CSE 673
COMPUTATIONAL VISION

VENU GOVINDARAJU
DEEN DAYAL MOHAN

University at Buffalo
Department of Computer Science and Engineering
School of Engineering and Applied Sciences

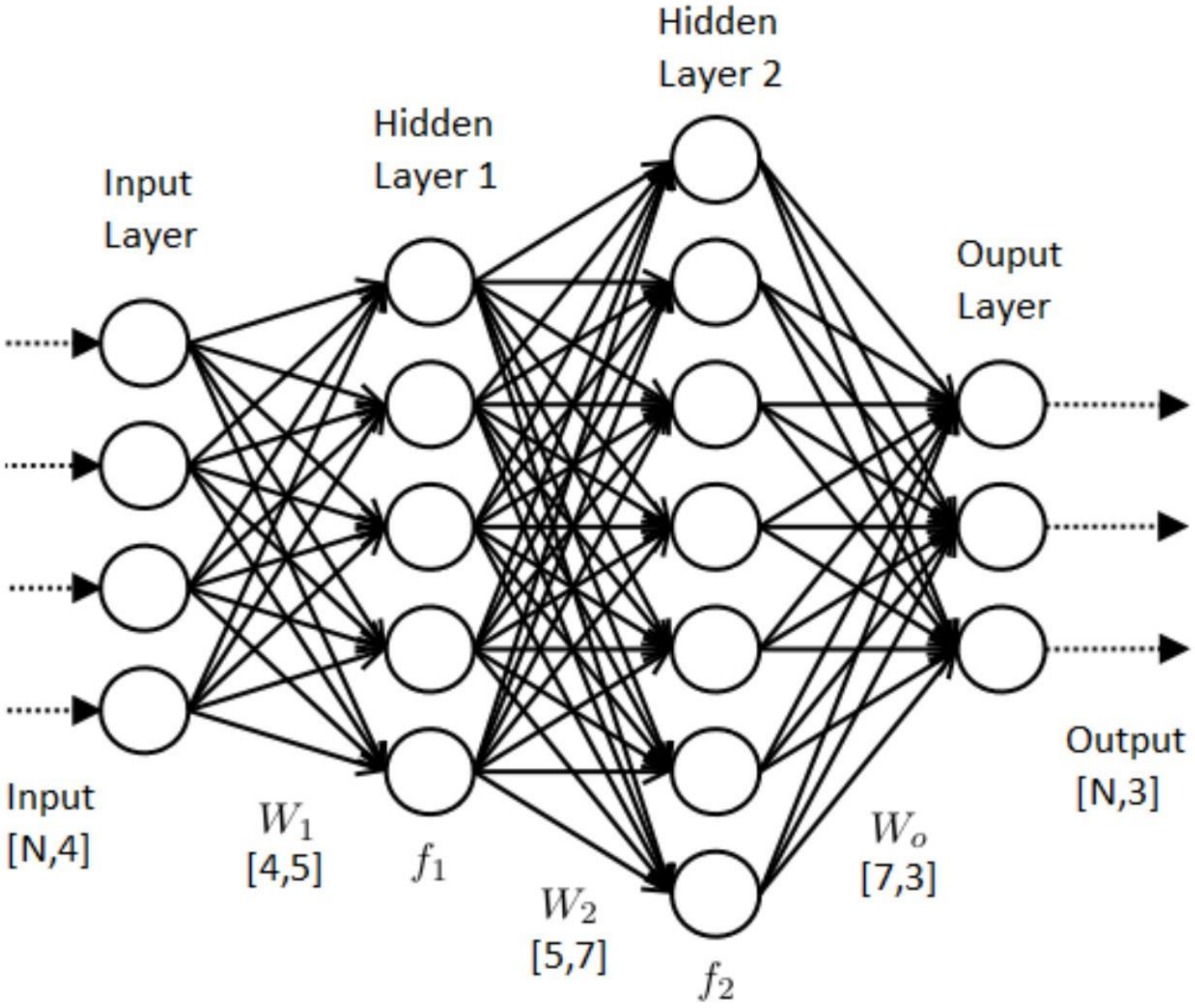
Covid-19 Guidelines

- Effective Aug. 3, the University at Buffalo will require all students, employees and visitors – regardless of their vaccination status – to wear face coverings while inside campus buildings. This includes classrooms, hallways, libraries and other common spaces, as well as UB buses and shuttles.

- Students are expected to wear mask in class during lectures (unless you have a UB approved exception)

- Public Health Behavior Expectations https://www.buffalo.edu/studentlife/who-we-are/departments/conduct/coronavirus-student-compliance-policy.html

Optimizers


Input Layer
Hidden Layer 1
Hidden Layer 2
Ouput Layer
Input
[N,4]
W_1
[4,5]
f_1
W_2
[5,7]
f_2
W_o
[7,3]
Output
[N,3]

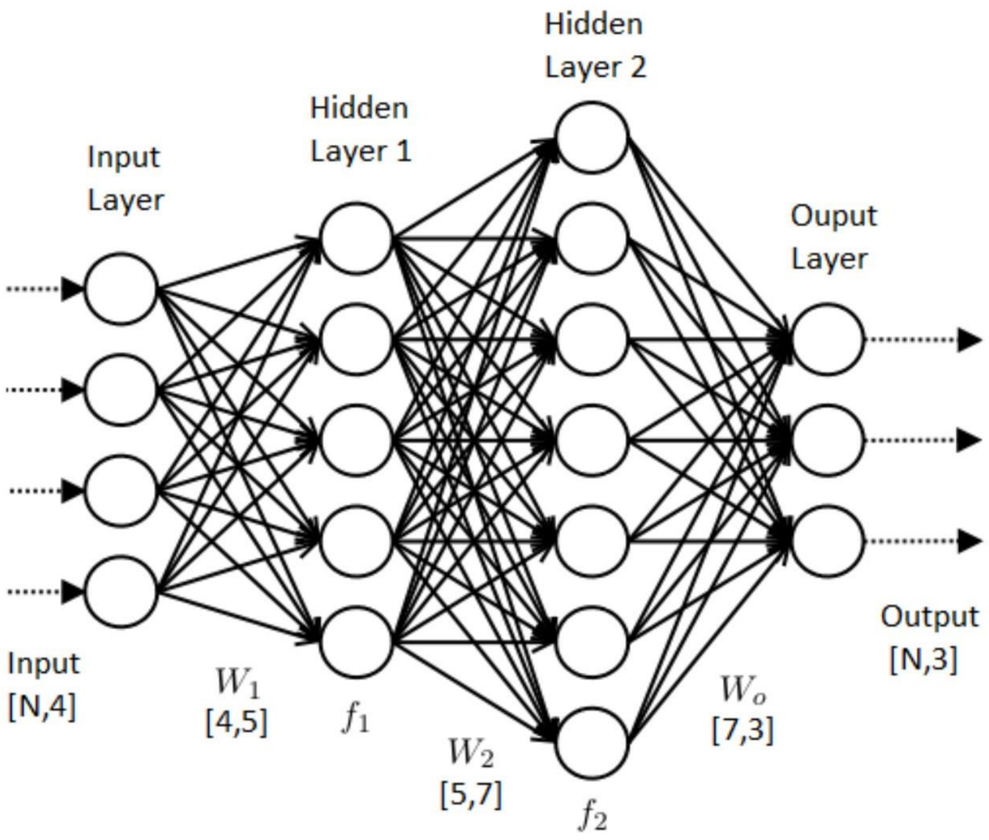
Optimizers

- $w_{new} = w_{Old} - \eta \frac{\partial J}{\partial w_{Old}}$

- This method of updating is called stochastic gradient descent

- In practice a mini batch of samples are passed through , the gradient is computed and then the parameters are updated based on this gradient

- Also referred as mini batch gradient descent



Optimizers

- $w_{new} = w_{Old} - \eta \frac{\partial J}{\partial w_{Old}}$

```
## mini batch stochastic Gradient Descent

for each in epoch:
    for batch in batchs:
        gradient_of_parameters = backward_through_graph(loss_function,batch,parameters)
        parameters = parameters - learning_rate * gradient_of_parameters
```

- The learning rate of all the parameters are the same

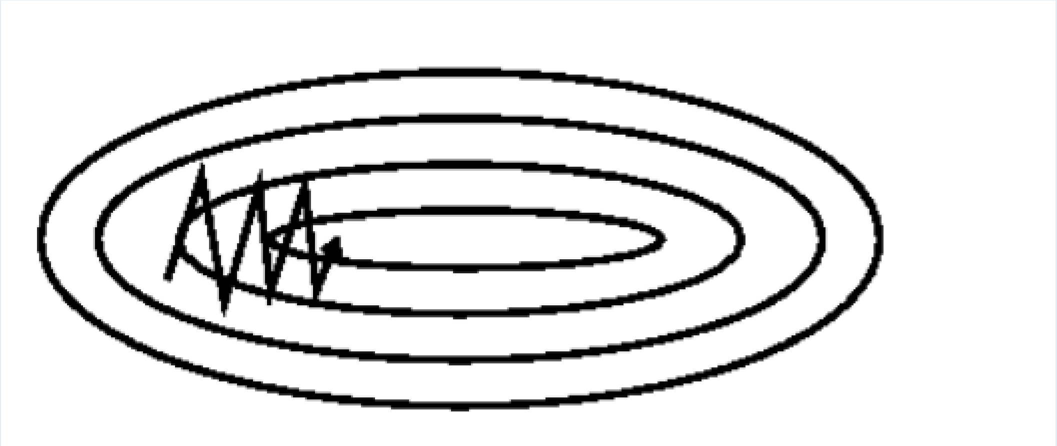
Momentum

- Vanilla SGD algorithm tend to suffer when the loss landscape is elongated in one direction

- SGD tends to jitter a lot when going towards the minimum

- Such landscapes are generally present near local minimum

- This is why we rarely use Vanilla SGD for training a Neural Network



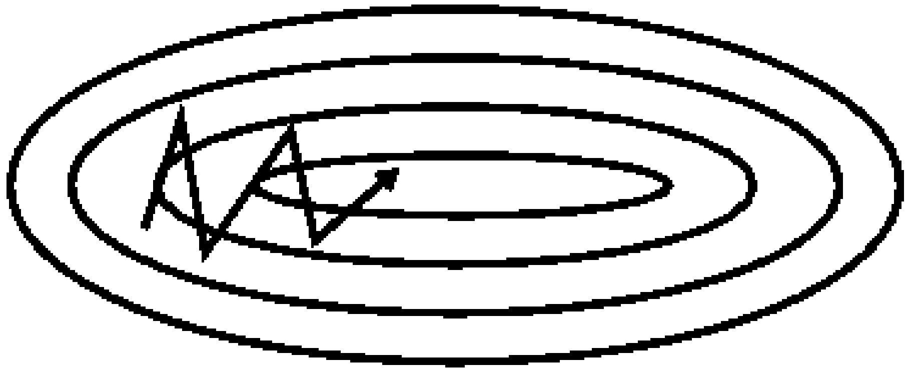
Momentum

- In order to avoid this we introduce momentum into the gradient update

- The idea of momentum is accumulate the power of gradient direction which does not change and dampen the direction which changes a lot

$$v_t = \gamma v_{t-1} + \eta \nabla_\theta J(\theta)$$
$$\theta = \theta - v_t$$

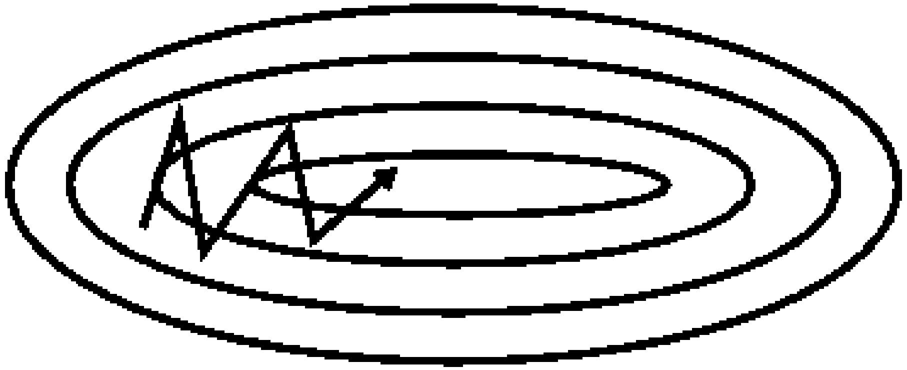
- Here 'v' the used as a term to accumulate the gradient across time steps and this term is used to update the parameters



Momentum

- γ is the momentum term, set between 0-1. (typically 0.9)

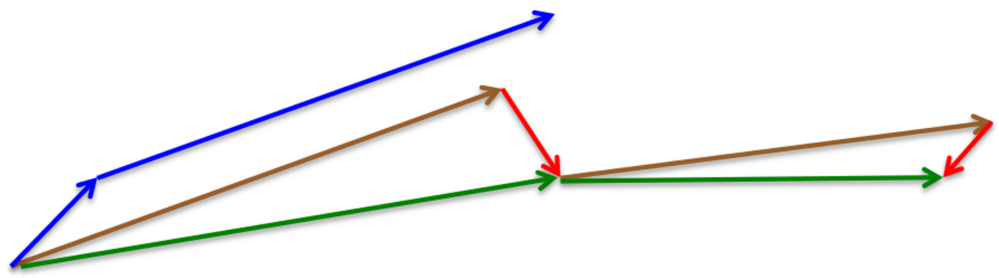
$$v_t = \gamma v_{t-1} + \eta \nabla_\theta J(\theta)$$
$$\theta = \theta - v_t$$



```
for each in epoch:
    for batch in batchs:
        gradient_of_parameters = backward_through_graph(loss_function,batch,parameters)
        v = lamda*v + learning_rate * gradient_of_parameters
        parameters = parameters - v
```

Momentum

- An improvement to the vanilla momentum is to use Nesterov Momentum.



brown vector = jump, red vector = correction, green vector = accumulated gradient

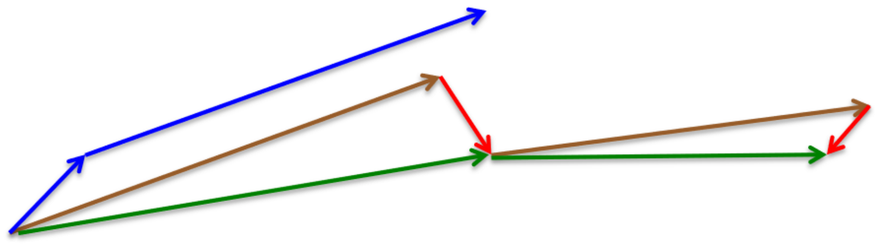
blue vectors = standard momentum

- The idea of Nesterov Momentum is to compute the gradient at the approximate next position in the loss landscape

Momentum

- Mathematically it can be written as

$$v_{t+1} = \mu v_t - \eta \nabla l(\theta + \mu v_t)$$
$$\theta_{t+1} = \theta_t + v_{t+1}$$



brown vector = jump, red vector = correction, green vector = accumulated gradient

blue vectors = standard momentum

- Here μ is the momentum parameter.

- The equation to simplified using some math tricks. (Look at reference 1 if you are interested)

- Mostly used as the first choice algorithm when using momentum

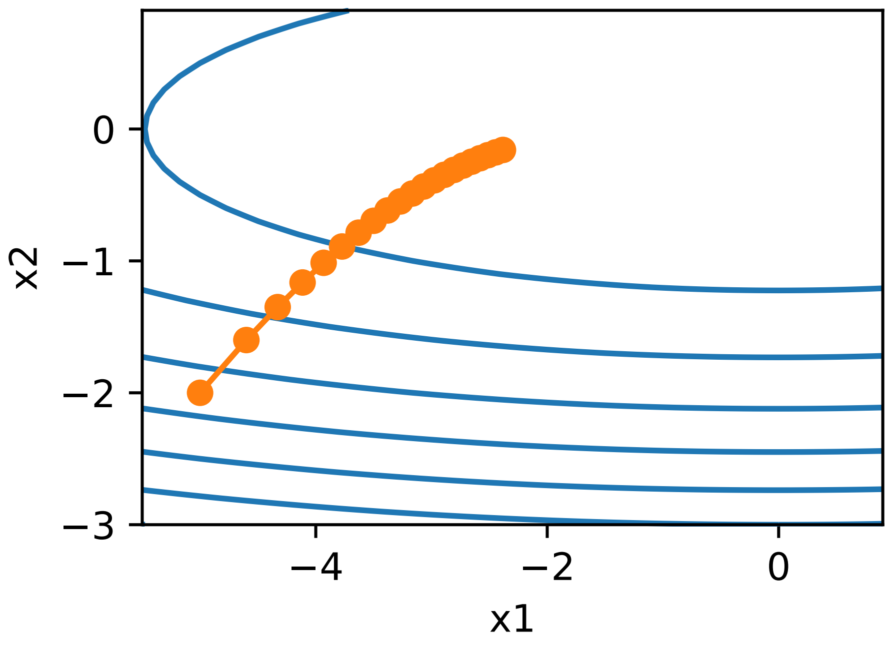
AdaGrad

- All the optimizers that we saw so far has a common learning rate for all the parameters

- Adaptive optimizers are able to have per-parameter learning rate. This is helpful as some parameters might need faster or slower update than the other

- The way this is achieved is by accumulating the square of the gradients and dividing the current gradient by the square root of the accumulated term

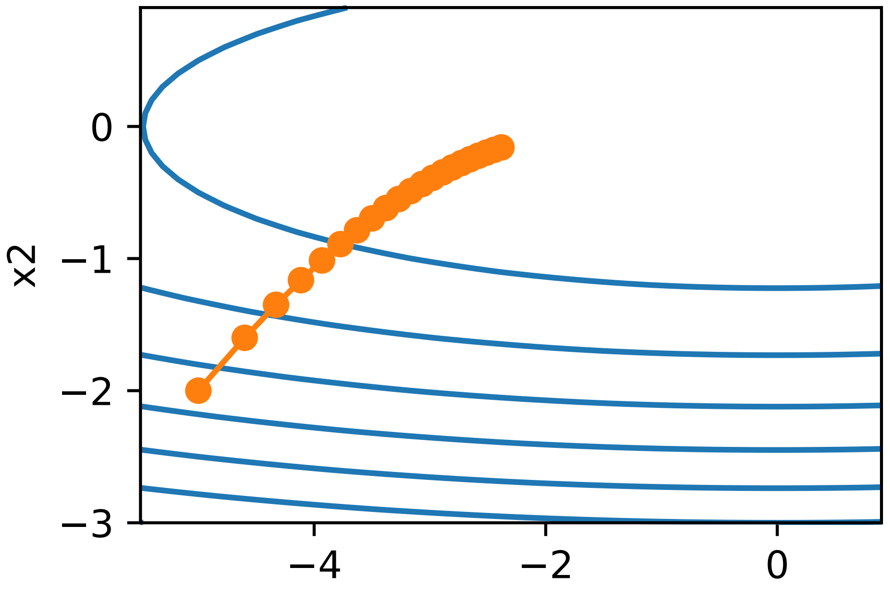
- Convergence is much faster than SGD based methods



AdaGrad

- Mathematically it is written as

$$\theta_{t+1,i} = \theta_{t,i} - \frac{\eta}{\sqrt{G_{t,ii} + \epsilon}} \cdot g_{t,i}.$$




```
for each in epoch:
    for batch in batchs:
        gradient_of_parameters = backward_through_graph(loss_function,batch,parameters)
        G = G + (gradient_of_parameters)**2
        parameters = parameters - learning_rate * gradient_of_parameters /(sqrt(G+1e-10))
```

RmsProp

- What is the disadvantage of AdaGrad?

- RmsProp was developed by Hinton in order to fix the issue with AdaGrad.

$$E[g^2]_t = 0.9E[g^2]_{t-1} + 0.1g_t^2$$

$$\theta_{t+1} = \theta_t - \frac{\eta}{\sqrt{E[g^2]_t + \epsilon}} g_t$$

```
for each in epoch:
    for batch in batchs:
        gradient_of_parameters = backward_through_graph(loss_function,batch,parameters)
        G =  G*lamda + (1-lamda)*(gradient_of_parameters)**2
        parameters = parameters - learning_rate * gradient_of_parameters /(sqrt(G+1e-10))
```

Adam

- So can we combine best of both worlds ?

- Combine the momentum based updates of the gradients and also scale the update down by second order moments to give a per-parameter learning rate

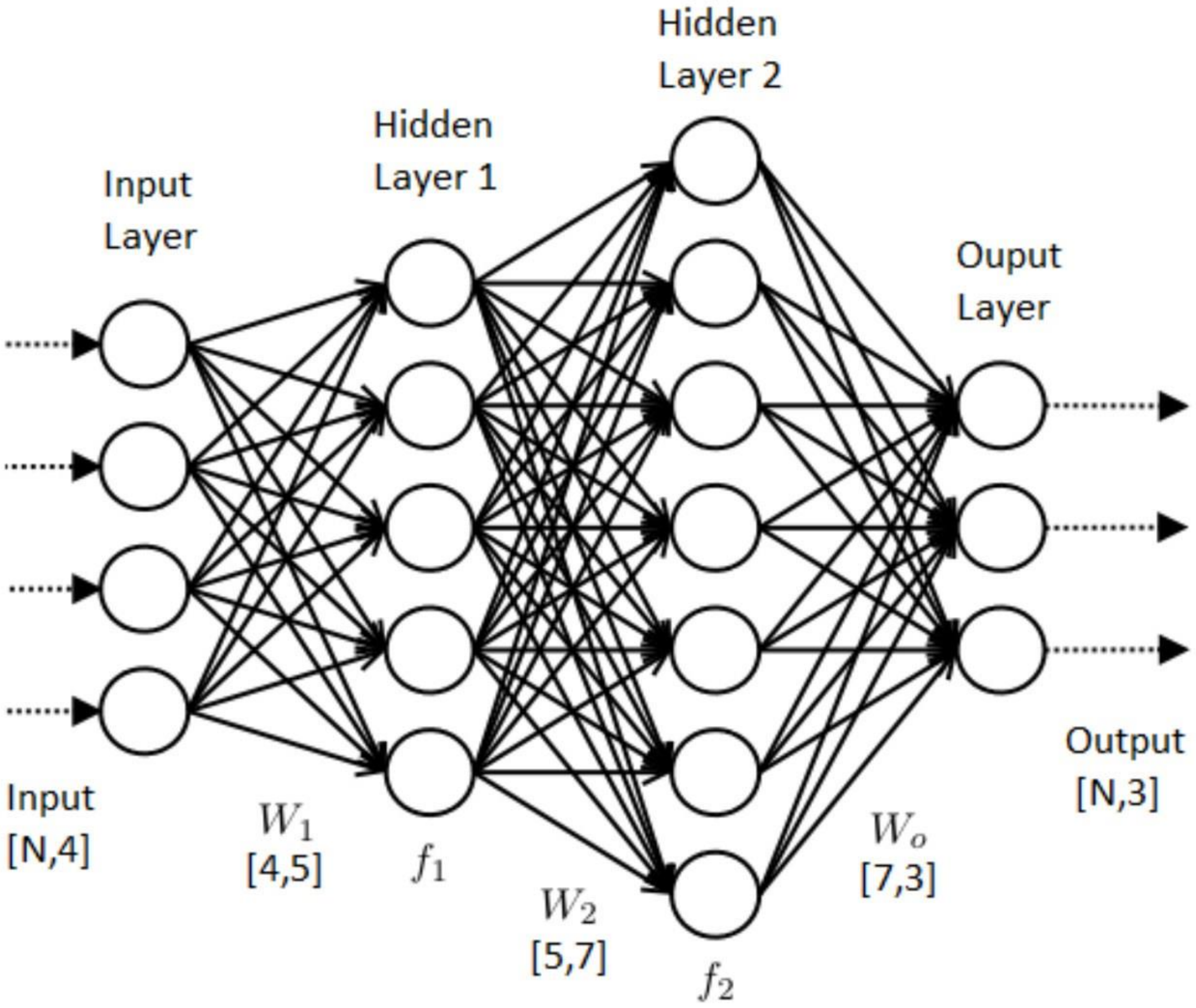
$$m_t = \beta_1 m_{t-1} + (1 - \beta_1) g_t$$
$$v_t = \beta_2 v_{t-1} + (1 - \beta_2) g_t^2$$

$$\theta_{t+1} = \theta_t - \frac{\eta}{\sqrt{\hat{v}_t} + \epsilon} \hat{m}_t.$$

```
for each in epoch:
    for batch in batchs:
        gradient_of_parameters = backward_through_graph(loss_function,batch,parameters)
        M = Beta1 * M + (1-Beta1)* gradient_of_parameters
        G = Beta2 * G + (1-Beta2)*(gradient_of_parameters)**2
        parameters = parameters - learning_rate * M /(sqrt(G+1e-10))
```

- There is a Bias correction step missing. Figure it out as HW

Activation Functions

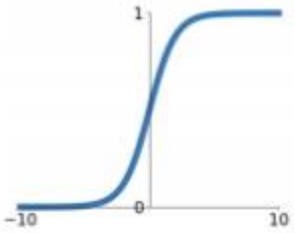


Artificial Neural Networks

- Today there are many activations

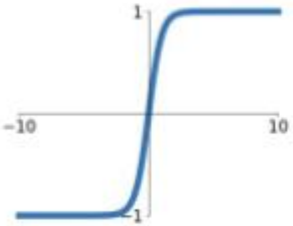
Sigmoid

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



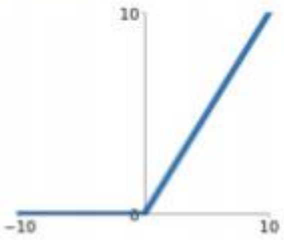
tanh

$\tanh(x)$



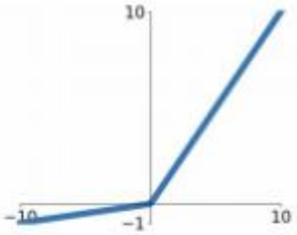
ReLU

$\max(0, x)$



Leaky ReLU

$\max(0.1x, x)$

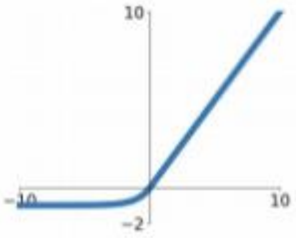


Maxout

$\max(w_1^T x + b_1, w_2^T x + b_2)$

ELU

$$\begin{cases} x & x \geq 0 \\ \alpha(e^x - 1) & x < 0 \end{cases}$$



Batch Normalization

- The data passed on to the input layer of the network is normalized
- In order to improve the performance, we like to maintain the normalization across layers
- But this is not guaranteed when using different layers
- In order to address this problem Batch Normalization is introduced
-

$$\widehat{x}^{(k)} = \frac{x^{(k)} - \mathrm{E}[x^{(k)}]}{\sqrt{\mathrm{Var}[x^{(k)}]}}$$

Batch Normalization

- The final algorithm for the batch norm layer would look like this .

Input: Values of x over a mini-batch: $\mathcal{B} = \{x_{1\ldots m}\}$;
Parameters to be learned: γ, β

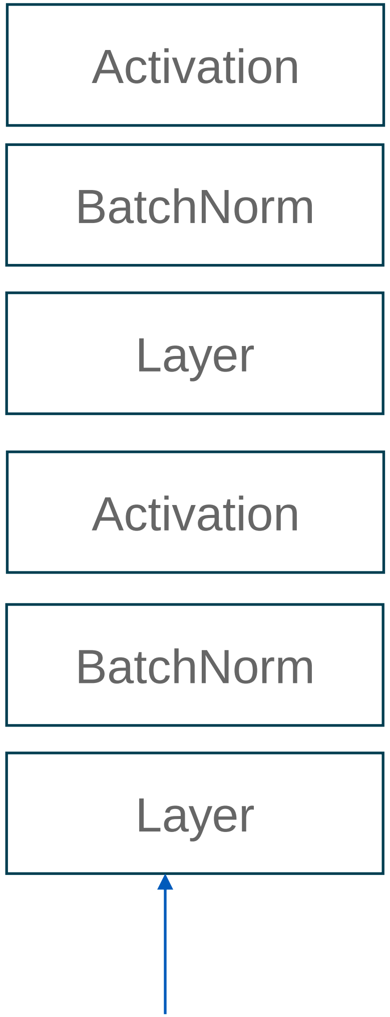
Output: $\{y_i = \mathrm{BN}_{\gamma,\beta}(x_i)\}$

$$\mu_{\mathcal{B}} \leftarrow \frac{1}{m}\sum_{i=1}^{m} x_i \qquad \text{// mini-batch mean}$$

$$\sigma_{\mathcal{B}}^2 \leftarrow \frac{1}{m}\sum_{i=1}^{m} (x_i - \mu_{\mathcal{B}})^2 \qquad \text{// mini-batch variance}$$

$$\widehat{x}_i \leftarrow \frac{x_i - \mu_{\mathcal{B}}}{\sqrt{\sigma_{\mathcal{B}}^2 + \epsilon}} \qquad \text{// normalize}$$

$$y_i \leftarrow \gamma \widehat{x}_i + \beta \equiv \mathrm{BN}_{\gamma,\beta}(x_i) \qquad \text{// scale and shift}$$



Batch Normalization

- Has the ability to adjust the parameters to help the training process.
- The idea of using Batch Norm often is said to reduce the internal covariance shift
- What happens at test time?
- If interested check out NormProp.

Input: Values of x over a mini-batch: $\mathcal{B} = \{x_{1\ldots m}\}$;
Parameters to be learned: γ, β

Output: $\{y_i = \mathrm{BN}_{\gamma,\beta}(x_i)\}$

$$\mu_{\mathcal{B}} \leftarrow \frac{1}{m}\sum_{i=1}^{m} x_i \qquad \text{// mini-batch mean}$$

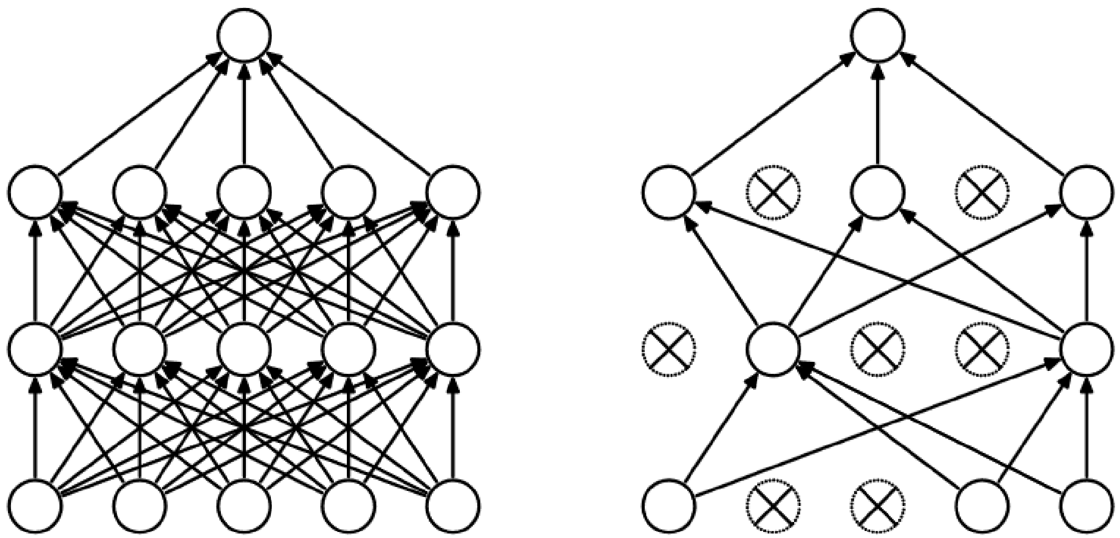
$$\sigma_{\mathcal{B}}^2 \leftarrow \frac{1}{m}\sum_{i=1}^{m} (x_i - \mu_{\mathcal{B}})^2 \qquad \text{// mini-batch variance}$$

$$\widehat{x}_i \leftarrow \frac{x_i - \mu_{\mathcal{B}}}{\sqrt{\sigma_{\mathcal{B}}^2 + \epsilon}} \qquad \text{// normalize}$$

$$y_i \leftarrow \gamma \widehat{x}_i + \beta \equiv \mathrm{BN}_{\gamma,\beta}(x_i) \qquad \text{// scale and shift}$$

Dropout

- The process of removing activations from a percentage of neurons
- Random neurons selected from the output of a layer and the value is set to 0.
- The neuron which were selected should be maintained to adjust in the backprop
- Why does this work?
- Not relying on just one neuron. Also ensemble intuition



(a) Standard Neural Net (b) After applying dropout.

Agenda

- Convolutional Neural Networks
- Convolution Arithmetic
- Backpropagation in CNN
- Pooling

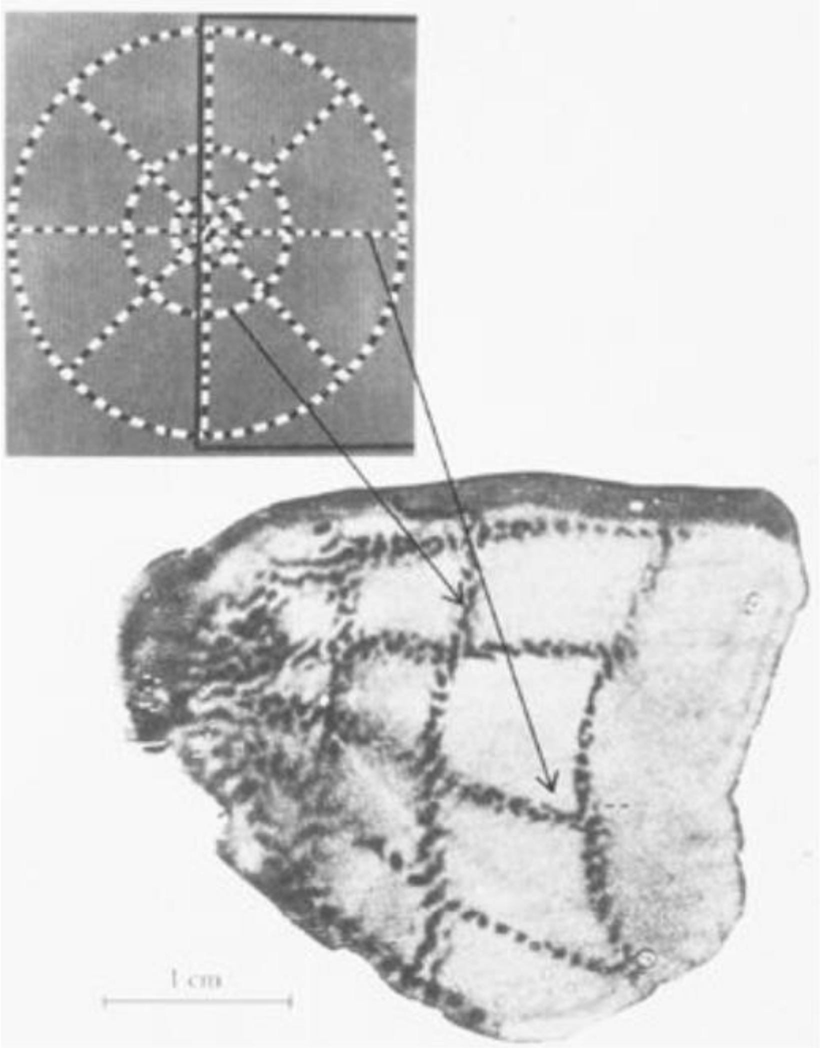
Convolutional Neural Network

- Inspired from visual cortex in the eye. Local cells in the cortex process close by information in the visual field

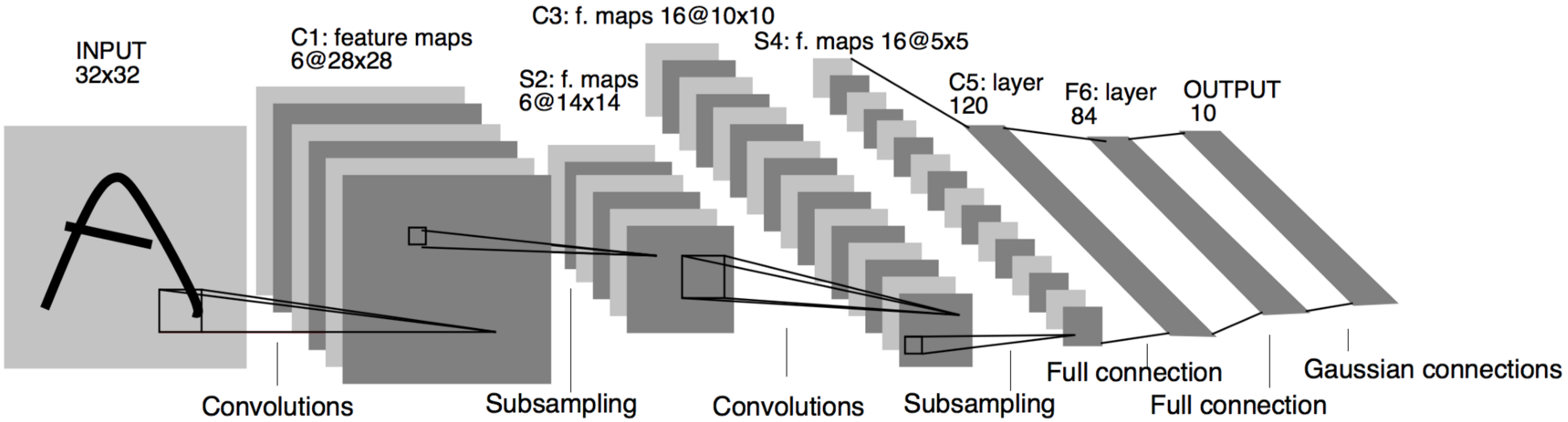
- Information processing has a hierarchical structure, in which the information is processed using a combination of simple and complex neurons

- Yann LeCun proposed the LeNet architecture in 1989, which used convolutional operations to capture the locality of information

- The network was trained using the Backpropagation algorithm

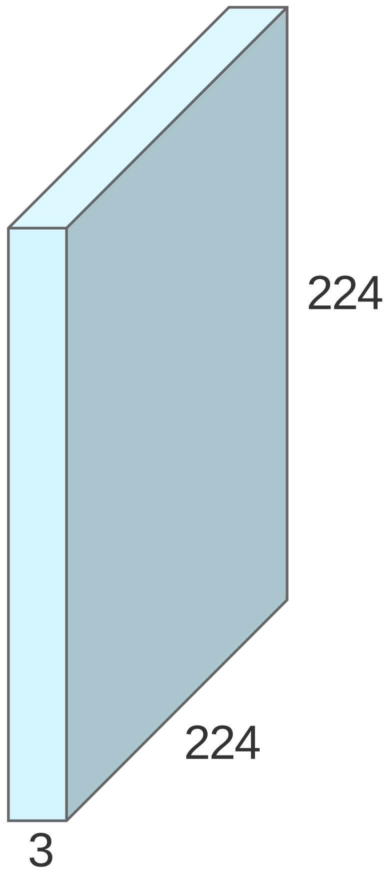


Convolutional Neural Network



Convolutional Neural Network

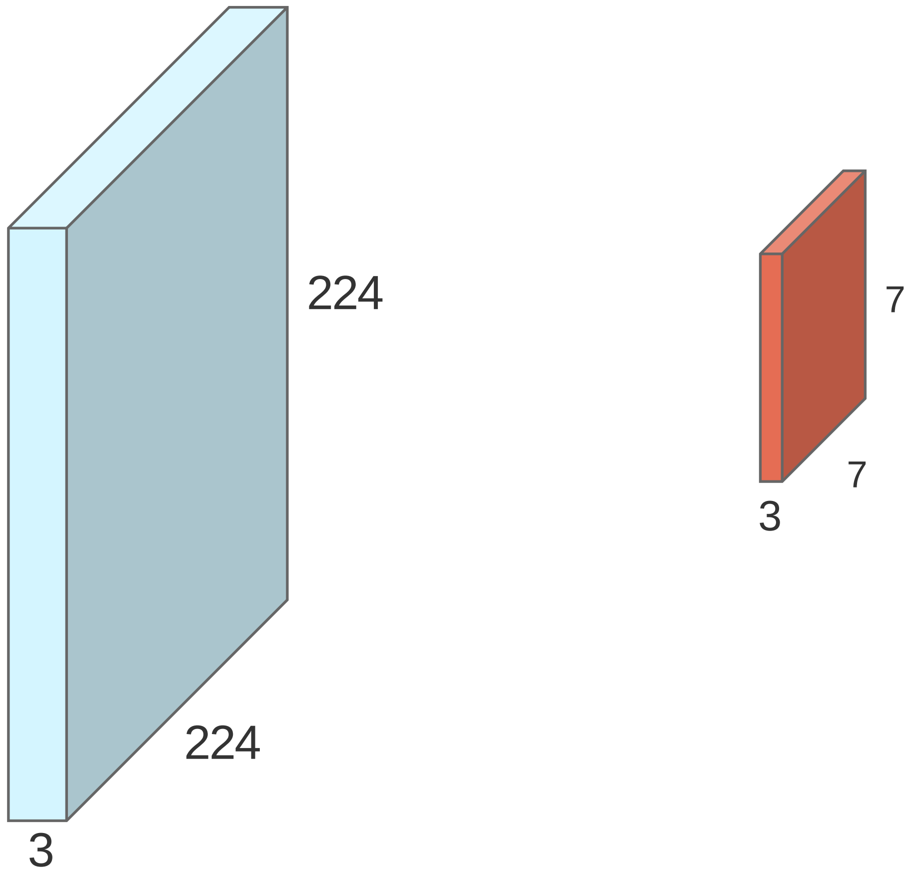
- Let us look at an image as a volume



- Width and height of the image is 224
- The Depth of the image is 3. This indicate the number of channels in the image.
- Total number of pixels in the image is 224 X 224 X 3

Convolutional Neural Network

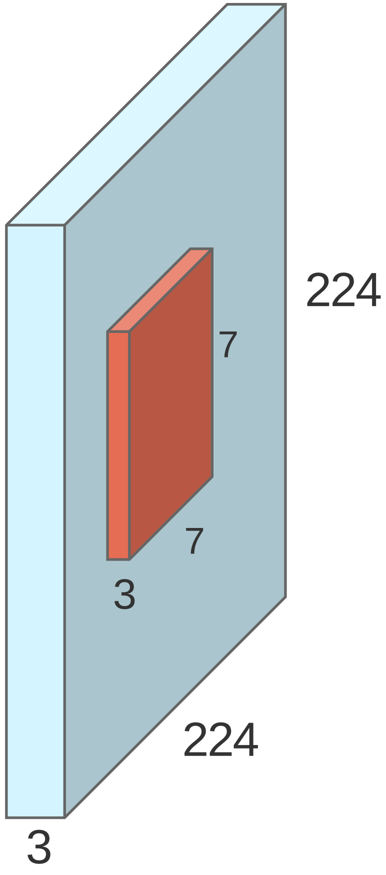
- Let us look at a Convolutional Layer



- Second small volume has a height and width of 7 and depth of 3.
- This image is often referred to as a filter.
- Sometimes in literature the filter size is only written as 7 X 7
- The depth of filter extends to the entire depth of the image
- In effect, an ordinary 7X7 filter is 7X7XD where D is the depth of the input volume

Convolutional Neural Network

- Let us look at a Convolutional Layer



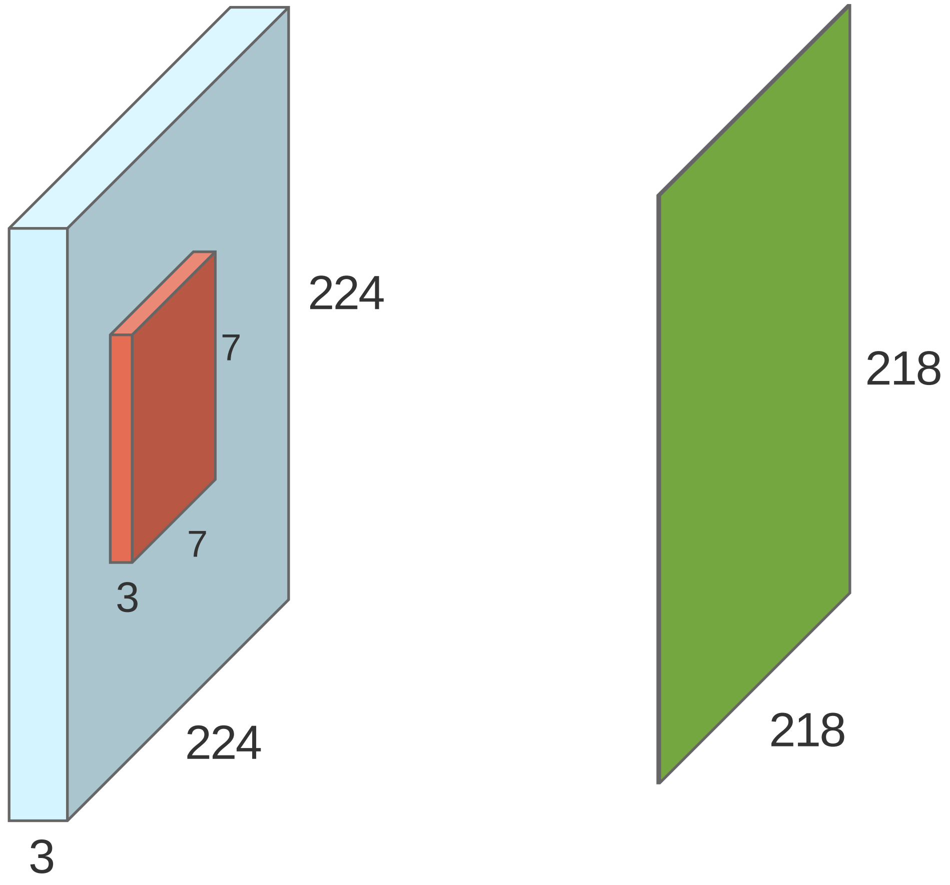
- The filter will be convolved with a part of the input volume
- At each location W^TX is computed
- That in effect would be taking element wise product of the filter with the part of the input volume that it overlaps
- This can be written as

$$\sum_{1=1}^{wxh \times D} X_i \times w_i$$

- Where X is the image and w is the filter
- This operation repeated by sliding the filter throughout the image

Convolutional Neural Network

- Let us look at a Convolutional Layer



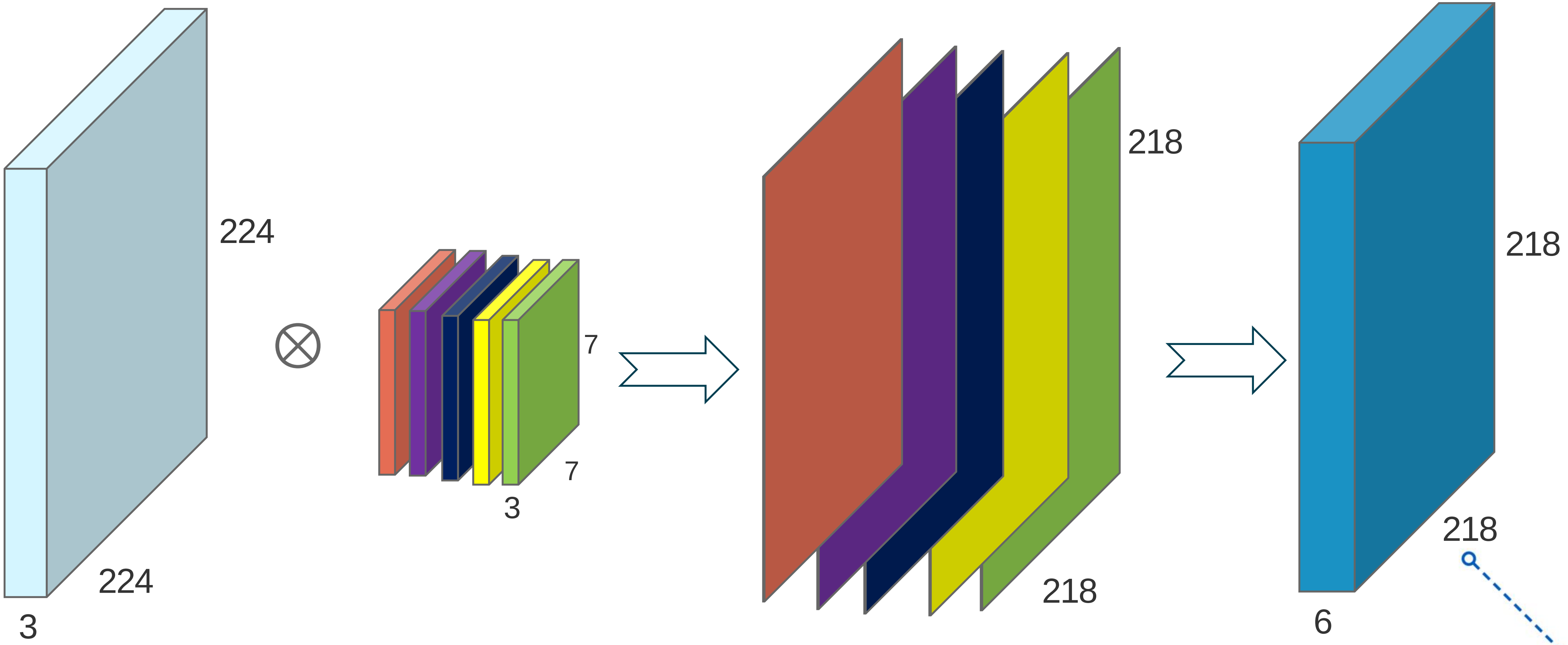
- The output volume after performing this operation has a size of

$$W^{new} \times H^{new} \times 1$$

- In this case we get an output of 218 X 218 X 1
- The value of width and height is determined by the number of unique location in which the filter can slide over
- The depth of the output is one, if we use one such filter
- Exact math of how we got 218 X 218, we will look at later.

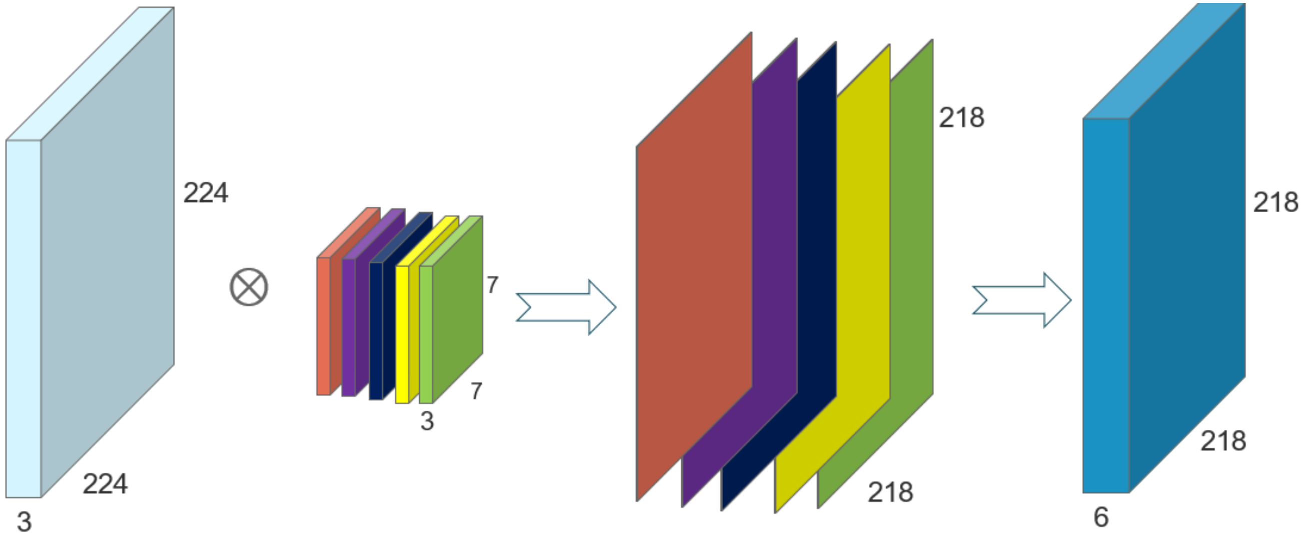
Convolutional Neural Network

- Let us look at a Convolutional Layer



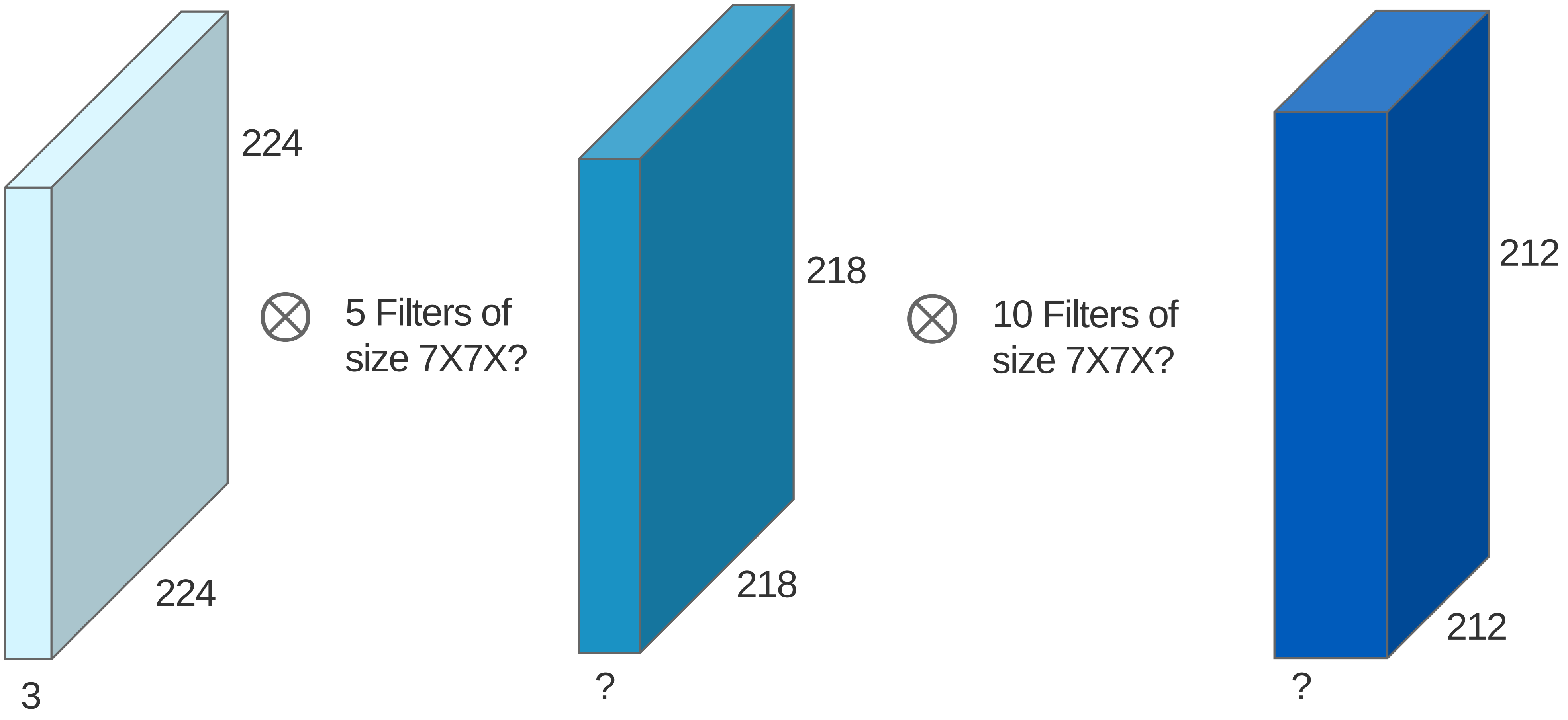
Convolutional Neural Network

- Each filter is applied separately on the input volume
- Each output created is stacked together to create the output volume
- Output volume is often called convolutional map or convolutional activation map
- The depth of the output volume is equal to the number of filters applied



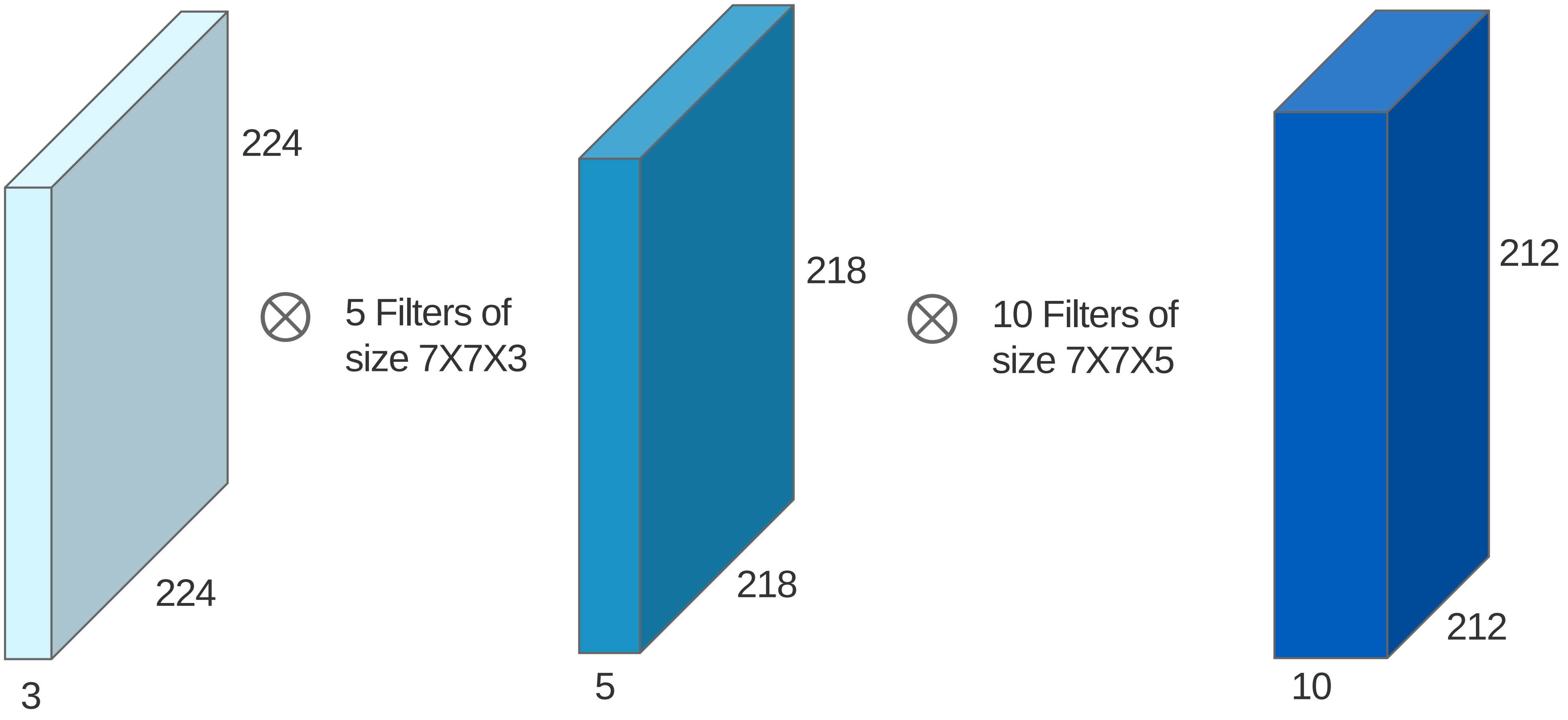
Convolutional Neural Network

- How are these convolutional layers arranged in a Neural Network?



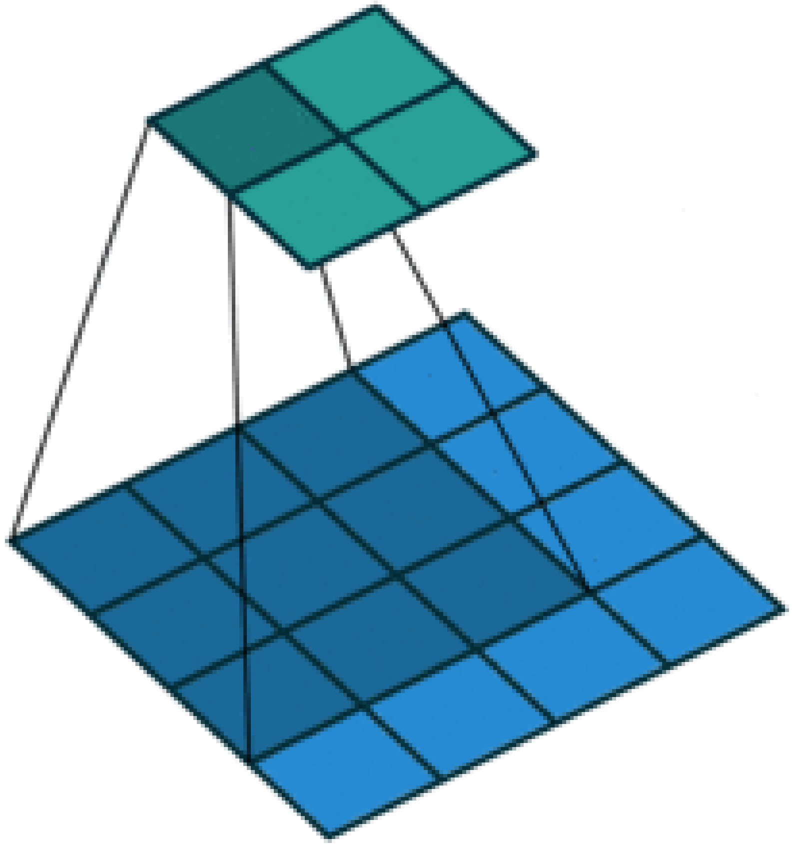
Convolutional Neural Network

- How are these convolutional layers arranged in a Neural Network?



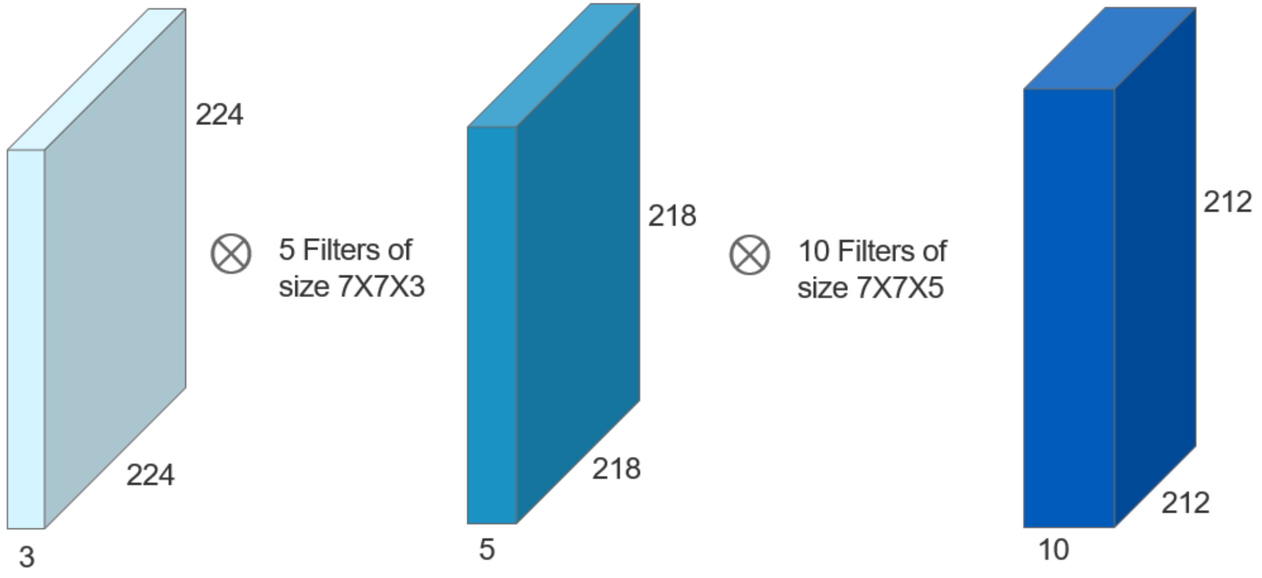
Convolutional Neural Network

- Let us look at top-down view of a convolutional layer
- What is the image size in this case?
- What is the filter size in this case?
- What is output spatial size in this case?
- What about depth?
- What is the stride?



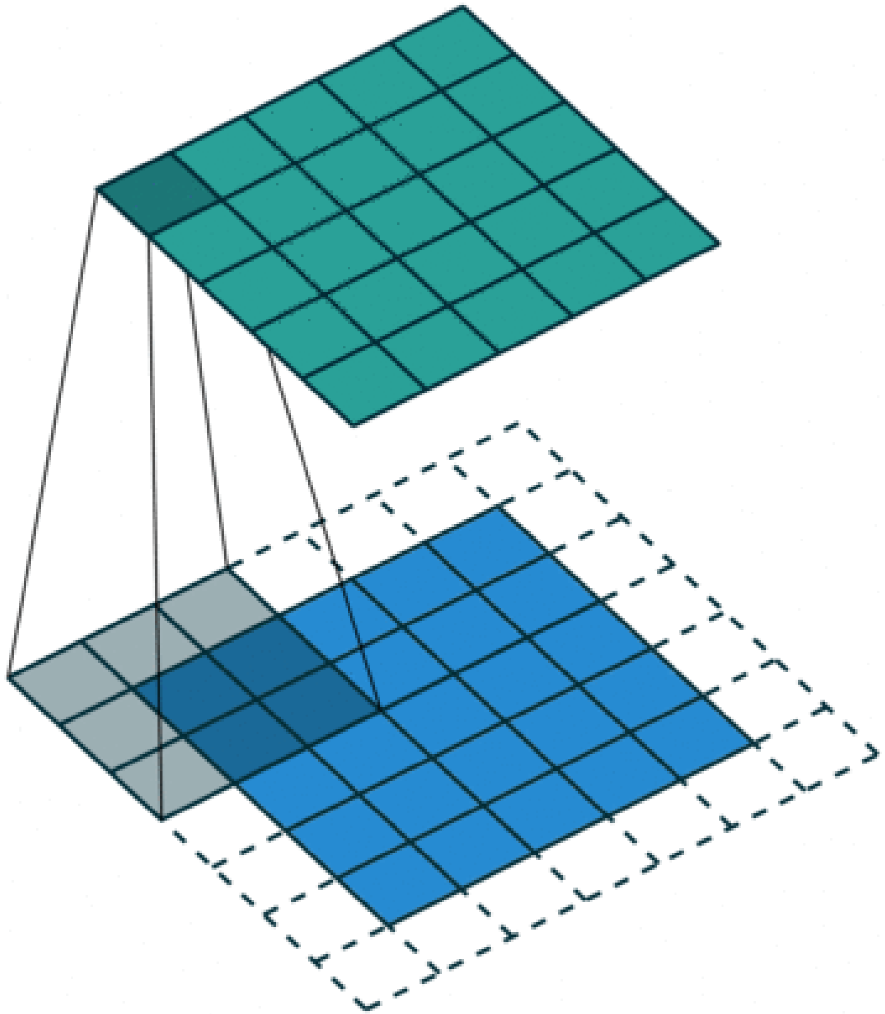
Convolutional Neural Network

- If you look closely the spatial size of the input decreases when a convolution filter is applied.
- This decrease in size is not ideal, since we want to build deep neural networks
- In order to avoid this size reduction, we use padding
- We pad the original image with zeros so that we get the original image size back after convolution



Convolutional Neural Network

- The image is padded with two rows and two columns of zeros
- This essentially gives the filter more unique locations to fit.
- The output spatial size created after the reduction associated with convolution operation is the same special size of the input
- Since the same spatial size is returned, this particular padding is sometimes referred as “same” padding



Convolutional Neural Network

- Let us now look at the arithmetic of the convolution operation
- Let the filter have width F_w and height of F_h
- Let the input image volume has a width of W, height of H and depth of K
- If the filter is applied with a stride of S and padding of the original image is P, then the final image size is given by the formula

$$W_{out} = [(W-F_w + 2P)/S] + 1$$

$$H_{out} = [(H-F_h + 2P)/S] + 1$$

C_{out} = Number of such filters applied

Convolutional Neural Network

- Let us look at an example

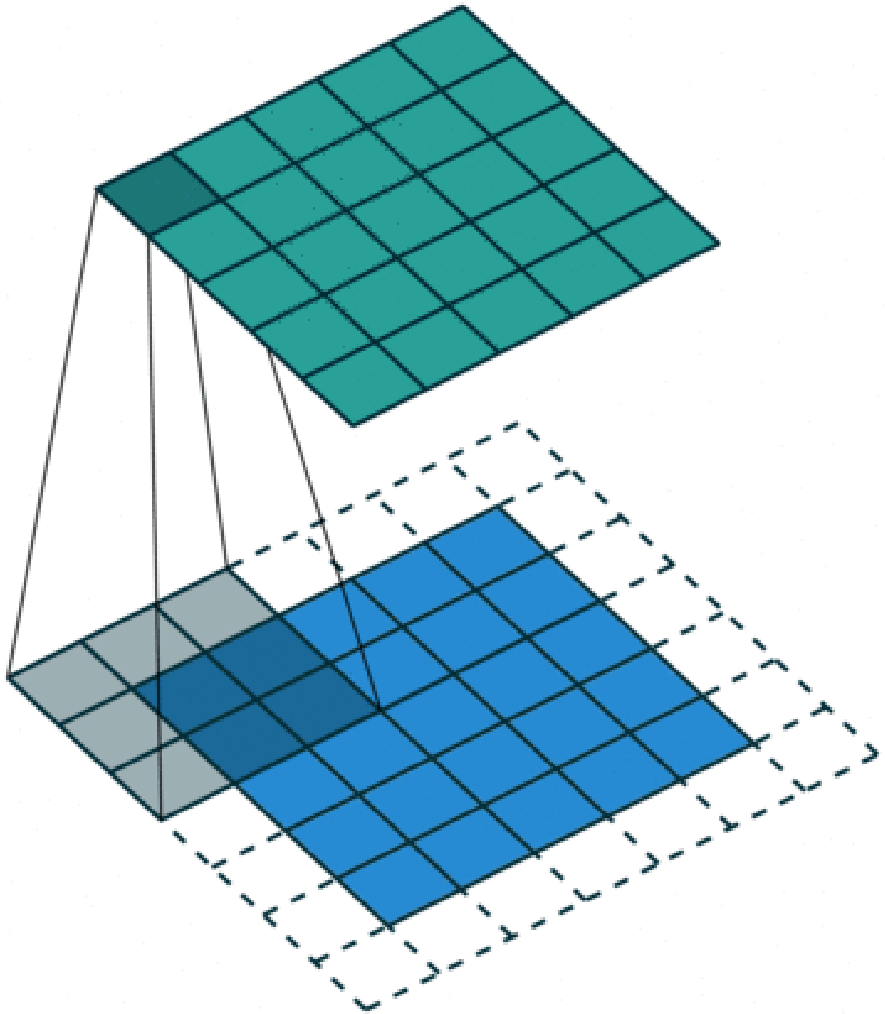
- Here input size of the image is 5 X 5. Assume the image has 3 channels. The filter size is 3X3X3 (since it extends the full depth)

- The padding in this particular case is 1 on each size of the input image and the stride is 1

- The final output size would be

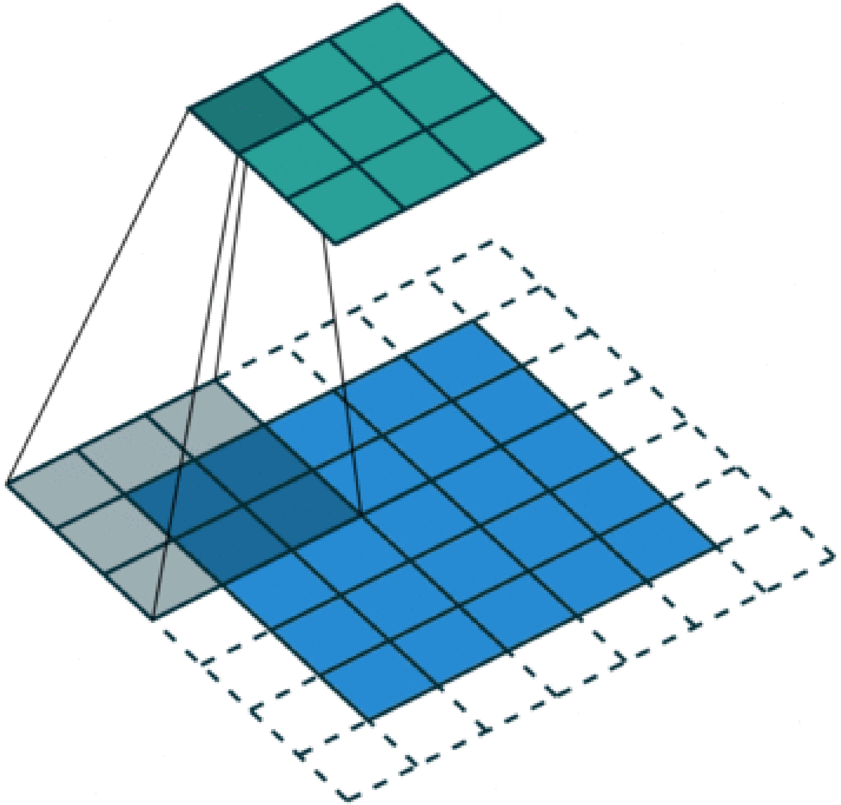
W_{out} = [((5-3) + 2*1)/1] + 1 = 3

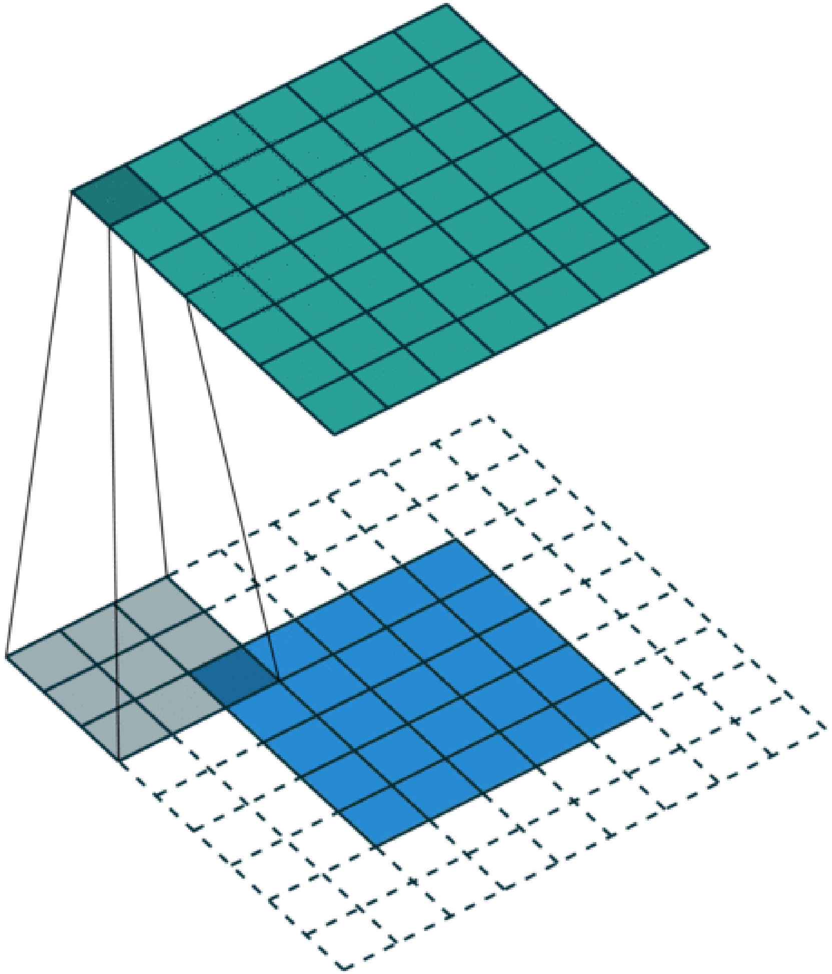
H_{out} = [((5-3) + 2*1)/1] + 1 = 3



Convolutional Neural Network

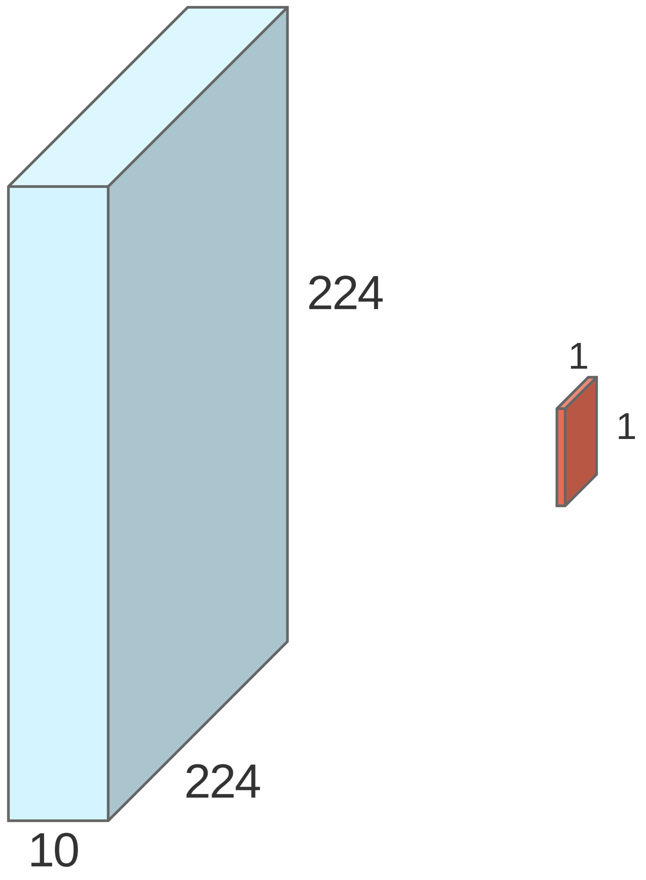
- What about these examples





Convolutional Neural Network

- Let us look at a convolutional layer



- Special type of convolution.
- Usually called 1x1 convolution.
- Does not aggregate spatial information.
- The filter is 1x1xD size.
- It performs a dot product at each pixel along the depth.

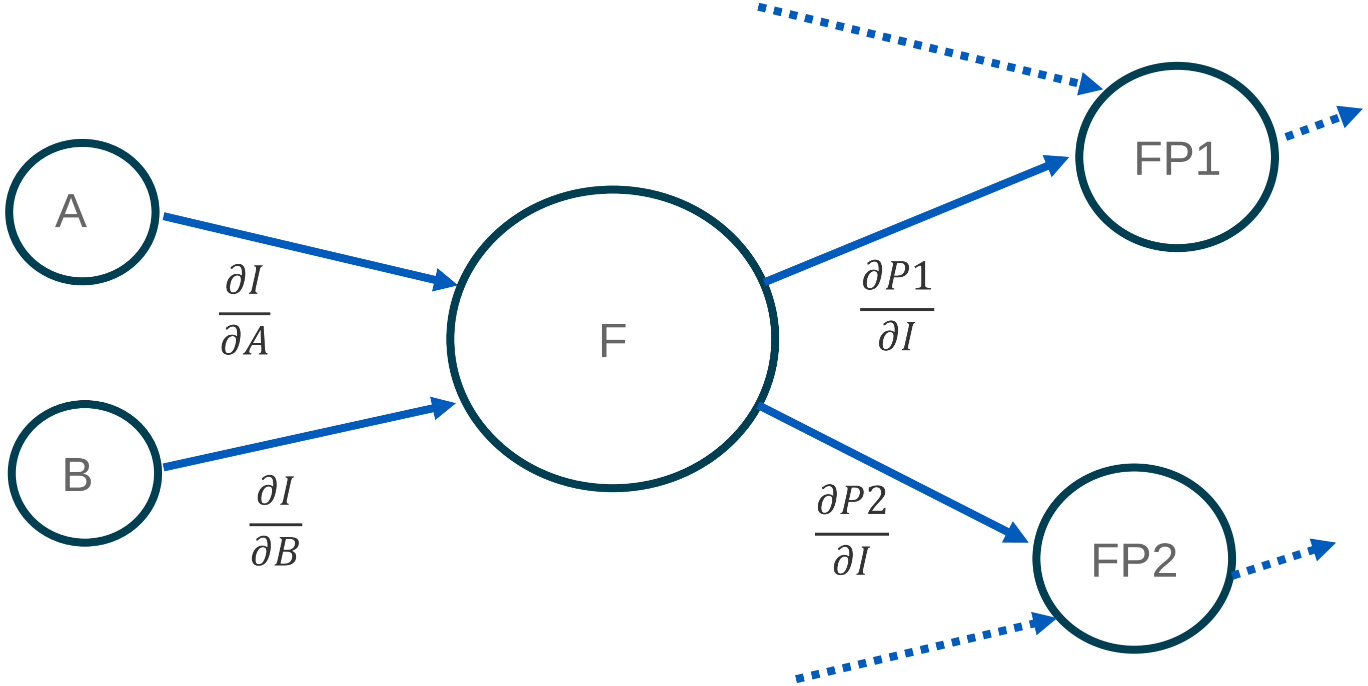
Convolutional Neural Network

- How does back propagation work in a convolutional neural network?

- Imagine if the intermediate result of the function is used in two different computations FP1 and FP2

- In order to compute the gradient $\frac{\partial output}{\partial A}$ and $\frac{\partial output}{\partial B}$, we will sum up all the gradients at F

- $$\frac{\partial Output}{\partial A} = \frac{\partial I}{\partial A} * (\frac{\partial P1}{\partial I} + \frac{\partial P2}{\partial I})$$
  $$\frac{\partial Output}{\partial B} = \frac{\partial I}{\partial B} * (\frac{\partial P1}{\partial I} + \frac{\partial P2}{\partial I})$$



Convolutional Neural Network

- How is the convolutional layer defined in the deep learning packages?

PyTorch

CONV2D

CLASS `torch.nn.Conv2d(in_channels, out_channels, kernel_size, stride=1, padding=0, dilation=1, groups=1, bias=True, padding_mode='zeros', device=None, dtype=None)` [SOURCE]

Applies a 2D convolution over an input signal composed of several input planes.

In the simplest case, the output value of the layer with input size (N, C_{in}, H, W) and output $(N, C_{\text{out}}, H_{\text{out}}, W_{\text{out}})$ can be precisely described as:

$$\text{out}(N_i, C_{\text{out}_j}) = \text{bias}(C_{\text{out}_j}) + \sum_{k=0}^{C_{\text{in}}-1} \text{weight}(C_{\text{out}_j}, k) \star \text{input}(N_i, k)$$

where $\star$ is the valid 2D cross-correlation operator, N is a batch size, C denotes a number of channels, H is a height of input planes in pixels, and W is width in pixels.

This module supports TensorFloat32.

- `stride` controls the stride for the cross-correlation, a single number or a tuple.
- `padding` controls the amount of padding applied to the input. It can be either a string {'valid', 'same'} or a tuple of ints giving the amount of implicit padding applied on both sides.
- `dilation` controls the spacing between the kernel points; also known as the à trous algorithm. It is harder to describe, but this link has a nice visualization of what `dilation` does.
- `groups` controls the connections between inputs and outputs. `in_channels` and `out_channels` must both be divisible by `groups`. For example,
  - At groups=1, all inputs are convolved to all outputs.
  - At groups=2, the operation becomes equivalent to having two conv layers side by side, each seeing half the input channels and producing half the output channels, and both subsequently concatenated.
  - At groups= `in_channels`, each input channel is convolved with its own set of filters (of size $\frac{\text{out\_channels}}{\text{in\_channels}}$).

Convolutional Neural Network

- How is the convolutional layer defined in the deep learning packages?

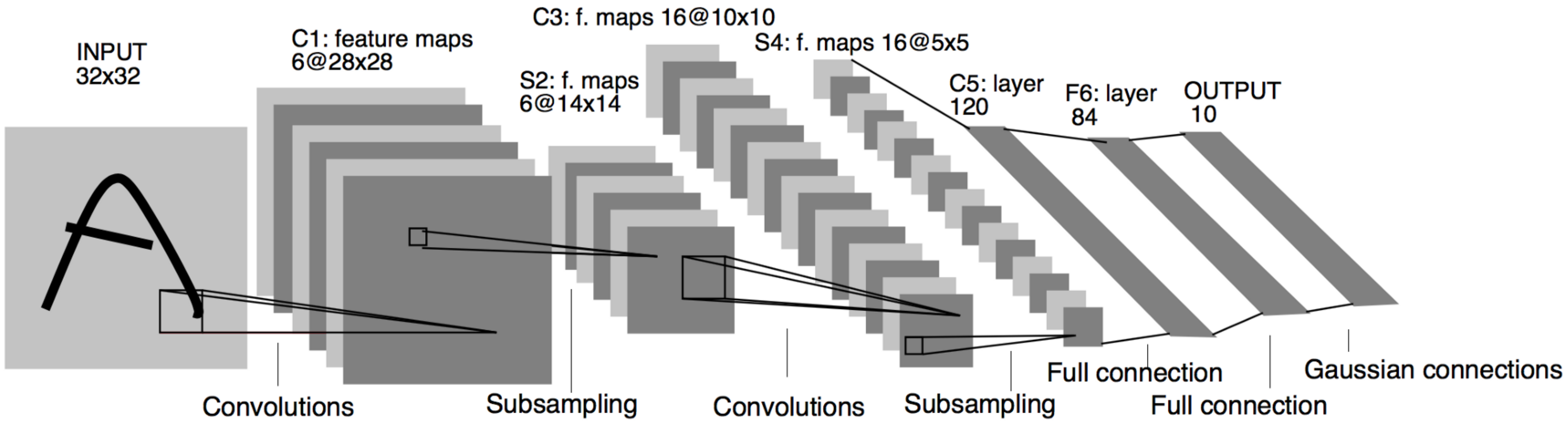
Keras

Conv2D layer

Conv2D class

```
tf.keras.layers.Conv2D(
    filters,
    kernel_size,
    strides=(1, 1),
    padding="valid",
    data_format=None,
    dilation_rate=(1, 1),
    groups=1,
    activation=None,
    use_bias=True,
    kernel_initializer="glorot_uniform",
    bias_initializer="zeros",
    kernel_regularizer=None,
    bias_regularizer=None,
    activity_regularizer=None,
    kernel_constraint=None,
    bias_constraint=None,
    **kwargs
)
```

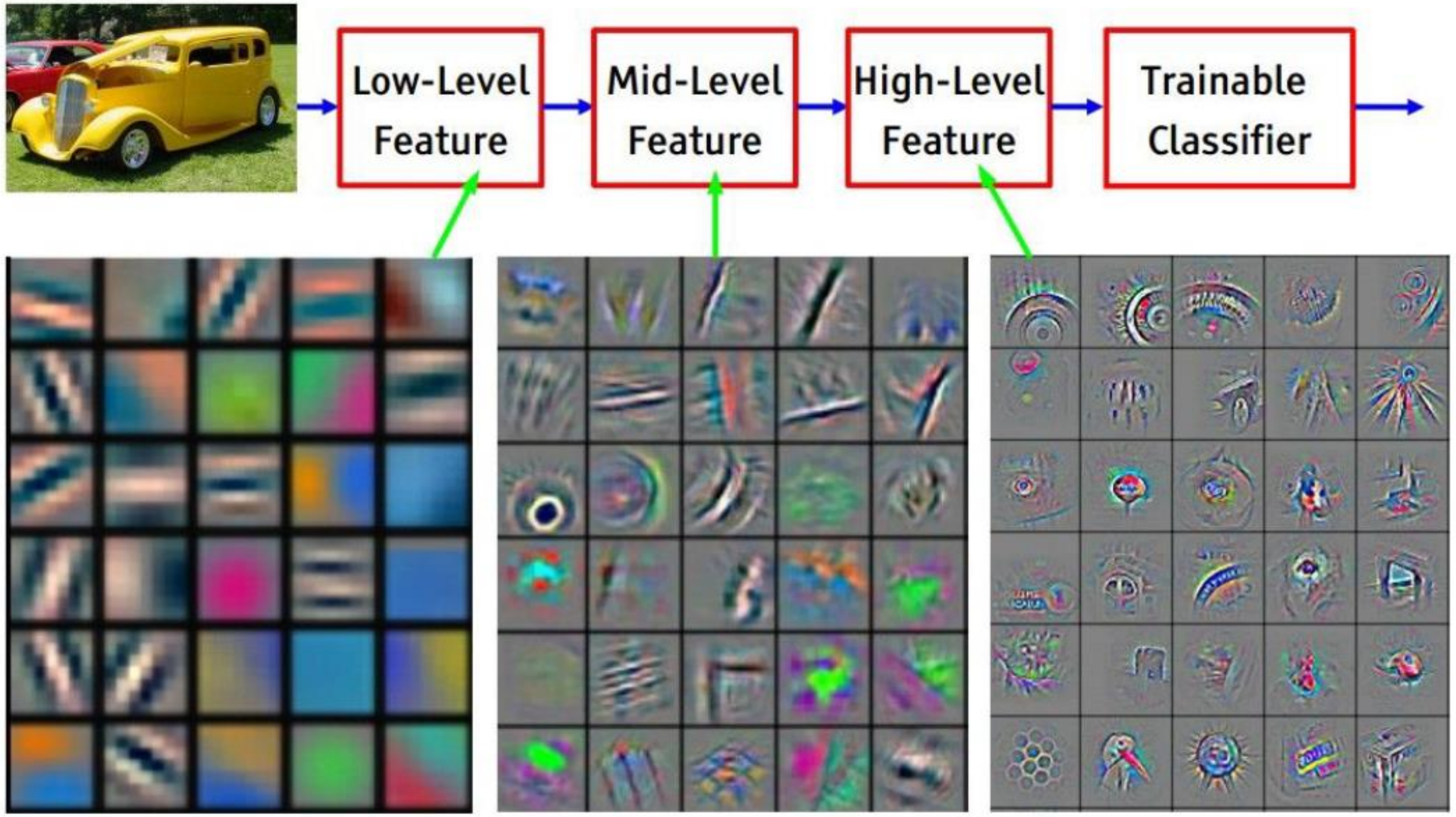
Convolutional Neural Network



- LeNet architecture

Convolutional Neural Network

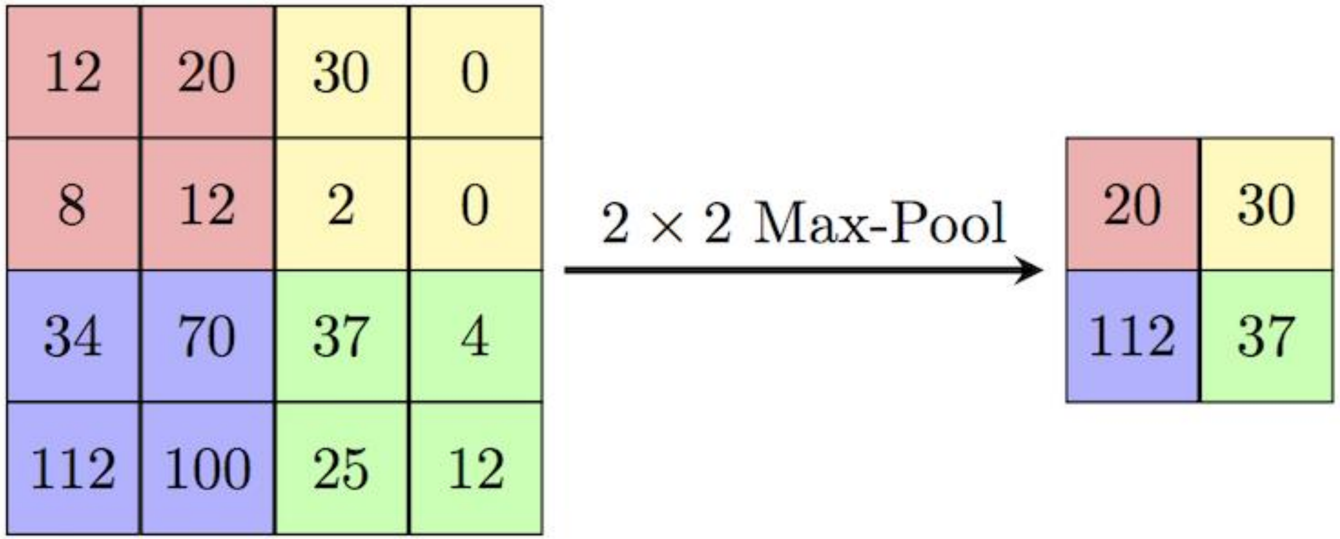
- What is intuition behind stacking convolutional filters?



Feature visualization of convolutional net trained on ImageNet from [Zeiler & Fergus 2013]

Convolutional Neural Network

- If we have large images, how do we make the information more manageable?
- We use pooling for the reducing the size of the image, works on each convolutional map independently
- Most used pooling technique is Max Pooling



Example uses max pooling with stride of 2

Convolutional Neural Network

- Let us look at an example

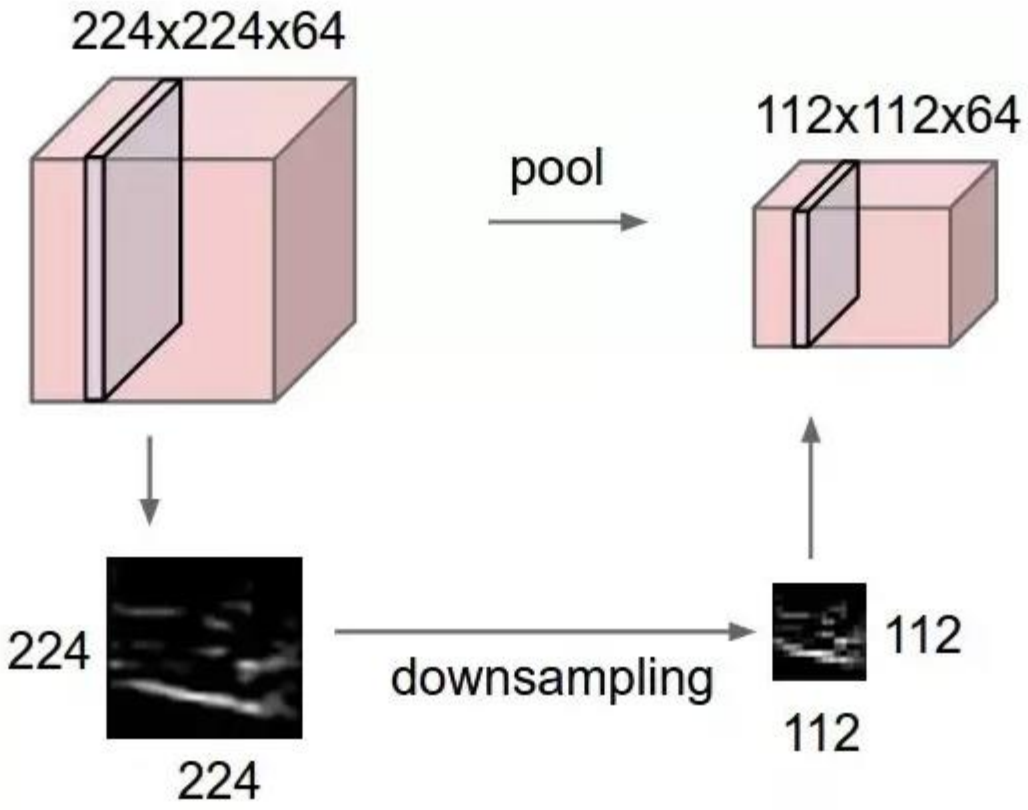
- Here input size of the image is 10 X 10. Assume the image has 3 channels. The max pooling size is 2X2 with stride of 2

- The final output size would be

  $W_{out} = [((10-2))/2] + 1 = 5$

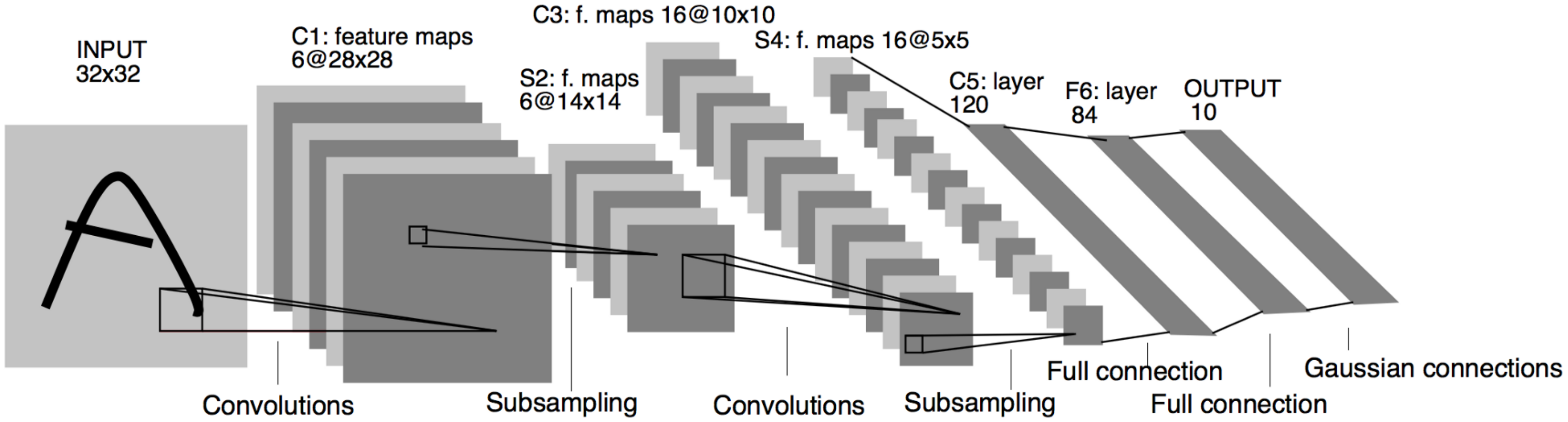
  $H_{out} = [((10-2))/2] + 1 = 5$

  $C_{out} =$ Number of input channels



Convolutional Neural Network

- So how many parameters does this neural network have?




ANY
QUESTIONS
?

References

- http://proceedings.mlr.press/v28/sutskever13.html
- This lecture is inspired from cse 231n https://www.youtube.com/watch?v=i94OvYb6noo&t=2051
- http://neuralnetworksanddeeplearning.com/chap5.html
- https://ruder.io/optimizing-gradient-descent/
- http://cs231n.stanford.edu/
- https://github.com/vdumoulin/conv_arithmetic
- https://www.google.com/imgres?imgurl=https%3A%2F%2Fmiro.medium.com%2Fmax%2F1400%2F1*Di4V69e4gC16ooF6PZPt-A.png&imgrefurl=https%3A%2F%2Ftowardsdatascience.com%2Feverything-you-need-to-know-about-neural-networks-and-backpropagation-machine-learning-made-easy-e5285bc2be3a&tbnid=PFKzBNejYXM4hM&vet=12ahUKEwism4aItO3yAhVrqnIEHSHUCNkQMyhDegQIARBf..i&docid=OXeL--Z4fRwo6M&w=1250&h=1057&q=neural%20networks%20with%20math&hl=en&client=firefox-b-1-d&ved=2ahUKEwism4aItO3yAhVrqnIEHSHUCNkQMyhDegQIARBf