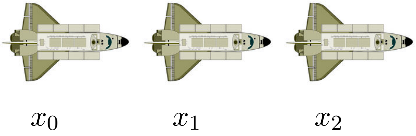
CS7015 (Deep Learning) : Lecture 11

Convolutional Neural Networks, LeNet, AlexNet, ZF-Net, VGGNet, GoogLeNet and ResNet

Mitesh M. Khapra

Department of Computer Science and Engineering
Indian Institute of Technology Madras

Module 11.1 : The convolution operation



$$s_t = \sum_{a=0}^{\infty} x_{t-a} w_{-a} = (x * w)_t$$

input

filter

convolution

- Suppose we are tracking the position of an aeroplane using a laser sensor at discrete time intervals
- Now suppose our sensor is noisy
- To obtain a less noisy estimate we would like to average several measurements
- More recent measurements are more important so we would like to take a weighted average

$$s_t = \sum_{a=0}^{6} x_{t-a} w_{-a}$$

	w_{-6}	w_{-5}	w_{-4}	w_{-3}	w_{-2}	w_{-1}	w_0					
W	0.01	0.01	0.02	0.02	0.04	0.4	0.5					
X	1.00	1.10	1.20	1.40	1.70	1.80	1.90	2.10	2.20	2.40	2.50	2.70
S							1.80					

$$s_6 = x_6 w_0 + x_5 w_{-1} + x_4 w_{-2} + x_3 w_{-3} + x_2 w_{-4} + x_1 w_{-5} + x_0 w_{-6}$$

- In practice, we would only sum over a small window
- The weight array (**w**) is known as the filter
- We just slide the filter over the input and compute the value of s_t based on a window around x_t
- Here the input (and the kernel) is one dimensional
- Can we use a convolutional operation on a 2D input also?

$$s_t = \sum_{a=0}^{6} x_{t-a} w_{-a}$$

		w_{-6}	w_{-5}	w_{-4}	w_{-3}	w_{-2}	w_{-1}	w_0				
W		0.01	0.01	0.02	0.02	0.04	0.4	0.5				
X	1.00	1.10	1.20	1.40	1.70	1.80	1.90	2.10	2.20	2.40	2.50	2.70
S							1.80	1.96				

$$s_6 = x_6 w_0 + x_5 w_{-1} + x_4 w_{-2} + x_3 w_{-3} + x_2 w_{-4} + x_1 w_{-5} + x_0 w_{-6}$$

- In practice, we would only sum over a small window
- The weight array (**w**) is known as the filter
- We just slide the filter over the input and compute the value of s_t based on a window around x_t
- Here the input (and the kernel) is one dimensional
- Can we use a convolutional operation on a 2D input also?

$$s_t = \sum_{a=0}^{6} x_{t-a} w_{-a}$$

			w_{-6}	w_{-5}	w_{-4}	w_{-3}	w_{-2}	w_{-1}	w_0			
W			0.01	0.01	0.02	0.02	0.04	0.4	0.5			
X	1.00	1.10	1.20	1.40	1.70	1.80	1.90	2.10	2.20	2.40	2.50	2.70
S							1.80	1.96	2.11			

$$s_6 = x_6 w_0 + x_5 w_{-1} + x_4 w_{-2} + x_3 w_{-3} + x_2 w_{-4} + x_1 w_{-5} + x_0 w_{-6}$$

- In practice, we would only sum over a small window
- The weight array (**w**) is known as the filter
- We just slide the filter over the input and compute the value of s_t based on a window around x_t
- Here the input (and the kernel) is one dimensional
- Can we use a convolutional operation on a 2D input also?

$$s_t = \sum_{a=0}^{6} x_{t-a} w_{-a}$$

					w_{-6}	w_{-5}	w_{-4}	w_{-3}	w_{-2}	w_{-1}	w_0	
W					0.01	0.01	0.02	0.02	0.04	0.4	0.5	
X	1.00	1.10	1.20	1.40	1.70	1.80	1.90	2.10	2.20	2.40	2.50	2.70
S							1.80	1.96	2.11	2.16	2.28	

$$s_6 = x_6w_0 + x_5w_{-1} + x_4w_{-2} + x_3w_{-3} + x_2w_{-4} + x_1w_{-5} + x_0w_{-6}$$

- In practice, we would only sum over a small window
- The weight array (**w**) is known as the filter
- We just slide the filter over the input and compute the value of s_t based on a window around x_t
- Here the input (and the kernel) is one dimensional
- Can we use a convolutional operation on a 2D input also?

$$s_t = \sum_{a=0}^{6} x_{t-a} w_{-a}$$

						w_{-6}	w_{-5}	w_{-4}	w_{-3}	w_{-2}	w_{-1}	w_0
W						0.01	0.01	0.02	0.02	0.04	0.4	0.5
X	1.00	1.10	1.20	1.40	1.70	1.80	1.90	2.10	2.20	2.40	2.50	2.70
S							1.80	1.96	2.11	2.16	2.28	2.42

$$s_6 = x_6 w_0 + x_5 w_{-1} + x_4 w_{-2} + x_3 w_{-3} + x_2 w_{-4} + x_1 w_{-5} + x_0 w_{-6}$$

- In practice, we would only sum over a small window
- The weight array ($\mathbf{w}$) is known as the filter
- We just slide the filter over the input and compute the value of s_t based on a window around x_t
- Here the input (and the kernel) is one dimensional
- Can we use a convolutional operation on a 2D input also?

$$S_{ij} = (I * K)_{ij} = \sum_{a=0}^{m-1} \sum_{b=0}^{n-1} I_{i-a,j-b} K_{a,b} I_{i+a,j+b} K_{a,b}$$

- We can think of images as 2D inputs
- We would now like to use a 2D filter $(m \times n)$
- First let us see what the 2D formula looks like
- This formula looks at all the preceding neighbours $(i-a, j-b)$
- In practice, we use the following formula which looks at the succeeding neighbours

Input

a	b	c	d
e	f	g	h
i	j	k	ℓ

Kernel

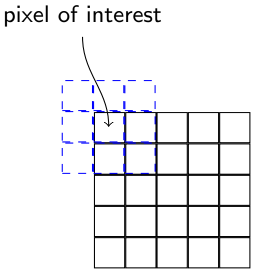
w	x
y	z

Output

aw+bx+ey+fz	bw+cx+fy+gz	cw+dx+gy+hz
ew+fx+iy+jz	fw+gx+jy+kz	gw+hx+ky+ℓz

- Let us apply this idea to a toy example and see the results

$$S_{ij} = (I * K)_{ij} = \sum_{a=\lfloor -\frac{m}{2} \rfloor}^{\lfloor \frac{m}{2} \rfloor} \sum_{b=\lfloor -\frac{n}{2} \rfloor}^{\lfloor \frac{n}{2} \rfloor} I_{i-a,j-b} K_{\frac{m}{2}+a,\frac{n}{2}+b}$$



- For the rest of the discussion we will use the following formula for convolution
- In other words we will assume that the kernel is centered on the pixel of interest
- So we will be looking at both preceeding and succeeding neighbors

Let us see some examples of 2D convolutions applied to images

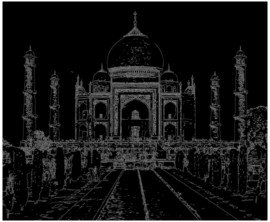
$$* \begin{matrix} 1 & 1 & 1 \\ 1 & 1 & 1 \\ 1 & 1 & 1 \end{matrix} =$$

blurs the image

$$* \begin{array}{ccc} 0 & -1 & 0 \\ -1 & 5 & -1 \\ 0 & -1 & 0 \end{array} =$$

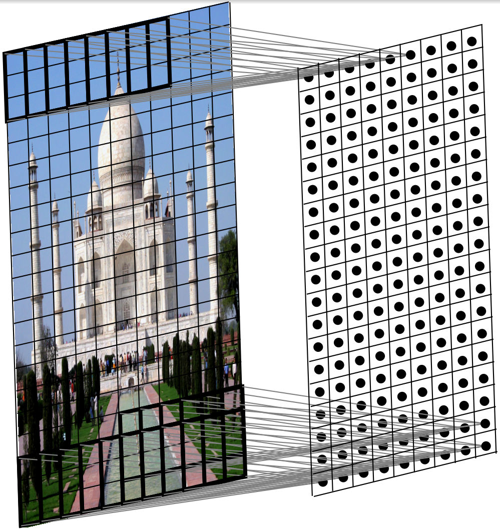
sharpens the image

$$* \begin{matrix} 1 & 1 & 1 \\ 1 & -8 & 1 \\ 1 & 1 & 1 \end{matrix} =$$



detects the edges

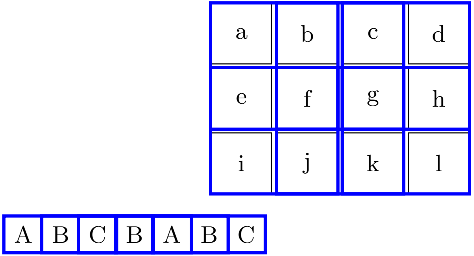
We will now see a working example of 2D convolution.

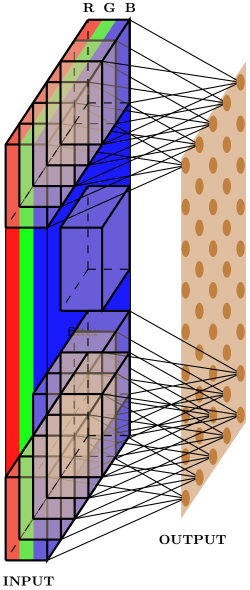


- We just slide the kernel over the input image
- Each time we slide the kernel we get one value in the output
- The resulting output is called a feature map.
- We can use multiple filters to get multiple feature maps.

Question

- In the 1D case, we slide a one dimensional filter over a one dimensional input
- In the 2D case, we slide a two dimensional filter over a two dimensional output
- What would happen in the 3D case?



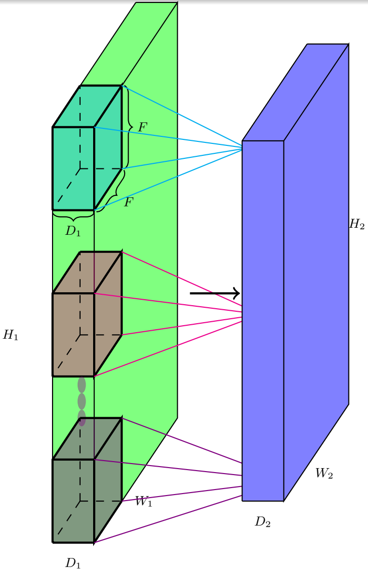


- What would a 3D filter look like?
- It will be 3D and we will refer to it as a volume
- Once again we will slide the volume over the 3D input and compute the convolution operation
- Note that in this lecture we will assume that the filter always extends to the depth of the image
- In effect, we are doing a 2D convolution operation on a 3D input (because the filter moves along the height and the width but not along the depth)
- As a result the output will be 2D (only width and height, no depth)
- Once again we can apply multiple filters to get multiple feature maps

Module 11.2 : Relation between input size, output size and filter size

- So far we have not said anything explicit about the dimensions of the
    1. inputs
    2. filters
    3. outputs

  and the relations between them
- We will see how they are related but before that we will define a few quantities



- We first define the following quantities
- Width (W_1), Height (H_1) and Depth (D_1) of the original input
- The Stride S (We will come back to this later)
- The number of filters K
- The spatial extent (F) of each filter (the depth of each filter is same as the depth of each input)
- The output is $W_2 \times H_2 \times D_2$ (we will soon see a formula for computing W_2, H_2 and D_2)

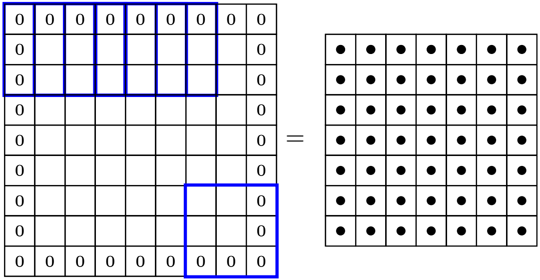
- Let us compute the dimension (W_2, H_2) of the output
- Notice that we can't place the kernel at the corners as it will cross the input boundary
- This is true for all the shaded points (the kernel crosses the input boundary)
- This results in an output which is of smaller dimensions than the input

In general, $W_2 = W_1 - F + 1$

$$H_2 = H_1 - F + 1$$

We will refine this formula further

- Let us compute the dimension (W_2, H_2) of the output
- Notice that we can't place the kernel at the corners as it will cross the input boundary
- This is true for all the shaded points (the kernel crosses the input boundary)
- This results in an output which is of smaller dimensions than the input
- As the size of the kernel increases, this becomes true for even more pixels
- For example, let's consider a 5 × 5 kernel
- We have an even smaller output now



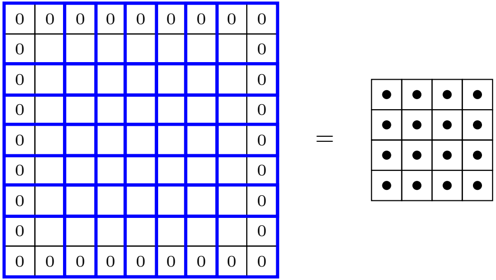
- What if we want the output to be of same size as the input?
- We can use something known as padding
- Pad the inputs with appropriate number of 0 inputs so that you can now apply the kernel at the corners
- Let us use pad P = 1 with a 3 × 3 kernel
- This means we will add one row and one column of 0 inputs at the top, bottom, left and right

$$\textit{We now have,}$$

$$W_2 = W_1 - F + 2P + 1$$

$$H_2 = H_1 - F + 2P + 1$$

We will refine this formula further

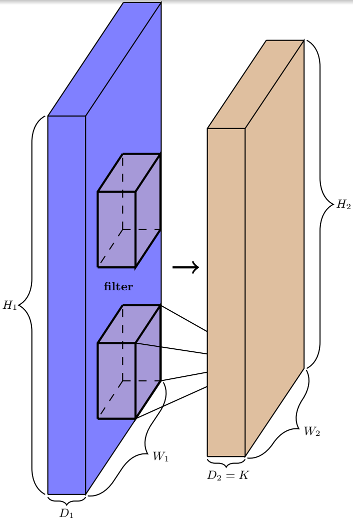


- What does the stride S do?
- It defines the intervals at which the filter is applied (here $S = 2$)
- Here, we are essentially skipping every 2nd pixel which will again result in an output which is of smaller dimensions

So what should our final formula look like,

$$W_2 = \frac{W_1 - F + 2P}{S} + 1$$

$$H_2 = \frac{H_1 - F + 2P}{S} + 1$$



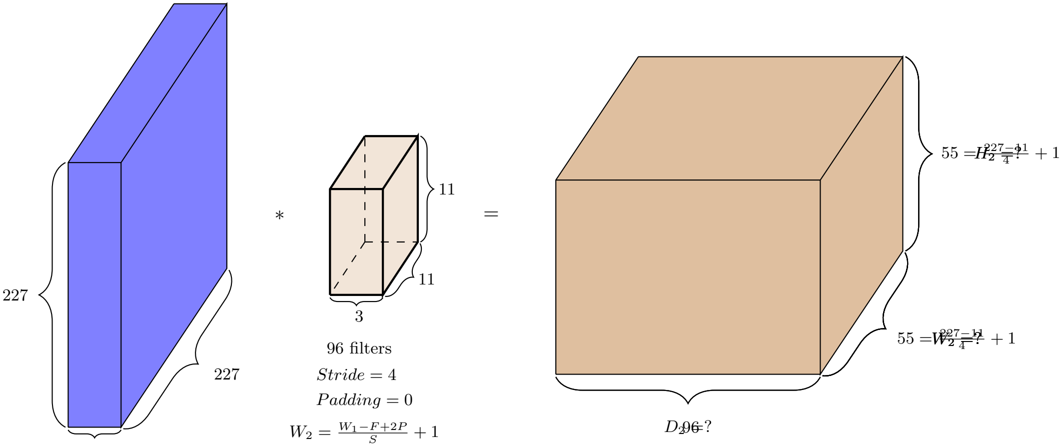
$$W_2 = \frac{W_1 - F + 2P}{S} + 1$$

$$H_2 = \frac{H_1 - F + 2P}{S} + 1$$

$$D_2 = K$$

- Finally, coming to the depth of the output.
- Each filter gives us one 2D output.
- K filters will give us K such 2D outputs
- We can think of the resulting output as $K \times W_2 \times H_2$ volume
- Thus $D_2 = K$

Let us do a few exercises



96 filters

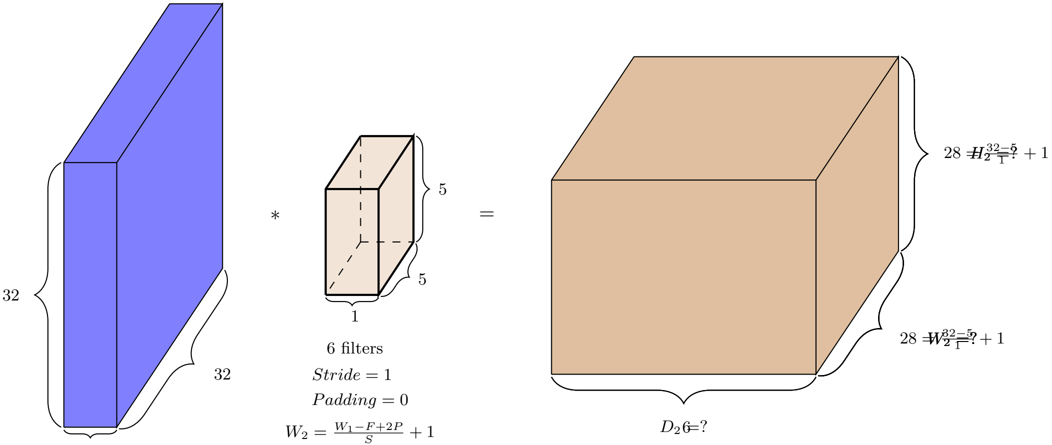
$Stride = 4$

$Padding = 0$

$$W_2 = \frac{W_1 - F + 2P}{S} + 1$$

$$H_2 = \frac{H_1 - F + 2P}{S} + 1$$

Let us do a few exercises



6 filters

$Stride = 1$

$Padding = 0$

$$W_2 = \frac{W_1 - F + 2P}{S} + 1$$

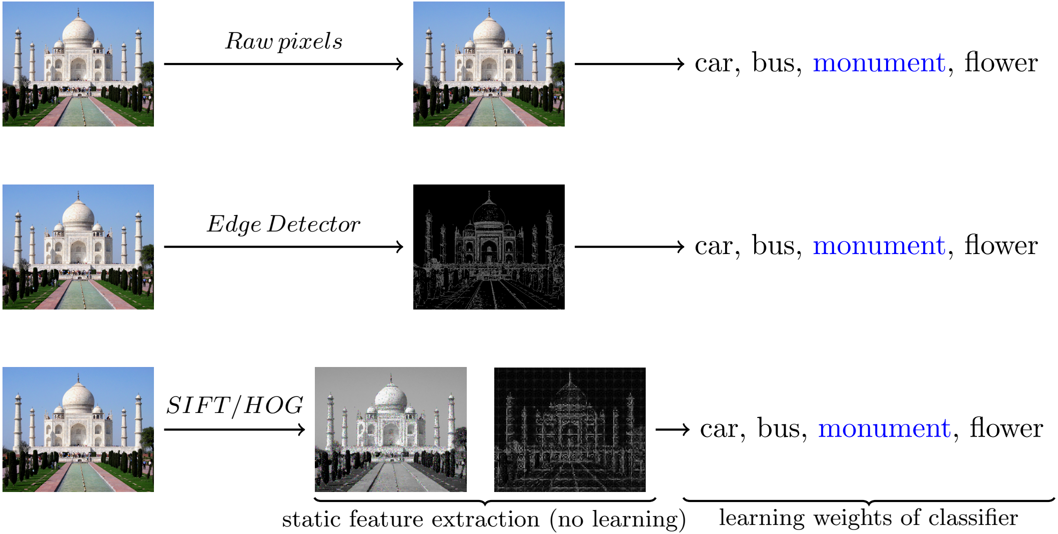
$$H_2 = \frac{H_1 - F + 2P}{S} + 1$$

Module 11.3 : Convolutional Neural Networks

Putting things into perspective

- What is the connection between this operation (convolution) and neural networks?
- We will try to understand this by considering the task of “image classification”

Features



Raw pixels → car, bus, monument, flower

Edge Detector → car, bus, monument, flower

SIFT/HOG → car, bus, monument, flower

static feature extraction (no learning)

learning weights of classifier

Input

Features

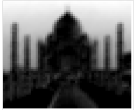
Classifier

$\longrightarrow$

$\longrightarrow$ car, bus, monument, flower

$$\begin{matrix} 0 & 0 & 0 & 0 & 0 \\ 0 & 1 & 1 & 1 & 0 \\ 0 & 1 & -8 & 1 & 0 \\ 0 & 1 & 1 & 1 & 0 \\ 0 & 0 & 0 & 0 & 0 \end{matrix}$$

$\longrightarrow$



$\longrightarrow$ car, bus, monument, flower

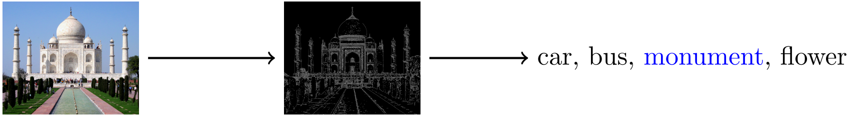
$$\begin{matrix} \text{-1.21358689e-03} & \text{3.23652686e-03} & \cdots & \cdots & \text{-2.06615720e-02} \\ \text{-1.52757822e-03} & \text{2.36130832e-03} & \cdots & \cdots & \text{-1.19824838e-02} \\ \vdots & \vdots & & & \vdots \\ \vdots & \vdots & & & \vdots \\ \text{-8.25322699e-04} & \text{-5.14897937e-03} & \cdots & \cdots & \text{-9.90395527e-03} \end{matrix} \longleftarrow \text{Learn these weights}$$

- Instead of using handcrafted kernels such as edge detectors **can we learn meaningful kernels/filters in addition to learning the weights of the classifier?**

Input

Features

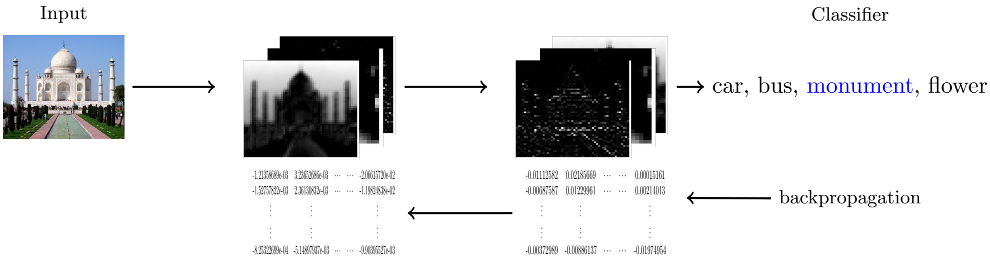
Classifier



$$\begin{matrix} 0 & 0 & 0 & 0 & 0 \\ 0 & 1 & 1 & 1 & 0 \\ 0 & 1 & -8 & 1 & 0 \\ 0 & 1 & 1 & 1 & 0 \\ 0 & 0 & 0 & 0 & 0 \end{matrix}$$

$$\begin{matrix} \text{[illegible]} & \text{[illegible]} & \cdots & \cdots & \text{[illegible]} \\ \text{[illegible]} & \text{[illegible]} & \cdots & \text{[illegible]} & \text{[illegible]} \\ \vdots & \vdots & & & \vdots \\ \vdots & \vdots & & & \vdots \\ \text{[illegible]} & \text{[illegible]} & \cdots & \cdots & \text{[illegible]} \end{matrix}$$

- **Even better:** Instead of using handcrafted kernels (such as edge detectors)**can we learn multiple meaningful kernels/filters in addition to learning the weights of the classifier?**

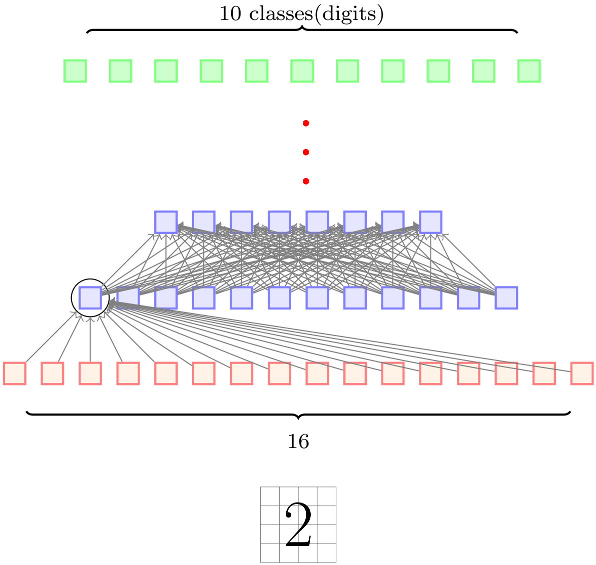


$$\begin{matrix} -1.21358689e\text{-}03 & 3.23652686e\text{-}03 & \cdots & \cdots & -2.06615720e\text{-}02 \\ -1.52757822e\text{-}03 & 2.36130832e\text{-}03 & \cdots & \cdots & -1.19824838e\text{-}02 \\ \vdots & \vdots & & & \vdots \\ \vdots & \vdots & & & \vdots \\ -8.25322699e\text{-}04 & -5.14897937e\text{-}03 & \cdots & \cdots & -9.90395527e\text{-}03 \end{matrix}$$

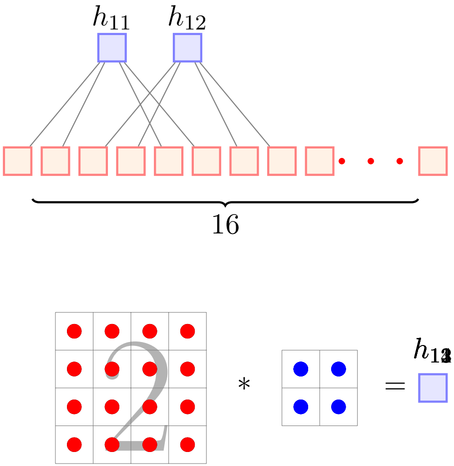
$$\begin{matrix} -0.01112582 & 0.02185669 & \cdots & \cdots & 0.00015161 \\ -0.00687587 & 0.01229961 & \cdots & \cdots & 0.00214013 \\ \vdots & \vdots & & & \vdots \\ \vdots & \vdots & & & \vdots \\ -0.00372989 & -0.00886137 & \cdots & \cdots & -0.01974954 \end{matrix}$$

- **Can we learn multiple layers of meaningful kernels/filters in addition to learning the weights of the classifier?**
- Yes, we can !
- Simply by treating these kernels as parameters and learning them in addition to the weights of the classifier (using back propagation)
- Such a network is called a Convolutional Neural Network.

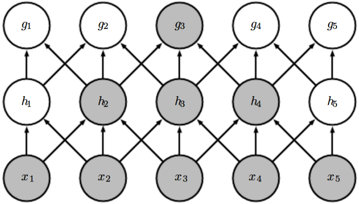
- Okay, I get it that the idea is to learn the kernel/filters by just treating them as parameters of the classification model
- But how is this different from a regular feedforward neural network
- Let us see



- This is what a regular feed-forward neural network will look like
- There are many dense connections here
- For example all the 16 input neurons are contributing to the computation of h_{11}
- Contrast this to what happens in the case of convolution



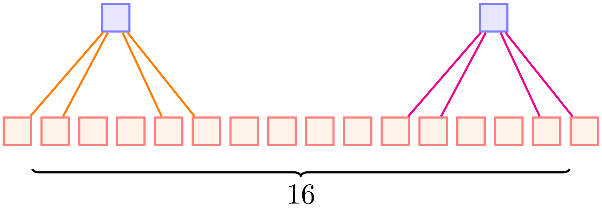
- Only a few local neurons participate in the computation of h_{11}
- For example, only pixels 1, 2, 5, 6 contribute to h_{11}
- The connections are much sparser
- We are taking advantage of the structure of the image(interactions between neighboring pixels are more interesting)
- This **sparse connectivity** reduces the number of parameters in the model

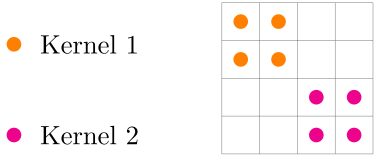


- But is sparse connectivity really good thing ?
- Aren't we losing information (by losing interactions between some input pixels)
- Well, not really
- The two highlighted neurons (x_1 & x_5)* do not interact in *layer* 1
- But they indirectly contribute to the computation of g_3 and hence interact indirectly

---

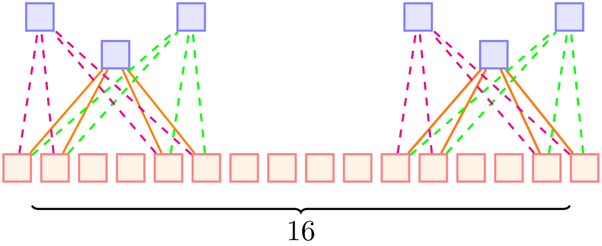
* Goodfellow-et-al-2016





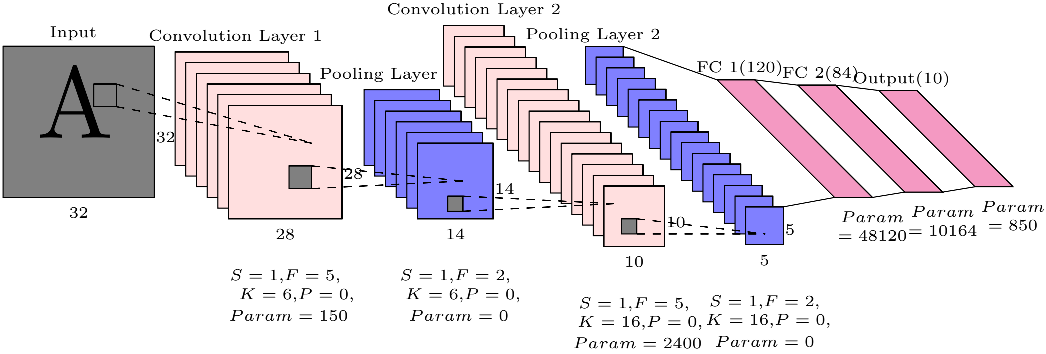
4x4 Image

- Another characteristic of CNNs is **weight sharing**
- Consider the following network
- Do we want the kernel weights to be different for different portions of the image?
- Imagine that we are trying to learn a kernel that detects edges
- Shouldn't we be applying the same kernel at all the portions of the image?

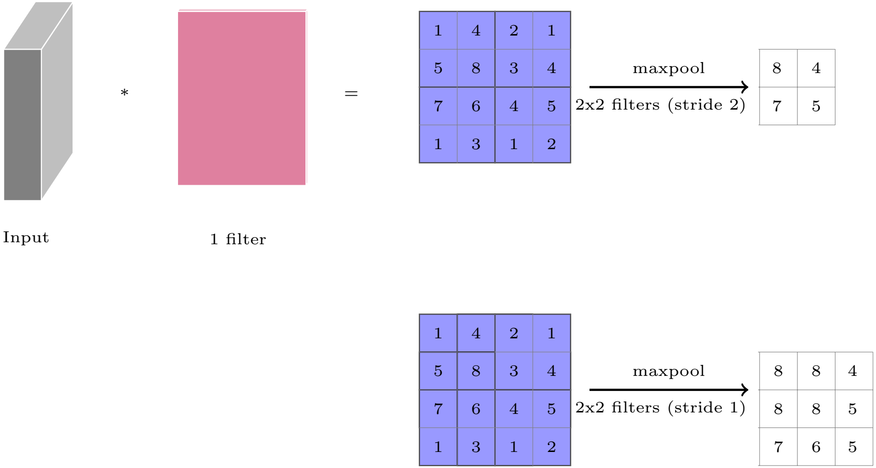


- In other words shouldn't the *orange* and *pink* kernels be the same
- Yes, indeed
- This would make the job of learning easier(instead of trying to learn the same weights/kernels at different locations again and again)
- But does that mean we can have only one kernel?
- No, we can have many such kernels but the kernels will be shared by all locations in the image
- This is called "weight sharing"

- So far, we have focused only on the convolution operation
- Let us see what a full convolutional neural network looks like



- It has alternate convolution and pooling layers
- What does a pooling layer do?
- Let us see



1	4	2	1
5	8	3	4
7	6	4	5
1	3	1	2

8	4
7	5

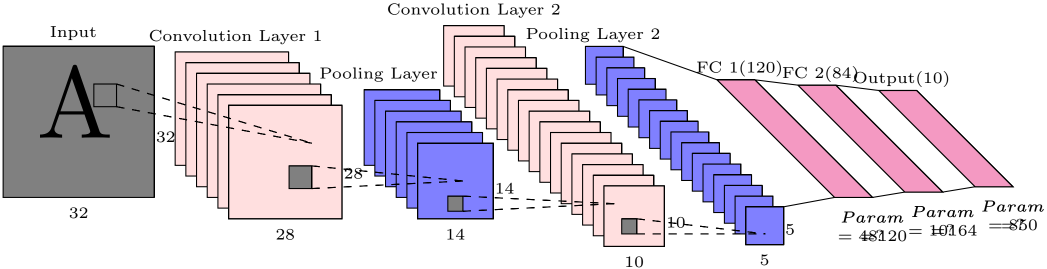
1	4	2	1
5	8	3	4
7	6	4	5
1	3	1	2

8	8	4
8	8	5
7	6	5

- Instead of max pooling we can also do average pooling

We will now see some case studies where convolution neural networks have been successful

LeNet-5 for handwritten character recognition



$S = 1, F = 5,$
$K = 6, P = 0,$
$Param = 150$

$S = 1, F = 2,$
$K = 6, P = 0,$
$Param = 0$

$S = 1, F = 5,$
$K = 16, P = 0,$
$Param = 2400$

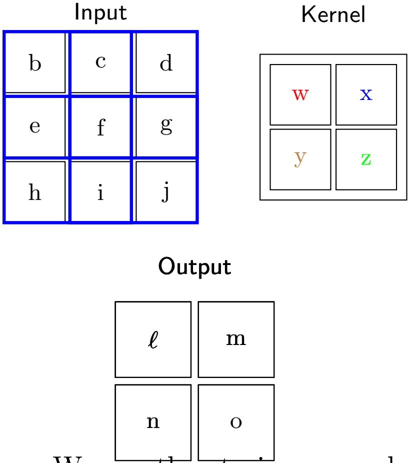
$S = 1, F = 2,$
$K = 16, P = 0,$
$Param = 0$

$Param = 48120$

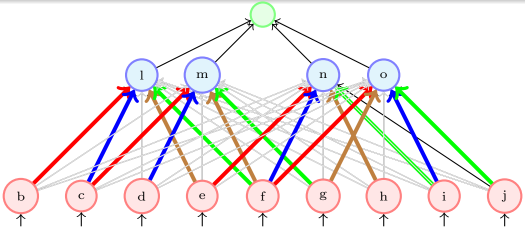
$Param = 10164$

$Param = 850$

- How do we train a convolutional neural network ?



- We can thus train a convolution neural network using backpropagation by thinking of it as a feedforward neural network with sparse connections

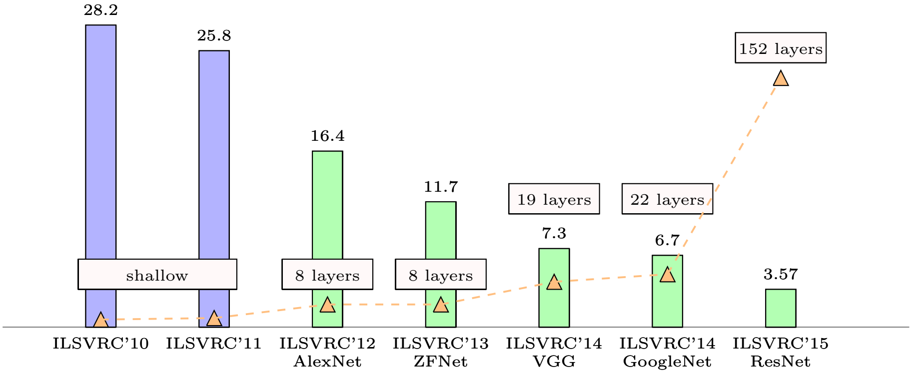


- A CNN can be implemented as a feedforward neural network
- wherein only a few weights(in color) are active
- the rest of the weights (in gray) are zero

Module 11.4 : CNNs (success stories on ImageNet)

ImageNet Success Stories(roadmap for rest of the talk)

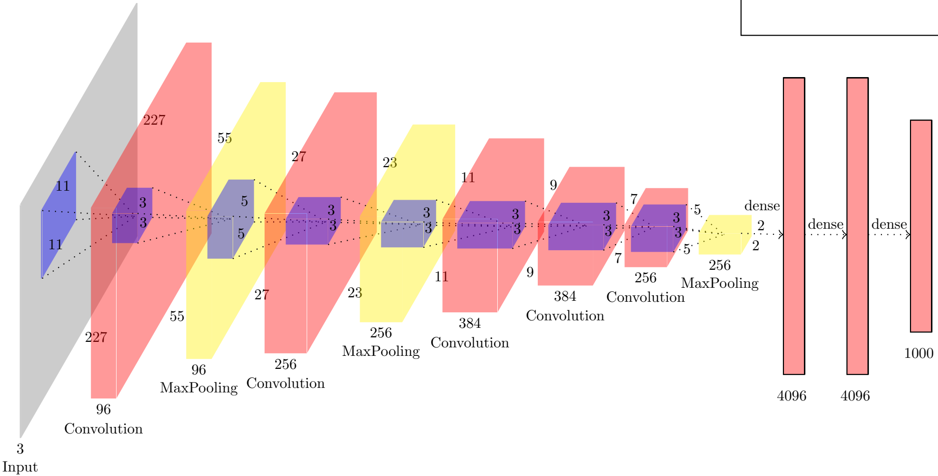
- AlexNet
- ZFNet
- VGGNet


28.2
25.8
16.4
11.7
7.3
6.7
3.57
shallow
8 layers
8 layers
19 layers
22 layers
152 layers
ILSVRC'10
ILSVRC'11
ILSVRC'12 AlexNet
ILSVRC'13 ZFNet
ILSVRC'14 VGG
ILSVRC'14 GoogleNet
ILSVRC'15 ResNet

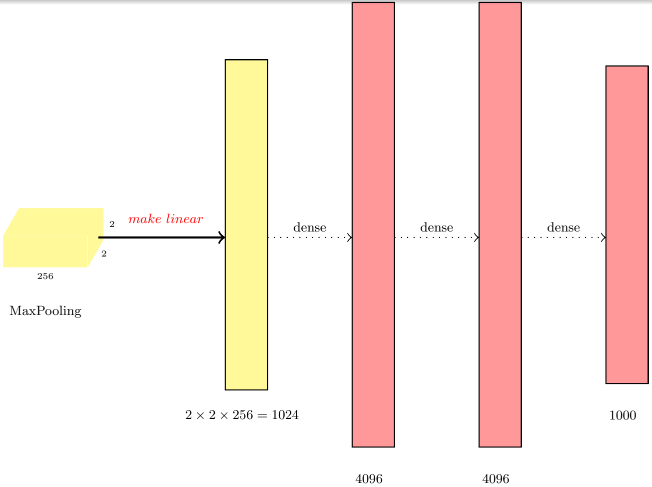
ImageNet Success Stories(roadmap for rest of the talk)

- AlexNet
- ZFNet
- VGGNet

Total Parameters: $27.55M$

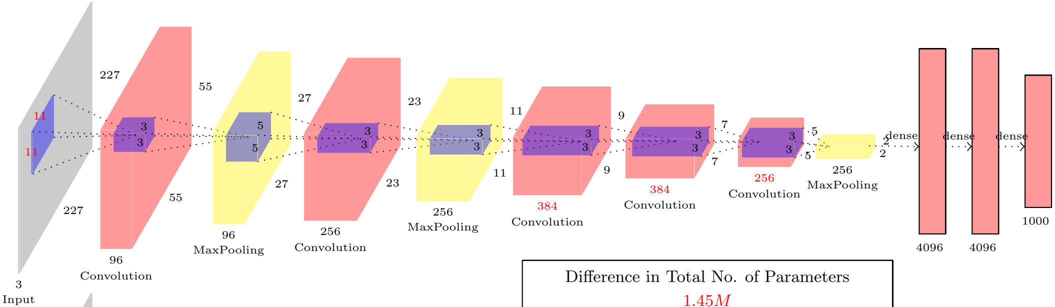


- Let us look at the connections in the fully connected layers in more detail
- We will first stretch out the last conv or maxpool layer to make it a 1d vector
- This 1d vector is then densely connected to other layers just as in a regular feedforward neural network

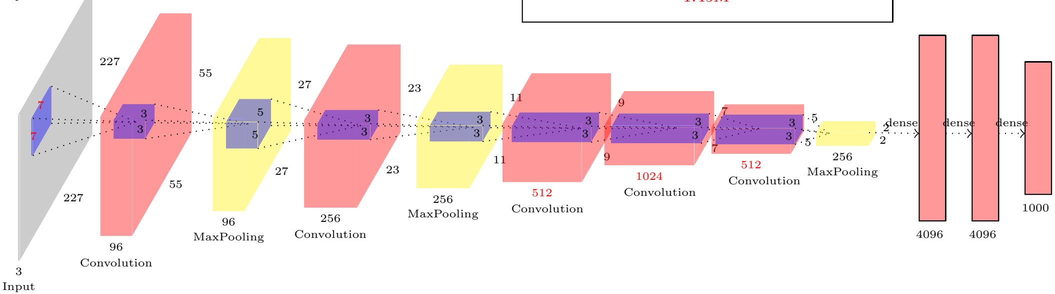


ImageNet Success Stories(roadmap for rest of the talk)

- AlexNet
- ZFNet
- VGGNet

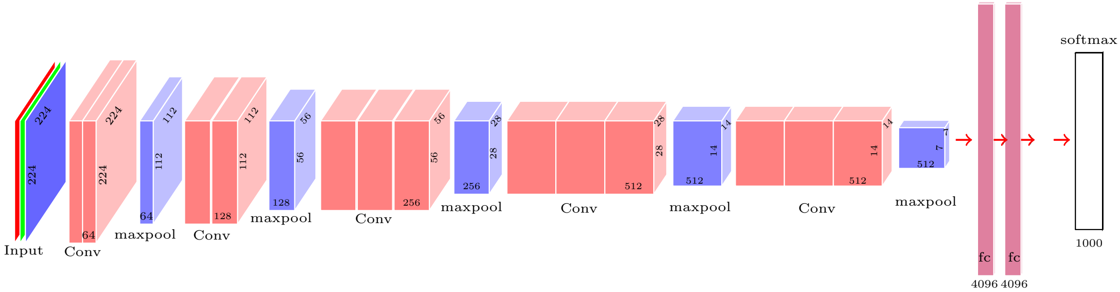


Difference in Total No. of Parameters
1.45M



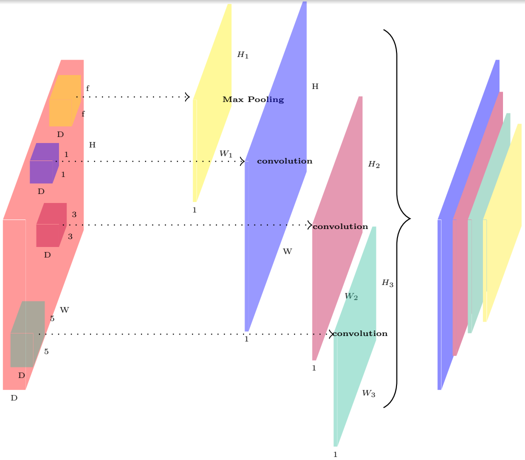
ImageNet Success Stories(roadmap for rest of the talk)

- AlexNet
- ZFNet
- VGGNet

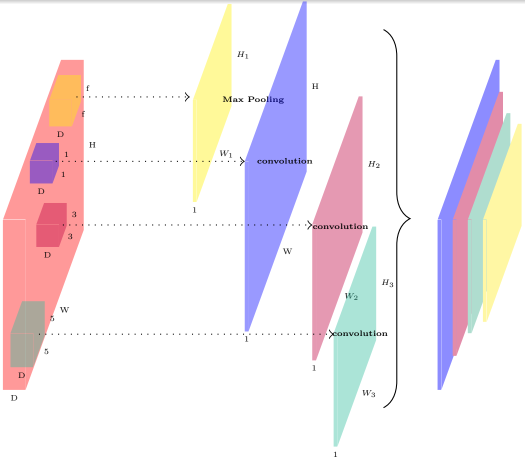


- Kernel size is 3 × 3 throughout
- Total parameters in non FC layers $= \sim 16M$
- Total Parameters in FC layers $= (512 \times 7 \times 7 \times 4096) + (4096 \times 4096) + (4096 \times 1024) = \sim 122M$
- Most parameters are in the first FC layer ($\sim$ 102M)

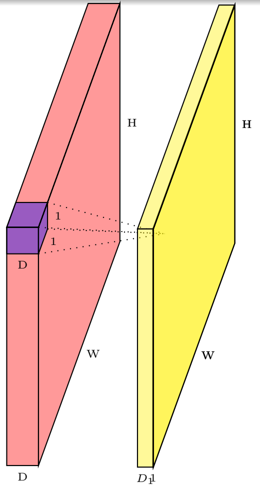
Module 11.5 : Image Classification continued (GoogLeNet and ResNet)



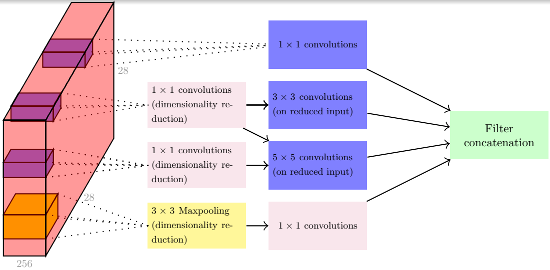
- Consider the output at a certain layer of a convolutional neural network
- After this layer we could apply a max-pooling layer
- Or a 1 × 1 convolution
- Or a 3 × 3 convolution
- Or a 5 × 5 convolution
- **Question:** Why choose between these options (convolution, maxpooling, filter sizes)?
- **Idea:** Why not apply all of them at the same time and then concatenate the feature maps?



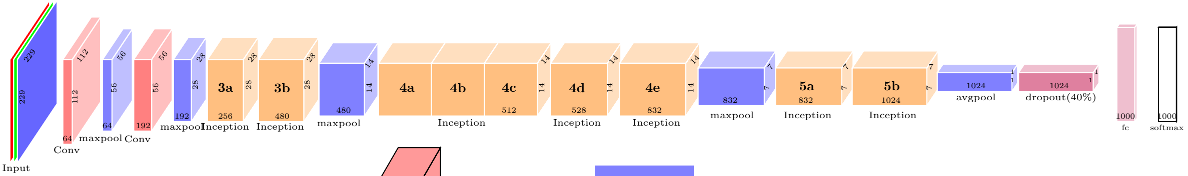
- Well this naive idea could result in a large number of computations
- If $P = 0$ & $S = 1$ then convolving a $W \times H \times D$ input with a $F \times F \times D$ filter results in a $(W - F + 1)(H - F + 1)$ sized output
- Each element of the output requires $O(F \times F \times D)$ computations
- Can we reduce the number of computations?

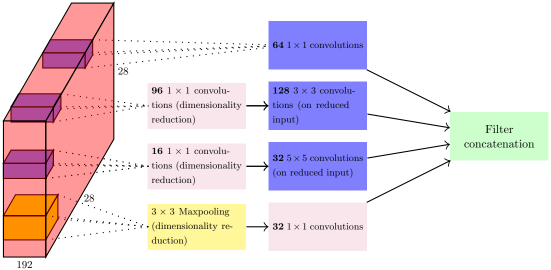


- Yes, by using 1×1 convolutions
- Huh?? What does a 1×1 convolution do ?
- It aggregates along the depth
- So convolving a $D \times W \times H$ input with D_1 1×1 ($D_1 < D$) filters will result in a $D_1 \times W \times H$ output ($S = 1, P = 0$)
- If $D_1 < D$ then this effectively reduces the dimension of the input and hence the computations
- Specifically instead of O($F \times F \times D$) we will need O($F \times F \times D_1$) computations
- We could then apply subsequent 3×3, 5×5 filter on this reduced output



- But we might want to use different dimensionality reductions before the 3×3 and 5×5 filters
- So we can use D_1 and D_2 1×1 filters before the 3×3 and 5×5 filters respectively
- We can then add the maxpooling layer followed by dimensionality reduction
- And a new set of 1×1 convolutions
- And finally we concatenate all these layers
- This is called the **Inception module**
- We will now see **GoogLeNet** which contains many such inception modules





64 1 × 1 convolutions

96 1 × 1 convolutions (dimensionality reduction)

128 3 × 3 convolutions (on reduced input)

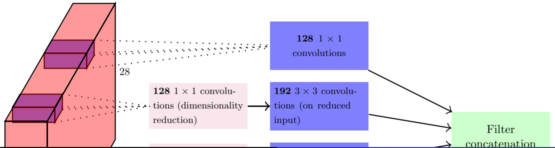
16 1 × 1 convolutions (dimensionality reduction)

32 5 × 5 convolutions (on reduced input)

3 × 3 Maxpooling (dimensionality reduction)

32 1 × 1 convolutions

Filter concatenation

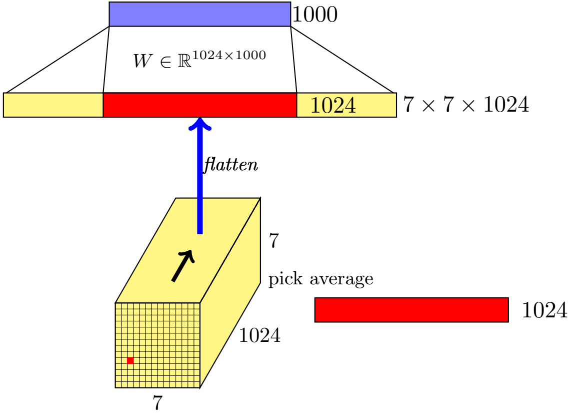


128 1 × 1 convolutions

128 1 × 1 convolutions (dimensionality reduction)

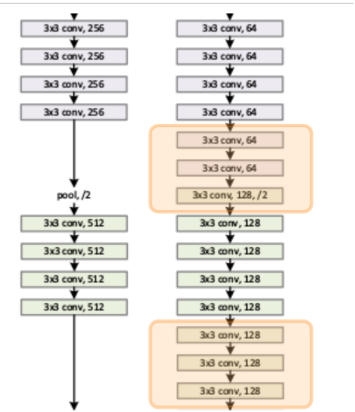
192 3 × 3 convolutions (on reduced input)

Filter concatenation

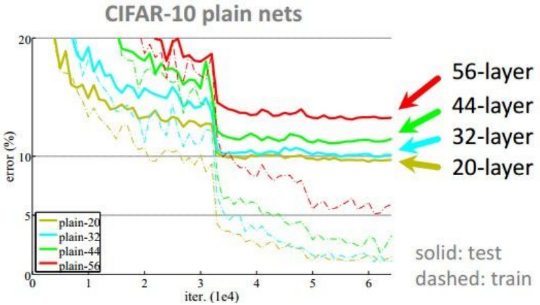


- **Important Trick:** Got rid of the fully connected layer
- Notice that output of the last layer is $7 \times 7 \times 1024$ dimensional
- What if we were to add a fully connected layer with 1000 nodes (for 1000 classes) on top of this
- We would have $7 \times 7 \times 1024 \times 1000 = 49M\ parameters$
- Instead they use an average pooling of size 7×7 on each of the 1024 feature maps
- This results in a 1024 dimensional output
- Significantly reduces the number of parameters

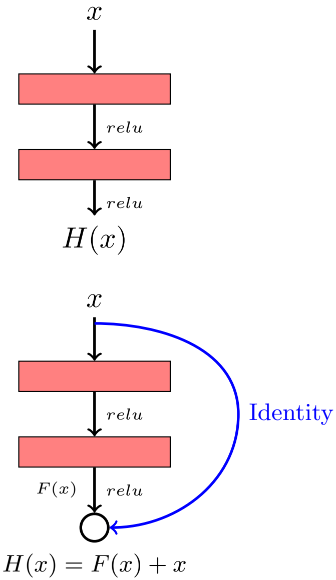
- GoogLeNet
- ResNet



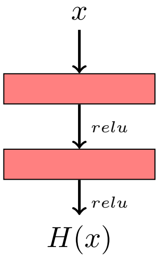
- Suppose we have been able to train a shallow neural network well
- Now suppose we construct a deeper network which has few more layers (in orange)
- Intuitively, if the shallow network works well then the deep network should also work well by simply learning to compute identity functions in the new layers
- Essentially, the solution space of a shallow neural network is a subset of the solution space of a deep neural network

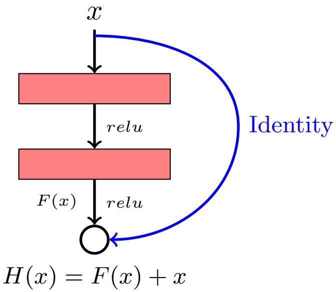


- But in practice it is observed that this doesn't happen
- Notice that the deep layers have a higher error rate on the test set



- Consider any two stacked layers in a CNN
- The two layers are essentially learning some function of the input
- What if we enable it to learn only a residual function of the input?





- Why would this help?
- Remember our argument that a deeper version of a shallow network would do just fine by learning identity transformations in the new layers
- This identity connection from the input allows a ResNet to retain a copy of the input
- Using this idea they were able to train really deep networks

ResNet, 152 layers

1^{st} place in all five main tracks

- **ImageNet Classification:** "Ultra-deep" 152-layer nets
- **ImageNet Detection:** 16% better than the 2nd best system
- **ImageNet Localization:** 27% better than the 2nd best system
- **COCO Detection:** 11% better than the 2nd best system
- **COCO Segmentation:** 12% better than the 2nd best system

ResNet, 152 layers

Bag of tricks

- Batch Normalizaton after every CONV layer
- Xavier/2 initialization from [He et al]
- SGD + Momentum(0.9)
- Learning rate:0.1, divided by 10 when validation error plateaus
- Mini-batch size 256
- Weight decay of 1e-5
- No dropout used