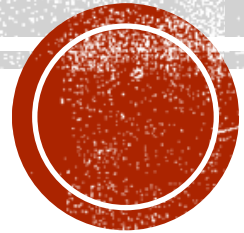
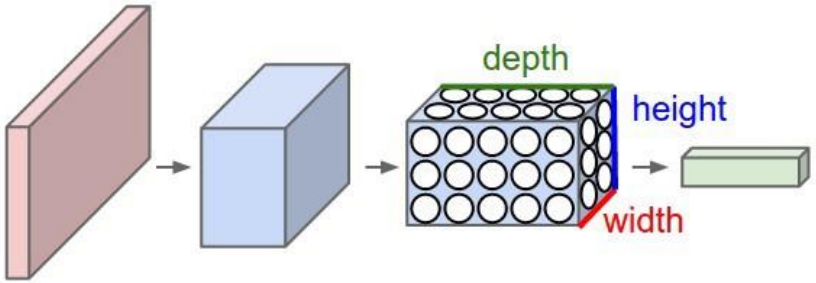
CNN CONCEPTS

Dr. Brian Mc Ginley



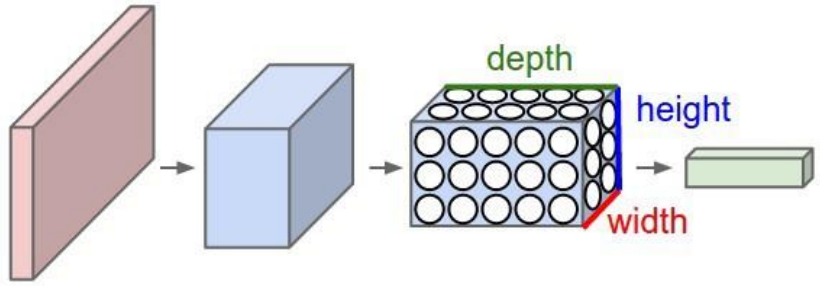
LAYERS TO BUILD CONVNETS/CNNS

- A ConvNet is made up of Layers. Every Layer has a simple API: It transforms an input 3D volume to an output 3D volume with some differentiable function that may or may not have parameters.
- Resemble early part of human visual cortex.
- Three main types of layers to build CNNs:
  1. Convolutional Layer
  2. Pooling Layer
  3. Fully-Connected Layer

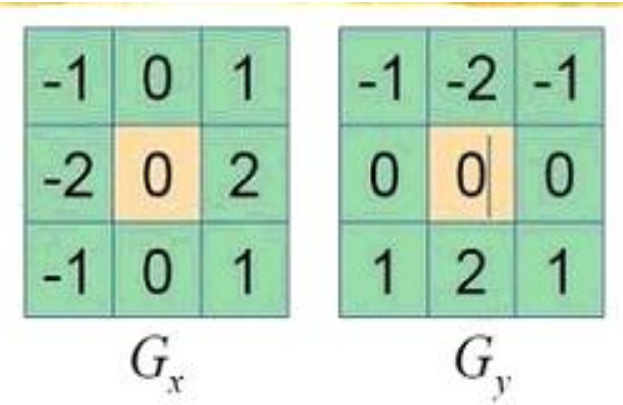


CNN CONVOLUTION LAYER

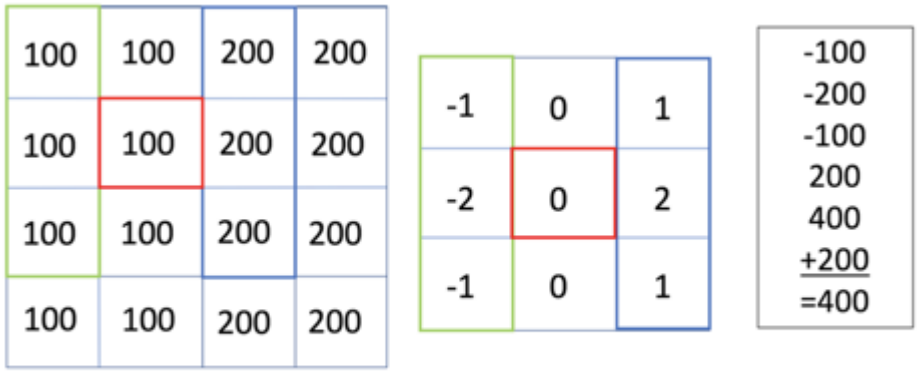
- Use a small filter (only a few pixels wide) and slide it over the image applying it in each different position.
- The weights of the filter are still learned but they learn the best response when every position of the image is taken into account.
- This means that the learned weights are shared by all positions.
- One filter wouldn't do the job because it would only find one thing in the image.
- So, we can have many filters that all learn to look for different things in the image and because of convolution they learn to find them no matter where they are in the image.



LINEAR OPERATION



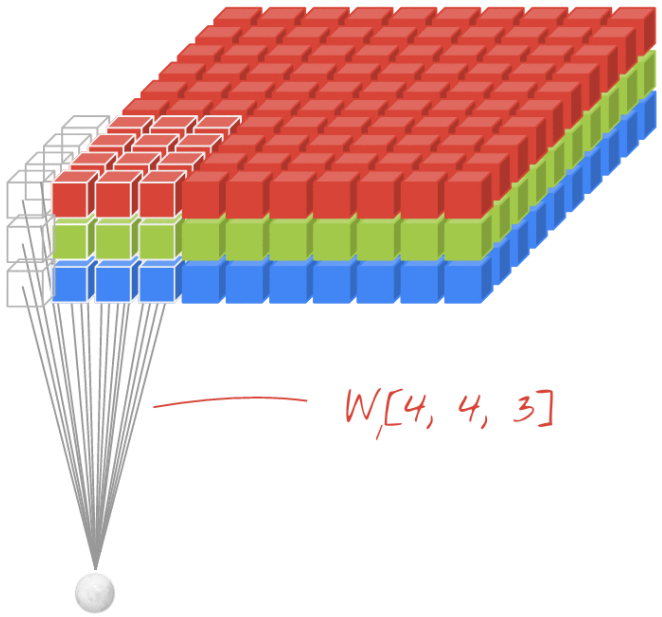
- Convolution is a linear operation and in CNNs it's linear at a layer and then passes through a non-linear activation function (act).
- It then passes to the next layer of convolution and non-linearity. Some refer to the idea of many layers sliding across the image.
- In this case the linear operation of convolution has no real meaning.
- FCNs are Fully Convolutional Networks that allow for the entire network to "slide" across the image.



Kernel Convolution: The bigger the value at the end, the more noticeable the edge will be.

SLIDING OPERATION

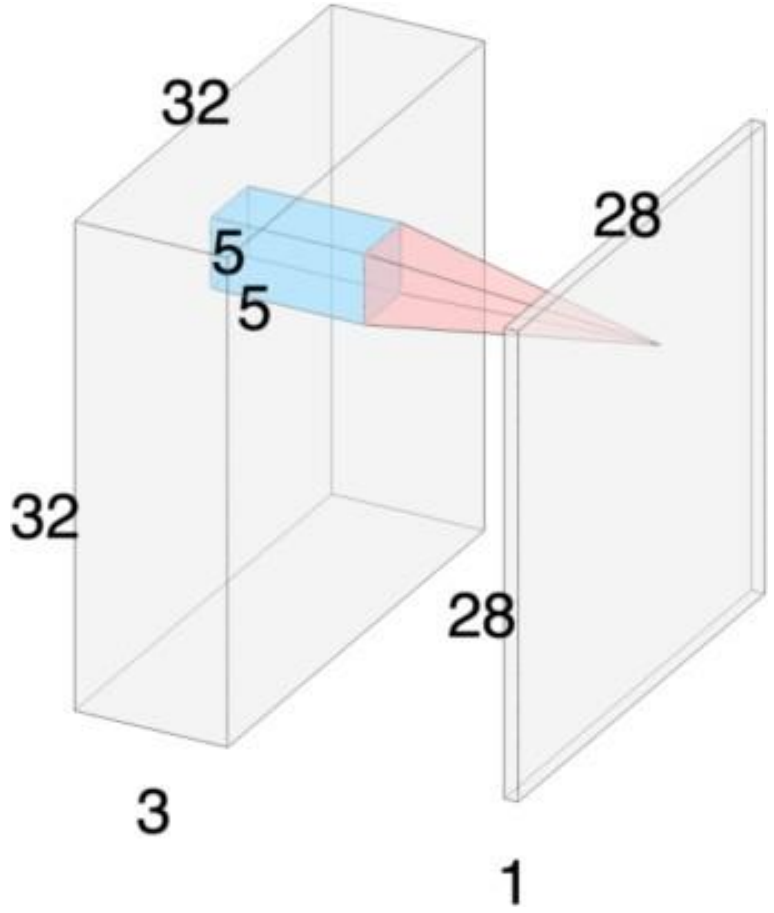
- Martin Gorner gif shows how depth is handled
  - FjvuN.gif (1090×780)



11	44	66	48	81
14	76	64	62	74
16	74	66	24	86
41	16	45	61	84
15	34	60	44	90

INDIVIDUAL FILTER

- Here we look at an example where the input is a 32 × 32 × 3 image.
- The filter is a 5 × 5 but must be the same depth as the input so it is a 5 × 5 × 3.
- We slide across the width and height, but we always use the full depth.
- So, our filter in this case will have 5 × 5 × 3 + 1 (bias) = 76 weights/parameters.
- It will produce a 28 × 28 × 1 activation map. You can view the activation like an image.



WHY 28 X 28

- Stride
- Zero Padding

11	11	11	44	66	48	81	81	81
11	11	11	44	66	48	81	81	81
11	11	11	44	66	48	81	81	81
14	14	14	76	64	62	74	74	74
16	16	16	74	66	24	86	86	86
41	41	41	16	45	61	84	84	84
15	15	15	34	60	44	90	90	90
15	15	15	34	60	44	90	90	90
15	15	15	34	60	44	90	90	90

76	14	14	44	66	24	74	74	74
44	11	11	44	66	48	81	81	48
44	11	11	44	66	48	81	81	48
76	14	14	76	64	62	74	74	62
74	16	16	74	66	24	86	86	24
16	41	41	16	45	61	84	84	61
34	15	15	34	60	44	90	90	44
34	15	15	34	60	44	90	90	90
16	41	41	16	45	61	84	84	61

ZERO PADDING

- So that we can get a value for every pixel value with a 3 × 3 filter - going off the edge

0	0	0	0	0	0	0
0	11	44	66	48	81	0
0	14	76	64	62	74	0
0	16	74	66	24	86	0
0	41	16	45	61	84	0
0	15	34	60	44	90	0
0	0	0	0	0	0	0

ZERO PADDING

- How much zero-padding to add for a 5x5 kernel?

11	11	11	44	66	48	81	81	81
11	11	11	44	66	48	81	81	81
11	11	11	44	66	48	81	81	81
14	14	14	76	64	62	74	74	74
16	16	16	74	66	24	86	86	86
41	41	41	16	45	61	84	84	84
15	15	15	34	60	44	90	90	90
15	15	15	34	60	44	90	90	90
15	15	15	34	60	44	90	90	90

76	14	14	44	66	24	74	74	74
44	11	11	44	66	48	81	81	48
44	11	11	44	66	48	81	81	48
76	14	14	76	64	62	74	74	62
74	16	16	74	66	24	86	86	24
16	41	41	16	45	61	84	84	61
34	15	15	34	60	44	90	90	44
34	15	15	34	60	44	90	90	90
16	41	41	16	45	61	84	84	61

STRIDE

- Stride refers to the number of steps the convolutional filter moves (or "slides") across the input feature map during convolution. Stride controls how much the kernel/receptive field overlaps as it processes the input.

11	11	11	44	66	48	81	81	81
11	11	11	44	66	48	81	81	81
11	11	11	44	66	48	81	81	81
14	14	14	76	64	62	74	74	74
16	16	16	74	66	24	86	86	86
41	41	41	16	45	61	84	84	84
15	15	15	34	60	44	90	90	90
15	15	15	34	60	44	90	90	90
15	15	15	34	60	44	90	90	90

76	14	14	44	66	24	74	74	74
44	11	11	44	66	48	81	81	48
44	11	11	44	66	48	81	81	48
76	14	14	76	64	62	74	74	62
74	16	16	74	66	24	86	86	24
16	41	41	16	45	61	84	84	61
34	15	15	34	60	44	90	90	44
34	15	15	34	60	44	90	90	90
16	41	41	16	45	61	84	84	61

STRIDE

- Calculations for output layer size

$$W_l = \frac{W_{l-1} - F_l + 2P_{l-1}}{S_l} + 1$$

$$H_l = \frac{H_{l-1} - F_l + 2P_{l-1}}{S_l} + 1$$

$$D_l = K_l$$

- where *l* is the layer number.
- W and H are the width and height.
- K is the number of filters
- F is the spatial extent of the filters e.g. 3 for a 3 × 3.
- S is the stride
- P is the amount of padding

STRIDE

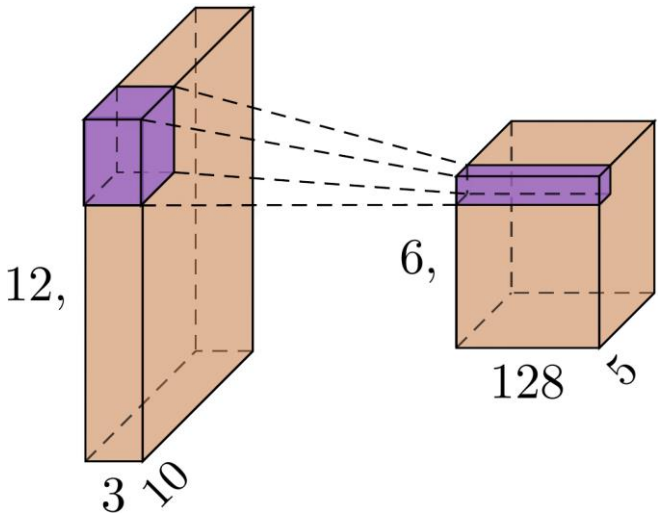
- Calculations for output layer size

$$W_l = \frac{W_{l-1} - F_l + 2P_{l-1}}{S_l} + 1$$

$$H_l = \frac{H_{l-1} - F_l + 2P_{l-1}}{S_l} + 1$$

$$D_l = K_l$$

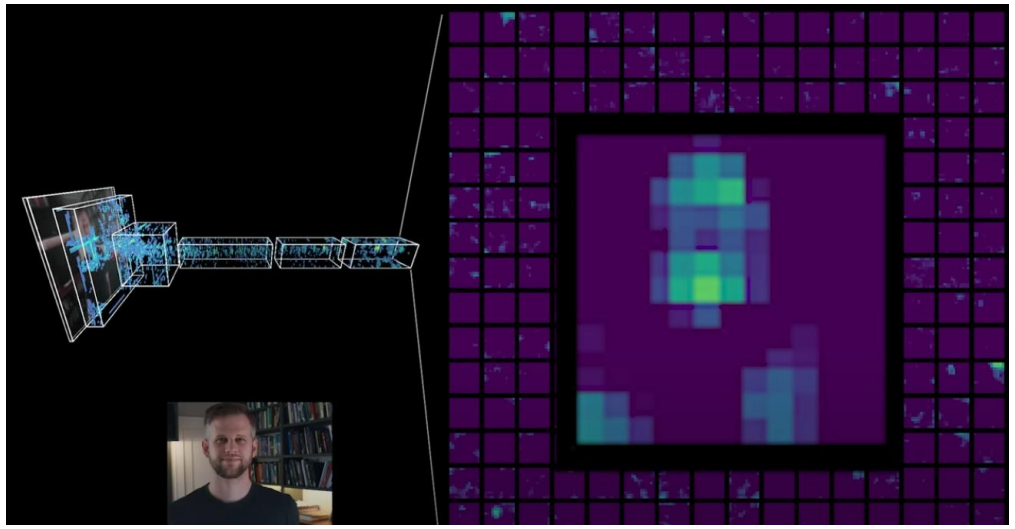
- where *l* is the layer number.
- W and H are the width and height.
- K is the number of filters
- F is the spatial extent of the filters e.g. 3 for a 3 × 3.
- S is the stride
- P is the amount of padding



From: Creating a Vectorized Convolution layer in NumPy | Cameron White - SWE

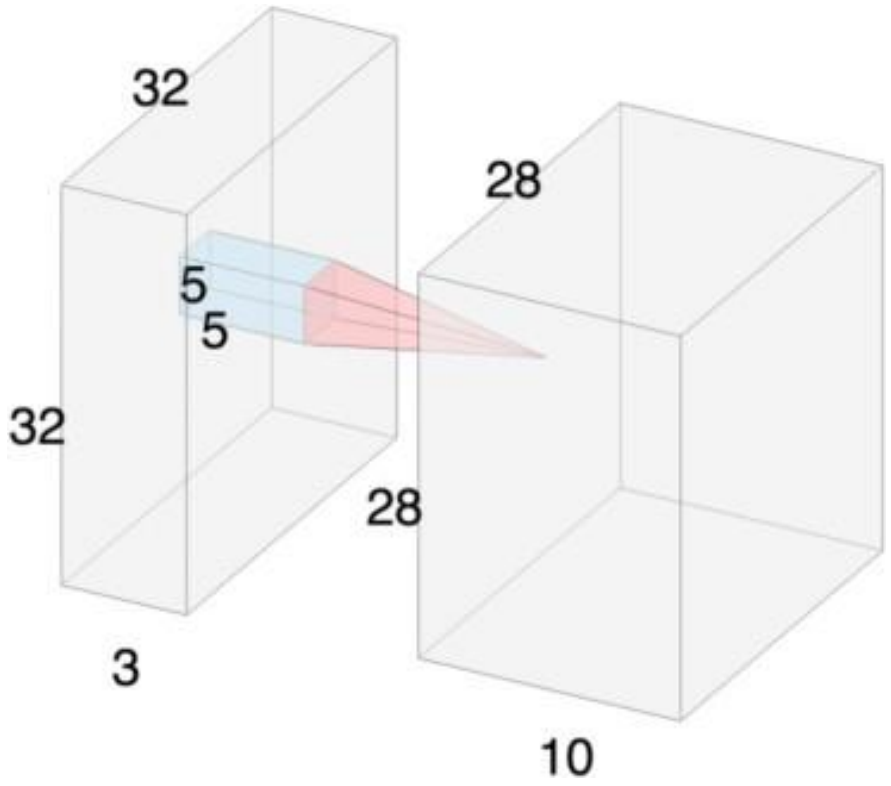
ACTIVATION MAP

- We have now a new representation of our original colour image, this change in representation has been brought about by the $5 \times 5 \times 3$ filter.
- The representation is the similarity at each point between the learned filter and the underlying image.
- The values in the activation map will not naturally have values that make it easy to display as an image i.e. $\in [0, 255]$ so they will normally have to be scaled for viewing on a computer.
- We can also look at the output before or after the non-linear activation to see what effect the activation function has on the map.

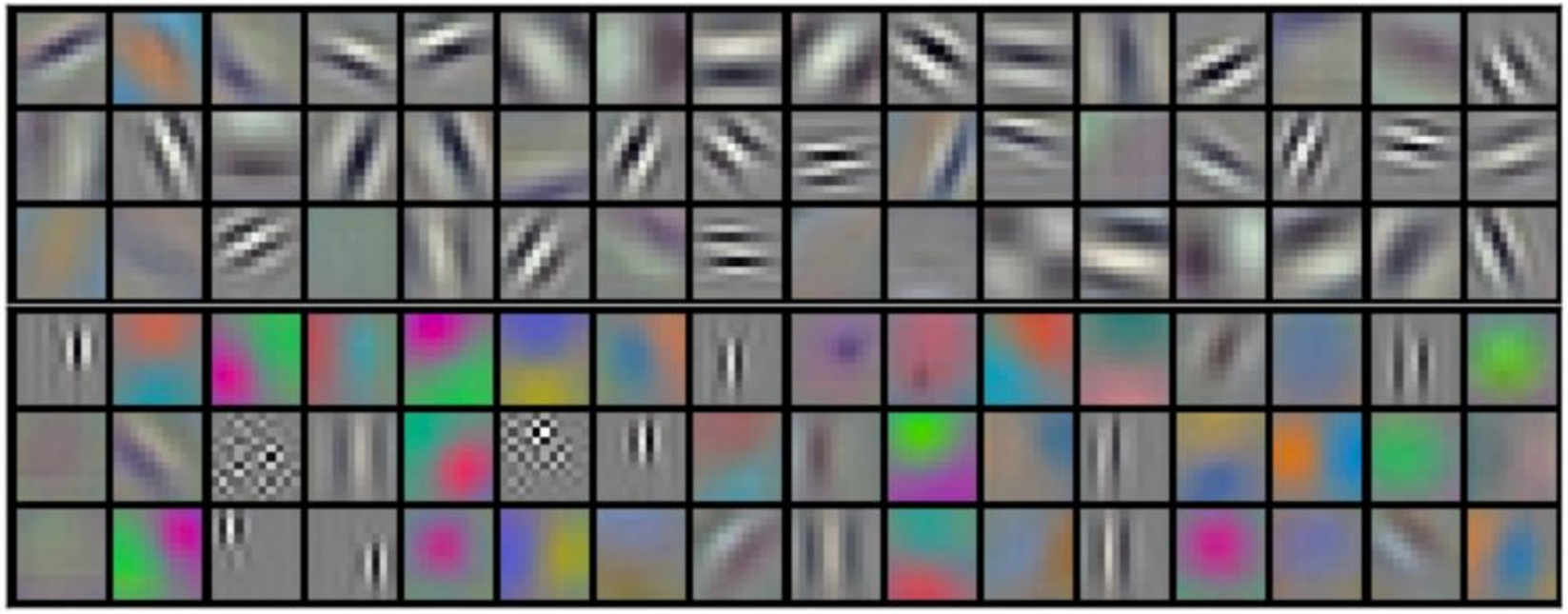


MULTIPLE ACTIVATION MAPS

- A single filter can only learn to look for one thing.
- Obviously, we want to learn to look for many things, therefore we will have many filters.
- Let's assume we decide (a hyperparameter) to have 10 filters of $5 \times 5 \times 3$ then we will get ten activation maps out.
- These ten activation maps can then be thought of as the new representation of the data.
- Instead of a 3-channel image it is now a 10-channel image.

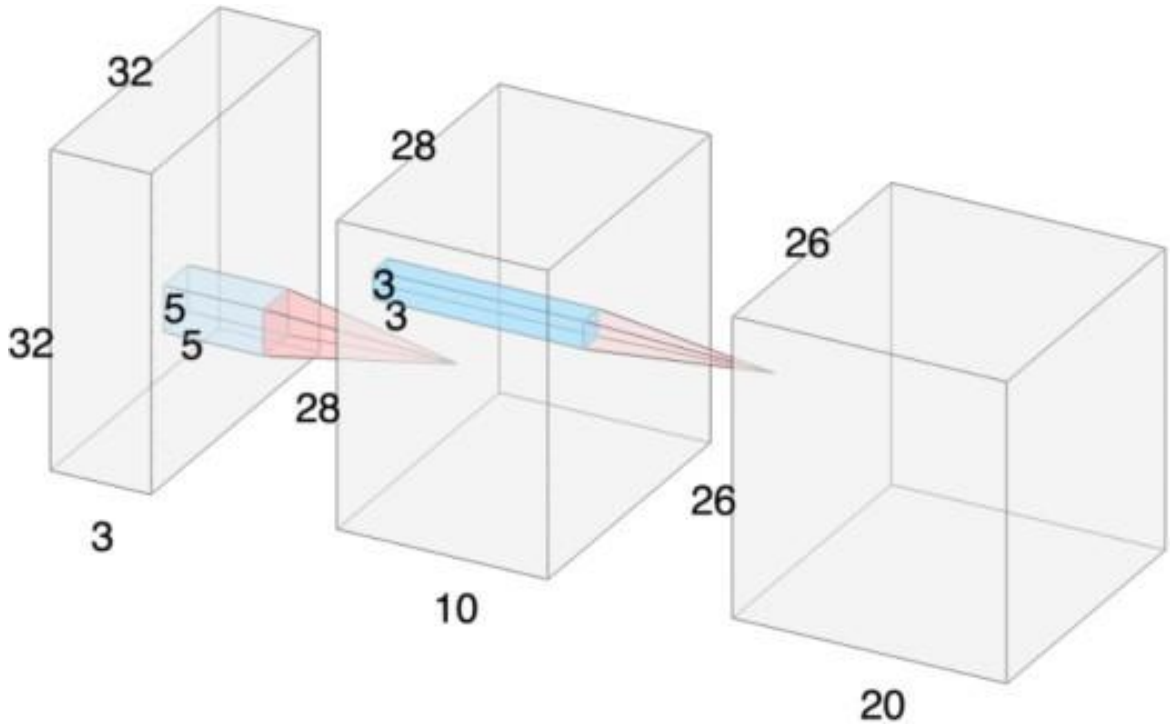


ALEXNET FEATURES



NEXT LAYER

- 3 × 3 × 10 filters and using 20 of them.

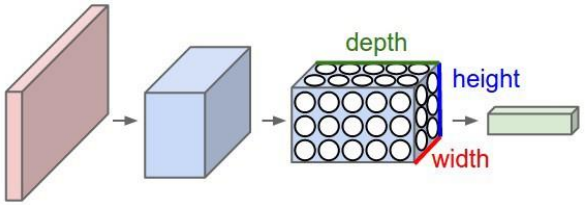


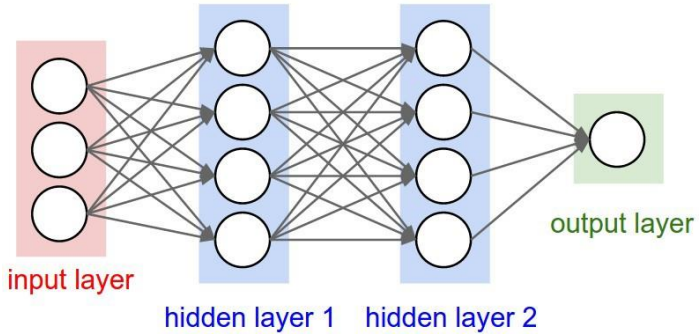
POPULAR FILTER SIZE

- n general filter sizes tend to be 3 × 3, 5 × 5.
- 7 × 7 and higher are used on occasion but are much more rare.
- 1 × 1 is also useful.
- 1 × 1 may seem strange as there is no spread from the point.
- Indeed, in ordinary image processing this would be a point operation only. However, we should remember that the convolution applies to the full depth of the previous layer and as such a 1 × 1 filter is another way of combining a weighted average of the activation maps in the previous layer.
- So, if we would expect a particular object to have high activations for four of the previous filter maps then these will get a high weighting for a 1 × 1 filter that is looking for this object.

CNN ON CIFAR-10 DATASET

- CIFAR-10 originates from Krishevsky and Hinton
- Dataset with 60,000 32 × 32 × 3 images divided equally into 10 classes
- Using a CNN we will see the neurons in a layer will only be connected to a small region of the layer before it, instead of all the neurons in a fully-connected manner. The final output layer would have dimensions 1 × 1 × 10 as we are reducing the full image into a single vector of class scores along the depth dimension. (Credit CS231n Stanford)





- Right: A regular 3-layer Neural Network. Left: A ConvNet arranges its neurons in three dimensions (width, height, depth), as visualized in one of the layers. Every layer of a ConvNet transforms the 3D input volume to a 3D output volume of neuron activations. In this example, the red input layer holds the image, so its width and height would be the dimensions of the image, and the depth would be 3 (Red, Green, Blue channels).

CNN EXAMPLE

- For the CIFAR-10 dataset example it could have a CNN architecture as follows [INPUT - CONV - RELU – POOL - FC]
  - INPUT , holds the raw pixel values: 32 × 32 × 3
  - CONV , compute the output of neurons that are connected to local regions in the input. This may result in volume such as [32x32x12] if we decided to use 12 filters.
  - RELU , the activation function. Leaves the size of the volume unchanged [32x32x12]
  - POOL , performs a downsampling operation along the spatial dimensions. [16x16x12]
  - FC , compute the class scores to [1x1x10] along the 10 categories of CIFAR-10. Each neuron is connected to all the ones in the previous volume.

LENET

- The LeNet (1989) by Yann leCun was the first successful use of a convolutional neural network. It was successfully deployed for use in postal services for reading hand written postal codes. It would be a while before they could be used at scale.
- Convolutional Network Demo from 1989 - YouTube

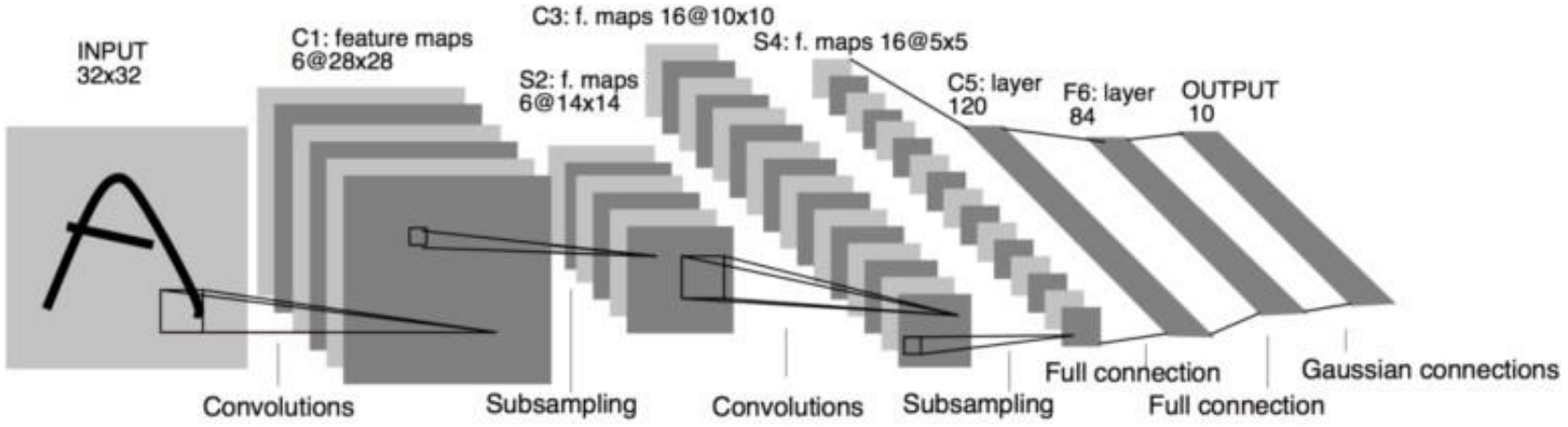
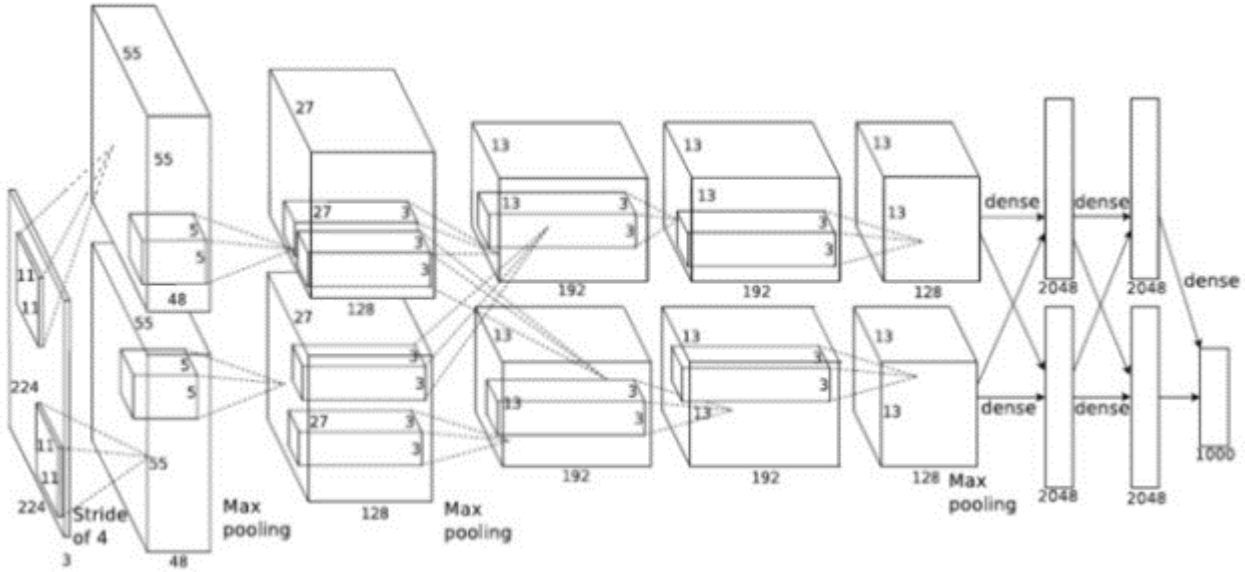


Figure: LeNet Architecture

ALEXNET

- AlexNet (2012) was the break through for CNNs. Alex Krishevsky et al. created an network that won the ImageNet challenge by a huge margin. It has not been won since by anything other than a CNN.
- The moment we stopped understanding AI [AlexNet]

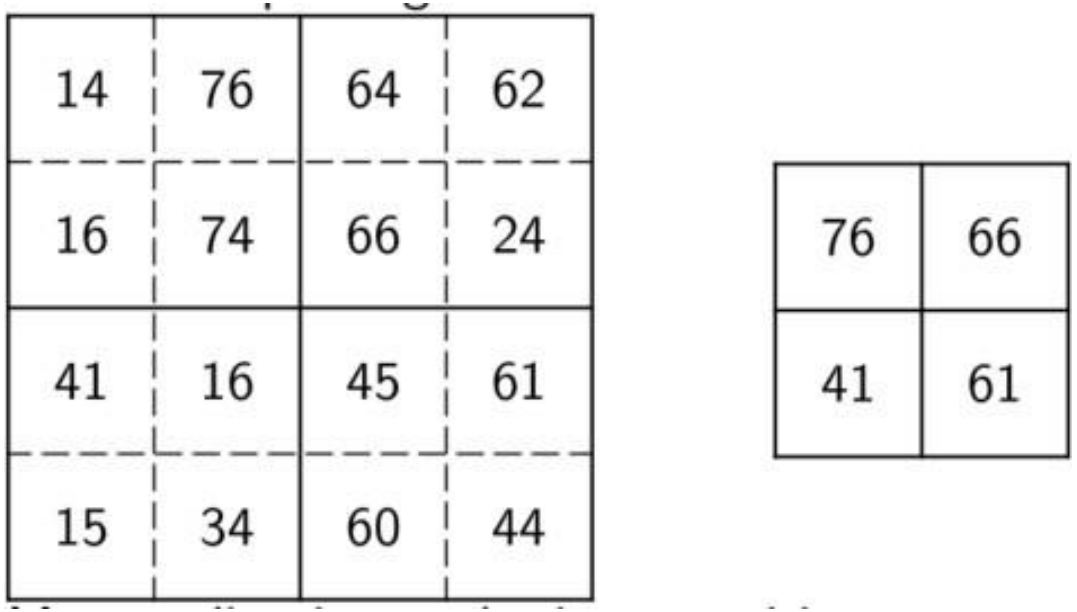


POOLING

- CNNs can do representational learning by extracting features.
- CNNs also add the power of dimensional reduction (like PCA which I mentioned before).
- Each layer can learn a new and improved representation but it can also learn/perform a reduction in dimensionality.
- The most straightforward way to do this is with a pooling layer.
- Here we take a section of pixels and reduce the number based on some criteria and by a pre-determined amount.

MAX-POOLING

- 2 × 2 max-pooling with a stride of 2
- Max-pooling is popular in recognition type networks



AVERAGE-POOLING

- 2 × 2 average-pooling with a stride of 2
- Average Pooling is popular in generative type networks.

14	76	64	62
16	74	66	24
41	16	45	61
15	34	60	44

45	54
26.5	52.5

CONVOLUTIONS WITH STRIDE GREATER THAN 1

- To some extent, the idea of simply using convolutions with a stride of greater than 1 have become popular for downscaling (as alternative to pooling).
- In generative networks there is sometimes a need to upscale and this can be achieved with fractional convolutions, sometimes called de-convolution or transpose convolutions.

DIMENSIONALITY REDUCTION

- Whatever the method of down-sampling, they all have the potential to learn a dimensionality reduction.
- Let's take the typical ImageNet Dataset Scenario.
- Images of approx $224 \times 224 \times 3$ must eventually end up giving a prediction of 1 out of 1000 class labels.
- This means that we want to reduce the dimensions from 150528 to 1000.
- It obviously makes sense to reduce the dimensions progressively while also changing the representation.
- So, we see a general trend towards Conv→Act→Pool→Conv→Act→Pool→Conv→Act→Pool etc.
- Sometimes Conv→Act→Conv→Act→Pool→Conv→Act→Conv→Act→Pool

FULLY CONNECTED LAYERS

- We previously discussed the idea of using hand engineered features to change the representation of data to make it easier to use in a standard machine learning algorithm.
- The convolutional layers and pooling layers are also used to bring the dimensionality and representation to a form that allows it to be input to some familiar machine learning mechanisms, such as logistic-regression/softmax function or SVM.
- To do this, we normally use a fully-connected layer (dense layer) after the convolutional layers and before the output activation.
- So we will normally get something like:
  Conv→Act→Conv→Act→Pool→Conv→Act→Conv→Act→Pool→FC→SVM
- We usually have a multi-class classifier on the end (e.g. 1 of 1000 classes) but it could also be a binary classifier, depending on the application.

FULL CONV NET