

Convolution ... Kernels & Filters

Convolution formal definition

1. Convolution is a specialized linear operation on an image (a function), used for feature extraction (edges)
2. The linear operation uses a small array of numbers called a **filter/ kernel** on the input image (tensor)
3. The operation of filter over the image is an **pixel-wise product** between each element of the filter and corresponding pixel in input image
4. The resulting products are summed to obtain the output value in the corresponding pixel position of the output image, also called a **feature map**
5. This procedure is repeated applying multiple kernels to form an arbitrary number of feature maps,
6. Each feature maps represent different characteristics of the input image. Thus the different kernels can be considered as different feature extractors
7. Two key hyperparameters that define the convolution operation are size and number of kernels. The former is typically 3×3 , but sometimes 5×5 or 7×7 . The latter is arbitrary, and determines the depth of output feature maps.

Convolution formal definition

Input image as a rank 2 tensor of shape 5X5

Local field size of 3X3

Assume stride of 1, we need 3X3 neurons in first convolution layer

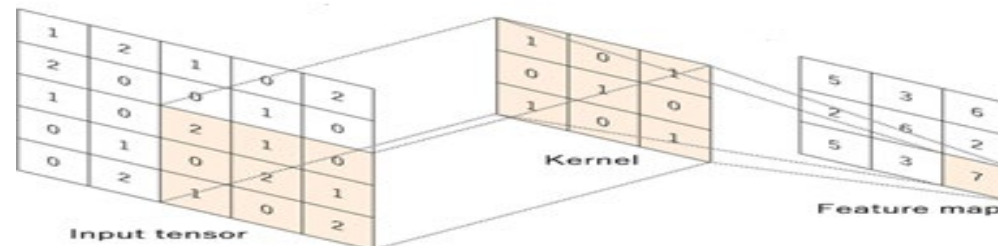
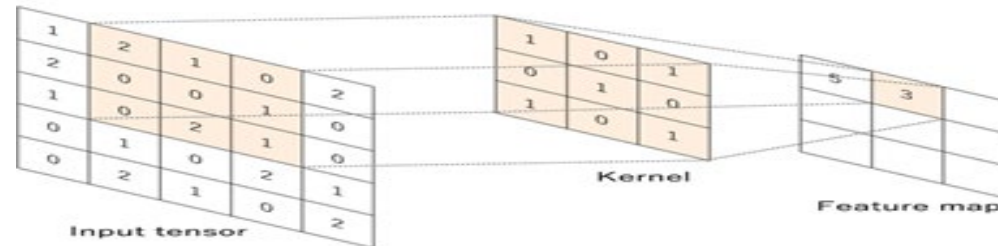
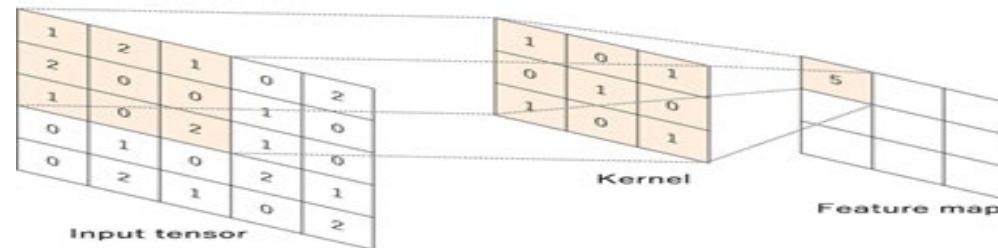
Each neuron is responsible for extracting features from it's local field

Each neuron will apply a bunch of kernels / filters to it's local field

Image as rank2 tensor of values

Single kernel applied across local fields

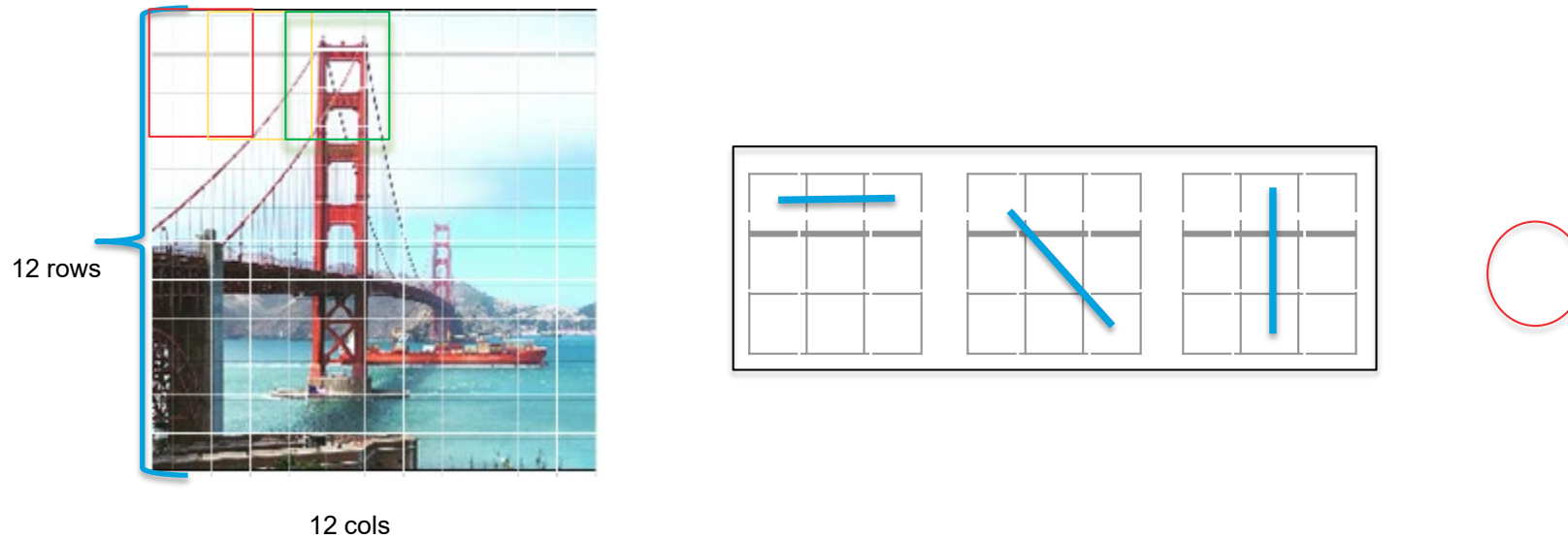
Resulting image with corresponding pixel values



Multiply each pixel of input image with corresponding element of filter

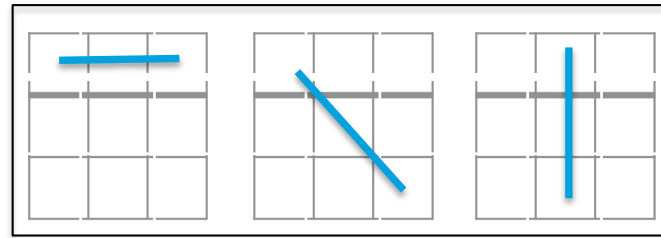
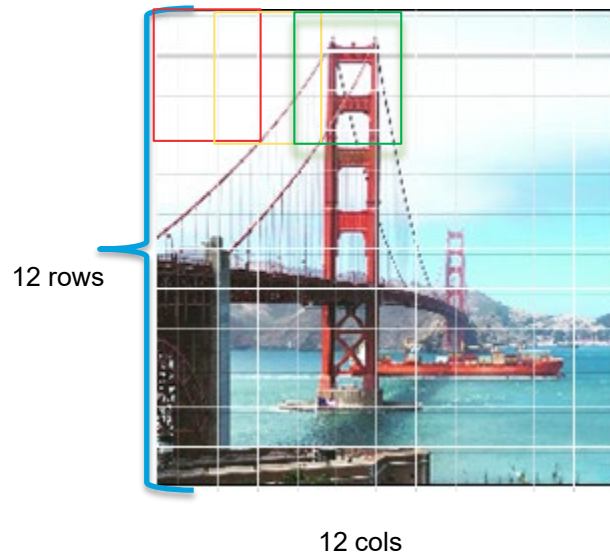
Add the products to fix value of the pixel in output image

Neurons Apply Filters in their regions respectively

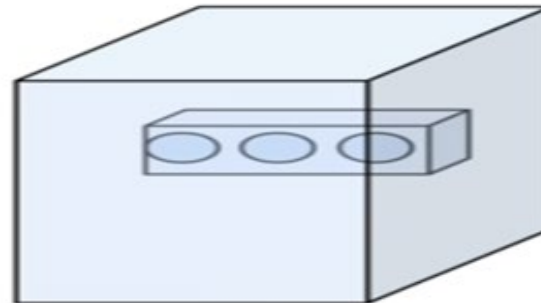
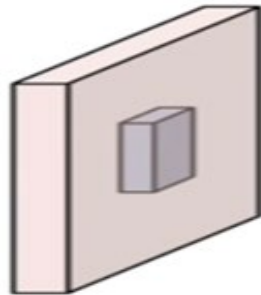


1. During training process, all the filters (as many specified in the architecture for a layer) are learnt through back prop. For e.g. 3 above
2. All the filters are initialized with weights using random gaussian or Glorot / Xavier initialization or He , methods. The weights are scaled between 0 – 1
3. Once the weights settle in each of the specified number of filters, the filters become grid of weights and bias
4. The filters are shared among all the neurons of the layer. This is known as shared weights and bias
5. Each neuron applies the bunch of filter on its region during the predict stage to extract features

Neurons Apply Features Maps in their regions respectively



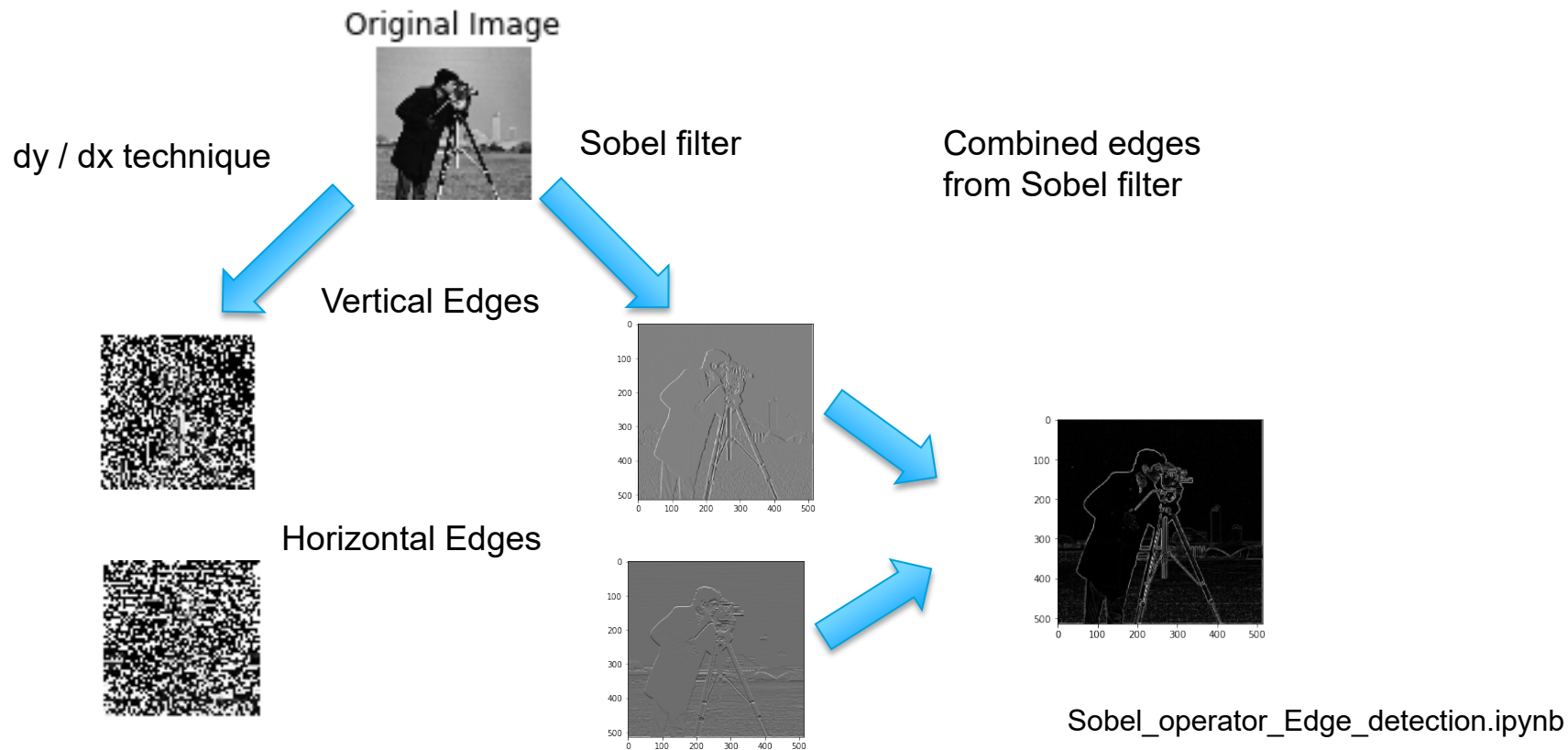
1. This fact is often pictorially shown as below.
2. The blue box is the feature map volume of size 10X10X3 in our case
3. The three neurons are basically the three filters with respective weights



Filters and Activation Maps

Convolution for Edge Detection / Horizontal edge detection

1. Simple filters subtract next pixel value from current to get the difference and may not always give good results (as in the case of camera man image)
2. Instead we use a different filter such as the Sobel filter which detects edges well

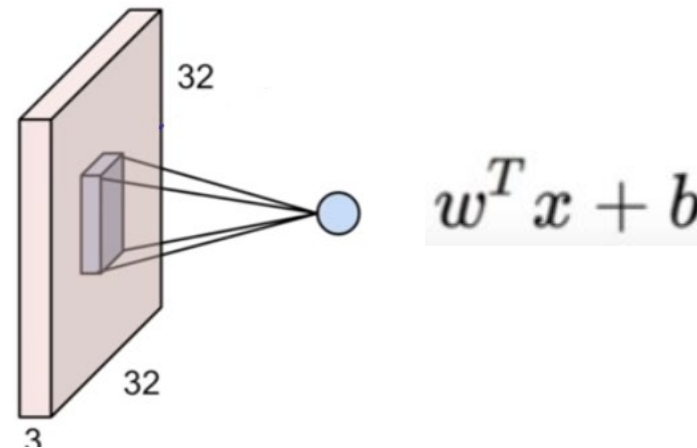


Filters and Activation Maps

Lenna_RGB_Filtering.ipynb

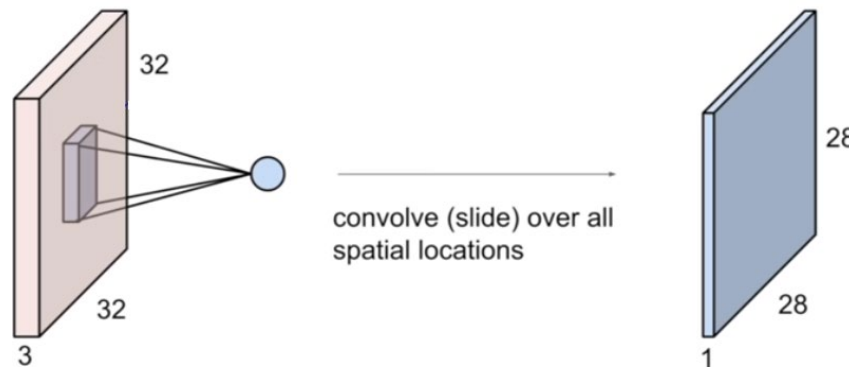
3. The sliding of the filter over the input image results in
 - a. W represents the filter weights of size $5 \times 5 \times 3$ (for color channels)
 - b. x represents the picture pixel values under the filter
 - c. b is a bias term
 - d. One neuron has $(5 \times 5 \times 3)$ coefficients + one bias term = 76 parameters

4. In this process, the filter is first centered on a pixel of the original image
 - a. When centered on the top left(0,0) pixel, the filter will go beyond the boundaries of the image
 - b. To prevent that, the original image will be padded



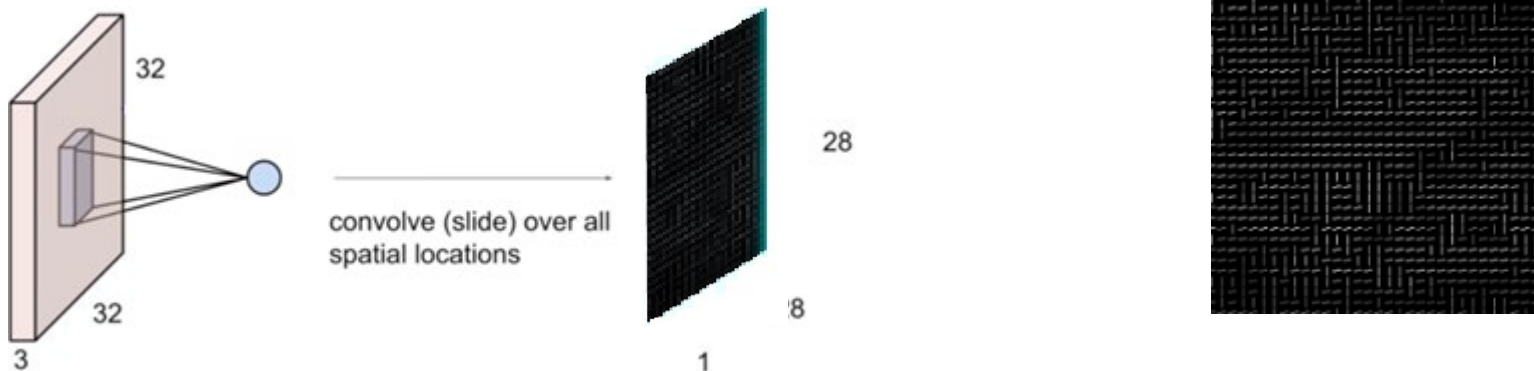
Filters and Activation Maps

5. The dot product between the filter elements and corresponding pixels underneath consist of a multiplication with addition
6. This operation is similar to finding the gradients around the centered pixel i.e. dy/dx . How much y (pixel intensity) changes in the neighborhood of a pixel
7. The result of this operation will be a transformed image (of lesser size if original is not padded)



Filters and Activation Maps

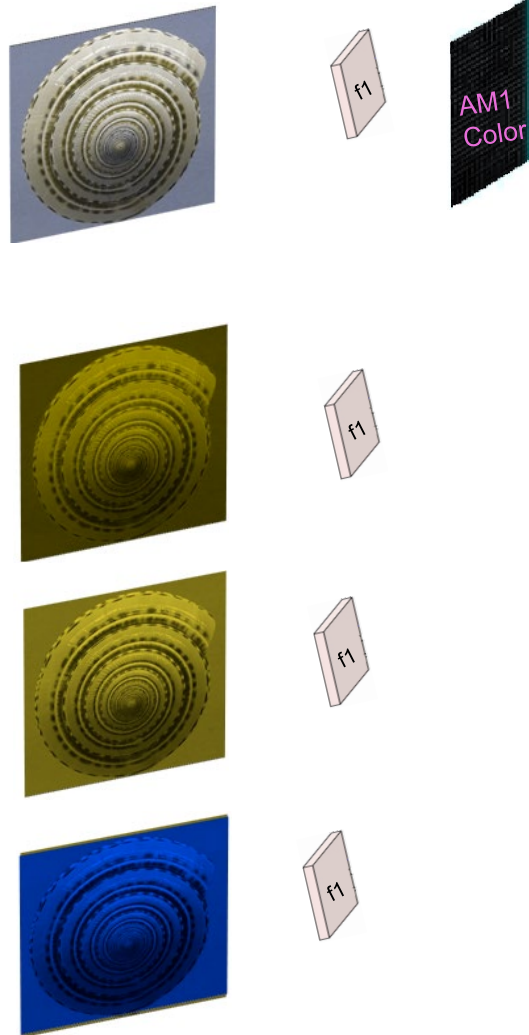
8. The filter is applied on the original picture on each color channel. Thus the gradient at a pixel is found independently for R,G,B.
9. The three gradients are combined (vector operation) to get resulting gradient at the pixel in RGB
10. The filter operation thus results in a gradient map (a.k.a Activation Map) at each pixel of the original picture



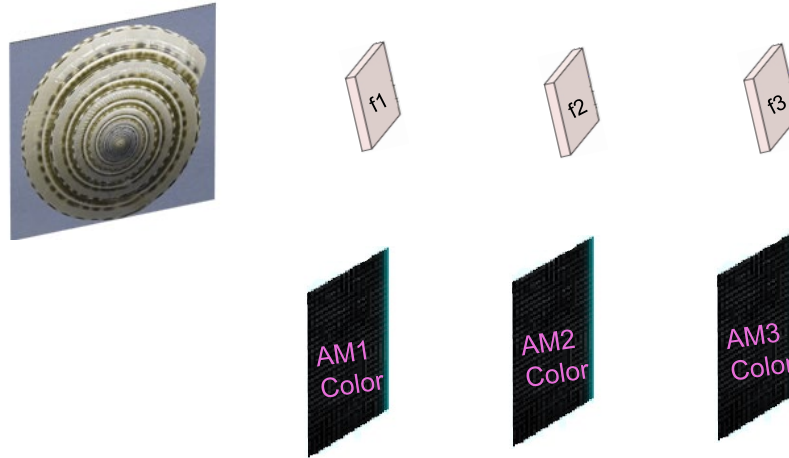
Filters and Activation Maps

- 11. One filter generates one activation map
- 12. One filter will not be able to capture all the gradients in original picture given the values in each filter element
- 13. A filter may detect gradient across vertical line while another may do so across horizontal line while another one detects gradient across angular lines in the input image
- 14. Thus we need a collection of filters, with each filter specializing in one direction of gradient to capture all the gradients into activation maps

Filters and Activation Maps



Filters and Activation Maps



- 15. As many filters that are applied on an image, that many Activation Maps are created.
- 16. Each activation map is the combined result of individual activation maps on the R,G,B

Filters and Activation Map Size

17. The filters are placed on the top left of the image.
The region of image covered is called receptive field

18. The mathematical operation results in a total value of the pixel in the new image (pink)

19. The filters cannot cross the boundary of the image

20. The filters can slide one pixel at a time or more (Stride)

21. Size of the new image in row and column is given by
 $R_{new} = ((image_rows - filter_rows) / stride) + 1$. Similarly,
 $C_{new} = ((image_cols - filter_cols) / stride) + 1$

22. Key words - Filter, filter size, stride, receptive field, activation map

Filter

1	0	1
0	1	0
1	0	1

Image

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

Activation Map

Filters and Activation Map Size (Exercise)

1. What is the size of the activation map
 - a. image size of 32 X 32 X 3
 - b. Filter size 5X5
 - c. Stride = 1
 - d. Ans : $\text{activation_row} = ((32 - 5) / 1) + 1 = 28$
 - e. $\text{activation_cols} = \text{ditto} = 28$
 - f. Activation map size = 28 X 28 X 1 (one map for RGB)
2. What is the volume of activation map if image size is 32 X 32 X 3 and the Filter size is 5 X 5 and number of filters = 10
 - a. Ans : 28 x 28 X 10 (as many filters, that many activation maps)
3. What is the size of activation region if image size is 7 X 7, filter size = 3 X 3 and Stride = 2 (Ans: 3 X 3)
4. What is the size of activation region if image size is 7 X 7, filter size = 3 X 3 and Stride = 3? (Error)

Filter

1	0	1
0	1	0
1	0	1

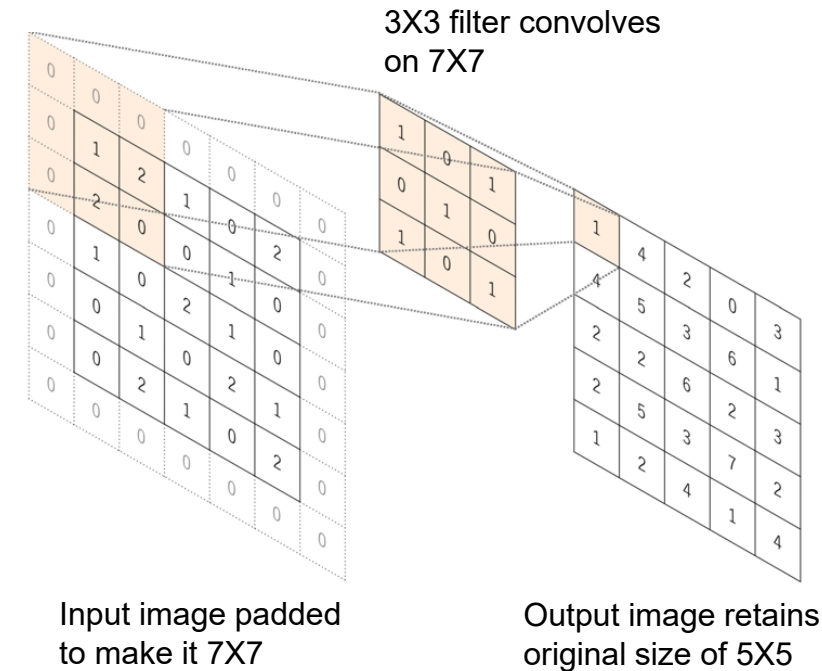
Image

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

Activation Map

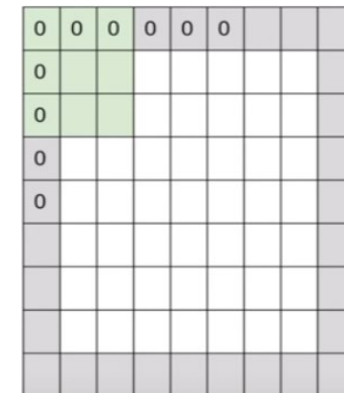
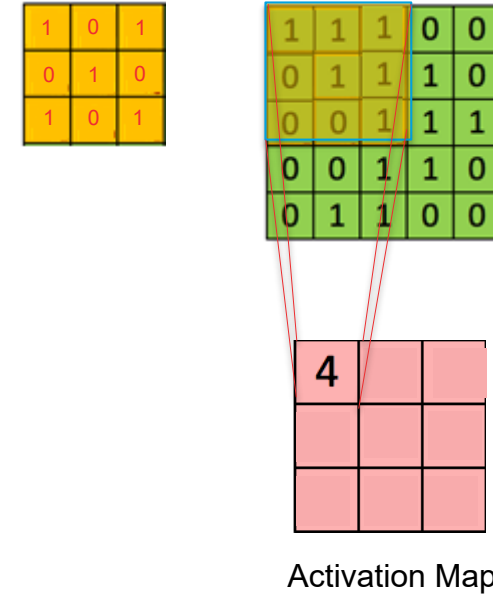
Filter convolution reduces output image size

1. In the convolution process, it is not possible to center the filter over the outermost pixels of the input image
2. As a result, it reduces the height and width of the output feature map compared to the input image
3. If not addressed, each successive convolution layer will reduce the image further and further
4. A technique called padding is used to prevent image size reduction. Padding, typically zero padding, is used where rows and columns of zeros are added on each side of the input image
5. This enables fitment of the center of a filter on the outermost pixel and keep the same image size throughout the convolution operations



Filters and Image padding for edge reading

1. The filter grid sliding over the image grid with any stride results in a value associated with a single neuron
2. The value represents the gradients / features surrounding the central pixels in the overlap between filter and image pixels
3. Features on the edge of the image may get unresolved! To avoid this, we use padding.
4. The image is padded on all sides by a dummy set of pixels which are usually assigned value of 0
5. The number of padding layers required will depend on the filter size
6. With appropriate padding the activation map size can remain same as original image size



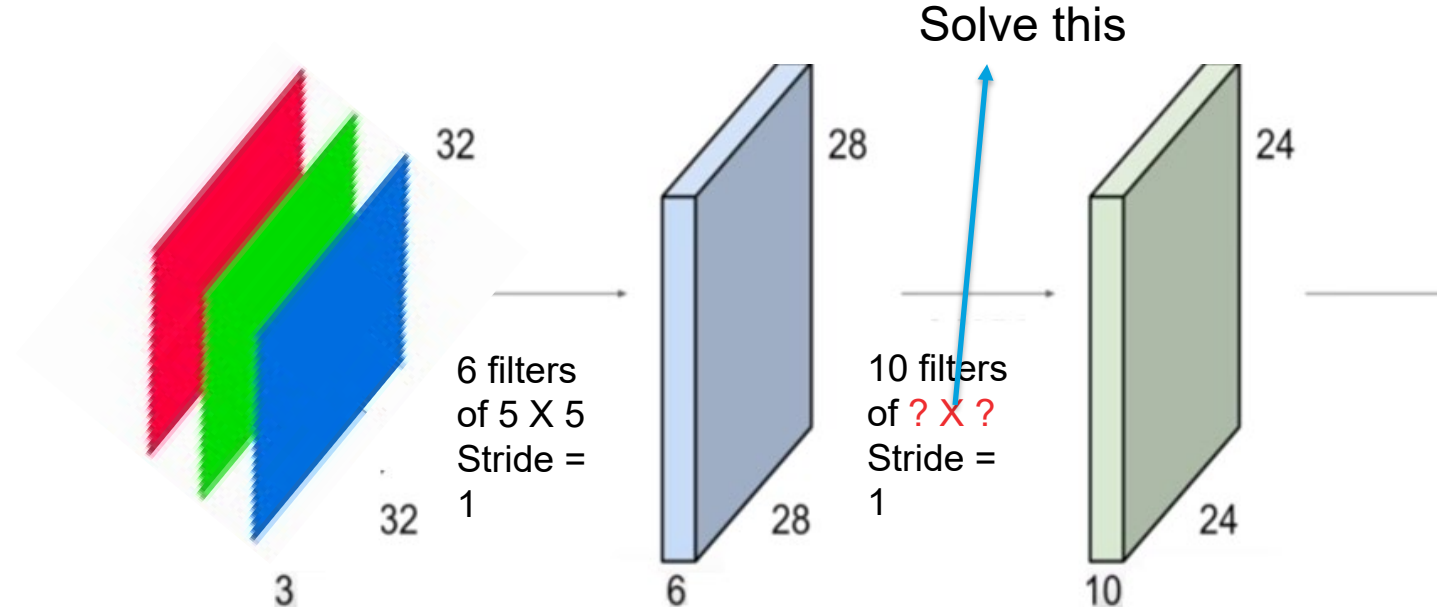
Filters and Image padding for edge reading

1. A 7 X 7 image with 3X 3 filter and stride of 1 and padding of one layer on all sides will result in activation region of
 - a. Activation row = $((7 + 2) - 3) / 1 + 1 = 7$
 - b. Activation col = ditto = 7
 - c. The 2 is extra bits added per row and per column
2. A 7 X 7 image with 3X 3 filter and stride of 1, padding of 1, 10 filters will result in activation region volume of ?
 - a. Ans : 7 X 7 X 10
3. It is common to see convnet with strides of 1, filters of size F X F and $(F-1)/2$ zero-padding preserves the size spatially. For e.g.
 - a. F = 3 , zero padding = $(3-1) / 2 = 1$
 - b. F = 5 , zero padding = $(5 - 1) / 2 = 2$

0	0	0	0	0	0	0			
0									
0									
0									
0									

Filters and Image padding for edge reading

4. Importance of padding can be understood by looking at the fast rate at which the size of activation region decreases when multiple convolutional layers are stacked



5. When we need large depth i.e. number of convolutional layers in the network, we need to pad the regions appropriately else we lose edge information in successive layers!

Filters - Important points

1. Each filter is designed to identify specific feature in the input space of the image.
2. Imagine a filter 3×3 to detect vertical lines. A filter at start may be poor at the job
3. Through backprop, it finally learns to identify the line where ever the line may be in the input space
4. At the point where it has learnt, recall it is simple dot product + a bias term (linear equation)
5. Thus the filter needed fine tuning on $(3 \times 3 \times 3) + 1$ parameters = 28 (Note: it has to learn the weights for R,G,B and
6. If there are 10 3×3 filters in conv layer, we need 100 parameters to learn (Why?)
7. The entire convnet's learning is in form of the weights and biases learnt at the conv layer

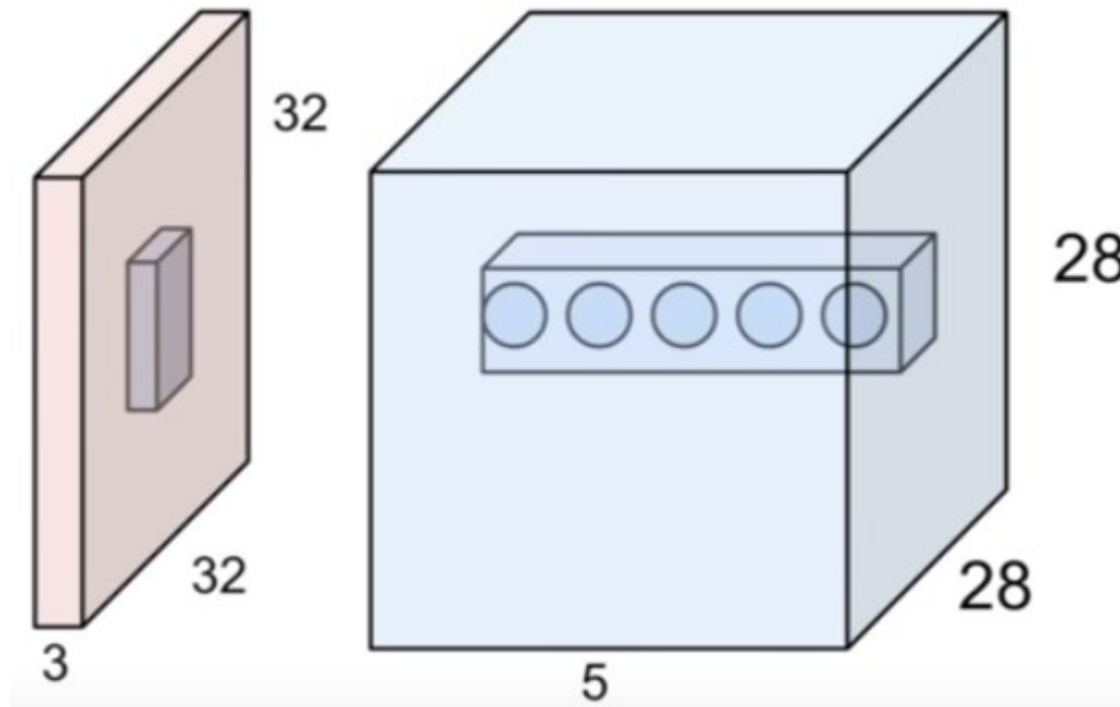
Note: filter is also known as receptive field for the neuron

Filters - Important points

8. Common settings
 - a. K (number of filters) = powers of 2 such as 32, 64, 128 etc
 - b. F (filter size) = 3,5,7
 - c. S (stride) = 1,2
 - d. P (padding) = 1, 2
9. Remember the relation between padding, filter size and the output size

Filters and Conv Layer

Pictorial view of conv layer with 5 filters each of size 3X3



1. Each neuron (shown as a circle in blue inside the activation map region) is dedicated to one filter
2. All of these neurons share same receptive field / region in the input space

Convolution Exercise

1. What is the volume of activation region given -
 - a. Input volume 32 X 32 X 3 (image size)
 - b. Filter size 5 X 5
 - c. Number of filters 10
 - d. Stride = 1
 - e. Pad = 2

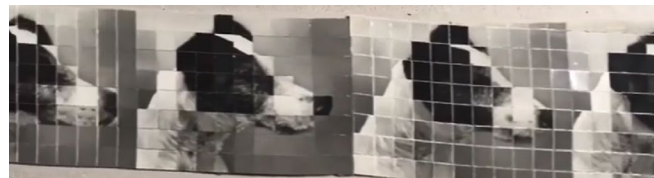
2. What is the number of parameters in this layer?

Convolution Layer Summary

1. Input volume i.e. image size $W1 \times H1 \times D1$
2. Hyperparameters to control the design of conv layer include
 - a. Number of filters K
 - b. Filter size $F \times F$ (all filters are of same size)
 - c. Stride S (same for all filters)
 - d. Zero padding P
3. Output of the conv layer is a volume of size $W2 \times H2 \times D2$ where
 - a. $D2 = K$
 - b. $W2 = H2 = ((W1 + 2P - F) / S) + 1$
4. With parameter sharing, number of parameters per filter = $(F \times F \times D1) + 1$
5. Number of learnable parameters at conv level = $((F \times F \times D1) \times K) + K$ biases
6. In the output of a conv layer, the volume = $W2 \times H2 \times K$
7. In the output volume, the d-th depth slice is the result of convolution of d-th filter over input volume with stride S and d-th bias

Pooling Layers

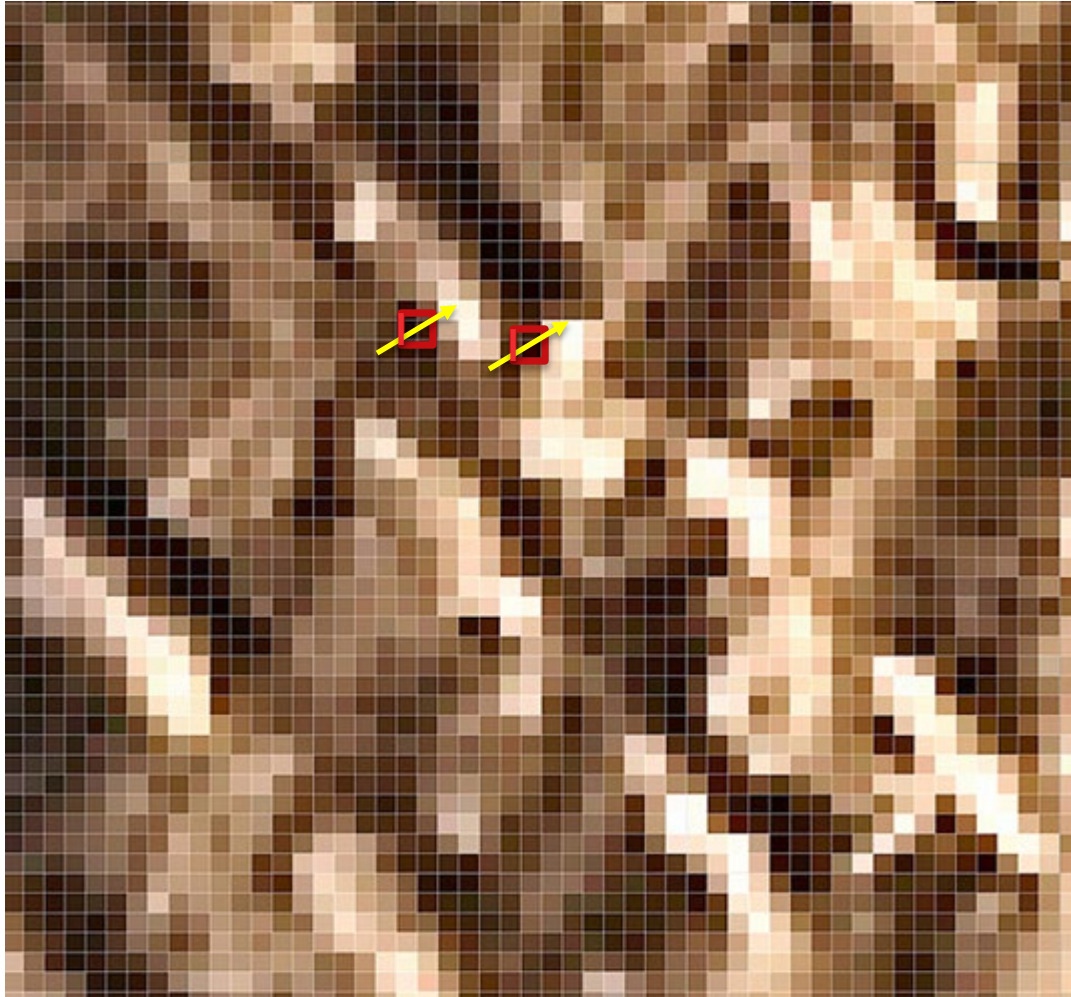
Why pool?



What does this tell us about images ?

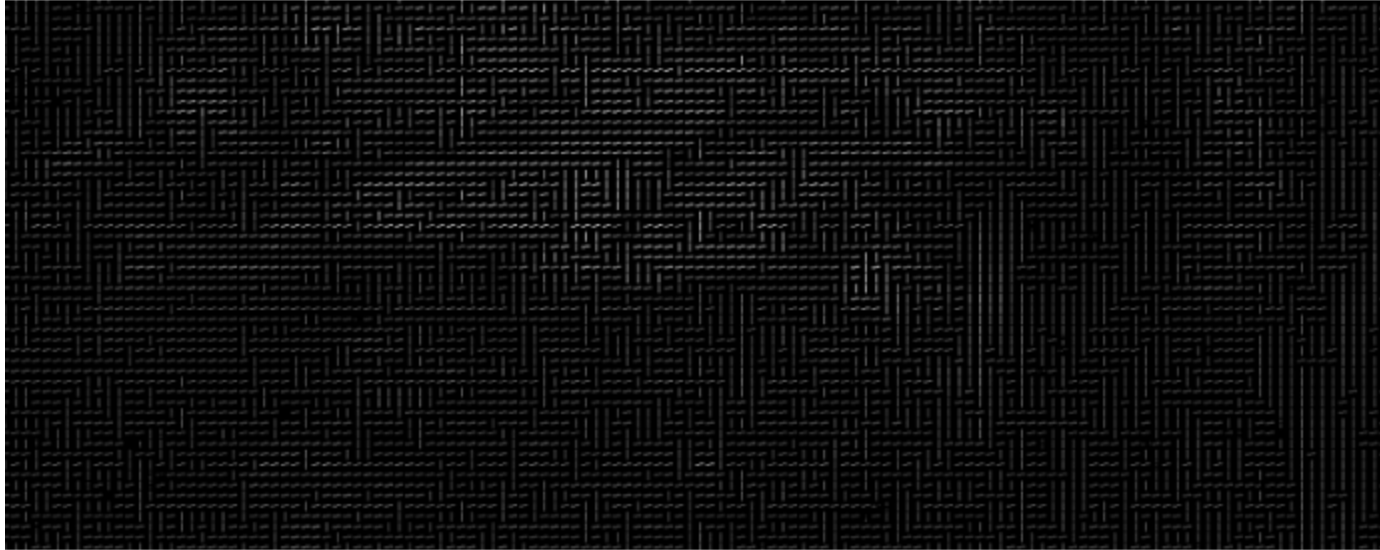
1. For object recognition we need high level features.
2. Many of the low level features are redundant
3. In this example, many of the pixels are unimportant
4. Only pixels that for the edge of the object are important and that too no all pixels on the edge

Why pool?



1. Raw pixel values to intensity gradients
2. Gradients point in the direction of max increase in pixel intensity
3. The entire image can be considered as a vector field of image gradients

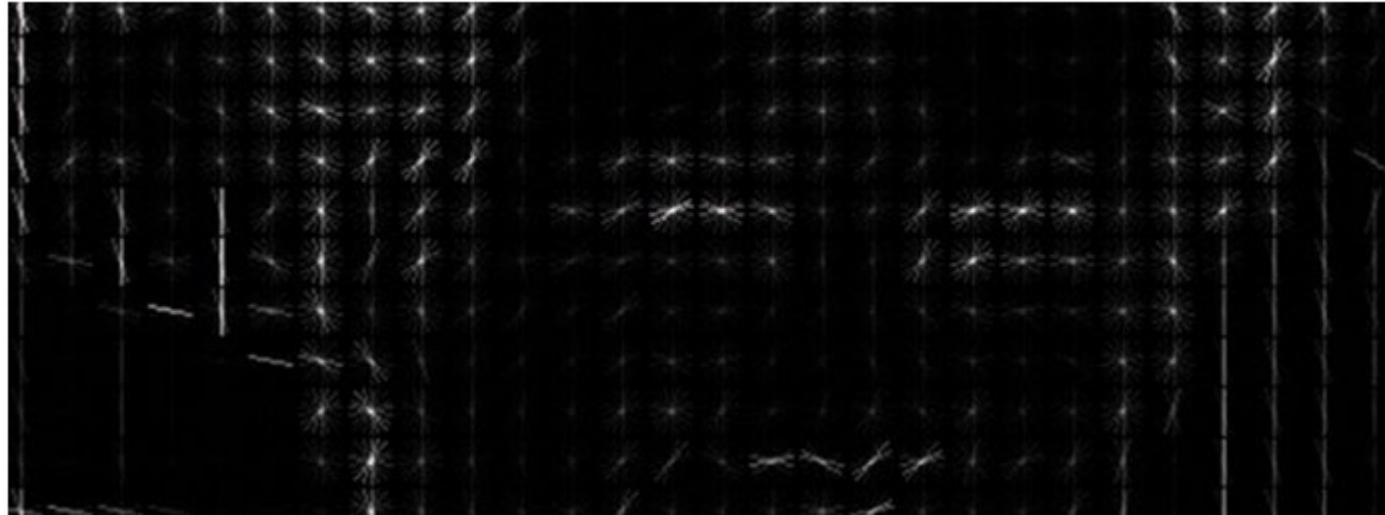
Replace the pixels with gradients



1. If we analyze by pixel values, same object under different lighting conditions will look different.
2. Instead replace the pixel based image with gradient based image
3. Considering the *direction* of change in pixel value, a picture under all lighting conditions will look the same (more or less)
4. Saving the gradient for every single pixel will be too detailed, many gradients will be redundant
5. Instead, if we capture the general flow from dark to bright and vice versa, we may have sufficient information. This is where pooling comes in

6. To get the overall flow, break up the image into small squares of $m \times n$ pixels each.
7. Usually the $m \times n$ is a 2×2 matrix with stride of two.
8. In each such square, count up how many gradients point in each major direction
9. Replace that square in the image with the arrow directions that were the strongest (max pooling).
10. We can also replace that square with an average direction (average pooling)

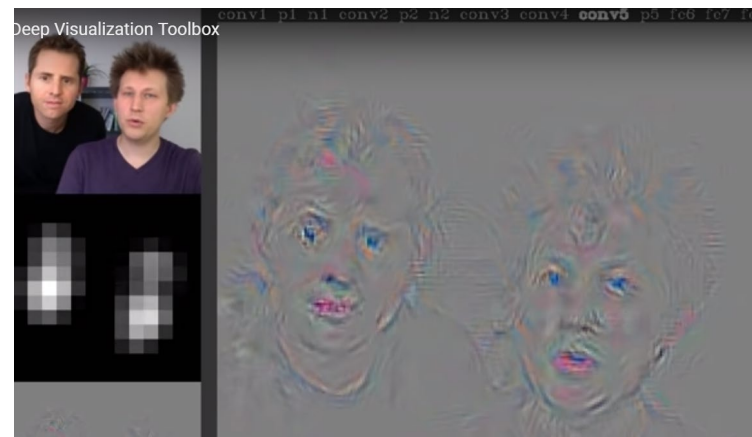
Max Pooled Gradients



11. Sometimes we may need to repeat the convolution and pooling multiple times in series i.e. convolve over a pooled image and pool again.
12. This is required when there are too many details in the given picture and we need only gross level features. For e.g. face identification Vs face recognition
13. Once we have sufficiently pooled information, we have to look for that part of the image that looks like a known pattern! This can be done using dense neural network with sigmoid activation and a softmax function

Pooling Layers

1. Deeper in the convnet, the activation maps represent more and more complex features
2. Those features are sort of agglomeration of microscopic features of earlier layers
3. In the last layer of the network, the filters activate when they see the whole picture. Ref: <https://www.youtube.com/watch?v=AgkfIQ4IGaM>



Pooling Layers

1. Pooling layers provide an approach to down sampling feature maps by summarizing the presence of features in patches of the feature map
2. Down sampling can be achieved with convolutional layers by changing the stride of the convolution across the image. However, a more common approach is to use a pooling layer
3. Two common pooling methods are average pooling and max pooling that summarize the average presence of a feature and the most activated presence of a feature respectively
4. A pooling layer is added after the convolutional layer. Specifically, after a nonlinearity (e.g. ReLU) has been applied to the feature maps output by a convolutional layer

Pooling Layers

5. The pooling layer operates upon each feature map separately to create a new set of the same number of pooled feature maps.
6. The size of the pooling operation or filter is smaller than the size of the feature map; specifically, it is almost always 2×2 pixels applied with a stride of 2 pixels

Pooling Layers Hyperparameters

1. Input volume of filter map – $W1 \times H1 \times D1$
2. Requires Filters size - $F \times F$
3. Stride – S
4. Produces output of volume $W2 \times H2 \times D2$
5. $W2 = (W1 - F)/S + 1$
6. $H2 = (H1 - F)/S + 1$
7. $D2 = D1$

8. Not common to use padding in pooling layer
9. Common to use $F = 2 \times 2$ or 3×3
10. Stride = 2

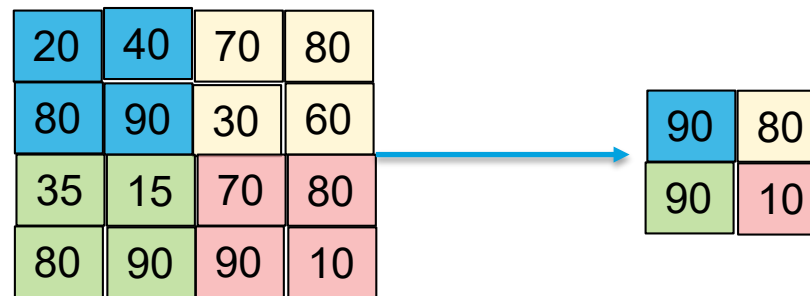
Maxpooling Layer

Maxpooling Layer

1. The most common type of pooling layer is a *maxpooling* layer
2. Maxpooling operation breaks convolutional layer output into smaller patches, often 2x2 pixel areas (pooling layer filter size) with a stride of 2
3. It moves the 2X2 window by the given stride across and down the image.
4. Stride of 1 will create overlapping in patches and a larger stride will miss some pixels in the input feature map
5. Patch size of two with stride of two ensures we do not have overlaps and do not miss any pixels in the input feature map

Maxpooling Layer

6. For each 2x2 patch, a maxpooling layer looks at each value in a patch and selects only the maximum gradient value
7. In the picture below, in the top left section, pooling window will select 90, in the top right it will select 80 and so on
8. As a result of this, we will have max gradients captured from the feature map into another sub-sampled feature map
9. Maxpooling helps feature selection by avoiding weak features and thus helps in dimensionality reduction



Maxpooling

Fully Connected Layer

Fully Connected Layer

1. At this layer, the high level features extracted by the previous conv layer are used for classification
2. This layer basically takes an input volume (whatever the output is of the conv or ReLU or pool layer preceding it) and outputs an N dimensional vector where N is the number of classes that the program has to choose from
3. This layer looks at the output of the previous layer i.e. the activation maps with high level features and determines which features most correlate to a particular class

Conv Layer Interface with Fully Connected Layer - Parameters

1. W_{cf} = Number of weights of a FC Layer which is connected to a Conv Layer
2. B_{cf} = Number of biases of a FC Layer which is connected to a Conv Layer
3. O = Size (width) of the output image of the previous Conv Layer
4. N = Number of kernels in the previous Conv Layer
5. F = Number of neurons in the FC Layer.

$$W_{cf} = O^2 \times N \times F$$

$$B_{cf} = F$$

$$P_{cf} = W_{cf} + B_{cf}$$

W_{cf} - weights from conv to fully connected layers

B_{cf} – Bias from conv to fully connected layers

P_{cf} – Parameters between conv and fully connected layers

Conv layer interface with fully connected layer - parameters

Example: The first fully connected layer of AlexNet is connected to a Conv Layer. For this layer, $O = 6$, $N = 256$ and $F = 4096$. Therefore,

$$W_{cf} = 6^2 \times 256 \times 4096 = 37,748,736$$

$$B_{cf} = 4096$$

$$P_{cf} = W_{cf} + B_{cf} = 37,752,832$$

Fully connected layer with fully connected layer - parameters

1. W_{ff} = Number of weights of a FC Layer which is connected to an FC Layer
2. B_{ff} = Number of biases of a FC Layer which is connected to an FC Layer
3. P_{ff} = Number of parameters of a FC Layer which is connected to an FC Layer
4. F = Number of neurons in the FC Layer
5. F_{-1} = Number of neurons in the previous FC Layer
6.
$$W_{ff} = F_{-1} \times F$$
$$B_{ff} = F$$
$$P_{ff} = W_{ff} + B_{ff}$$
7. In the above equation, $F_{-1} \times F$ is the total number of connection weights from neurons of the previous FC Layer the neurons of the current FC Layer. The total number of biases is the same as the number of neurons (F)

Fully connected layer with fully connected layer - parameters

Example: The last fully connected layer of AlexNet is connected to an FC Layer. For this layer, $F_{-1} = 4096$ and $F = 1000$. Therefore,

$$W_{ff} = 4096 \times 1000 = 4,096,000$$

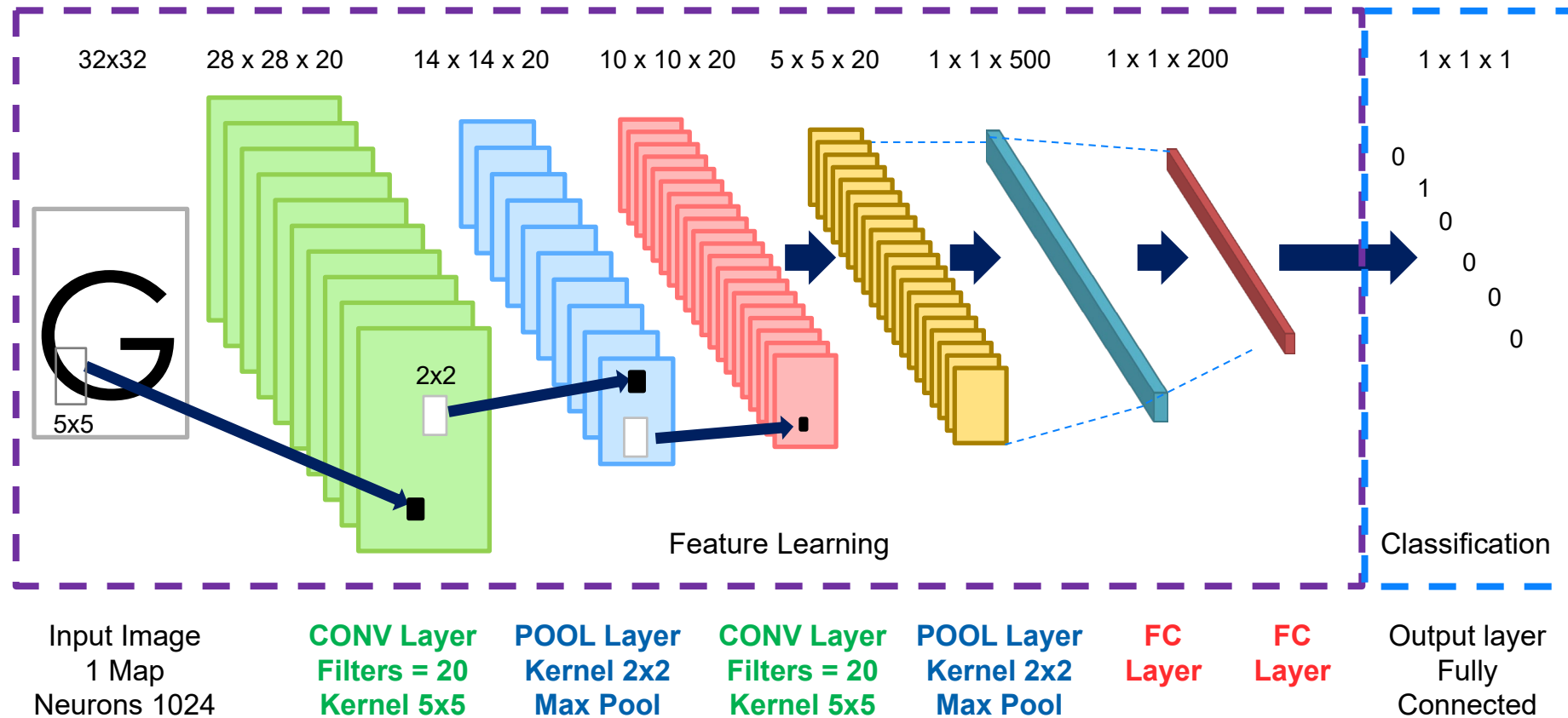
$$B_{ff} = 1,000$$

$$P_{ff} = W_{ff} + B_{ff} = 4,097,000$$

Convolutional Network

1. The different layers in the CNN model may be
 - a. Input layer (the image)
 - b. Convolutional Layer
 - c. Nonlinearity (activation function)
 - d. Pooling Layer
 - e. Dense layer

Convolution Network



CNN in Keras Step_By_Step

Convolutional Network

Generic minimalistic CNN structure

1. `input_shape = (28,28,1)`
2. `num_classes = 10`
3. `model = Sequential()`
4. `model.add(Conv2D(32, kernel_size=(5, 5), strides=(1, 1),
activation='relu', input_shape=input_shape))`
5. `model.add(MaxPooling2D(pool_size=(2, 2), strides=(2,
2)))`
6. `model.add(Conv2D(64, (5, 5), activation='relu'))`
7. `model.add(MaxPooling2D(pool_size=(2,2)))`
8. `model.add(Flatten())`
9. `model.add(Dense(1000, activation='relu'))`
10. `model.add(Dense(num_classes, activation='softmax'))`

Structure of input

28

Number of target
class

Two ways of creating a
CNN – Sequential(),
Functional(). allows you to

add a 2D conv layer. first
argument is number of filters
to learn. Next is kernel size.
Next is strides in the x and y

add a 2D max pooling
layer. specify the size of the
pooling in the x and y

Add another conv and pooling
layers, add as many as
required. In this case two

Flatten the tensor to feed into
the fully connected dense
layer (ANN)

Add the dense layer with
RELU activation function

Add another dense layer with
softmax layer for classification

Convolutional Network

Generic minimalistic CNN structure (Contd..)

```
11. model.compile(loss=keras.losses.categorical_crossentropy,  
optimizer=keras.optimizers.SGD(lr=0.01),  
metrics=['accuracy'])
```

Wrap the model with a loss function, algorithm to use to optimize, hyper parameters to make the algorithms efficient and overall metrics to assess the model while the loss is minimized

```
12. x_train=xtrain.reshape(xtrain.shape[0],28,28,  
1).astype('float32')  
13. x_test=xtest.reshape(xtest.shape[0],28,28,1).  
astype('float32')
```

Reshaping the data as floating point, 28 X 28 grid for the convolutional layer.

The digit 1 signifies the images are in grayscale

Convolutional Network

Generic minimalistic CNN structure (Contd..)

14. `x_train/=255`

15. `x_test/=255` # standardize the data for train
and test to be between 0 -1

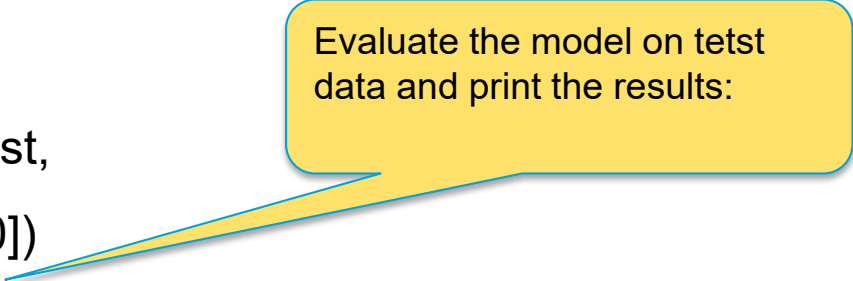
16. `model.fit(x_train, y_train,`
`batch_size=batch_size, epochs=epochs,`
`verbose=1, validation_data=(x_test, y_test),`
`callbacks=[history])`

Standardize the data to scale of 0 to 1

Call the fit function on the training data. Batch size is for applying GD on minibatch, epochs is number of iterations on the training set, verbose =1 to get all details
Validation data for testing

Convolutional Network

```
17.score = model.evaluate(x_test, y_test,  
    verbose=0) print('Test loss:', score[0])  
    print('Test accuracy:', score[1])
```



Evaluate the model on test data and print the results:

