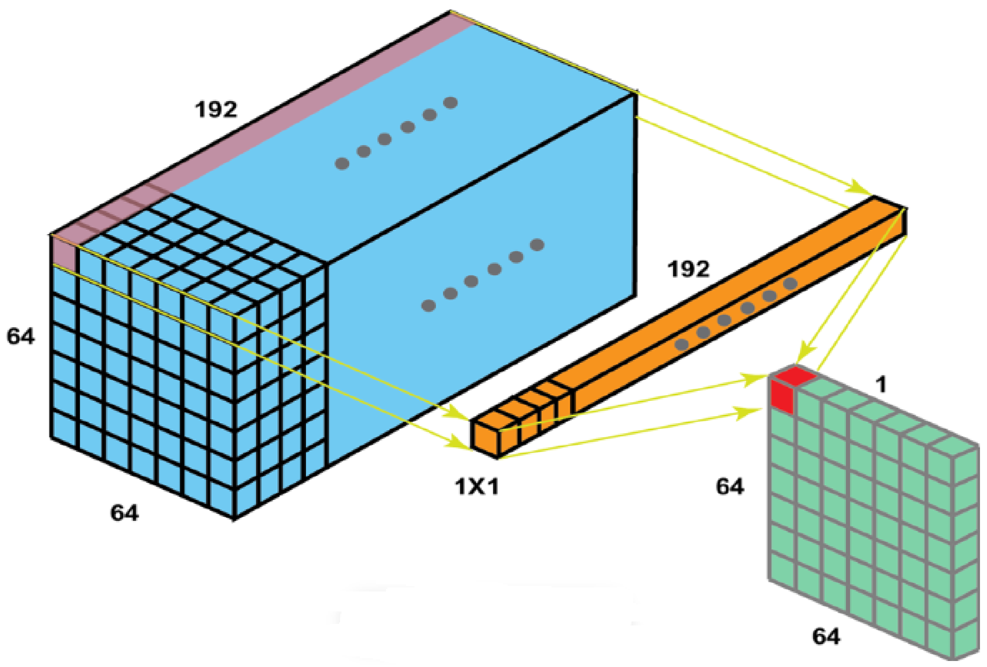
In 1X1 Convolution simply means the filter is of size 1X1 This 1X1 filter will convolve over the entire input image pixel by pixel.

Example-1: Input of 64X64X3, if we choose a 1X1 filter (which would be 1X1X3), then the output will have the same Height and Weight as input but only one channel 64X64X1.

1X1 Convolution Advantages

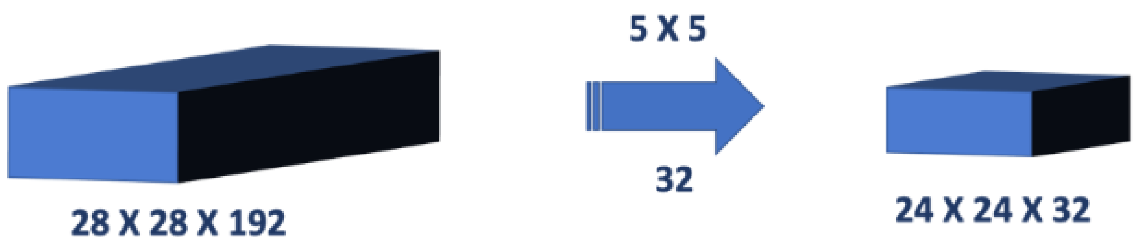
1. Provides down sampling, called **‘Dimensionality reduction’.**

   Now consider inputs with large number of channels — 192. If we want to reduce the depth and but keep the Height X Width of the feature maps (Receptive field) the same, then we can choose 1X1 filters (with Number of filters = Output Channels) to achieve this effect.



2. Reducing dimension will reduce computational load.

Suppose we need to convolve 28 X 28 X 192 input feature maps with 5 X 5 X 32 filters.

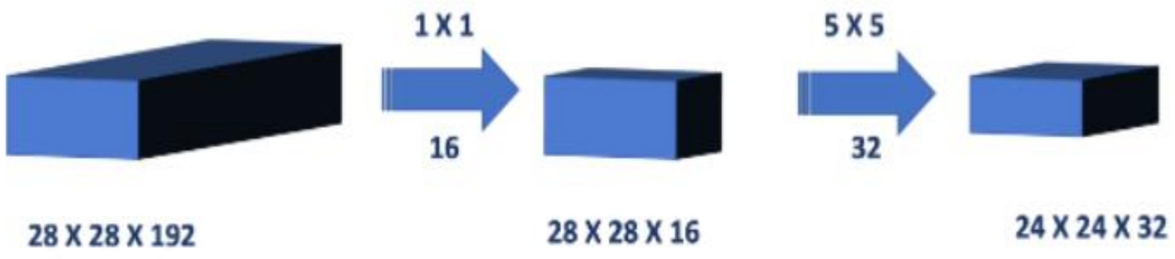


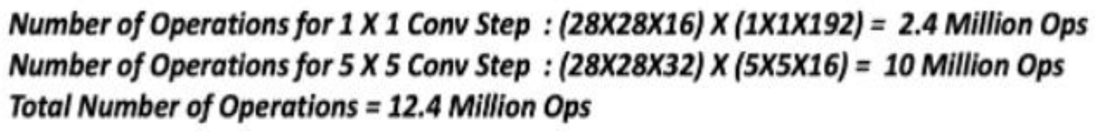
Number of Operations = (28*28*32) * (5*5*192)

= 120,422,400

= 120.422 Million operations

Now with the same input feature maps but using 1X1 Convolution layer before the 5 X 5 convolution layer




Number of Operations for 1 X 1 Conv Step : (28X28X16) X (1X1X192) = 2.4 Million Ops
Number of Operations for 5 X 5 Conv Step : (28X28X32) X (5X5X16) = 10 Million Ops
Total Number of Operations = 12.4 Million Ops

By adding 1X1 convolution layer before the 5X5 convolution, while keeping the height and width of the feature map, we have reduced the number of operations by a factor of 10.

This will reduce the computational needs and in turn will end up being more efficient.

3. CNN architectures- GoogLeNet

Google Net or (Inception V1) was proposed by research at Google in 2014 with the research paper titled "Going deeper with convolutions".

The first version of Inception network is called Google Net.

The architecture is the winner of ILSVRC 2014 image classification and detection challenge. (ImageNet Large-Scale Visual Recognition Challenge).

The architecture takes image of size 224 * 224 with RGB color channels.

All the convolutions inside the architecture uses ReLU activation function.

Provides a significant decrease in the error rate when compared with AlexNet and VGG Network.

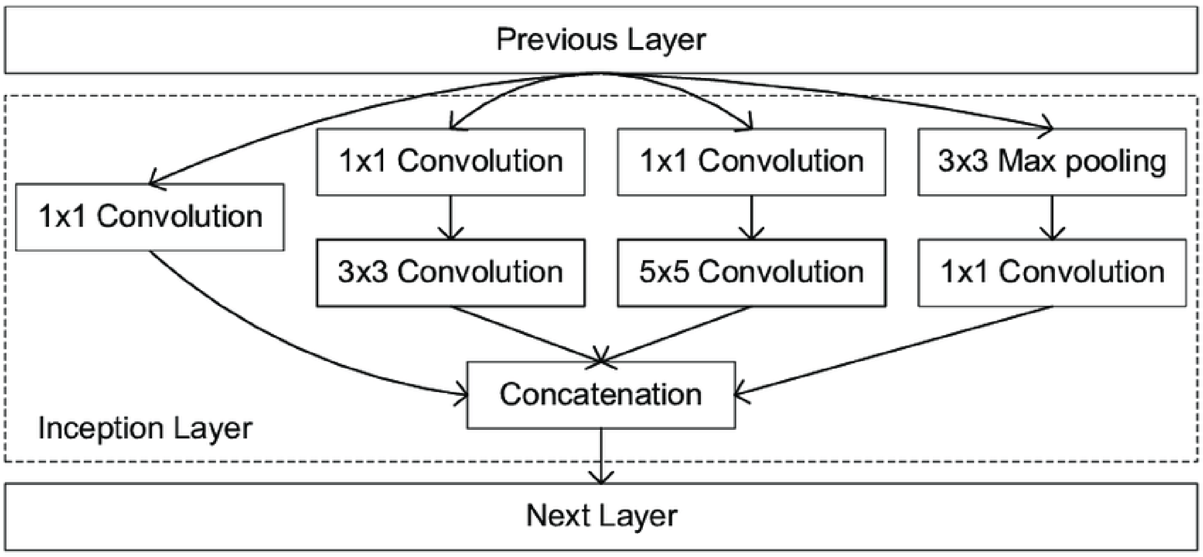
- If a network is built with many deep layers it might face the problem of overfitting. To solve this problem, uses filters with multiple sizes that can operate on the same level. With this idea, the network actually becomes wider rather than deeper.
- The main hallmark of this architecture is the improved utilization of the computing resources inside the network. By a carefully crafted design, we increased the depth and width of the network while keeping the computational budget constant.

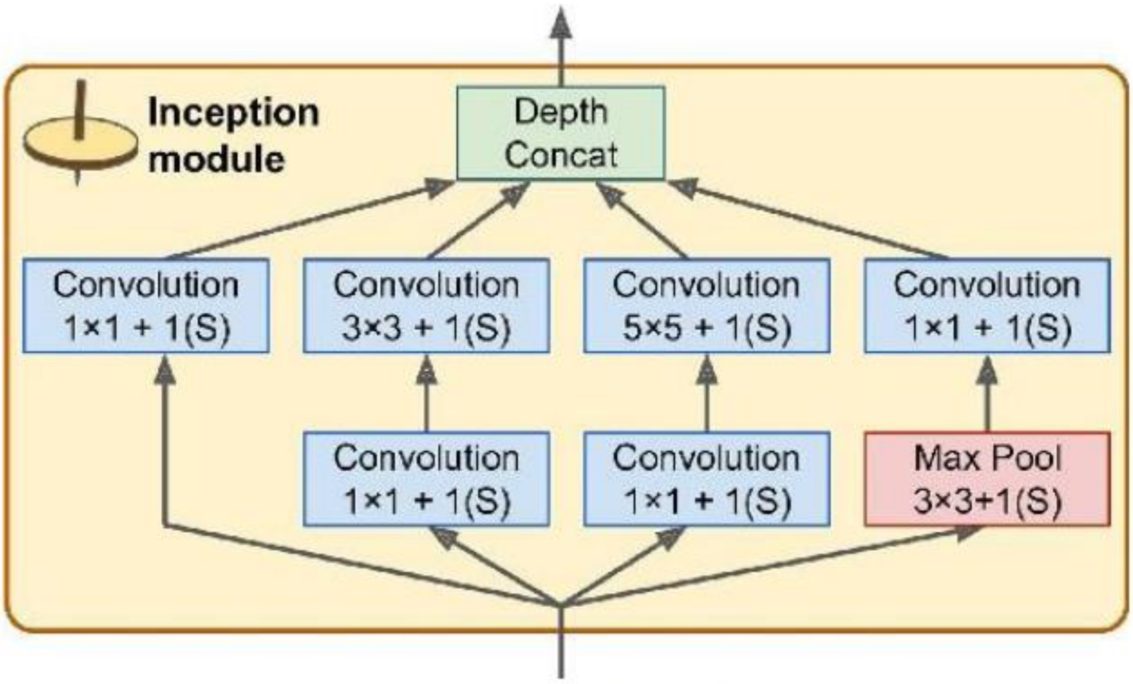
This great performance came in large part from the fact that the network was much deeper than previous CNNs.

The major introduction in this network is

1. **Inception Layers**
2. **Global Average Pooling.**

This was made possible by sub-networks called inception modules which allow GoogLeNet to use parameters much more efficiently than previous architectures:





Inception module

The above diagram shows the architecture of an inception module.

The notation "3 × 3 + 2(S)" means that the layer uses a 3 × 3 kernel, stride 2, and SAME padding.

The input signal is first copied and fed to four different layers. All convolutional layers use the ReLU activation function.

The second set of convolutional layers uses different kernel sizes (1 × 1, 3 × 3, and 5 × 5), allowing them to capture patterns at different scales. Also every single layer uses a stride of 1 and SAME padding (including the max pooling layer), so their outputs all have the same height and width as their inputs.

This makes it possible to concatenate all the outputs along the depth dimension in the final depth concat layer (i.e., stack the feature maps from all four top convolutional layers).

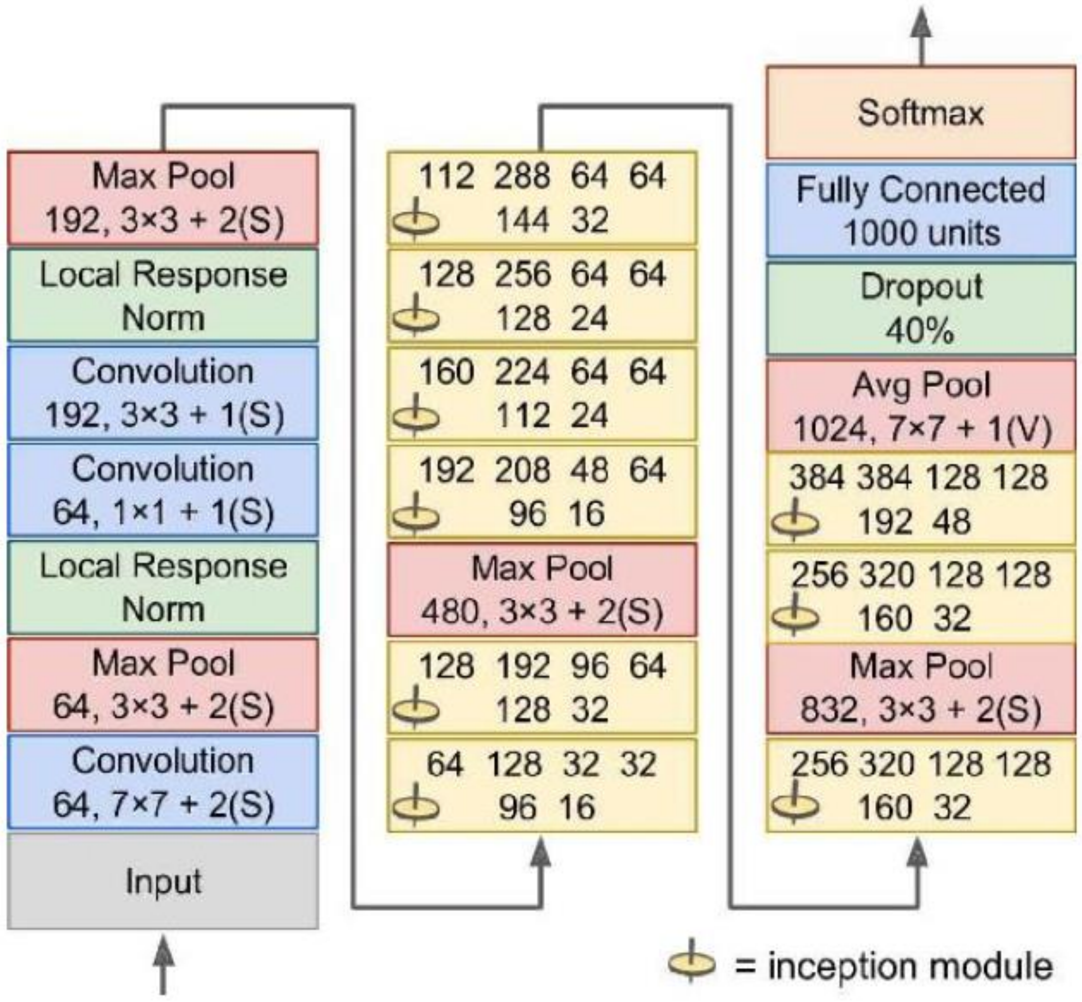
The inception modules have convolutional layers with 1 × 1 kernels. These layers cannot capture any features since they look at only one pixel at a time? In fact, these layers serve two purposes:

1. These are configured to output many fewer feature maps than their inputs, so they serve as bottleneck layers, meaning they reduce dimensionality. This is particularly useful before the 3 × 3 and 5 × 5 convolutions, since these are very computationally expensive layers.

2. **Second, each pair of convolutional layers ([1 × 1, 3 × 3] and [1 × 1, 5 × 5]) acts like a single, powerful convolutional layer, capable of capturing more complex patterns.**

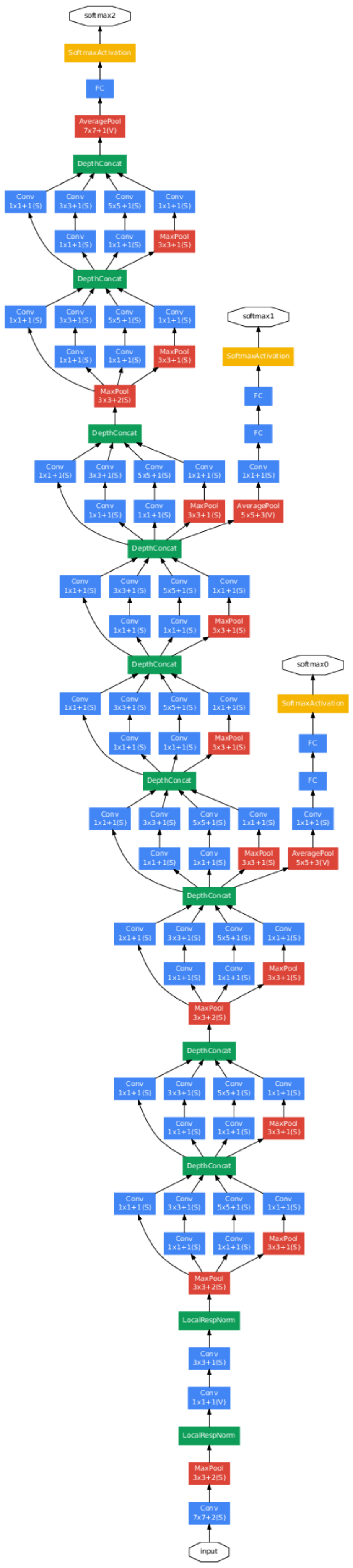
The Inception module able to output feature maps that capture complex patterns at various scales.

The architecture of GoogleNet CNN has 22 Layers with the following sequence of operations.



GoogLeNet architecture

- The first two layers divide the image’s height and width by 4 (so its area is divided by 16), to reduce the computational load.
- Then the local response normalization layer ensures that the previous layers learn a wide variety of features.
- Two convolutional layers follow, where the first acts like a bottleneck layer. As explained earlier, you can think of this pair as a single smarter convolutional layer.
- Again, a local response normalization layer ensures that the previous layers capture a wide variety of patterns.
- Next a max pooling layer reduces the image height and width by 2, again to speed up computations.
- Then comes the tall stack of nine inception modules, interleaved with a couple max pooling layers to reduce dimensionality and speed up the net.
- Next, the average pooling layer uses a kernel the size of the feature maps with VALID padding, outputting 1 × 1 feature maps: this surprising strategy is called global average pooling. It effectively forces the previous layers to produce feature maps that are actually confidence maps for each target class (since other kinds of features would be destroyed by the averaging step). This makes it unnecessary to have several fully connected layers at the top of the CNN (like in AlexNet), considerably reducing the number of parameters in the network and limiting the risk of overfitting.
- The last layer dropout for regularization, then a fully connected layer with a softmax activation function to output estimated class probabilities.


softmax2
SoftmaxActivation
FC
AveragePool 7x7+1(V)
DepthConcat
Conv 1x1+1(S)
Conv 3x3+1(S)
Conv 5x5+1(S)
Conv 1x1+1(S)
Conv 1x1+1(S)
Conv 1x1+1(S)
MaxPool 3x3+1(S)
DepthConcat
Conv 1x1+1(S)
Conv 3x3+1(S)
Conv 5x5+1(S)
Conv 1x1+1(S)
softmax1
Conv 1x1+1(S)
Conv 1x1+1(S)
MaxPool 3x3+1(S)
SoftmaxActivation
MaxPool 3x3+2(S)
FC
DepthConcat
FC
Conv 1x1+1(S)
Conv 3x3+1(S)
Conv 5x5+1(S)
Conv 1x1+1(S)
Conv 1x1+1(S)
Conv 1x1+1(S)
Conv 1x1+1(S)
MaxPool 3x3+1(S)
AveragePool 5x5+3(V)
DepthConcat
Conv 1x1+1(S)
Conv 3x3+1(S)
Conv 5x5+1(S)
Conv 1x1+1(S)
Conv 1x1+1(S)
Conv 1x1+1(S)
MaxPool 3x3+1(S)
DepthConcat
softmax0
Conv 1x1+1(S)
Conv 3x3+1(S)
Conv 5x5+1(S)
Conv 1x1+1(S)
SoftmaxActivation
Conv 1x1+1(S)
Conv 1x1+1(S)
MaxPool 3x3+1(S)
FC
DepthConcat
FC
Conv 1x1+1(S)
Conv 3x3+1(S)
Conv 5x5+1(S)
Conv 1x1+1(S)
Conv 1x1+1(S)
Conv 1x1+1(S)
Conv 1x1+1(S)
MaxPool 3x3+1(S)
AveragePool 5x5+3(V)
DepthConcat
Conv 1x1+1(S)
Conv 3x3+1(S)
Conv 5x5+1(S)
Conv 1x1+1(S)
Conv 1x1+1(S)
Conv 1x1+1(S)
MaxPool 3x3+1(S)
MaxPool 3x3+2(S)
DepthConcat
Conv 1x1+1(S)
Conv 3x3+1(S)
Conv 5x5+1(S)
Conv 1x1+1(S)
Conv 1x1+1(S)
Conv 1x1+1(S)
MaxPool 3x3+1(S)
DepthConcat
Conv 1x1+1(S)
Conv 3x3+1(S)
Conv 5x5+1(S)
Conv 1x1+1(S)
Conv 1x1+1(S)
Conv 1x1+1(S)
MaxPool 3x3+1(S)
MaxPool 3x3+2(S)
LocalRespNorm
Conv 3x3+1(S)
Conv 1x1+1(V)
LocalRespNorm
MaxPool 3x3+2(S)
Conv 7x7+2(S)
input

CNN Architecture: ResNet

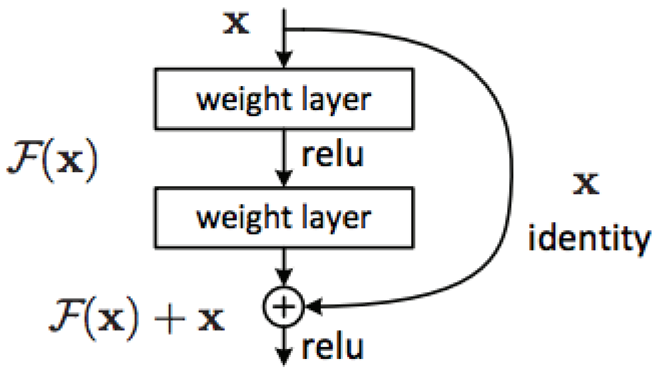
Deep Residual Learning is used for Image Recognition.

The intuition behind deep neural networks with several layers is these layers progressively learn more complex features. The first layer learns edges, the second layer learns shapes, the third layer learns objects, the fourth layer learns other parts, and so on. Hence a deep neural network with several layers of convolutions is used.

The limitation of deep convolutions is choosing optimization function, initialization of the network, or the vanishing/exploding gradient problem.

Using batch normalization vanishing/exploding gradient problem is resolved as each layer normalizes the data.

The problem of training very deep networks has been minimized with the introduction of a new neural network layer — The Residual Block.


x
weight layer
$\mathcal{F}(\mathbf{x})$
relu
x
weight layer
identity
$\mathcal{F}(\mathbf{x}) + \mathbf{x}$
relu

Residual learning: a building block.

The most important modification in this architecture is the 'Skip Connection', identity mapping.

This identity mapping does not have any parameters and is just there to add the output from the previous layer to the layer ahead.

However, sometimes x and F(x) will not have the same dimension. Recall that a convolution operation typically shrinks the spatial resolution of an image, e.g. a 3x3 convolution on a 32 x 32 image results in a 30 x 30 image.

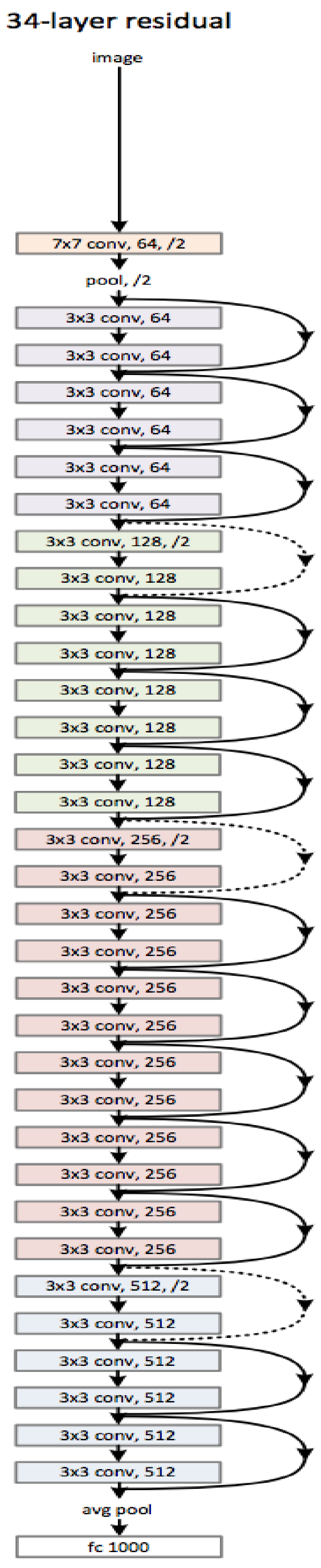
The identity mapping is multiplied by a linear projection W to expand the channels of shortcut to match the residual. This allows for the input x and F(x) to be combined as input to the next layer.

$$\mathbf{y} = \mathcal{F}(\mathbf{x}, \{W_i\}) + W_s\mathbf{x}.$$

when F(x) and x have a different dimensionality such as 32x32 and 30x30. This Ws term can be implemented with 1x1 convolutions, this introduces additional parameters to the model.

The Skip Connections between layers add the outputs from previous layers to the outputs of stacked layers. This results in the ability to train much deeper networks than what was previously possible.

In the image below, the dotted skip connections represent multiplying the identity mapping by the Ws linear projection


34-layer residual
image
7x7 conv, 64, /2
pool, /2
3x3 conv, 64
3x3 conv, 64
3x3 conv, 64
3x3 conv, 64
3x3 conv, 64
3x3 conv, 64
3x3 conv, 128, /2
3x3 conv, 128
3x3 conv, 128
3x3 conv, 128
3x3 conv, 128
3x3 conv, 128
3x3 conv, 128
3x3 conv, 128
3x3 conv, 256, /2
3x3 conv, 256
3x3 conv, 256
3x3 conv, 256
3x3 conv, 256
3x3 conv, 256
3x3 conv, 256
3x3 conv, 256
3x3 conv, 256
3x3 conv, 256
3x3 conv, 256
3x3 conv, 256
3x3 conv, 512, /2
3x3 conv, 512
3x3 conv, 512
3x3 conv, 512
3x3 conv, 512
3x3 conv, 512
avg pool
fc 1000

Variants of CNN: DenseNet

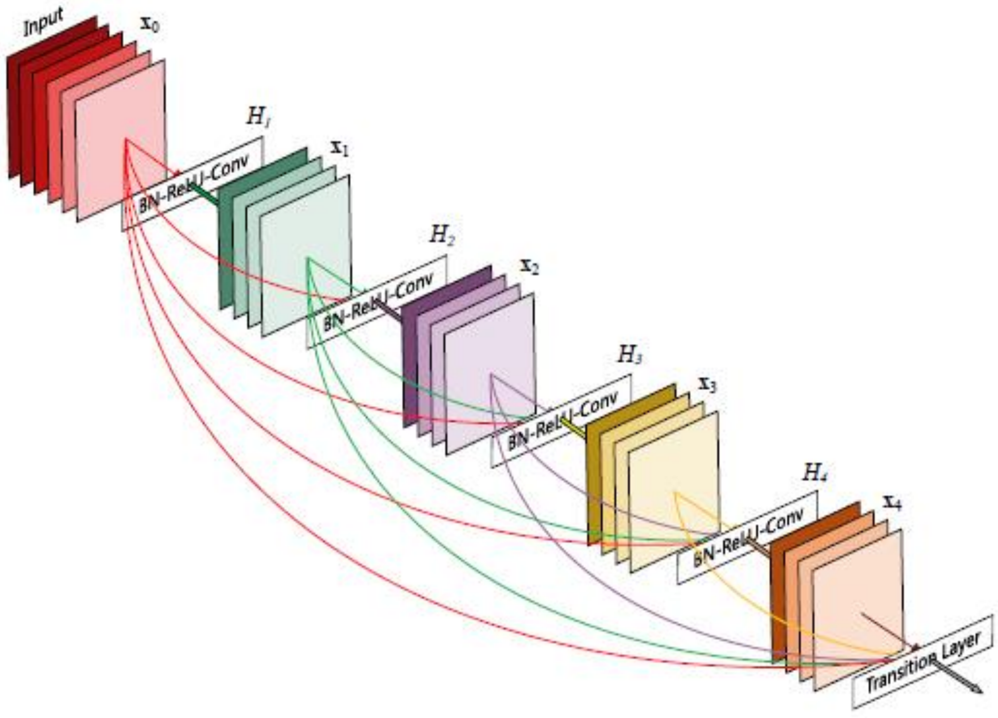
Densely Connected Convolutional Networks, DenseNets, are the next step on the way to keep increasing the depth of deep convolutional networks.

The problems arise with CNNs when they go deeper. This is because the path for information from the input layer until the output layer becomes so big, that they can get vanished before reaching the other side. DenseNets simplify the connectivity pattern between layers.

Instead of drawing representational power from extremely deep or wide architectures, DenseNets exploit the potential of the network through feature reuse.

The Dense Convolutional Network (DenseNet), which connects each layer to every other layer in a feed-forward fashion.

The architecture ensures maximum information flow between layers in the network, we connect all layers (with matching feature-map sizes) directly with each other. To preserve the feed-forward nature, each layer obtains additional inputs from all preceding layers and passes on its own feature-maps to all subsequent layers.



A 5-layer dense block with a growth rate of $k = 4$. Each layer takes all preceding feature-maps as input.

DenseNets do not sum the output feature maps of the layer with the incoming feature maps but concatenate them.

DenseNets are divided into DenseBlocks, where the dimensions of the feature maps remains constant within a block, but the number of filters changes between them. These layers between them are called Transition Layers and take care of the downsampling applying a batch normalization, a 1x1 convolution and a 2x2 pooling layers.

Since we are concatenating feature maps, this channel dimension is increasing at every layer. If we make h_l to produce k feature maps every time, then we can generalize for the l-th layer:

$$k_l = k_0 + k * (l - 1)$$

This hyperparameter k is the growth rate. The growth rate regulates how much information is added to the network each layer.

i.e We could see the feature maps as the information of the network. Every layer has access to its preceding feature maps, and therefore, to the collective knowledge. Each layer is then adding a new information to this collective knowledge, in concrete k feature maps of information.

Variants of CNN: PixelNet

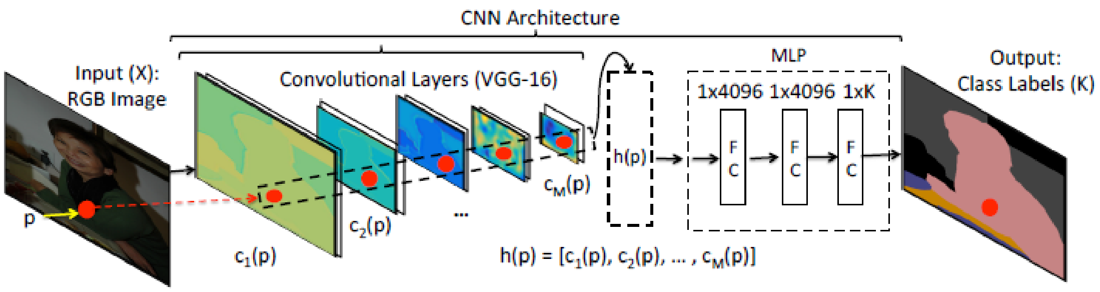
“Representation of the pixels, by the pixels, and for the pixels.”

The architectures for general pixel-level prediction problems, from low-level edge detection to mid-level surface normal estimation to high-level semantic segmentation. Convolutional predictors, such as the fully-convolutional network (FCN), have achieved remarkable success by exploiting the spatial redundancy of neighboring pixels through convolutional processing. Though computationally efficient, we point out that such approaches are not statistically efficient during learning precisely because spatial redundancy limits the information learned from neighboring pixels.

This architecture uses

(1) stratified (Layer below a Layer) sampling allows us to add diversity during batch updates

(2) Sampled multi-scale features allow us to explore more nonlinear predictors (multiple fully-connected layers followed by ReLU) that improve overall accuracy.



In PixelNet the input is an image to a convolutional neural network, and extract hyper column descriptor for a sampled pixel from multiple convolutional layers. Sparse sampling of hyper column features allows for exploration of highly nonlinear nature, which in turn significantly boosts performance.

The hyper column descriptor is then fed to a multi-layer perceptron (MLP) for the non-linear optimization, and the last layer of MLP outputs the required response for the task.

Higher convolutional layers are typically associated with larger receptive fields that capture high-level global context. Because such features may miss low-level details, numerous approaches have built predictors based on multiscale features extracted from multiple layers of a CNN.

One approach is “hyper columns” to refer to features extracted from multiple layers that correspond to the same pixel.

Let

$$hp(X) = [c1(p), c2(p), . . . , cM(p)]$$

denote the multi-scale hypercolumn feature computed for pixel p, where ci(p) denotes the feature vector of convolutional responses from layer i centered at pixel p.