Convolutional Neural Networks

CSE 4311 – Neural Networks and Deep Learning

Vassilis Athitsos
Computer Science and Engineering Department
University of Texas at Arlington

Design Choices

- Some important design choices in a neural network:
  - Number of layers.
    - Too many: slow to train, risk of overfitting.

Design Choices

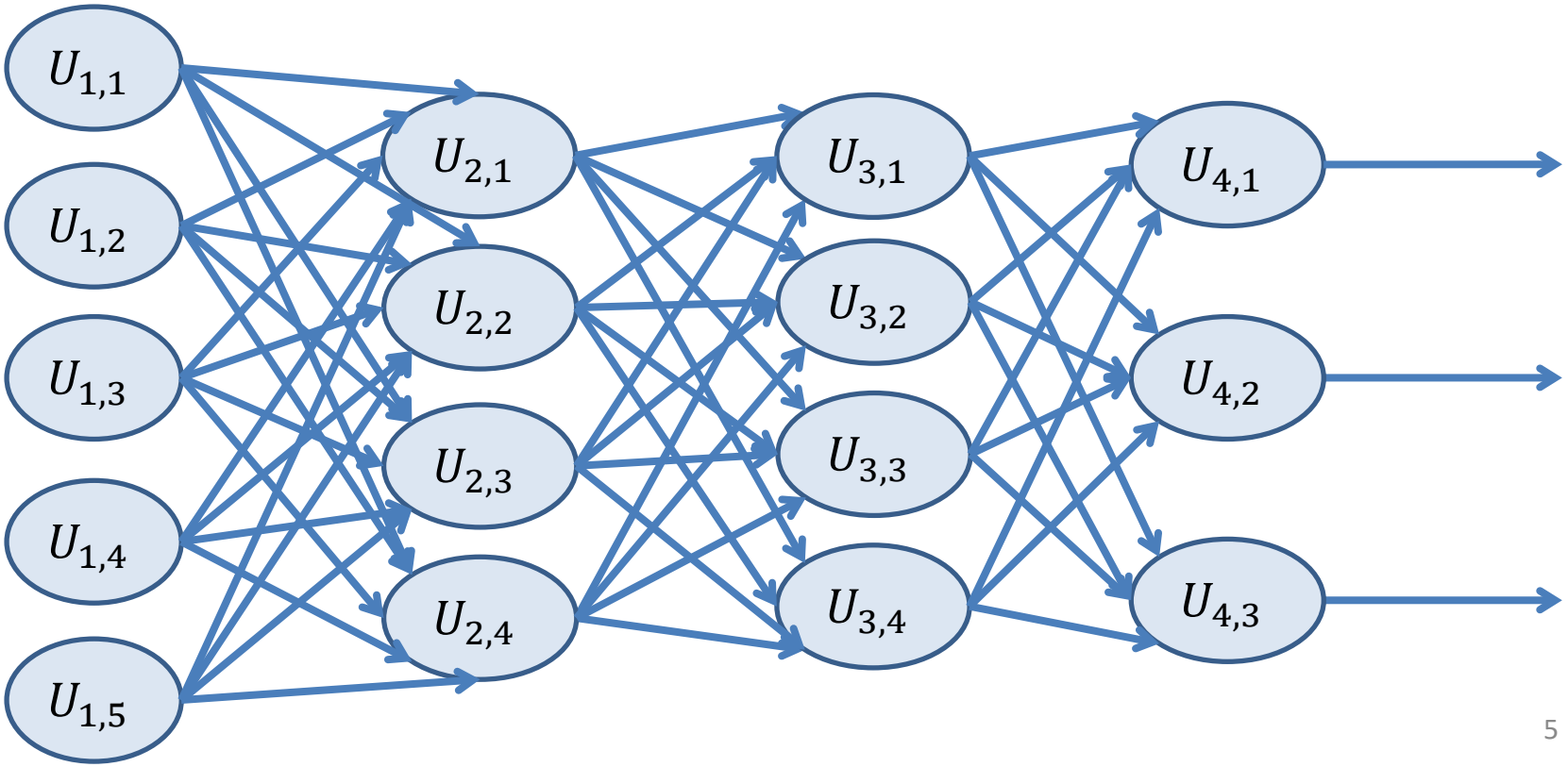
- Some important design choices in a neural network:
  - Number of layers.
    - Too many: slow to train, risk of overfitting.
    - Too few: may be too simple to learn an accurate model.

Design Choices

- Some important design choices in a neural network:
  - Number of layers.
    - Too many: slow to train, risk of overfitting.
    - Too few: may be too simple to learn an accurate model.
  - Number of units per layer.
    - We have a separate choice for each layer.
    - Too many: slow to train, dangers of overfitting.
    - Too few: may be too simple to learn an accurate model.
    - We have no choice for input layer (number of units = number of dimensions of input vectors) and output layer (number of units = number of classes).
  - Connectivity: what outputs are connected to what inputs?

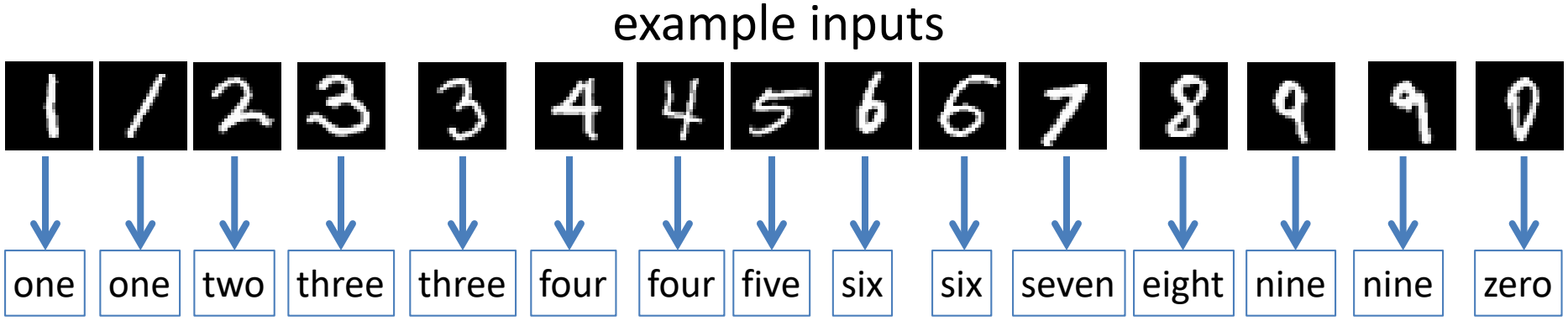
Fully Connected Layer

- A fully connected layer is a layer where every unit has as inputs ALL the outputs of the previous layer.
- In the picture, all layers (except the first one) are fully connected.

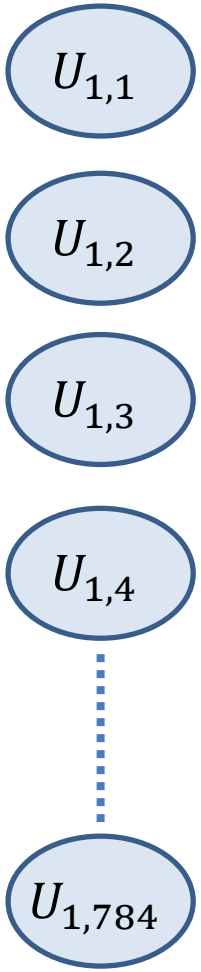


Example of Fully Connected Layer

- Consider the MNIST dataset.
- Each image is of size 28x28.
  - Each image is a 784-dimensional vector.

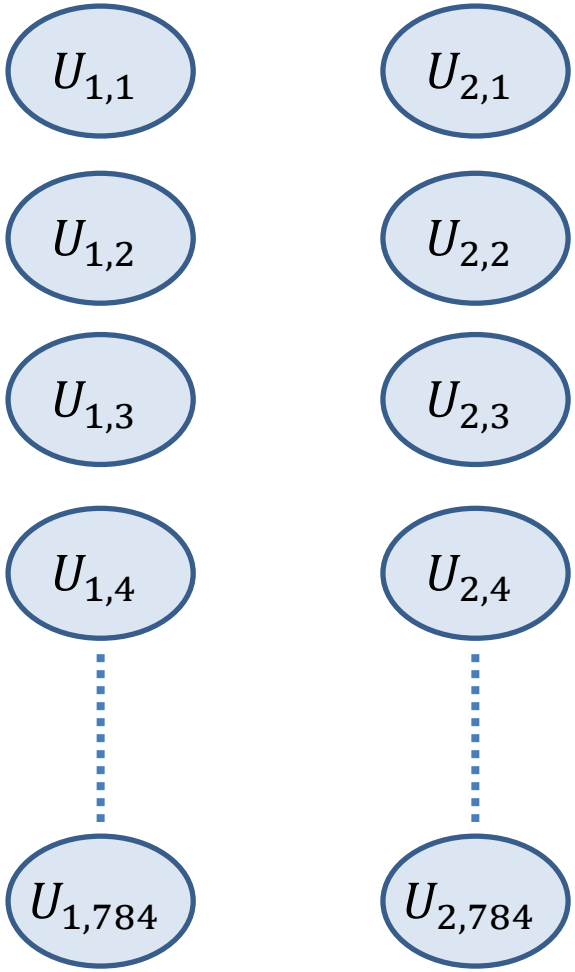


Example of Fully Connected Layer



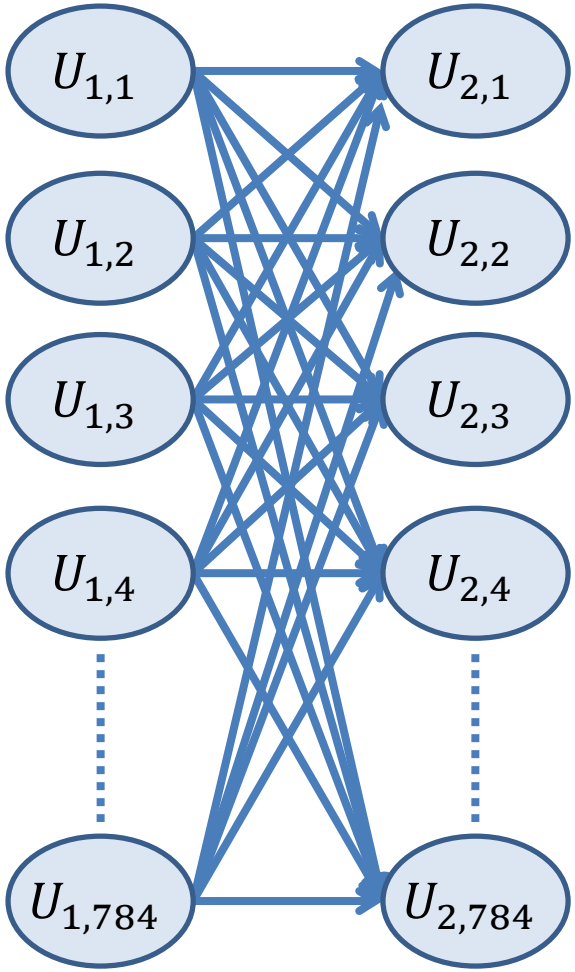
- Thus, the input layer has 784 units.

Example of Fully Connected Layer



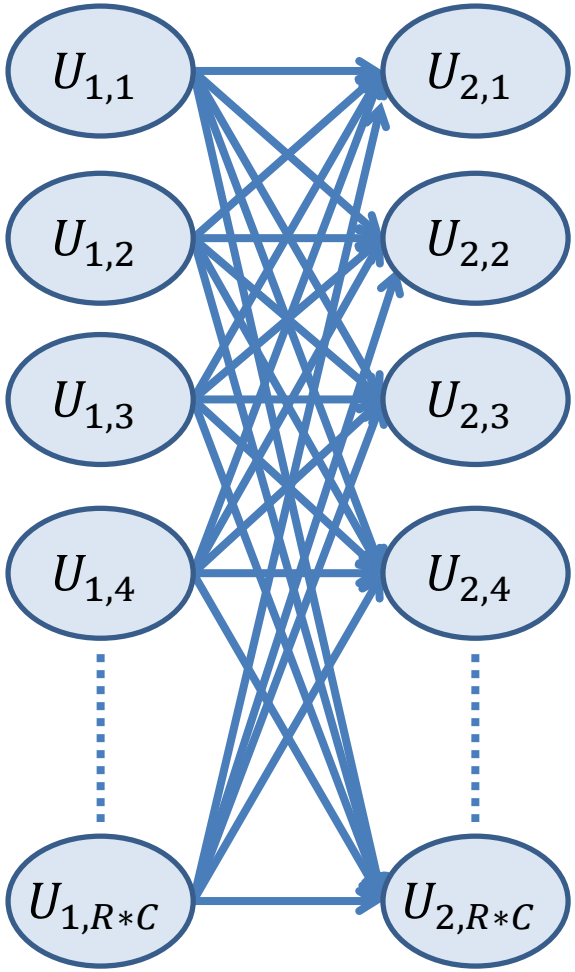
- Thus, the input layer has 784 units.
- What about the next layer?
  - How many units? There is no standard way to decide. Let's pick 784, to match the input layer.

Example of Fully Connected Layer



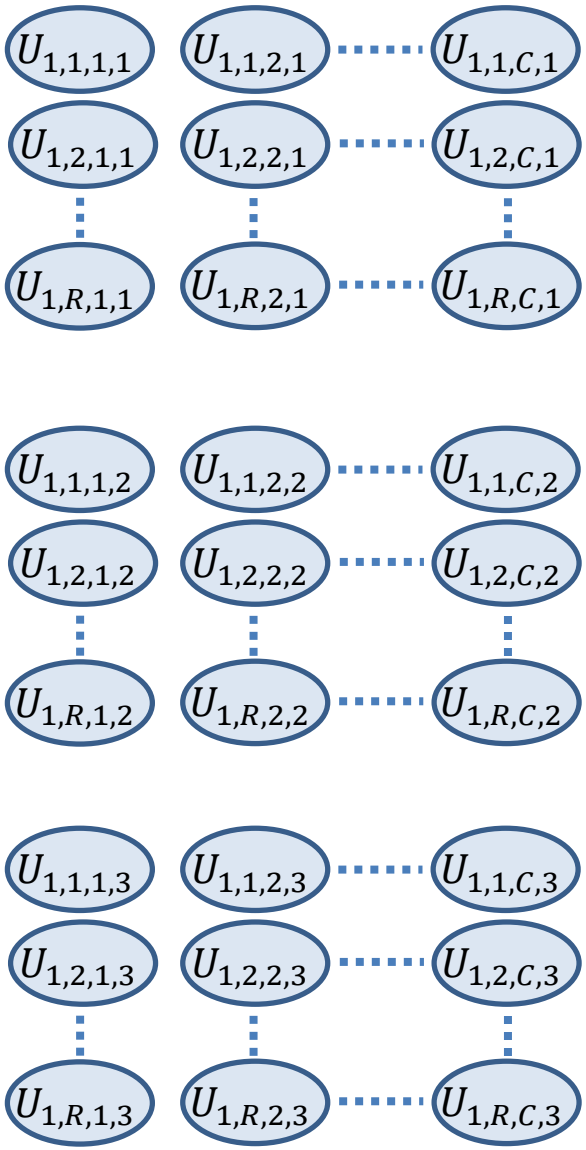
- Thus, the input layer has 784 units.
- What about the next layer?
  - How many units? There is no standard way to decide. Let's pick 784, to match the input layer.
- Connectivity? Let's make it fully connected.
- How many weights do we need to learn?
  - 784*784 = 614,656 weights.

Example of Fully Connected Layer



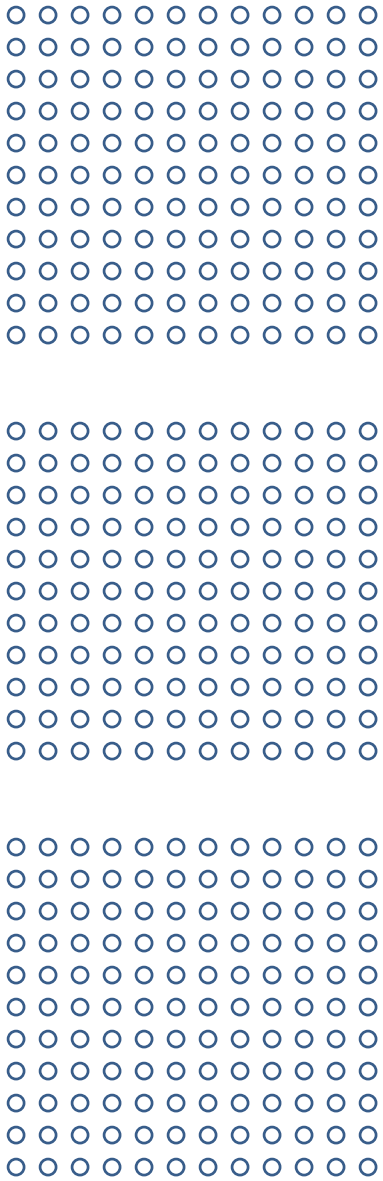
- For larger images, let's say 400x500 pixels, the number of weights could be enormous.
  - It would be 400*500*400*500, which is 40 billion weights.
- Such a layer would require:
  - Large storage to store the weights.
  - Long time to train.
  - Long time to classify an input image.

Visualizing the Input Layer in 3D



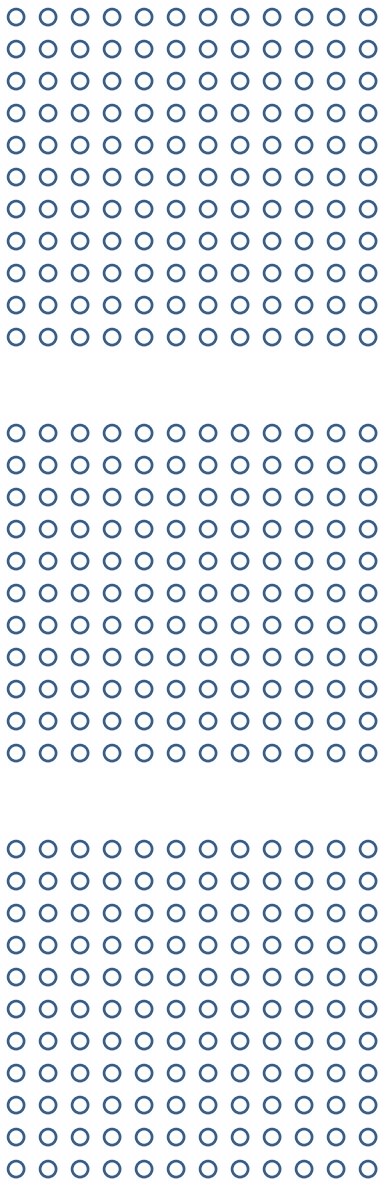
- To describe a convolutional layer, we need to change our notation a bit.
- The input layer will be a 3D array, to better reflect the fact that the input is an RGB image (of size R*C*3).
- So, what you see on the left is a single layer: the input layer, with R*C*3 units.
  - Unit $P_{1,i,j,k}$ is the unit corresponding to layer 1 (the input layer), row i, column j, channel k.

Visualizing the Input Layer in 3D



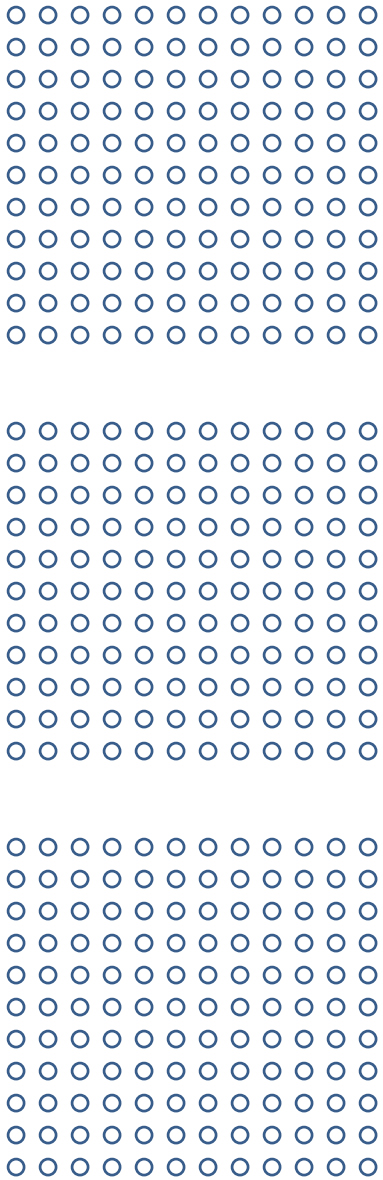
- Sometimes it is useful to visualize a larger number of units.
- In that case, we just show them as dots, or little circles, or something of that sort…
- So, the image on the left still shows the input layer:
  - R*C*3 units, corresponding to an RGB image with R rows and C columns.

Visualizing the Input Layer in 3D



- Before we define convolutional layers in their general forms, we will see a specific example.
- In this example, the convolutional layer will be the second layer, right after the input layer.
- The convolutional layer is based on (but not identical to) a convolution operation.

Example of a Convolutional Layer

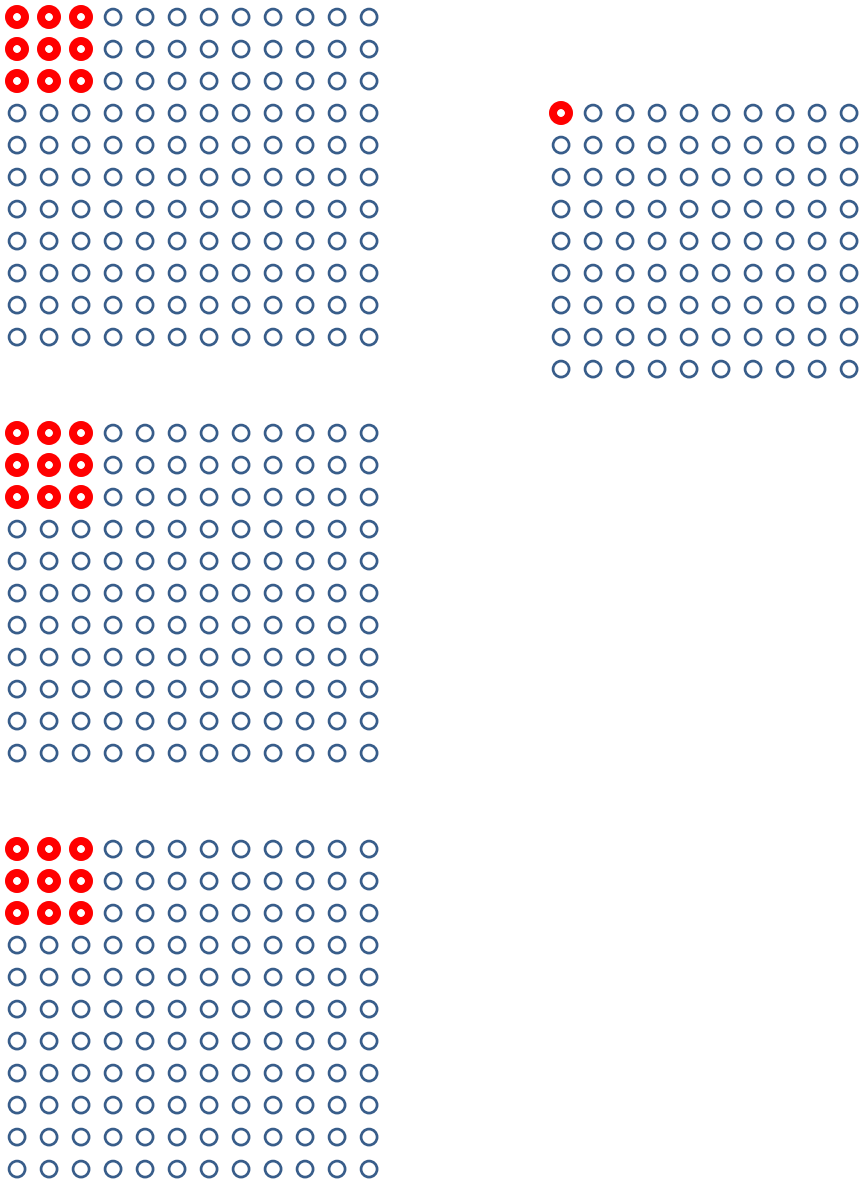


- A convolution is defined by an M*N*C array of weights.
- In our example, we will use M=3 and N=3.
- Let’s see what happens with a single such filter.

Example of a Convolutional Layer

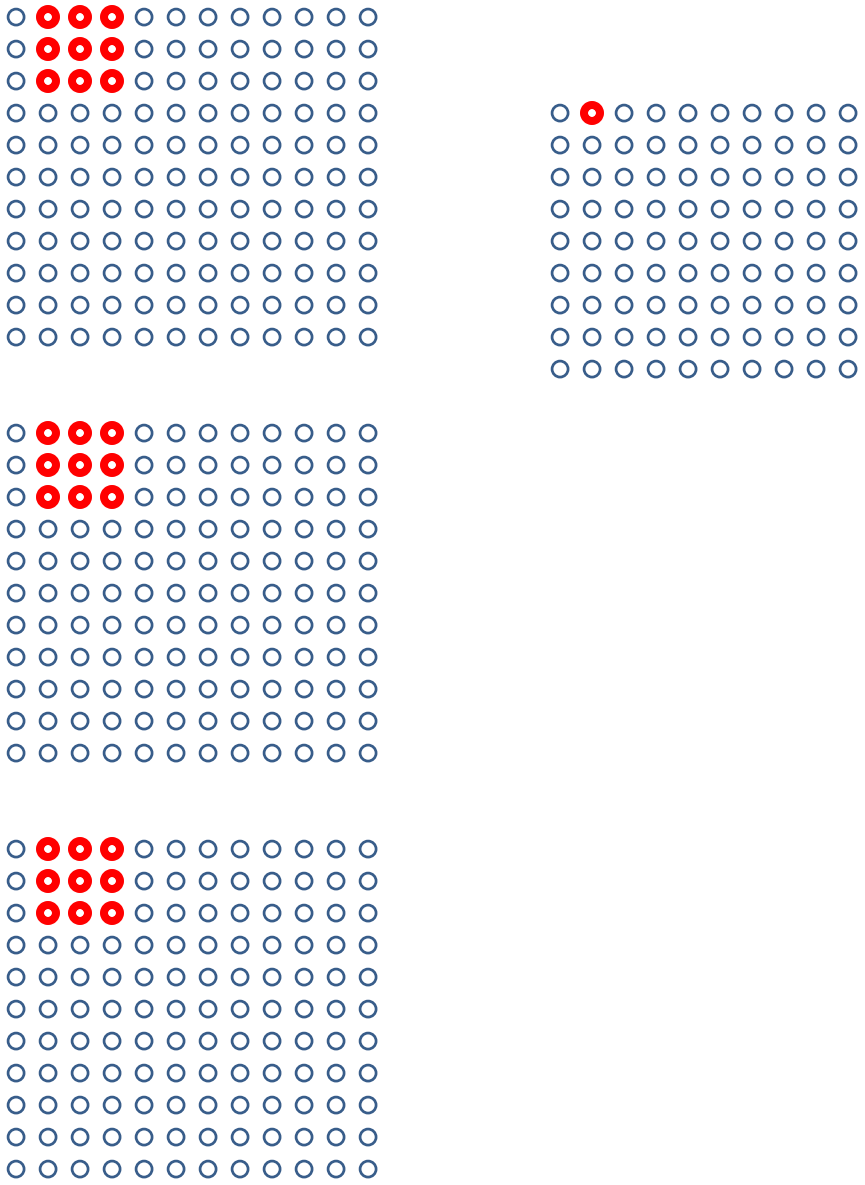
- We are using a 3*3*3 filter.
- In the result, we will throw away boundary values, where part of the filter does not align with any image pixels.
- So, if the image has R rows and C columns, the result will have R-2 rows and C-2 columns.

Example of a Convolutional Layer



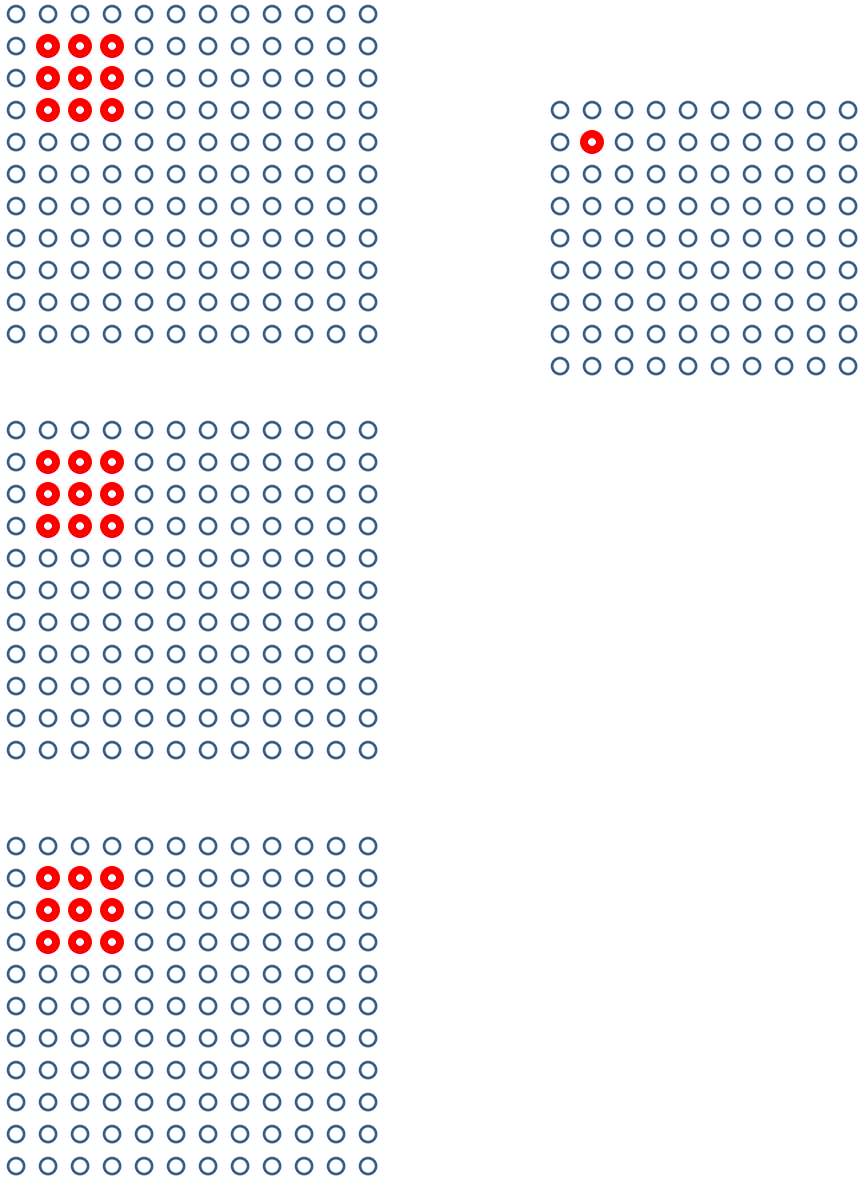
- The convolution result at location (1,1) corresponds to matching the 3x3x3 filter values with the highlighted values in the input layer.

Example of a Convolutional Layer



- Similarly, the convolution result at location (1,2) corresponds to matching the 3x3x3 filter values with the highlighted values in the input layer.

Example of a Convolutional Layer



- Similarly, the convolution result at location (2,2) corresponds to matching the 3x3x3 filter values with the highlighted values in the input layer.

Example With Numbers

Input image: 7x9x3

58	17	75	9	51	54	21	18	91
6	65	19	93	52	36	31	23	98
24	74	69	78	82	94	48	44	44
36	65	19	49	80	88	24	32	12
83	46	37	44	65	56	85	93	26
2	55	63	45	38	63	20	44	41
5	30	79	31	82	59	23	19	60

53	99	34	78	4	86	49	69	73
24	4	68	72	75	81	17	5	15
49	89	14	91	51	58	98	8	66
63	92	73	90	48	19	72	53	52
68	80	11	34	91	24	51	10	98
40	10	66	70	61	89	48	82	65
37	27	50	20	62	3	6	82	81

38	74	31	13	20	38	39	99	82
20	27	71	99	37	20	59	74	27
49	43	67	18	47	43	26	35	60
34	55	54	4	99	49	30	59	3
96	95	70	57	16	13	62	11	43
93	42	67	89	86	59	27	91	32
6	99	18	67	65	23	83	88	17

Filter: 3x3x3

5	4	1
2	1	2
6	9	7

7	2	5
6	1	5
9	3	6

9	8	5
6	4	3
4	3	2

Result: ??x??x??

Example With Numbers

Input image: 7x9x3

58	17	75	9	51	54	21	18	91
6	65	19	93	52	36	31	23	98
24	74	69	78	82	94	48	44	44
36	65	19	49	80	88	24	32	12
83	46	37	44	65	56	85	93	26
2	55	63	45	38	63	20	44	41
5	30	79	31	82	59	23	19	60

53	99	34	78	4	86	49	69	73
24	4	68	72	75	81	17	5	15
49	89	14	91	51	58	98	8	66
63	92	73	90	48	19	72	53	52
68	80	11	34	91	24	51	10	98
40	10	66	70	61	89	48	82	65
37	27	50	20	62	3	6	82	81

38	74	31	13	20	38	39	99	82
20	27	71	99	37	20	59	74	27
49	43	67	18	47	43	26	35	60
34	55	54	4	99	49	30	59	3
96	95	70	57	16	13	62	11	43
93	42	67	89	86	59	27	91	32
6	99	18	67	65	23	83	88	17

Filter: 3x3x3

5	4	1
2	1	2
6	9	7

7	2	5
6	1	5
9	3	6

9	8	5
6	4	3
4	3	2

Result: 5x7x1

- How do we compute result(1,1)?

Example With Numbers

Input image: 7x9x3

58	**17**	**75**	9	51	54	21	18	91
6	**65**	**19**	93	52	36	31	23	98
24	**74**	**69**	78	82	94	48	44	44
36	65	19	49	80	88	24	32	12
83	46	37	44	65	56	85	93	26
2	55	63	45	38	63	20	44	41
5	30	79	31	82	59	23	19	60

53	**99**	**34**	78	4	86	49	69	73
24	**4**	**68**	72	75	81	17	5	15
49	**89**	**14**	91	51	58	98	8	66
63	92	73	90	48	19	72	53	52
68	80	11	34	91	24	51	10	98
40	10	66	70	61	89	48	82	65
37	27	50	20	62	3	6	82	81

38	**74**	**31**	13	20	38	39	99	82
20	**27**	**71**	99	37	20	59	74	27
49	**43**	**67**	18	47	43	26	35	60
34	55	54	4	99	49	30	59	3
96	95	70	57	16	13	62	11	43
93	42	67	89	86	59	27	91	32
6	99	18	67	65	23	83	88	17

Filter: 3x3x3

5	4	1
2	1	2
6	9	7

7	2	5
6	1	5
9	3	6

9	8	5
6	4	3
4	3	2

Result: 5x7x1

5849						

- How do we compute result(1,1)?
- We must sum up all these values:

58*5	17*4	75*1
6*2	65*1	19*2
24*6	74*9	69*7
53*7	99*2	34*5
24*6	4*1	68*5
49*9	89*3	14*6
38*9	74*8	31*5
20*6	27*4	71*3
49*4	43*3	67*2

Example With Numbers

Input image: 7x9x3

58	17	75	9	51	54	21	18	91
6	65	19	93	52	36	31	23	98
24	74	69	78	82	94	48	44	44
36	65	19	49	80	88	24	32	12
83	46	37	44	65	56	85	93	26
2	55	63	45	38	63	20	44	41
5	30	79	31	82	59	23	19	60

53	99	34	78	4	86	49	69	73
24	4	68	72	75	81	17	5	15
49	89	14	91	51	58	98	8	66
63	92	73	90	48	19	72	53	52
68	80	11	34	91	24	51	10	98
40	10	66	70	61	89	48	82	65
37	27	50	20	62	3	6	82	81

38	74	31	13	20	38	39	99	82
20	27	71	99	37	20	59	74	27
49	43	67	18	47	43	26	35	60
34	55	54	4	99	49	30	59	3
96	95	70	57	16	13	62	11	43
93	42	67	89	86	59	27	91	32
6	99	18	67	65	23	83	88	17

Filter: 3x3x3

5	4	1
2	1	2
6	9	7

7	2	5
6	1	5
9	3	6

9	8	5
6	4	3
4	3	2

Result: 5x7x1

5849	???					

- How do we compute result(1,2)?

Example With Numbers

Input image: 7x9x3

58	**17**	**75**	**9**	51	54	21	18	91
6	**65**	**19**	**93**	52	36	31	23	98
24	**74**	**69**	**78**	82	94	48	44	44
36	65	19	49	80	88	24	32	12
83	46	37	44	65	56	85	93	26
2	55	63	45	38	63	20	44	41
5	30	79	31	82	59	23	19	60

53	**99**	**34**	**78**	4	86	49	69	73
24	**4**	**68**	**72**	75	81	17	5	15
49	**89**	**14**	**91**	51	58	98	8	66
63	92	73	90	48	19	72	53	52
68	80	11	34	91	24	51	10	98
40	10	66	70	61	89	48	82	65
37	27	50	20	62	3	6	82	81

38	**74**	**31**	**13**	20	38	39	99	82
20	**27**	**71**	**99**	37	20	59	74	27
49	**43**	**67**	**18**	47	43	26	35	60
34	55	54	4	99	49	30	59	3
96	95	70	57	16	13	62	11	43
93	42	67	89	86	59	27	91	32
6	99	18	67	65	23	83	88	17

Filter: 3x3x3

5	4	1
2	1	2
6	9	7

7	2	5
6	1	5
9	3	6

9	8	5
6	4	3
4	3	2

Result: 5x7x1

5849	**7463**					

- How do we compute result(1,2)?
- We must sum up all these values:

17*5	75*4	9*1
65*2	19*1	93*2
74*6	69*9	78*7
99*7	34*2	78*5
4*6	68*1	72*5
89*9	14*3	91*6
74*9	31*8	13*5
27*6	71*4	99*3
43*4	67*3	18*2

Example With Numbers

Input image: 7x9x3

58	17	75	9	51	54	21	18	91
6	65	19	93	52	36	31	23	98
24	74	69	78	82	94	48	44	44
36	65	19	49	80	88	24	32	12
83	46	37	44	65	56	85	93	26
2	55	63	45	38	63	20	44	41
5	30	79	31	82	59	23	19	60

53	99	34	78	4	86	49	69	73
24	4	68	72	75	81	17	5	15
49	89	14	91	51	58	98	8	66
63	92	73	90	48	19	72	53	52
68	80	11	34	91	24	51	10	98
40	10	66	70	61	89	48	82	65
37	27	50	20	62	3	6	82	81

38	74	31	13	20	38	39	99	82
20	27	71	99	37	20	59	74	27
49	43	67	18	47	43	26	35	60
34	55	54	4	99	49	30	59	3
96	95	70	57	16	13	62	11	43
93	42	67	89	86	59	27	91	32
6	99	18	67	65	23	83	88	17

Filter: 3x3x3

5	4	1
2	1	2
6	9	7

7	2	5
6	1	5
9	3	6

9	8	5
6	4	3
4	3	2

Result: 5x7x1

5849	7463	6193	7261	6177	6309	6484
5580	7161	7311	7902	6699	5329	5375
6690	7301	6211	6464	7450	5865	6553
6826	7003	6740	6804	7072	5875	5869
6917	6605	6838	6247	6121	5410	6179

- This way we can compute all values in the result.

Example With Numbers

Input image: 7x9x3

58	17	75	9	51	54	21	18	91
6	65	19	93	52	36	31	23	98
24	74	69	78	82	94	48	44	44
36	65	19	49	80	88	24	32	12
83	46	37	44	65	56	85	93	26
2	55	63	45	38	63	20	44	41
5	30	79	31	82	59	23	19	60

53	99	34	78	4	86	49	69	73
24	4	68	72	75	81	17	5	15
49	89	14	91	51	58	98	8	66
63	92	73	90	48	19	72	53	52
68	80	11	34	91	24	51	10	98
40	10	66	70	61	89	48	82	65
37	27	50	20	62	3	6	82	81

38	74	31	13	20	38	39	99	82
20	27	71	99	37	20	59	74	27
49	43	67	18	47	43	26	35	60
34	55	54	4	99	49	30	59	3
96	95	70	57	16	13	62	11	43
93	42	67	89	86	59	27	91	32
6	99	18	67	65	23	83	88	17

Filter: 3x3x3

5	4	1
2	1	2
6	9	7

7	2	5
6	1	5
9	3	6

9	8	5
6	4	3
4	3	2

Result: 5x7x1

5849	7463	6193	7261	6177	6309	6484
5580	7161	7311	7902	6699	5329	5375
6690	7301	6211	6464	7450	5865	6553
6826	7003	6740	6804	7072	5875	5869
6917	6605	6838	6247	6121	5410	6179

- How do these operations correspond to a layer in a neural network?

Example With Numbers

Input image: 7x9x3

58	**17**	**75**	9	51	54	21	18	91
6	**65**	**19**	93	52	36	31	23	98
24	**74**	**69**	78	82	94	48	44	44
36	65	19	49	80	88	24	32	12
83	46	37	44	65	56	85	93	26
2	55	63	45	38	63	20	44	41
5	30	79	31	82	59	23	19	60

53	**99**	**34**	78	4	86	49	69	73
24	**4**	**68**	72	75	81	17	5	15
49	**89**	**14**	91	51	58	98	8	66
63	92	73	90	48	19	72	53	52
68	80	11	34	91	24	51	10	98
40	10	66	70	61	89	48	82	65
37	27	50	20	62	3	6	82	81

38	**74**	**31**	13	20	38	39	99	82
20	**27**	**71**	99	37	20	59	74	27
49	**43**	**67**	18	47	43	26	35	60
34	55	54	4	99	49	30	59	3
96	95	70	57	16	13	62	11	43
93	42	67	89	86	59	27	91	32
6	99	18	67	65	23	83	88	17

Filter: 3x3x3

5	4	1
2	1	2
6	9	7

7	2	5
6	1	5
9	3	6

9	8	5
6	4	3
4	3	2

Result: 5x7x1

5849						

- Consider the value at position (1,1) of the result.
- How do we obtain that value?
- It is a dot product between a subset of the input numbers, and a specific set of weights.
- This like having a perceptron.
  - What are the inputs of the perceptron?
  - What are the weights of the perceptron?

Example With Numbers

Input image: 7x9x3

58	17	75	9	51	54	21	18	91
6	65	19	93	52	36	31	23	98
24	74	69	78	82	94	48	44	44
36	65	19	49	80	88	24	32	12
83	46	37	44	65	56	85	93	26
2	55	63	45	38	63	20	44	41
5	30	79	31	82	59	23	19	60

53	99	34	78	4	86	49	69	73
24	4	68	72	75	81	17	5	15
49	89	14	91	51	58	98	8	66
63	92	73	90	48	19	72	53	52
68	80	11	34	91	24	51	10	98
40	10	66	70	61	89	48	82	65
37	27	50	20	62	3	6	82	81

38	74	31	13	20	38	39	99	82
20	27	71	99	37	20	59	74	27
49	43	67	18	47	43	26	35	60
34	55	54	4	99	49	30	59	3
96	95	70	57	16	13	62	11	43
93	42	67	89	86	59	27	91	32
6	99	18	67	65	23	83	88	17

Filter: 3x3x3

5	4	1
2	1	2
6	9	7

7	2	5
6	1	5
9	3	6

9	8	5
6	4	3
4	3	2

Result: 5x7x1

5849						

- Consider the value at position (1,1) of the result.
- How do we obtain that value?
- It is a dot product between a subset of the input numbers, and a specific set of weights.
- This like having a perceptron.
  - Perceptron inputs are the red values
  - Perceptron weights are the filter values.

Example With Numbers

Input image: 7x9x3

58	17	75	9	51	54	21	18	91
6	65	19	93	52	36	31	23	98
24	74	69	78	82	94	48	44	44
36	65	19	49	80	88	24	32	12
83	46	37	44	65	56	85	93	26
2	55	63	45	38	63	20	44	41
5	30	79	31	82	59	23	19	60

53	99	34	78	4	86	49	69	73
24	4	68	72	75	81	17	5	15
49	89	14	91	51	58	98	8	66
63	92	73	90	48	19	72	53	52
68	80	11	34	91	24	51	10	98
40	10	66	70	61	89	48	82	65
37	27	50	20	62	3	6	82	81

38	74	31	13	20	38	39	99	82
20	27	71	99	37	20	59	74	27
49	43	67	18	47	43	26	35	60
34	55	54	4	99	49	30	59	3
96	95	70	57	16	13	62	11	43
93	42	67	89	86	59	27	91	32
6	99	18	67	65	23	83	88	17

Filter: 3x3x3

5	4	1
2	1	2
6	9	7

7	2	5
6	1	5
9	3	6

9	8	5
6	4	3
4	3	2

Result: 5x7x1

5849	7463	6193	7261	6177	6309	6484
5580	7161	7311	7902	6699	5329	5375
6690	7301	6211	6464	7450	5865	6553
6826	7003	6740	6804	7072	5875	5869
6917	6605	6838	6247	6121	5410	6179

- So, the convolution operation can be thought of as defining the second layer of a neural network.
  - Each unit in that layer is connected to 27 units in the input layer.
  - **ALL UNITS HAVE IDENTICAL WEIGHTS,** specified by the values in the 3x3x3 filter.

Example With Numbers

Input image: 7x9x3

58	17	75	9	51	54	21	18	91
6	65	19	93	52	36	31	23	98
24	74	69	78	82	94	48	44	44
36	65	19	49	80	88	24	32	12
83	46	37	44	65	56	85	93	26
2	55	63	45	38	63	20	44	41
5	30	79	31	82	59	23	19	60

53	99	34	78	4	86	49	69	73
24	4	68	72	75	81	17	5	15
49	89	14	91	51	58	98	8	66
63	92	73	90	48	19	72	53	52
68	80	11	34	91	24	51	10	98
40	10	66	70	61	89	48	82	65
37	27	50	20	62	3	6	82	81

38	74	31	13	20	38	39	99	82
20	27	71	99	37	20	59	74	27
49	43	67	18	47	43	26	35	60
34	55	54	4	99	49	30	59	3
96	95	70	57	16	13	62	11	43
93	42	67	89	86	59	27	91	32
6	99	18	67	65	23	83	88	17

Filter: 3x3x3

5	4	1
2	1	2
6	9	7

7	2	5
6	1	5
9	3	6

9	8	5
6	4	3
4	3	2

Result: 5x7x1

5849	7463	6193	7261	6177	6309	6484
5580	7161	7311	7902	6699	5329	5375
6690	7301	6211	6464	7450	5865	6553
6826	7003	6740	6804	7072	5875	5869
6917	6605	6838	6247	6121	5410	6179

- However, there is an important difference between a convolution operation and the output of a neural network layer.
  - The convolution corresponds to taking, for each perceptron, the dot product between inputs and weights.
  - Does a perceptron output just that dot product?

Example With Numbers

Input image: 7x9x3

58	17	75	9	51	54	21	18	91
6	65	19	93	52	36	31	23	98
24	74	69	78	82	94	48	44	44
36	65	19	49	80	88	24	32	12
83	46	37	44	65	56	85	93	26
2	55	63	45	38	63	20	44	41
5	30	79	31	82	59	23	19	60

53	99	34	78	4	86	49	69	73
24	4	68	72	75	81	17	5	15
49	89	14	91	51	58	98	8	66
63	92	73	90	48	19	72	53	52
68	80	11	34	91	24	51	10	98
40	10	66	70	61	89	48	82	65
37	27	50	20	62	3	6	82	81

38	74	31	13	20	38	39	99	82
20	27	71	99	37	20	59	74	27
49	43	67	18	47	43	26	35	60
34	55	54	4	99	49	30	59	3
96	95	70	57	16	13	62	11	43
93	42	67	89	86	59	27	91	32
6	99	18	67	65	23	83	88	17

Filter: 3x3x3

5	4	1
2	1	2
6	9	7

7	2	5
6	1	5
9	3	6

9	8	5
6	4	3
4	3	2

Result: 5x7x1

5849	7463	6193	7261	6177	6309	6484
5580	7161	7311	7902	6699	5329	5375
6690	7301	6211	6464	7450	5865	6553
6826	7003	6740	6804	7072	5875	5869
6917	6605	6838	6247	6121	5410	6179

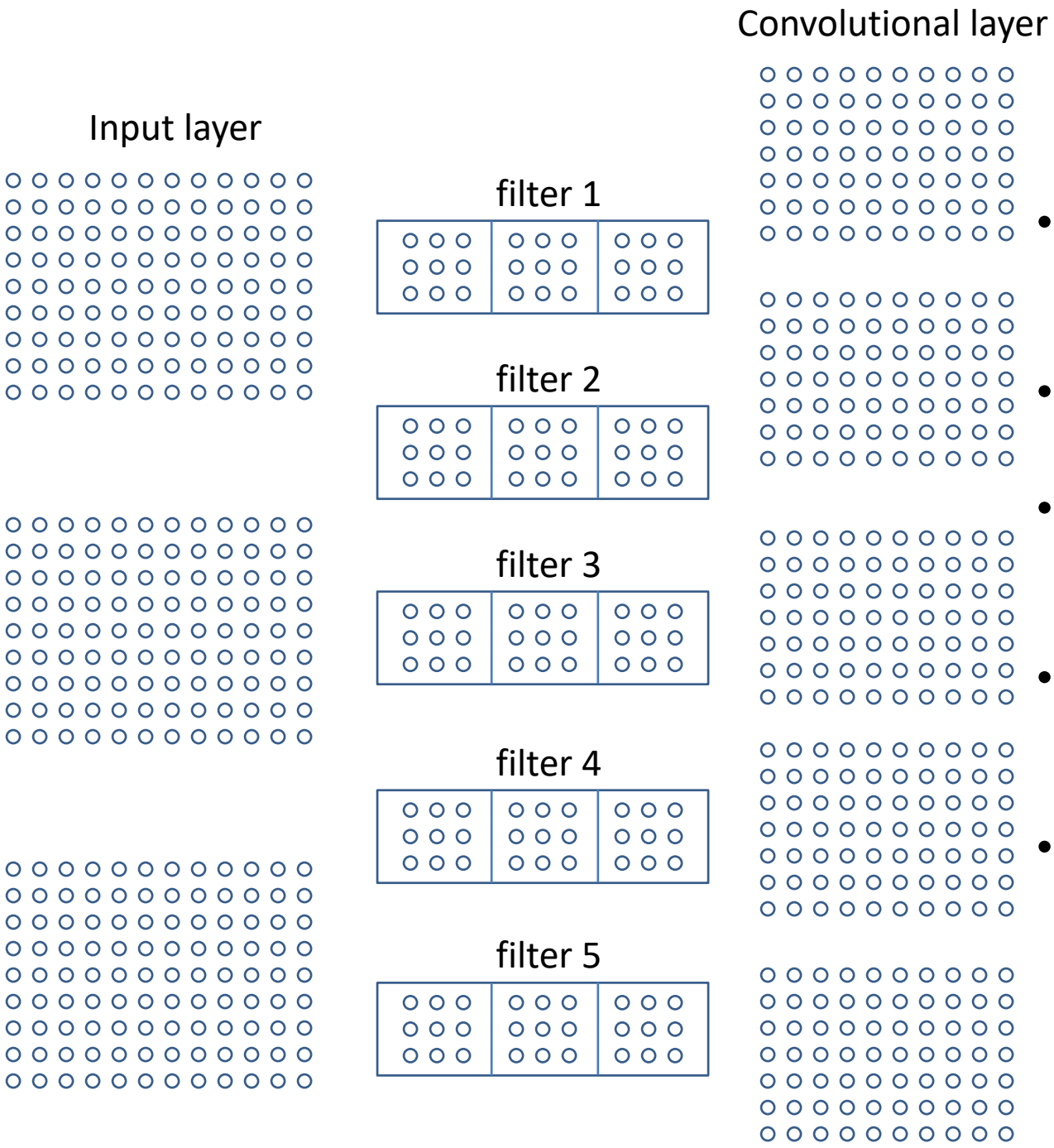
- However, there is an important difference between a convolution operation and the output of a neural network layer.
  - The convolution corresponds to taking, for each perceptron, the dot product between inputs and weights.
  - The output of a perceptron is obtained by **applying an activation function to the dot product**.

Convolutional Layers

- So, a convolution can be thought of as specifying a neural network layer.
- The filter values specify the weights of every single perceptron in the layer.
- However, remember that **the output of a convolutional layer is not the result of the convolution**.
  - Each dot product passes through an activation function (usually ReLU) in order to produce an output.

Applying Multiple Convolutions

- In our previous example:
  - We started with a 3-channel input image (an RGB image).
    - This defined a 3D array of input units, of dimensions R*C*3.
  - We applied a single 3x3x3 convolutional filter.
  - We obtained a 1-channel output.
    - This defined a 2D array of units in the convolutional layer.
- We can apply multiple convolutions at the same time.
  - We can apply K different 3x3x3 convolutional filters.
  - This leads to a K-channel output.
    - This defines a 3D array of units in the convolutional layer, of dimensions (R-2)*(C-2)*K.
    - Each 2D slice of that array corresponds to a single 3x3x3 filter.

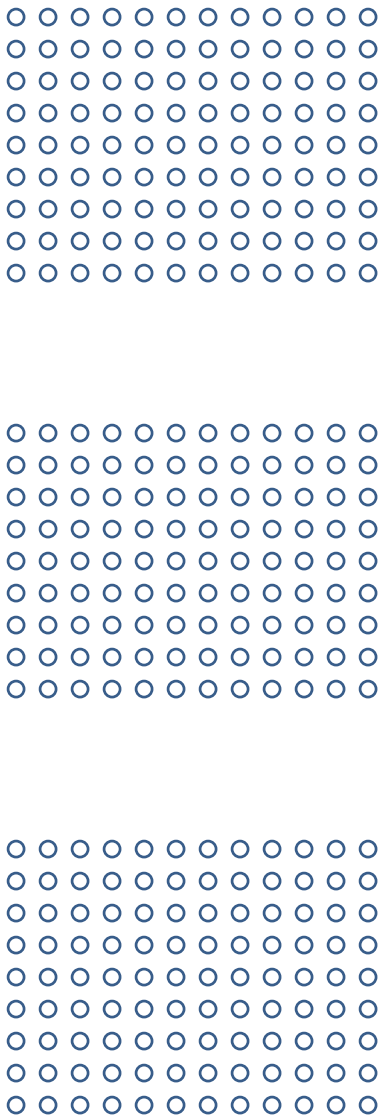


- R*C*3 input units (one unit for each pixel value).
- (R-2)*(C-2)*5 units in the convolutional layer.
- The convolutional layer consists of five 2D arrays of units.
- Each such 2D array corresponds to a 3x3x3 filter.
- Overall, the weights (filter values) can be thought of as a 4D array, of size 3x3x3x4 in this example.

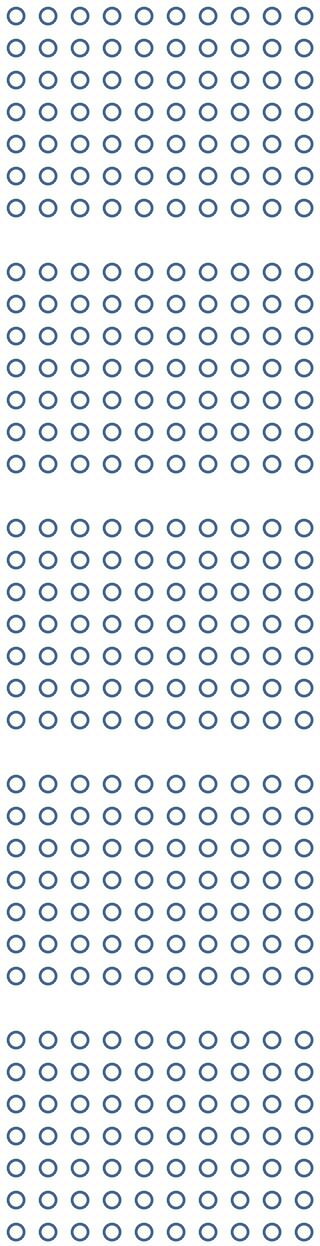
Multiple Convolutional Layers

- In our previous example:
  - We started with a 3-channel input image (an RGB image).
    - This defined a 3D array of input units, of dimensions R*C*3.
  - We applied 5 3x3x3 convolutional filters.
  - We obtained a 5-channel output.
    - This defined a convolutional layer with (R-2)*(C-2)*5 units.
- We can put a new convolutional layer after the previous one.
  - We can define K 3x3x5 filters.
    - K is whatever we want.
    - 5 is there to match the number of channels in the previous layer.
  - We end up with a layer of size (R-4)*(C-4)*K units.

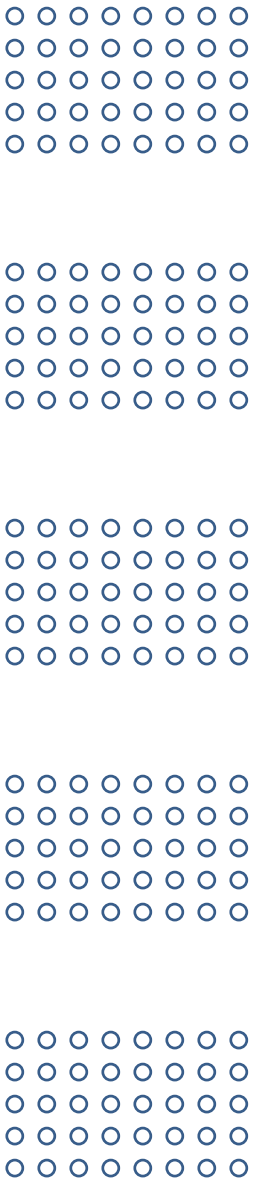
Input layer



1st Convolutional layer
(R-2)*(C-2)*5



2nd Convolutional layer
(R-4)*(C-4)*5

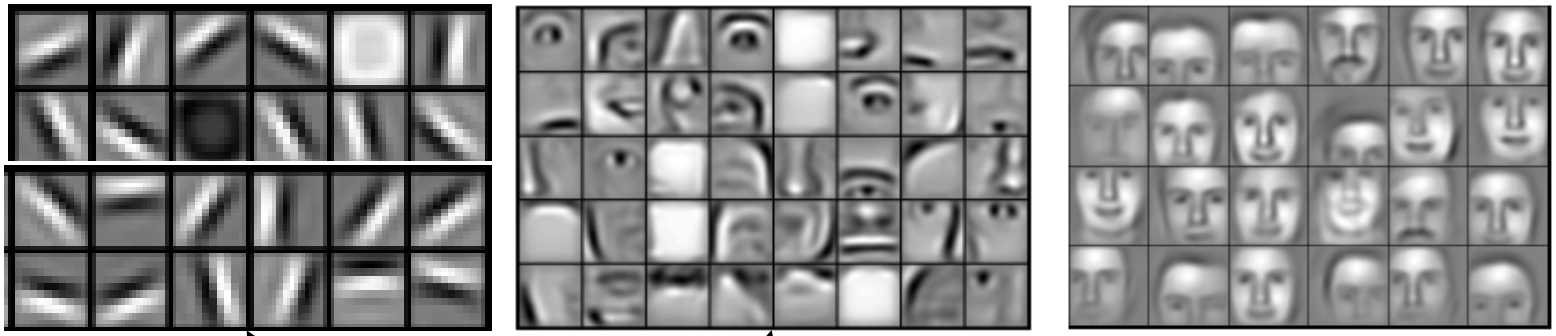


Multiple Convolutional Layers

- Overall, we can put as many convolutional layers in sequence as we want.

CNN Visualization

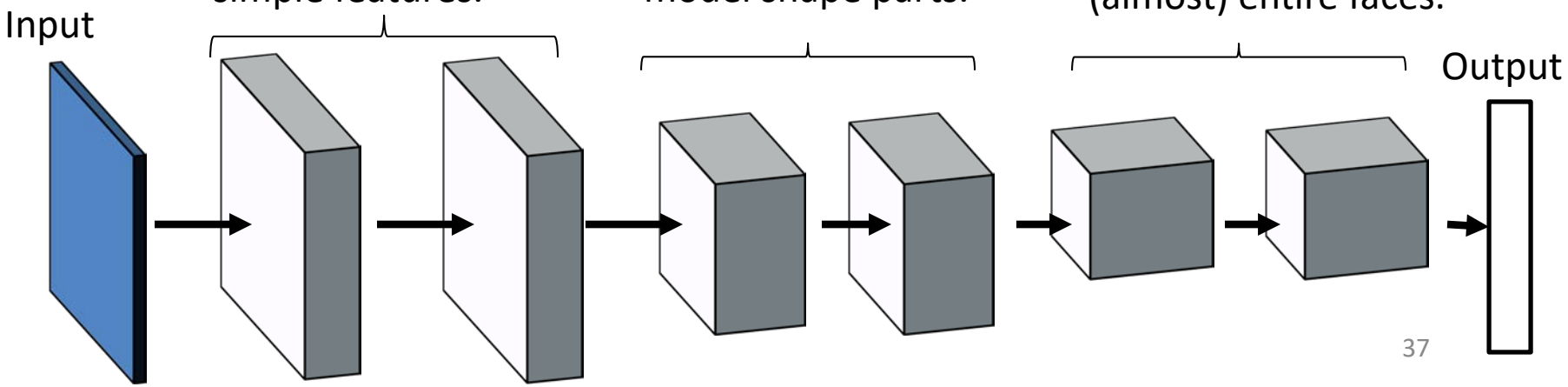
- Visualization of a deep neural network trained with face images.
  - Credit for feature visualizations: [Lee09] Honglak Lee, Roger Grosse, Rajesh Ranganath and Andrew Y. Ng., ICML 2009.



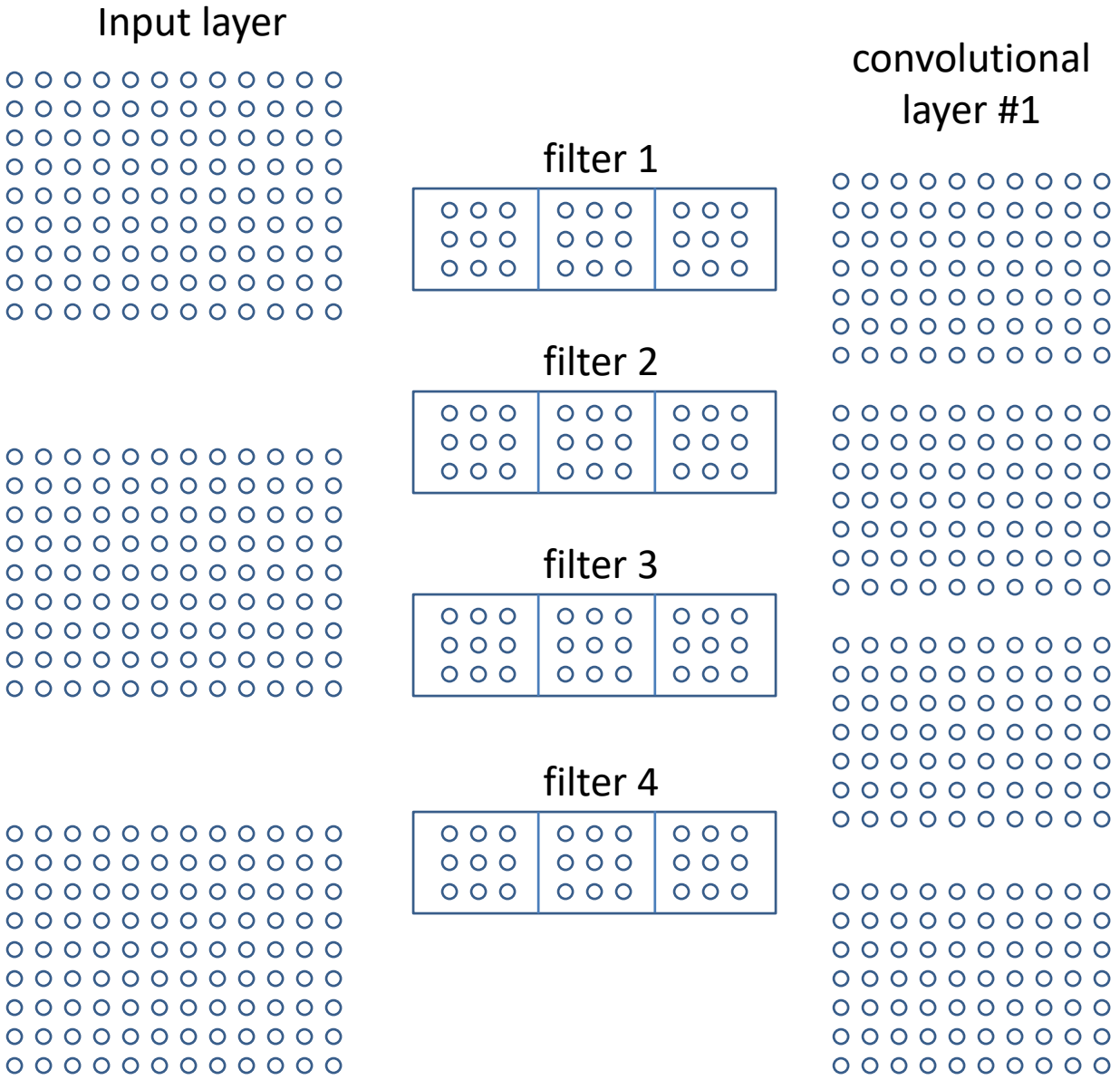
Early layers, extract simple features.

Middle layers, model shape parts.

Last layers, model (almost) entire faces.

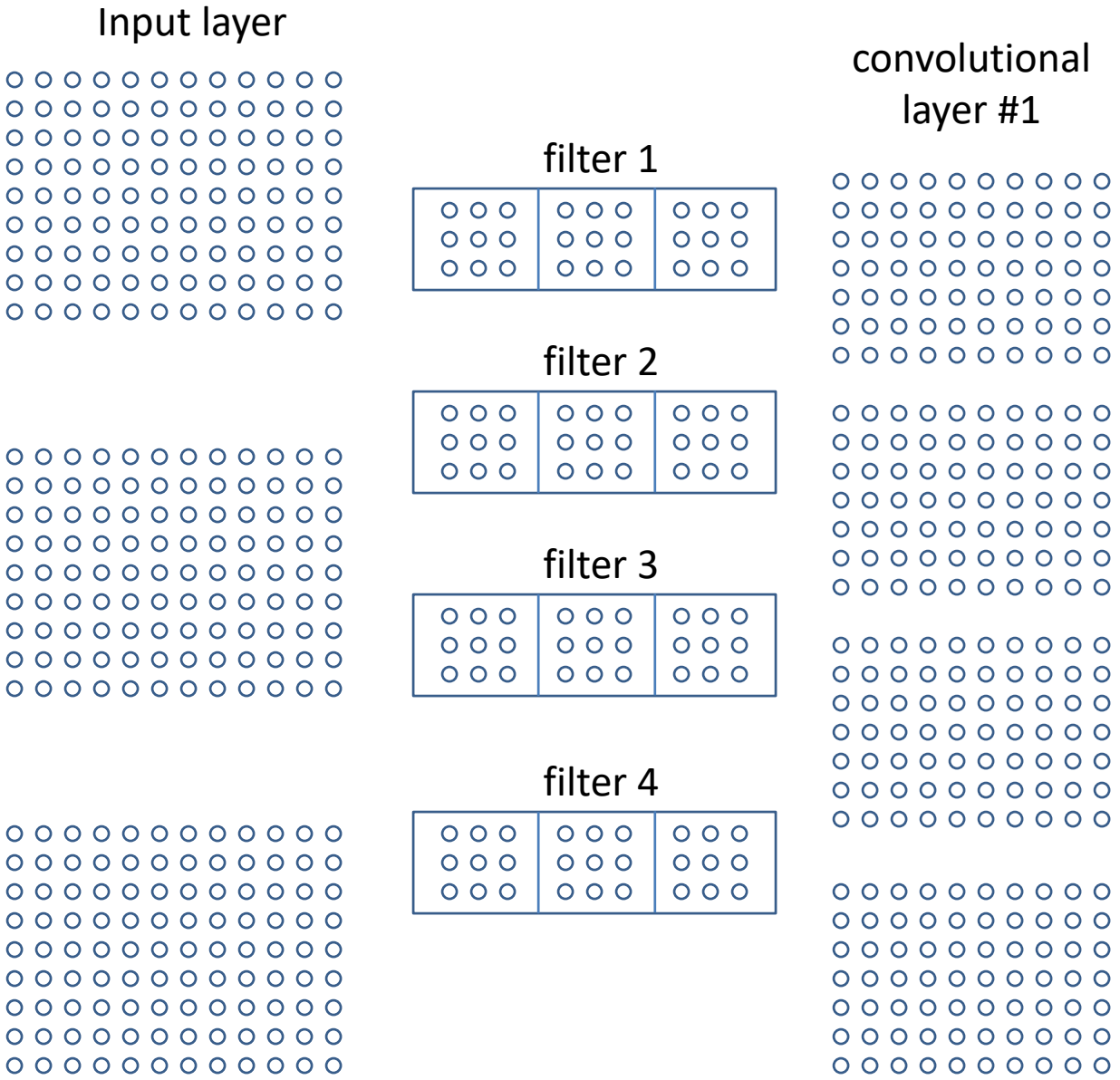


Convolutional Vs. Fully Connected



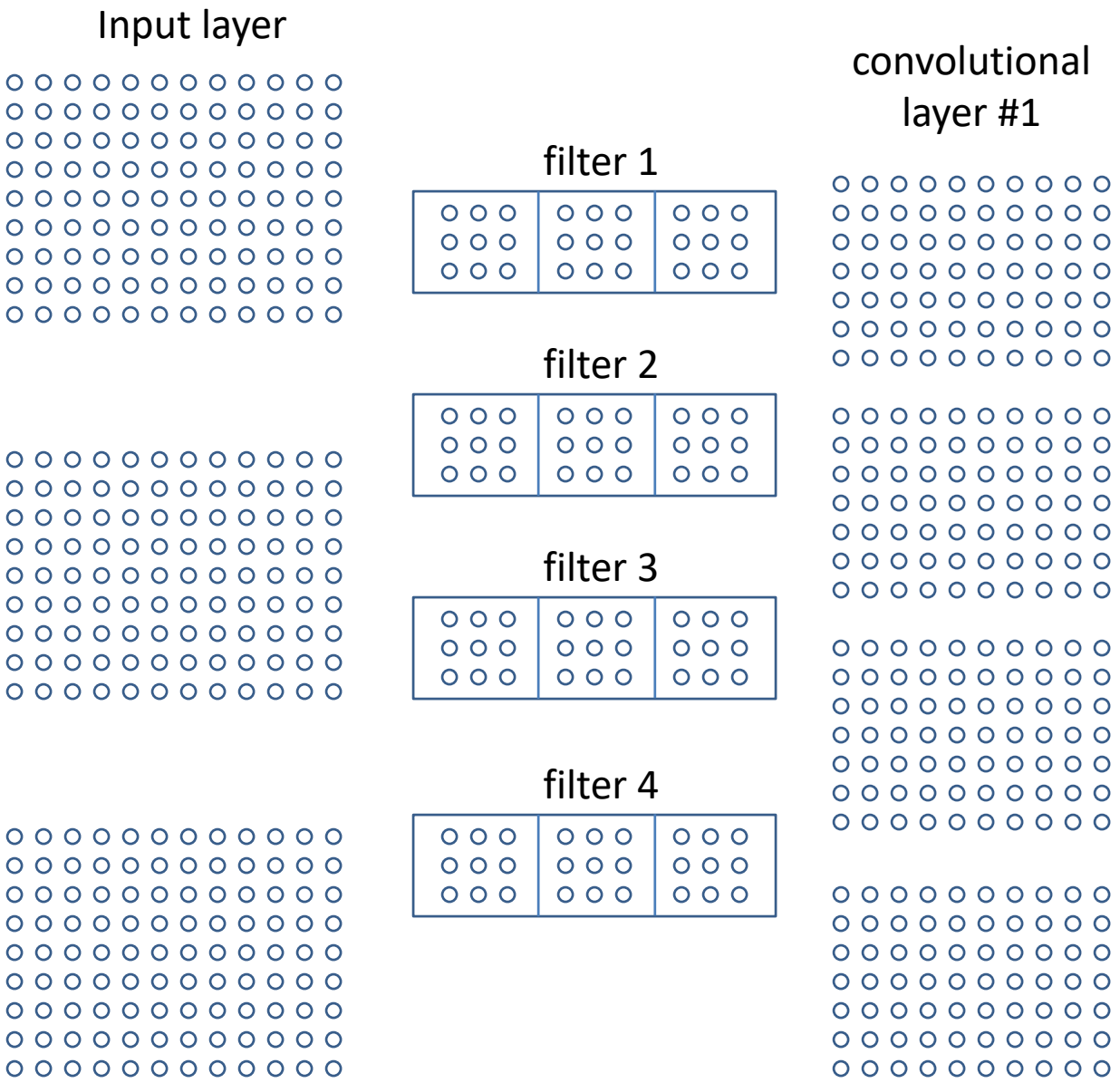
- R*C*3 input units (one unit for each pixel value).
- (R-2)*(C-2)*4 units in the convolutional layer.
- How many weights do we need to learn for this layer?
- How many weights would we need to learn for a fully-connected layer with the same number of units?

Convolutional Vs. Fully Connected



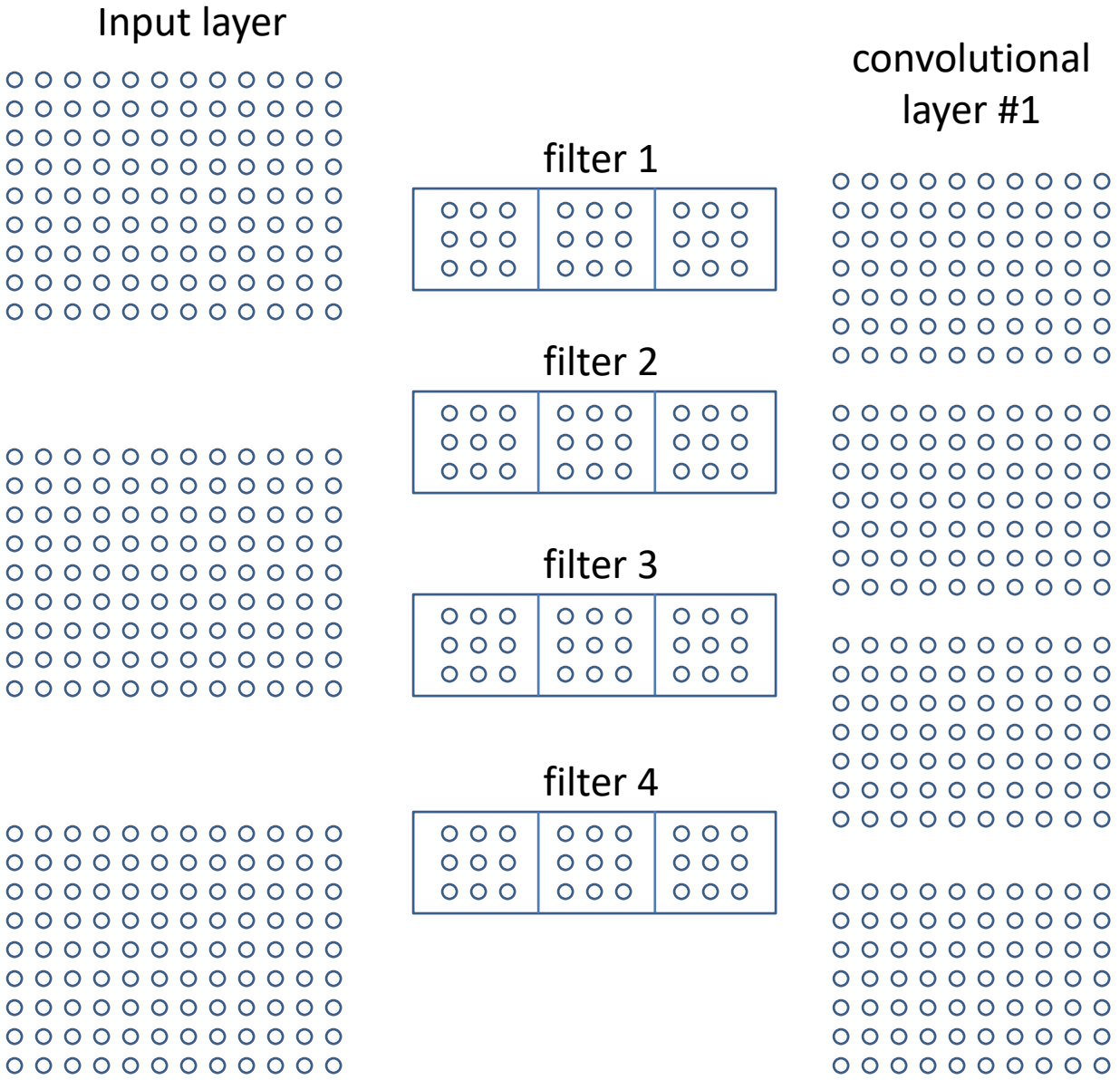
- R*C*3 input units (one unit for each pixel value).
- (R-2)*(C-2)*4 units in the convolutional layer.
- The convolutional layer requires learning four filters.
- Each filter has 3x3x3=27 weights.
- So, in total we need to learn 4*27 = 108 weights.

Convolutional Vs. Fully Connected



- R*C*3 input units (one unit for each pixel value).
- (R-2)*(C-2)*4 units in the convolutional layer.
- For a fully connected layer, there is an edge connecting each input unit to each fully connected layer unit.
- Number of edges: R*C*3*(R-2)*(C-2)*4.
- We need to learn that many weights.
- For a 400x500 image: 475 billion weights.

Convolutional Vs. Fully Connected



- So, for the convolutional layer we need to learn 108 weights.
- For a similarly sized fully connected layer we would need 475 billion weights.
- Thus, convolutional layers are much more practical to use in terms of:
  - Storage
  - training time
  - runtime on test data
  - amount of training data required.

Max Pooling Layer

- A max pooling layer is a layer where every unit corresponds to a 2x2 (or 3x3) neighborhood in the previous layer, and the output is the maximum value in that neighborhood.

Max Pooling Example

Input image:
7x7x3

58	17	75	9	51	54	21
6	65	19	93	52	36	31
24	74	69	78	82	94	48
36	65	19	49	80	88	24
83	46	37	44	65	56	85
2	55	63	45	38	63	20
5	30	79	31	82	59	23

53	99	34	78	4	86	49
24	4	68	72	75	81	17
49	89	14	91	51	58	98
63	92	73	90	48	19	72
68	80	11	34	91	24	51
40	10	66	70	61	89	48
37	27	50	20	62	3	6

38	74	31	13	20	38	39
20	27	71	99	37	20	59
49	43	67	18	47	43	26
34	55	54	4	99	49	30
96	95	70	57	16	13	62
93	42	67	89	86	59	27
6	99	18	67	65	23	83

Filter:
3x3x3

5	4	1
2	1	2
6	9	7

7	2	5
6	1	5
9	3	6

9	8	5
6	4	3
4	3	2

Convolutional Layer
Output: 5x5x1

5849	7463	6193	7261	6177
5580	7161	7311	7902	6699
6690	7301	6211	6464	7450
6826	7003	6740	6804	7072
6917	6605	6838	6247	6121

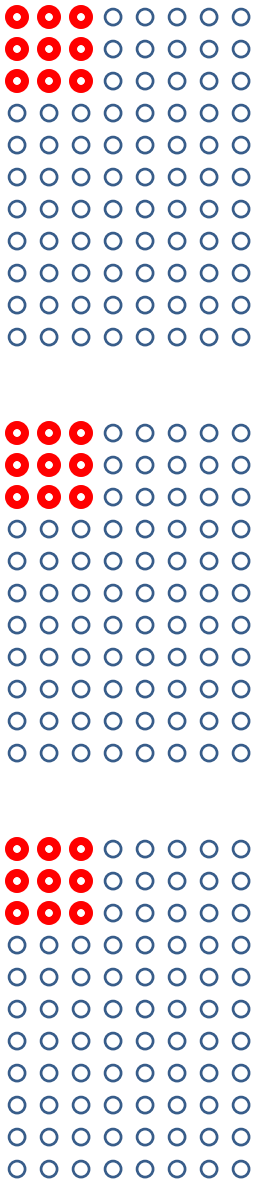
Max Pool Layer
Output: 3x3x1

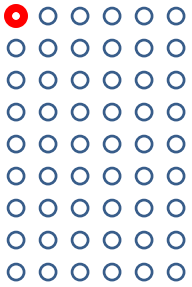
7463	7902	6699
7301	6804	7450
6917	6838	6121

Stride

- For each convolutional layer, we can pick a **stride** *S*.
- *S* is a single number, that influences the number of outputs of the convolutional layer.
  - It determines how many locations to shift the filter to the left or to the bottom to compute the next value of the convolution result.
- Stride = 1: we shift the filter one location at a time, so no rows or columns are skipped.
- Stride = 2: we shift the filter two locations at a time. Half the rows and half the columns are skipped.
  - This means that the convolutional layer will have about half the number of rows and columns of the previous layer.

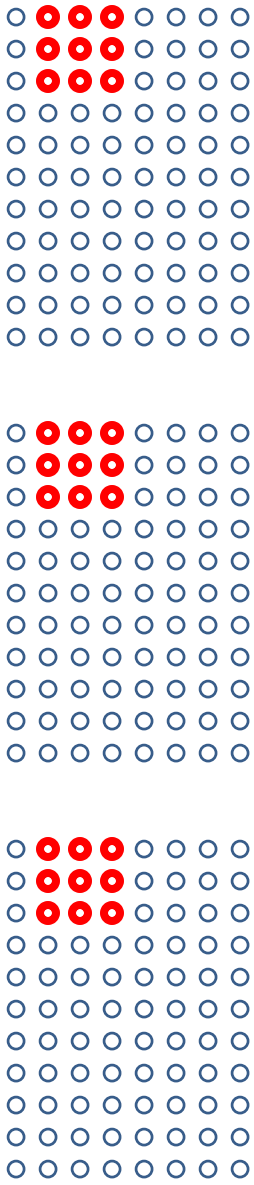
Example of Stride 1

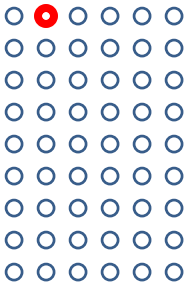




- The convolution result at location (1,1) corresponds to matching the 3x3x3 filter values with the highlighted values in the input layer.

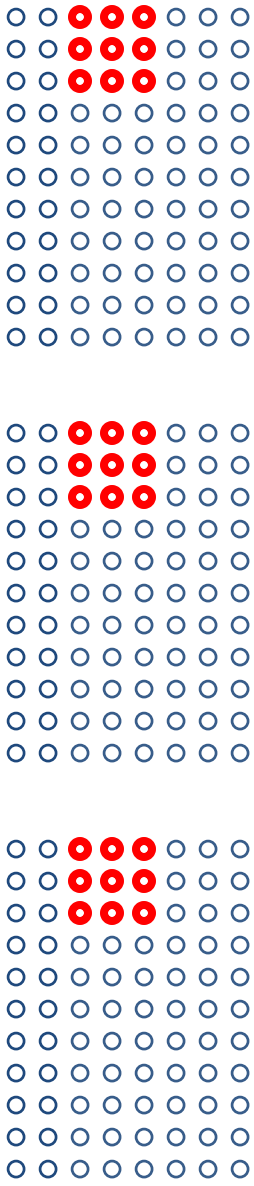
Example of Stride 1

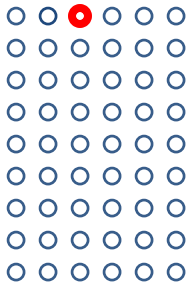




- The convolution result at location (1,2) corresponds to matching the 3x3x3 filter values with the highlighted values in the input layer.

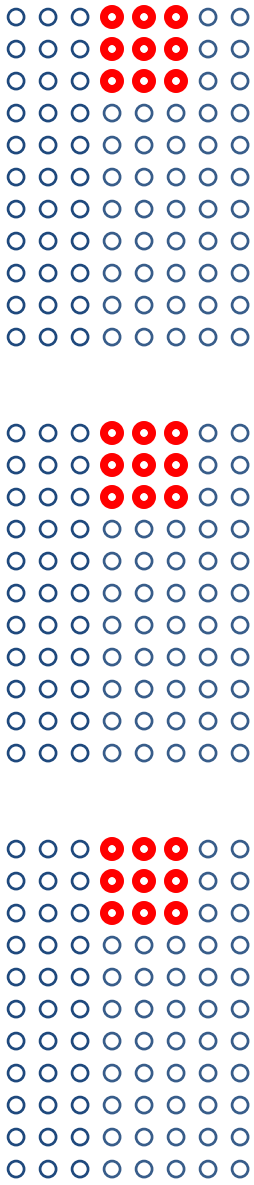
Example of Stride 1

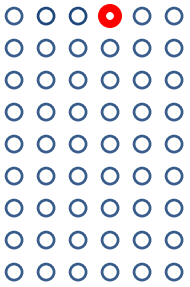




- The convolution result at location (1,3) corresponds to matching the 3x3x3 filter values with the highlighted values in the input layer.

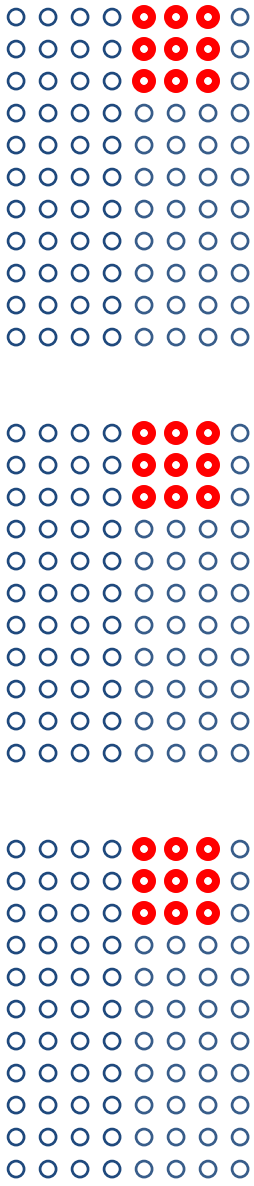
Example of Stride 1

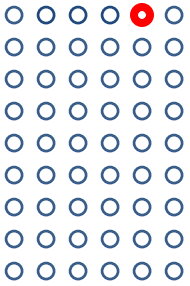




- The convolution result at location (1,4) corresponds to matching the 3x3x3 filter values with the highlighted values in the input layer.

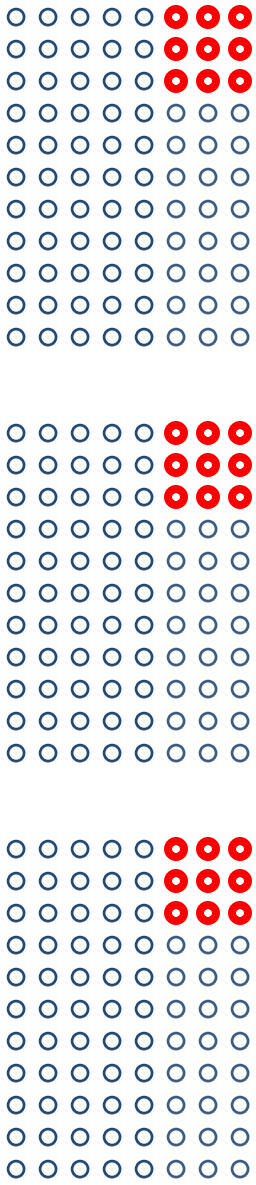
Example of Stride 1

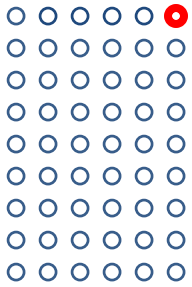




- The convolution result at location (1,5) corresponds to matching the 3x3x3 filter values with the highlighted values in the input layer.

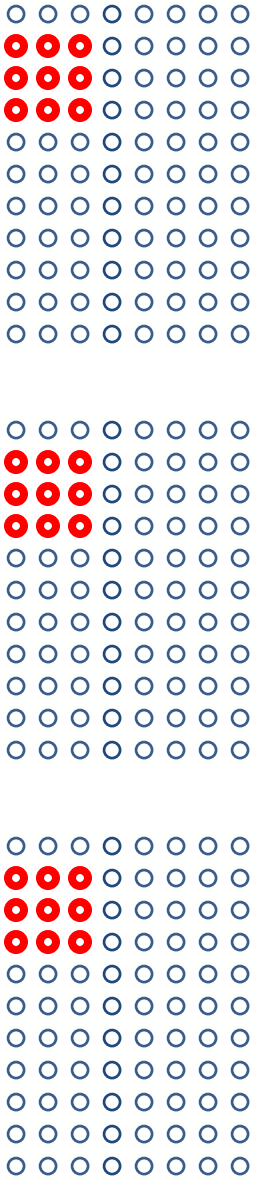
Example of Stride 1

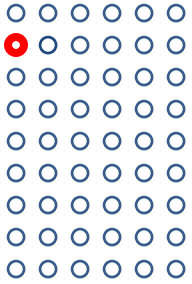




- The convolution result at location (1,6) corresponds to matching the 3x3x3 filter values with the highlighted values in the input layer.

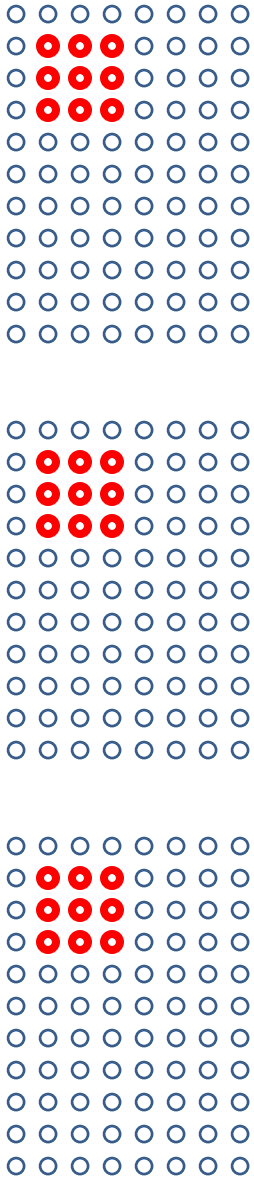
Example of Stride 1

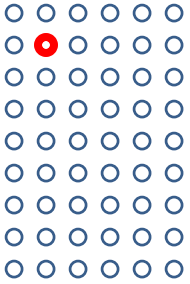




- The convolution result at location (2,1) corresponds to matching the 3x3x3 filter values with the highlighted values in the input layer.

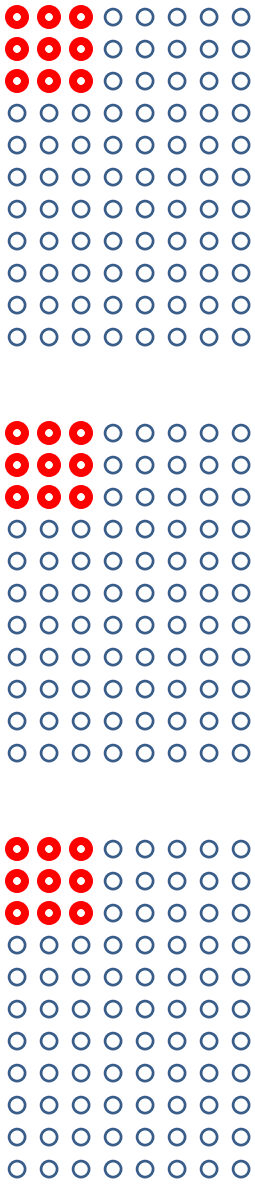
Example of Stride 1





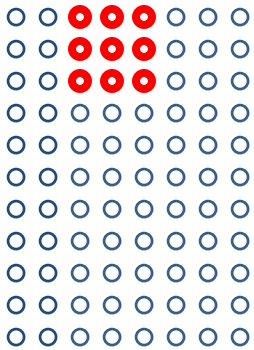
- The convolution result at location (2,2) corresponds to matching the 3x3x3 filter values with the highlighted values in the input layer.
- And so on…

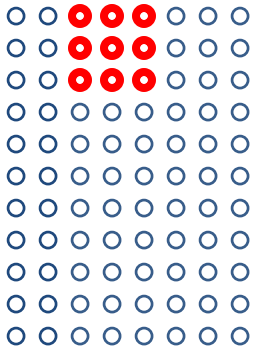
Example of Stride 2

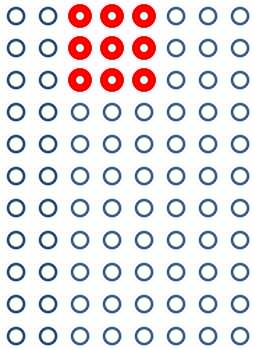


- The convolution result at location (1,1) corresponds to matching the 3x3x3 filter values with the highlighted values in the input layer.

Example of Stride 2

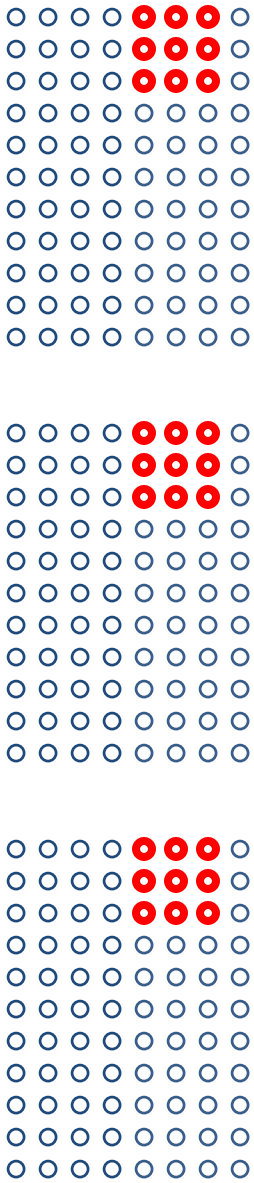






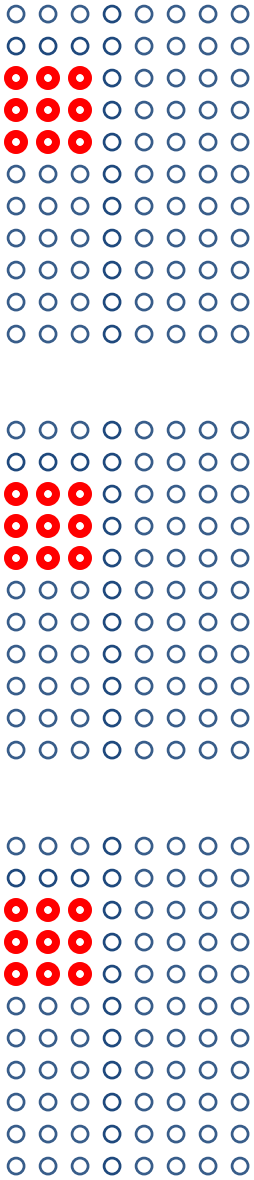
- The convolution result at location (1,2) corresponds to matching the 3x3x3 filter values with the highlighted values in the input layer.

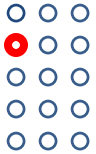
Example of Stride 2



- The convolution result at location (1,3) corresponds to matching the 3x3x3 filter values with the highlighted values in the input layer.

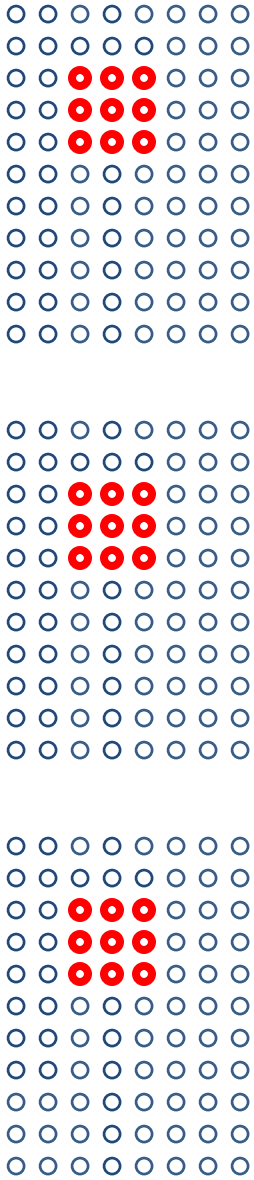
Example of Stride 2

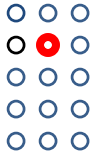




- The convolution result at location (2,1) corresponds to matching the 3x3x3 filter values with the highlighted values in the input layer.

Example of Stride 2

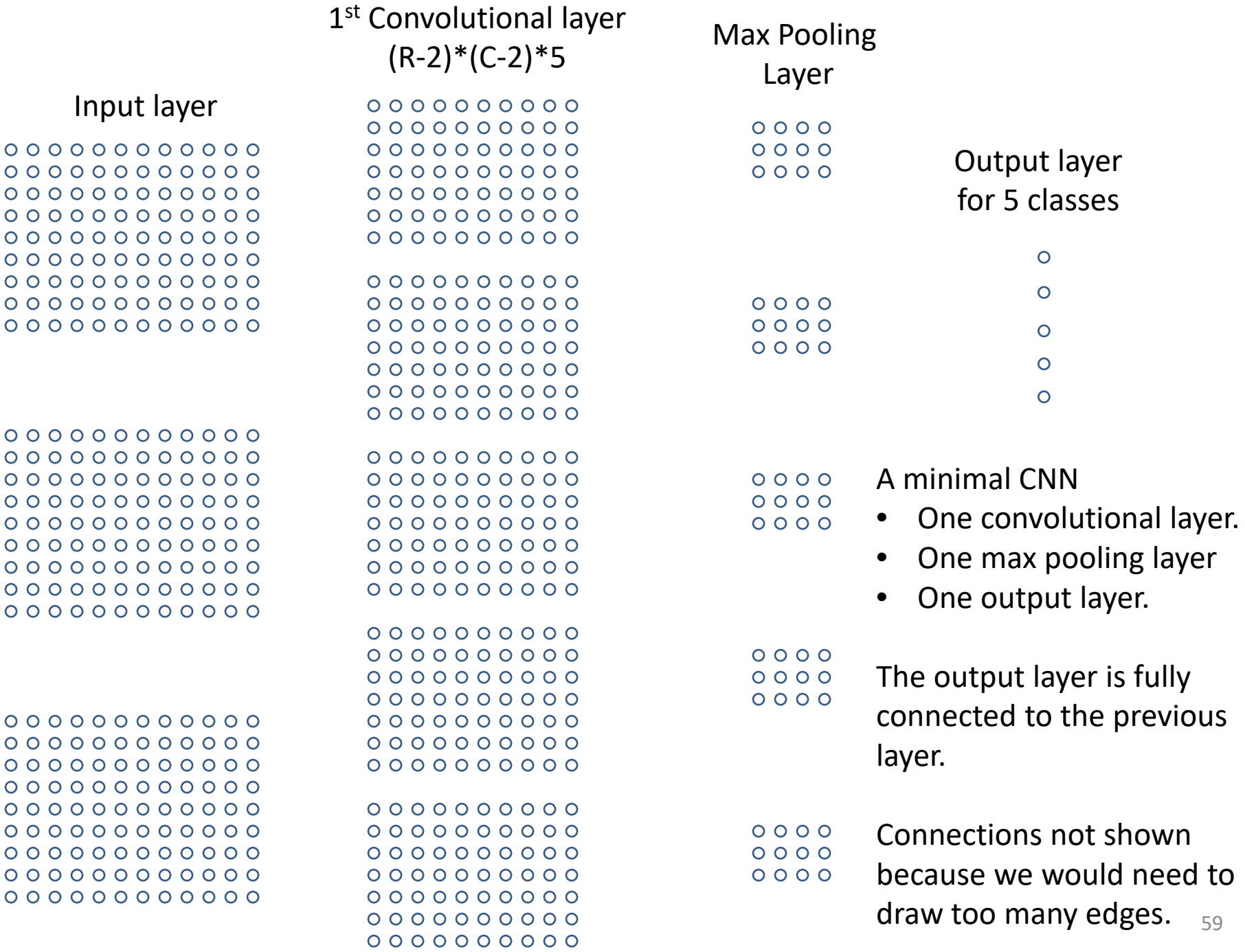




- The convolution result at location (2,2) corresponds to matching the 3x3x3 filter values with the highlighted values in the input layer.

Fully Connected Output Layer

- As usual, the output layer has as many units as the number of classes.
  - Since we have a multiclass problem, we apply the softmax function to convert outputs to class probabilities.
- The layer preceding the output layer is called the **last hidden layer**.
- Usually the output layer is fully connected to its previous layer.
- The last hidden layer can be a convolutional layer or a max pooling layer.



A minimal CNN

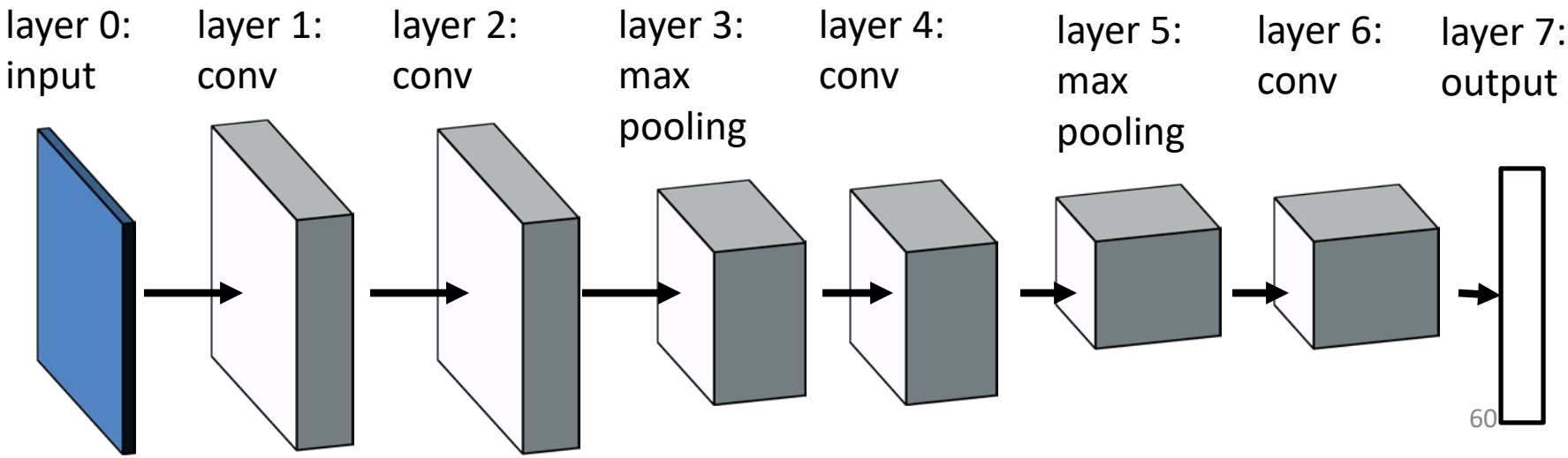
- One convolutional layer.
- One max pooling layer
- One output layer.

The output layer is fully connected to the previous layer.

Connections not shown because we would need to draw too many edges.

Example of a CNN Layout

- This is an example of a CNN with six hidden layers.
  - Four convolutional layers with stride 1.
  - Two max pooling layers.



CNNs in Keras: keras_cnn_mnist.py

```
import numpy as np
from tensorflow import keras
import matplotlib.pyplot as plt

dataset = keras.datasets.mnist
(x_train, train_labels), (x_test, test_labels) = dataset.load_data()
number_of_classes = np.max([np.max(train_labels), np.max(test_labels)]) + 1

# Scale images to the [0, 1] range
x_train = x_train.astype("float32") / 255
x_test = x_test.astype("float32") / 255
```

- This is the first part of the code.
- There is nothing new in these lines. We load the mnist dataset and we normalize the pixel values to the [0,1] range.

CNNs in Keras: keras_cnn_mnist.py

```
# Make sure images have shape (28, 28, 1)
x_train = np.expand_dims(x_train, -1)
x_test = np.expand_dims(x_test, -1)
```

- This is the second part of the code.
- Here we are doing something new. We use the numpy **expand_dims()** function to convert training and test inputs to 4D arrays.
  - In MNIST, each input is originally a 2D array of size rows x cols.
  - These lines convert each input to a 3D array, of size rows x cols x 1.
- Why are we doing this? Because in Keras, a convolutional layer expects each input to be a 3D array.

CNNs in Keras: keras_cnn_mnist.py

```
model = keras.Sequential([
        keras.Input(shape=input_shape),
        keras.layers.Conv2D(32, kernel_size=(3, 3), activation="relu"),
        keras.layers.MaxPooling2D(pool_size=(2, 2)),
        keras.layers.Conv2D(64, kernel_size=(3, 3), activation="relu"),
        keras.layers.MaxPooling2D(pool_size=(2, 2)),
        keras.layers.Flatten(),
        keras.layers.Dropout(0.5),
        keras.layers.Dense(number_of_classes, activation="softmax"),
    ])
```

- This is the third part of the code.
- Notice the use of various Keras layers: Conv2D, MaxPooling, Flatten, Dropout, Dense.

CNNs in Keras: keras_cnn_mnist.py

```
model = keras.Sequential([
        keras.Input(shape=input_shape),
        keras.layers.Conv2D(32, kernel_size=(3, 3), activation="relu"),
        keras.layers.MaxPooling2D(pool_size=(2, 2)),
        keras.layers.Conv2D(64, kernel_size=(3, 3), activation="relu"),
        keras.layers.MaxPooling2D(pool_size=(2, 2)),
        keras.layers.Flatten(),
        keras.layers.Dropout(0.5),
        keras.layers.Dense(number_of_classes, activation="softmax"),
    ])
```

- Both the input and output of a Conv2D layer are 3D arrays.
  - The first argument specifies the third dimension of the output array.
  - The other two dimensions depend on the size of the input array.

CNNs in Keras: keras_cnn_mnist.py

```
model.compile(loss=keras.losses.SparseCategoricalCrossentropy(),
optimizer="adam", metrics=["accuracy"])

model.fit(x_train, train_labels, epochs=15, batch_size=128)
test_loss, test_acc = model.evaluate(x_test,  test_labels, verbose=0)
print('\nTest accuracy: %.2f‘%% % (test_acc * 100))
```

- This is the fourth and last part of the code.
- Everything here should be familiar, we have used identical code in previous examples.
- The classification accuracy is 99.27%.
  - Obviously, each time you run the training you may get a different test accuracy.

Summary of the Model

Output:

```
_________________________________________________________________
 Layer (type)                 Output Shape              Param #
=================================================================
conv2d_18 (Conv2D)           (None, 26, 26, 32)        320
max_pooling2d_18 (MaxPooling2D)  (None, 13, 13, 32)        0
conv2d_19 (Conv2D)           (None, 11, 11, 64)        18496
max_pooling2d_19 (MaxPooling2D)  (None, 5, 5, 64)          0
flatten_9 (Flatten)          (None, 1600)              0
dropout_9 (Dropout)          (None, 1600)              0
dense_9 (Dense)              (None, 10)                16010
=================================================================
```

- It is important to understand the values of the output shapes.
- The None at the beginning of each output shape simply indicates that the first dimension is equal to the batch size.
  - It is more simple to ignore that first dimension. The rest of the shape is what we get when we process a single training input at a time.

Summary of the Model

Output:

```
_________________________________________________________________
 Layer (type)                 Output Shape              Param #
=================================================================
conv2d_18 (Conv2D)           (None, 26, 26, 32)        320
max_pooling2d_18 (MaxPooling2D)  (None, 13, 13, 32)        0
conv2d_19 (Conv2D)           (None, 11, 11, 64)        18496
max_pooling2d_19 (MaxPooling2D)  (None, 5, 5, 64)          0
flatten_9 (Flatten)          (None, 1600)              0
dropout_9 (Dropout)          (None, 1600)              0
dense_9 (Dense)              (None, 10)                16010
=================================================================
```

- For the first Conv2D layer, why is the output shape 26x26x32?
- We created that layer with this line:

```
keras.layers.Conv2D(32, kernel_size=(3, 3), activation="relu")
```

Summary of the Model

Output:

```
_________________________________________________________________
 Layer (type)                 Output Shape              Param #
=================================================================
conv2d_18 (Conv2D)           (None, 26, 26, 32)        320
max_pooling2d_18 (MaxPooling2D)  (None, 13, 13, 32)        0
conv2d_19 (Conv2D)           (None, 11, 11, 64)        18496
max_pooling2d_19 (MaxPooling2D)  (None, 5, 5, 64)          0
flatten_9 (Flatten)          (None, 1600)              0
dropout_9 (Dropout)          (None, 1600)              0
dense_9 (Dense)              (None, 10)                16010
=================================================================
```

- For the first Conv2D layer, why is the output shape 26x26x32?
- Each input image has 28 rows and 28 columns.
- Convolving with a 3x3 kernel gives a result of 26 rows and 26 columns.
- We convolve with 32 kernels, so we get 32 arrays of size 26x26.

Summary of the Model

Output:

```
_________________________________________________________________
 Layer (type)                 Output Shape              Param #
=================================================================
conv2d_18 (Conv2D)           (None, 26, 26, 32)        320
max_pooling2d_18 (MaxPooling2D)  (None, 13, 13, 32)        0
conv2d_19 (Conv2D)           (None, 11, 11, 64)        18496
max_pooling2d_19 (MaxPooling2D)  (None, 5, 5, 64)          0
flatten_9 (Flatten)          (None, 1600)              0
dropout_9 (Dropout)          (None, 1600)              0
dense_9 (Dense)              (None, 10)                16010
=================================================================
```

- For the first Conv2D layer, why is the number of parameters 320?
- We created that layer with this line:

```
keras.layers.Conv2D(32, kernel_size=(3, 3), activation="relu")
```

Summary of the Model

Output:

```
_________________________________________________________________
 Layer (type)                 Output Shape              Param #
=================================================================
conv2d_18 (Conv2D)           (None, 26, 26, 32)        320
max_pooling2d_18 (MaxPooling2D)  (None, 13, 13, 32)        0
conv2d_19 (Conv2D)           (None, 11, 11, 64)        18496
max_pooling2d_19 (MaxPooling2D)  (None, 5, 5, 64)          0
flatten_9 (Flatten)          (None, 1600)              0
dropout_9 (Dropout)          (None, 1600)              0
dense_9 (Dense)              (None, 10)                16010
=================================================================
```

- For the first Conv2D layer, why is the number of parameters 320?
- We have 32 channels in that layer, 1 channel in the previous layer, and a 3x3 filter size.
- That gives us 32*1*3*3 = 278 weights for the filter values.
- We also get 32 bias weights, one for each channel. 278+32=320

Summary of the Model

Output:

```
_________________________________________________________________
 Layer (type)                 Output Shape              Param #
=================================================================
conv2d_18 (Conv2D)           (None, 26, 26, 32)        320
max_pooling2d_18 (MaxPooling2D)  (None, 13, 13, 32)        0
conv2d_19 (Conv2D)           (None, 11, 11, 64)        18496
max_pooling2d_19 (MaxPooling2D)  (None, 5, 5, 64)          0
flatten_9 (Flatten)          (None, 1600)              0
dropout_9 (Dropout)          (None, 1600)              0
dense_9 (Dense)              (None, 10)                16010
=================================================================
```

- For the first MaxPooling2D layer, why is the output shape 13x13x32?
- We created that layer with this line:

keras.layers.MaxPooling2D(pool_size=(2, 2))

Summary of the Model

Output:

```
_________________________________________________________________
 Layer (type)                 Output Shape              Param #
=================================================================
conv2d_18 (Conv2D)           (None, 26, 26, 32)        320
max_pooling2d_18 (MaxPooling2D)  (None, 13, 13, 32)        0
conv2d_19 (Conv2D)           (None, 11, 11, 64)        18496
max_pooling2d_19 (MaxPooling2D)  (None, 5, 5, 64)          0
flatten_9 (Flatten)          (None, 1600)              0
dropout_9 (Dropout)          (None, 1600)              0
dense_9 (Dense)              (None, 10)                16010
=================================================================
```

- For the first MaxPooling2D layer, why is the output shape 13x13x32?
  - We split each 2D slice of the input into 2x2 regions, and we pick the max value out of each region.
  - So, if the input has 26 rows and 26 columns, the output has 13 rows and 13 columns.
  - The third dimension in the output has size 32, like the input.

Summary of the Model

Output:

```
_________________________________________________________________
 Layer (type)                 Output Shape              Param #
=================================================================
conv2d_18 (Conv2D)           (None, 26, 26, 32)        320
max_pooling2d_18 (MaxPooling2D)  (None, 13, 13, 32)        0
conv2d_19 (Conv2D)           (None, 11, 11, 64)        18496
max_pooling2d_19 (MaxPooling2D)  (None, 5, 5, 64)          0
flatten_9 (Flatten)          (None, 1600)              0
dropout_9 (Dropout)          (None, 1600)              0
dense_9 (Dense)              (None, 10)                16010
=================================================================
```

- For the second Conv2D layer, why is the output shape 11x11x64?
- We created that layer with this line:

keras.layers.Conv2D(64, kernel_size=(3, 3), activation="relu")

Summary of the Model

Output:

```
_________________________________________________________________
 Layer (type)                 Output Shape              Param #
=================================================================
conv2d_18 (Conv2D)           (None, 26, 26, 32)        320
max_pooling2d_18 (MaxPooling2D)  (None, 13, 13, 32)        0
conv2d_19 (Conv2D)           (None, 11, 11, 64)        18496
max_pooling2d_19 (MaxPooling2D)  (None, 5, 5, 64)          0
flatten_9 (Flatten)          (None, 1600)              0
dropout_9 (Dropout)          (None, 1600)              0
dense_9 (Dense)              (None, 10)                16010
=================================================================
```

- For the second Conv2D layer, why is the output shape 11x11x64?
- The output of the previous layer has 13 rows and 13 columns.
- Convolving with a 3x3 kernel gives a result of 11 rows and 11 columns.
- We convolve with 64 kernels, so we get 64 arrays of size 11x11.

Summary of the Model

Output:

```
_________________________________________________________________
 Layer (type)                 Output Shape              Param #
=================================================================
conv2d_18 (Conv2D)           (None, 26, 26, 32)        320
max_pooling2d_18 (MaxPooling2D)  (None, 13, 13, 32)        0
conv2d_19 (Conv2D)           (None, 11, 11, 64)        18496
max_pooling2d_19 (MaxPooling2D)  (None, 5, 5, 64)          0
flatten_9 (Flatten)          (None, 1600)              0
dropout_9 (Dropout)          (None, 1600)              0
dense_9 (Dense)              (None, 10)                16010
=================================================================
```

- For the second Conv2D layer, why is the number of parameters 18496?
- We created that layer with this line:

```
keras.layers.Conv2D(64, kernel_size=(3, 3), activation="relu")
```

Summary of the Model

Output:

```
_________________________________________________________________
 Layer (type)                 Output Shape              Param #
=================================================================
conv2d_18 (Conv2D)           (None, 26, 26, 32)        320
max_pooling2d_18 (MaxPooling2D)  (None, 13, 13, 32)        0
conv2d_19 (Conv2D)           (None, 11, 11, 64)        18496
max_pooling2d_19 (MaxPooling2D)  (None, 5, 5, 64)          0
flatten_9 (Flatten)          (None, 1600)              0
dropout_9 (Dropout)          (None, 1600)              0
dense_9 (Dense)              (None, 10)                16010
=================================================================
```

- For the second Conv2D layer, why is the number of parameters 18496?
- We have 64 channels in that layer, 32 channel in the previous layer, and a 3x3 filter size.
- That gives us 64*32*3*3 = 18432 weights for the filter values.
- We also get 64 bias weights, one for each channel. 18432+64=18496.

Summary of the Model

Output:

```
_________________________________________________________________
 Layer (type)                 Output Shape              Param #
=================================================================
conv2d_18 (Conv2D)           (None, 26, 26, 32)        320
max_pooling2d_18 (MaxPooling2D)  (None, 13, 13, 32)        0
conv2d_19 (Conv2D)           (None, 11, 11, 64)        18496
max_pooling2d_19 (MaxPooling2D)  (None, 5, 5, 64)          0
flatten_9 (Flatten)          (None, 1600)              0
dropout_9 (Dropout)          (None, 1600)              0
dense_9 (Dense)              (None, 10)                16010
=================================================================
```

- For the second MaxPooling2D layer, why is the output shape 5x5x64?
- We created that layer with this line:

keras.layers.MaxPooling2D(pool_size=(2, 2))

Summary of the Model

Output:

```
_________________________________________________________________
 Layer (type)                 Output Shape              Param #
=================================================================
conv2d_18 (Conv2D)           (None, 26, 26, 32)        320
max_pooling2d_18 (MaxPooling2D)  (None, 13, 13, 32)        0
conv2d_19 (Conv2D)           (None, 11, 11, 64)        18496
max_pooling2d_19 (MaxPooling2D)  (None, 5, 5, 64)          0
flatten_9 (Flatten)          (None, 1600)              0
dropout_9 (Dropout)          (None, 1600)              0
dense_9 (Dense)              (None, 10)                16010
=================================================================
```

- For the second MaxPooling2D layer, why is the output shape 5x5x64?
  - We split each 2D slice of the input into 2x2 regions, and we pick the max value out of each region.
  - So, if the input has 11 rows and 11 columns, the output has 5 rows and 5 columns. (Values in the last row and last column of the input got ignored).
  - The third dimension in the output has size 64, like the input.

Summary of the Model

Output:

```
_________________________________________________________________
 Layer (type)                 Output Shape              Param #
=================================================================
conv2d_18 (Conv2D)           (None, 26, 26, 32)        320
max_pooling2d_18 (MaxPooling2D)  (None, 13, 13, 32)        0
conv2d_19 (Conv2D)           (None, 11, 11, 64)        18496
max_pooling2d_19 (MaxPooling2D)  (None, 5, 5, 64)          0
flatten_9 (Flatten)          (None, 1600)              0
dropout_9 (Dropout)          (None, 1600)              0
dense_9 (Dense)              (None, 10)                16010
=================================================================
```

- For the Flatten layer, why is the output shape 1600?
- We created that layer with keras.layers.Flatten().
- The Flatten layer simply converts the input to a 1D output.
- The input had 5*5*64 = 1600 values, so the output is a 1D array of size 1600.

Summary of the Model

Output:

```
_________________________________________________________________
 Layer (type)                 Output Shape              Param #
=================================================================
conv2d_18 (Conv2D)           (None, 26, 26, 32)        320
max_pooling2d_18 (MaxPooling2D)  (None, 13, 13, 32)        0
conv2d_19 (Conv2D)           (None, 11, 11, 64)        18496
max_pooling2d_19 (MaxPooling2D)  (None, 5, 5, 64)          0
flatten_9 (Flatten)          (None, 1600)              0
dropout_9 (Dropout)          (None, 1600)              0
dense_9 (Dense)              (None, 10)                16010
=================================================================
```

- For the Dropout layer, why is the output shape 1600?
- We created that layer with keras.layers.Dropout(0.5).
- The Dropout layer is not a real layer, it just tells Keras to ignore 50% of the outputs of the previous layer during training.
- This is why the output has the same size as the input.

Summary of the Model

Output:

```
_________________________________________________________________
 Layer (type)                 Output Shape              Param #
=================================================================
conv2d_18 (Conv2D)           (None, 26, 26, 32)        320
max_pooling2d_18 (MaxPooling2D)  (None, 13, 13, 32)        0
conv2d_19 (Conv2D)           (None, 11, 11, 64)        18496
max_pooling2d_19 (MaxPooling2D)  (None, 5, 5, 64)          0
flatten_9 (Flatten)          (None, 1600)              0
dropout_9 (Dropout)          (None, 1600)              0
dense_9 (Dense)              (None, 10)                16010
=================================================================
```

- In the Param # column, notice that there are zero parameters for MaxPooling, Flatten, and Dropout layers.
- That makes sense, these layers do not have any weights, they just perform some fixed mathematical functions.