CS/DS 541: Class 9

Jacob Whitehill

Group Exercise

https://www.wellandgood.com/missing-richard-simmons-podcast-fitness-history/

Group Exercise

Suppose you have a very deep NN (e.g., where the number of layers $l = 10000$). Let the element-wise activation function of every hidden layer be σ, and suppose that, given the current set of network weights $\mathcal{W} = \{\mathbf{W}^{(k)}\}_{k=1}^{l}$ and some input $\mathbf{x}$, the pre-activation vector $\mathbf{z}^{(k)}$ at each layer k contains all zeros. What happens to $\nabla_{\mathbf{W}^{(k)}} f(\mathbf{x}; \mathcal{W})$, where f is the loss function (e.g., MSE), as $k \to 0$, for the following 3 cases:

- $\sigma = \tanh$.
- $\sigma = \text{ReLU}$.
- $\sigma(z) = (1 + \exp(-z))^{-1}$, i.e., logistic sigmoid.

To get started, first (1) draw each of these activation functions on paper and/or the screen. Next, (2) determine each of their derivatives evaluated at 0. Finally, (3) think about how the chain-rule yields each $\nabla_{\mathbf{W}^{(k)}} f(\mathbf{x}; \mathcal{W})$, and how the corresponding product-of-Jacobians (some of which are directly affected by σ') changes for each of the three cases above.

Data normalization

Data normalization

- In many ML settings (not just DL), it is important or useful to normalize the input data to improve accuracy or ease of training.

- Common normalization methods include:
  - Mapping *each* feature into a fixed range (e.g., [0,1], [-1,1]).
  - Z-scoring *each* feature (so that the mean and stddev are 0 and 1, respectively).
  - Whitening *all* features jointly (to have 0 mean and *I*-covariance).

Data normalization

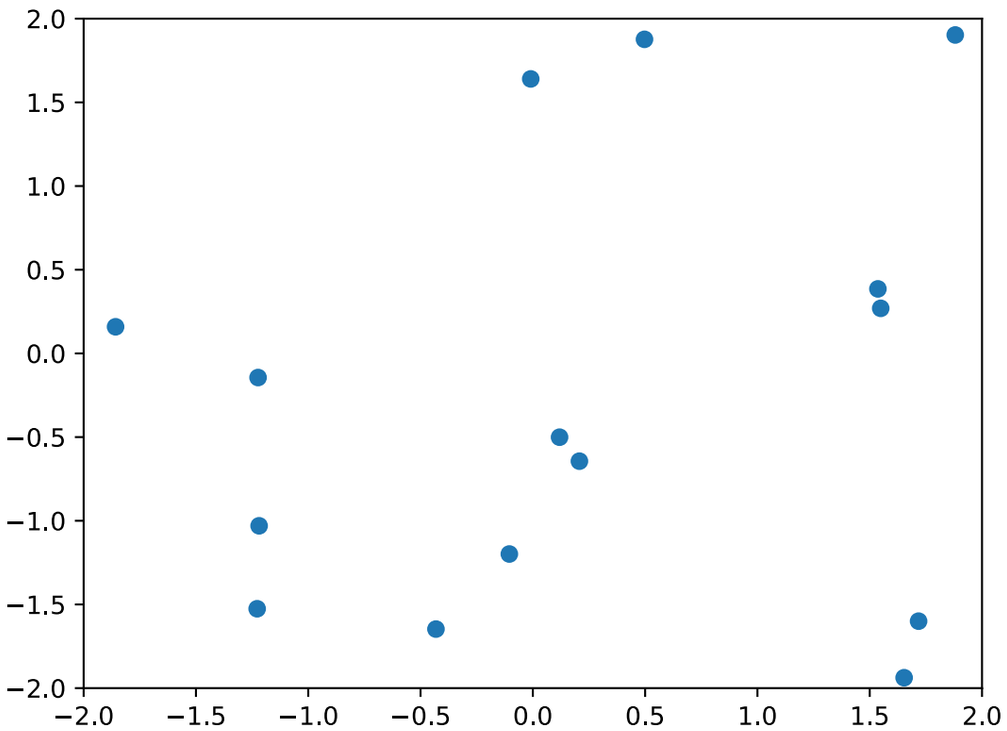
- There are (at least) two ways to achieve this.
- Strategy 1: Learn transformation on training data:
  - Compute the normalization parameters on the training set; save them.
  - Apply the normalization to the training data *and* each of the testing data.

Data normalization

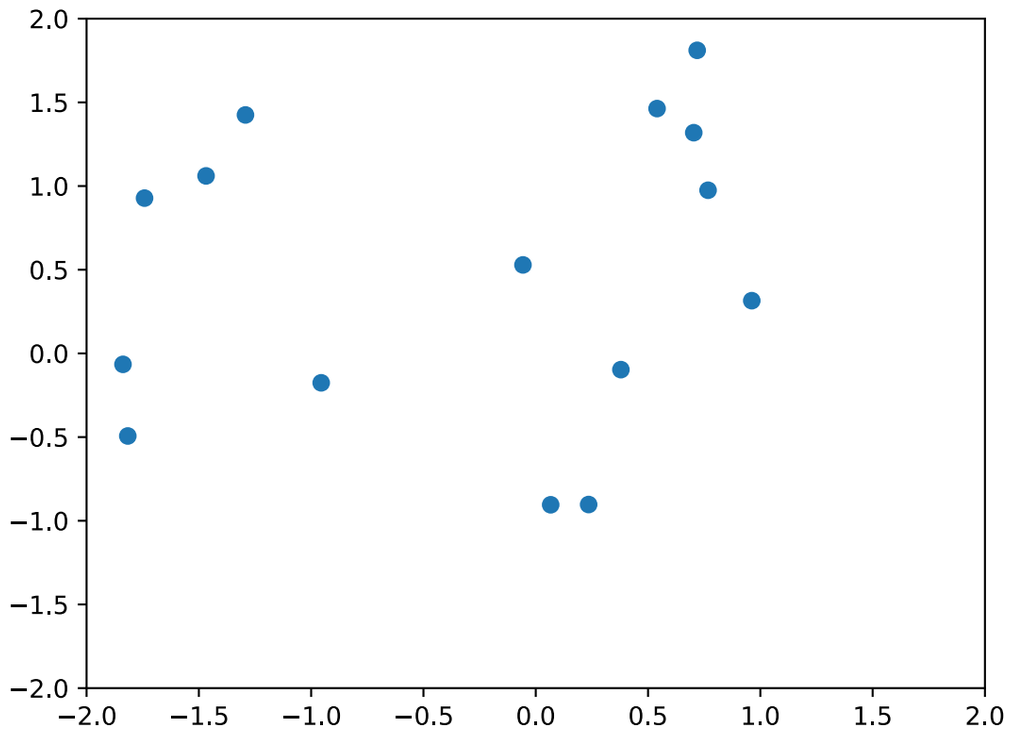
- There are (at least) two ways to achieve this.
- Strategy 2: Learn transformations for training and testing data *separately*:
  - Compute the normalization parameters on the *training* set, and apply it to each of the *training* data.
  - Compute the normalization parameters on the *testing* set, and apply it to each of the *testing* data.

Strategy 1

Unnormalized training data

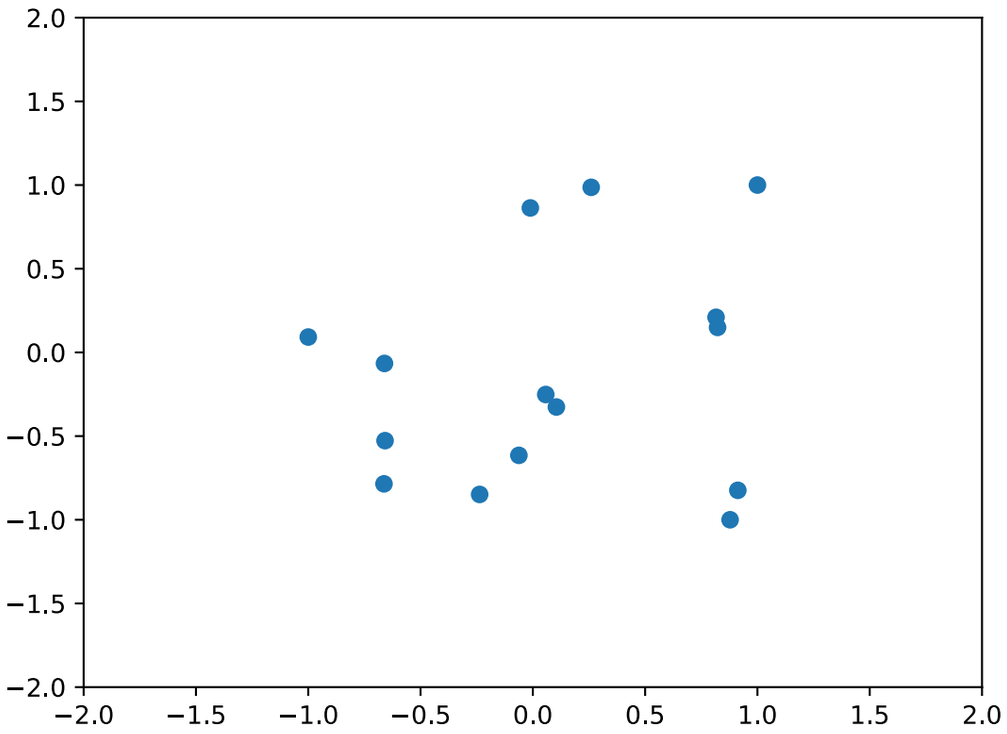


Unnormalized testing data

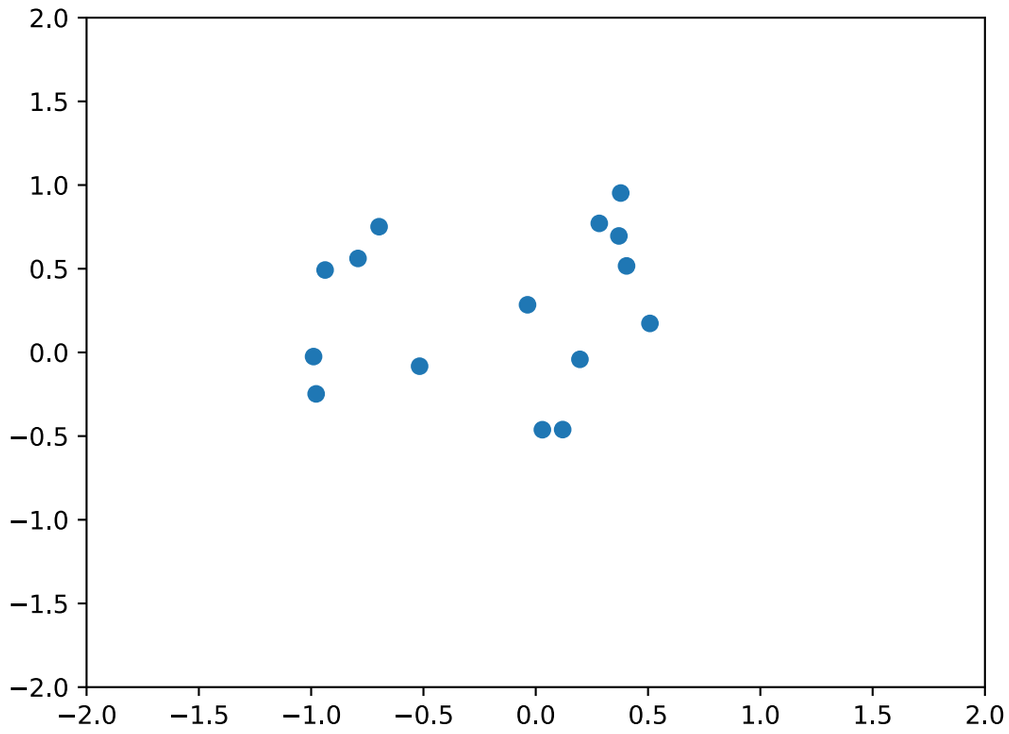


Strategy 1

Normalized training data

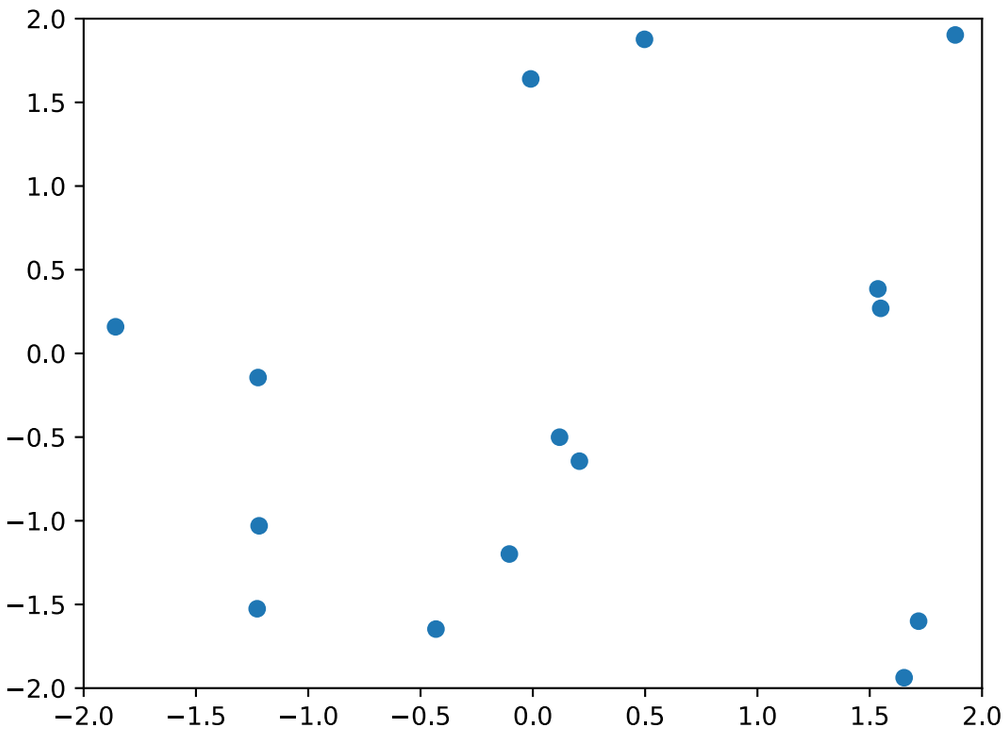


Normalized testing data

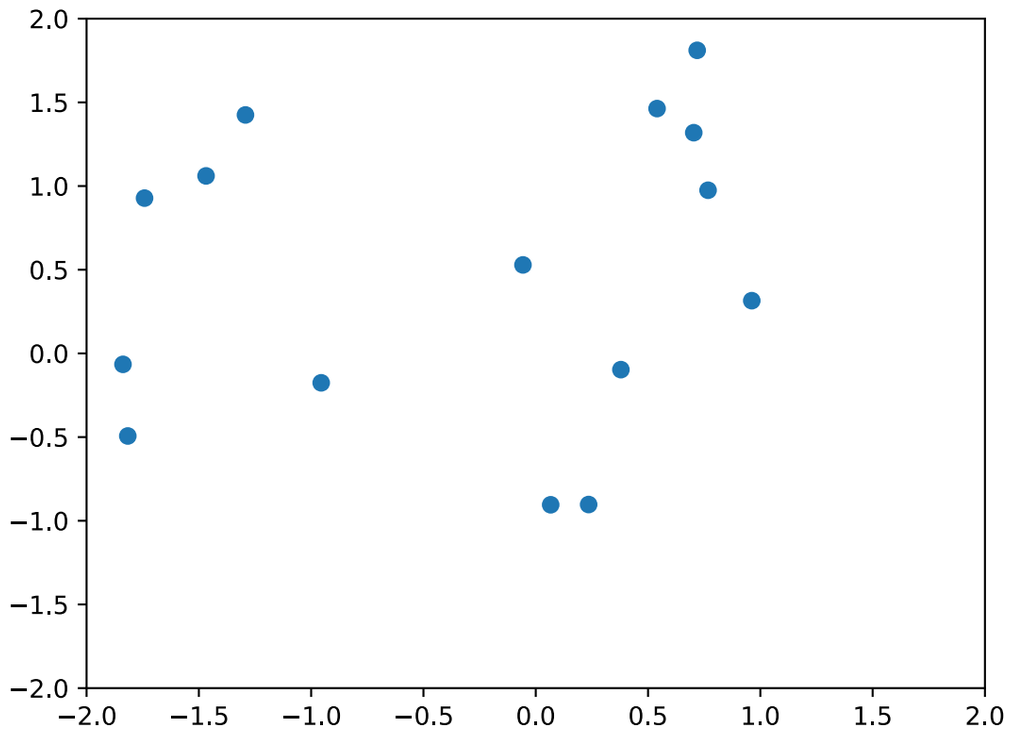


Strategy 2

Unnormalized training data

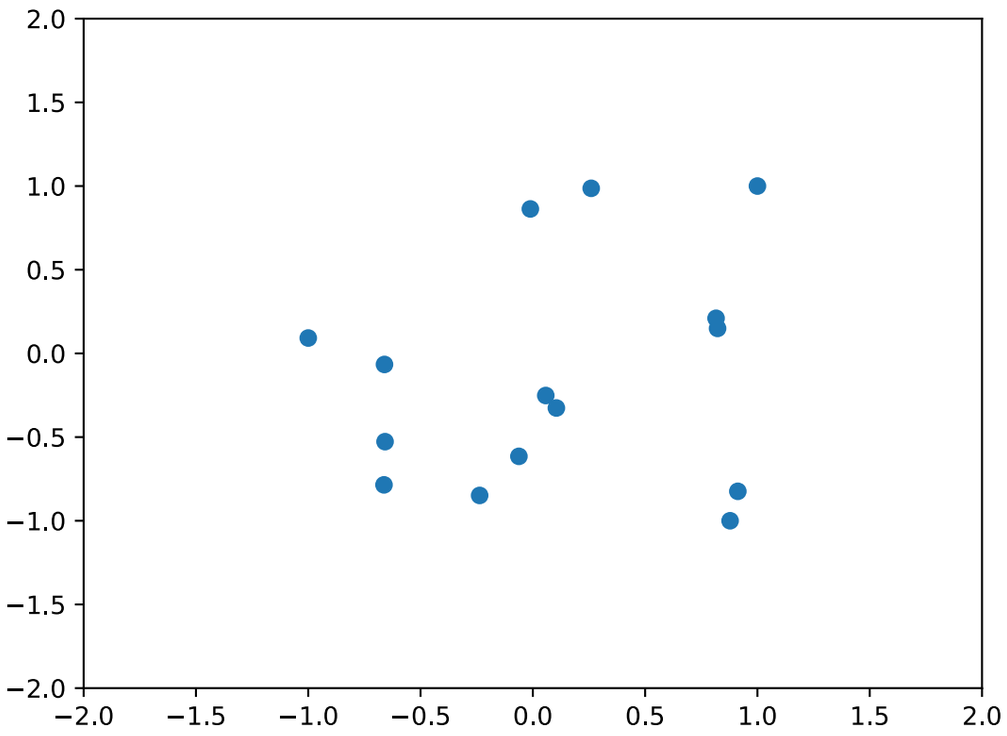


Unnormalized testing data

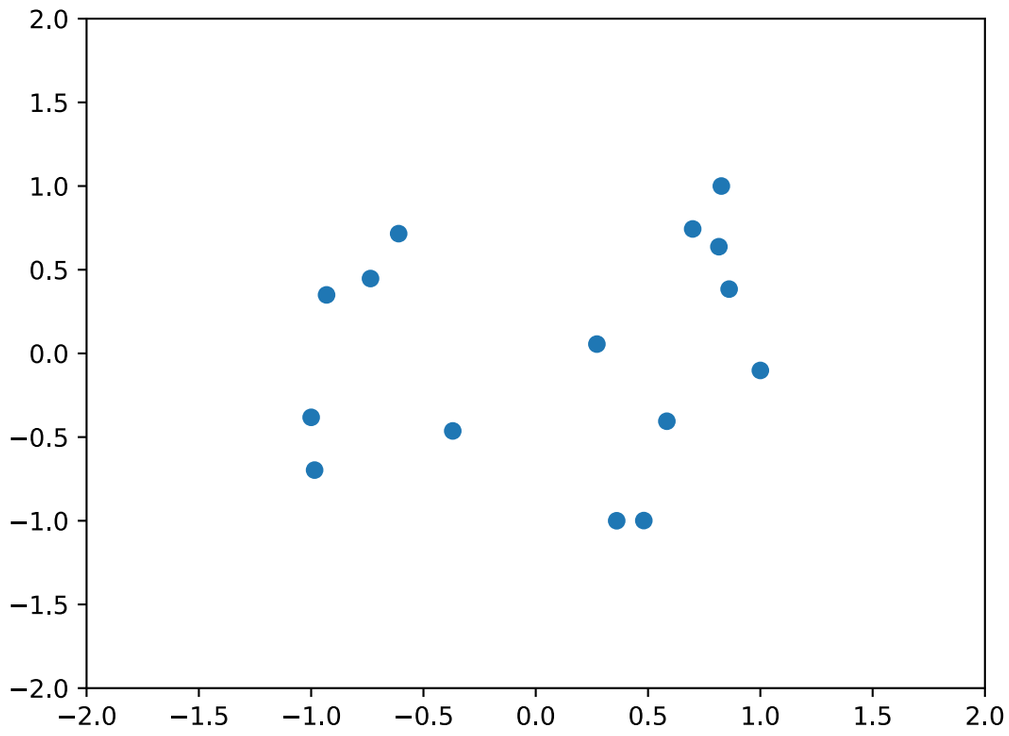


Strategy 2

Normalized training data



Normalized testing data



Strategy 1 vs. Strategy 2

- Strategy 2 results in normalized testing data that exactly fit the range [-1, +1].

Strategy 1 vs. Strategy 2

- Strategy 2 results in normalized testing data that exactly fit the range [-1, +1].

- However, normalizing in this way requires knowledge of the test distribution.

- This means it is necessary to collect a sufficiently large sample of testing data *before* running the classifier.

Data augmentation

Data augmentation

- The more training data you have, the less is the risk of overfitting.
- Unfortunately, training data are often hard to find.
- Can we synthesize new training examples automatically?

Data augmentation

- **Data augmentation** is the creation of new examples based on existing ones.

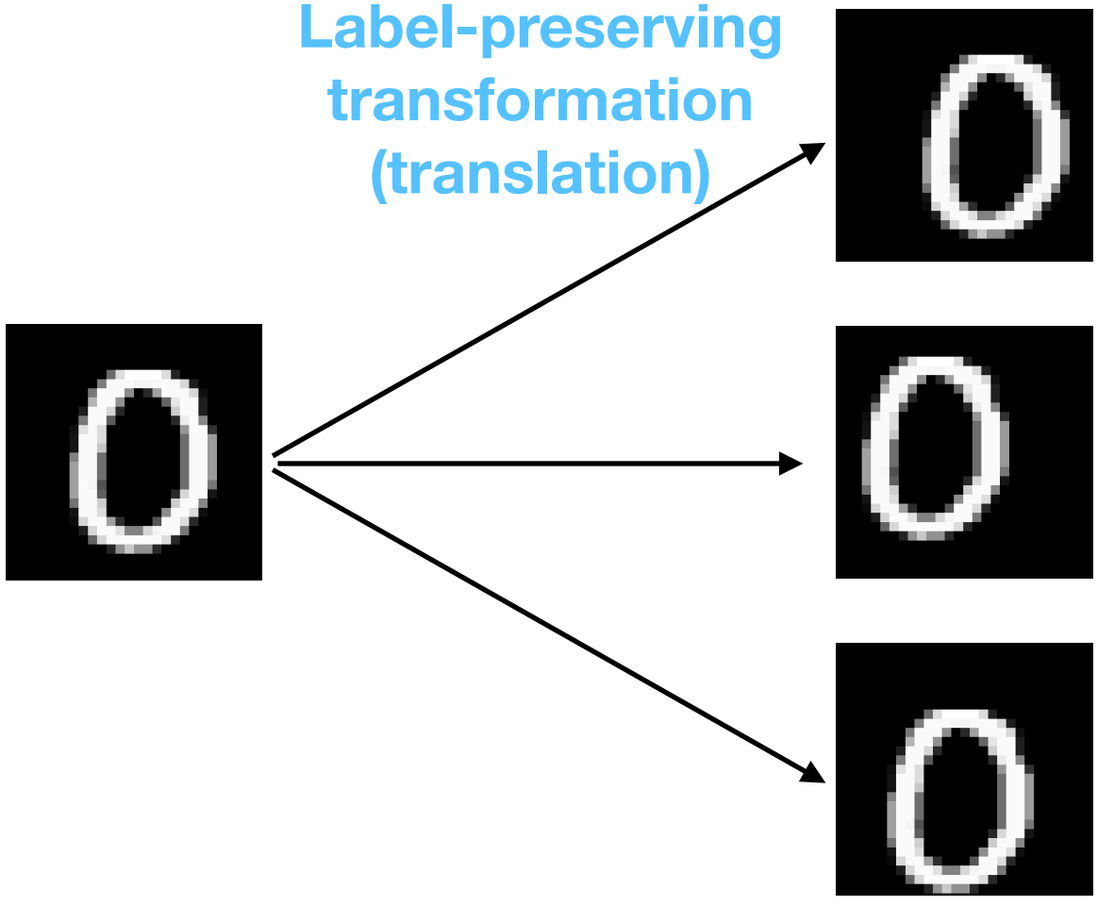
- If we can alter an existing training example without affecting its associated label, then we can generate many new training examples and train on them.

Data augmentation

- Several commonly used methods of data augmentation:
  - Adding noise to existing examples (e.g., Gaussian, Laplacian).
  - Geometric transformations (e.g., flip left/right, rotate, translate).

Example: translation

- From an existing MNIST image, translate all the pixels by some random amount (dx, dy).



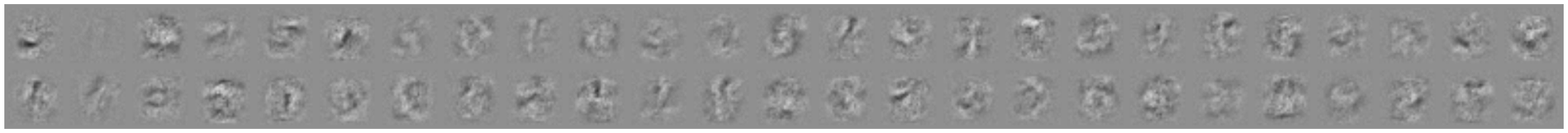
Example: translation

- Data augmentation via translation encourages the NN to learn **translation-invariant** features — they are useful for classification no matter where in the image they occur.

Example: translation

- Here are the weights $\mathbf{W}^{(1)}$ (transformed to 100x28x28) of a MNIST classification network **without** data augmentation:

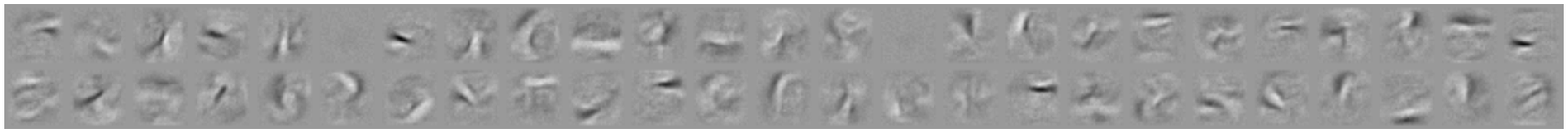
Acc=98.07



Example: translation

- Here are the weights $\mathbf{W}^{(1)}$ (transformed to 100x28x28) of a MNIST classification network **with** data augmentation:

Acc=98.44



- Compared to the previously shown weights, these show visually more well-defined contours.

(Supervised)
pre-training

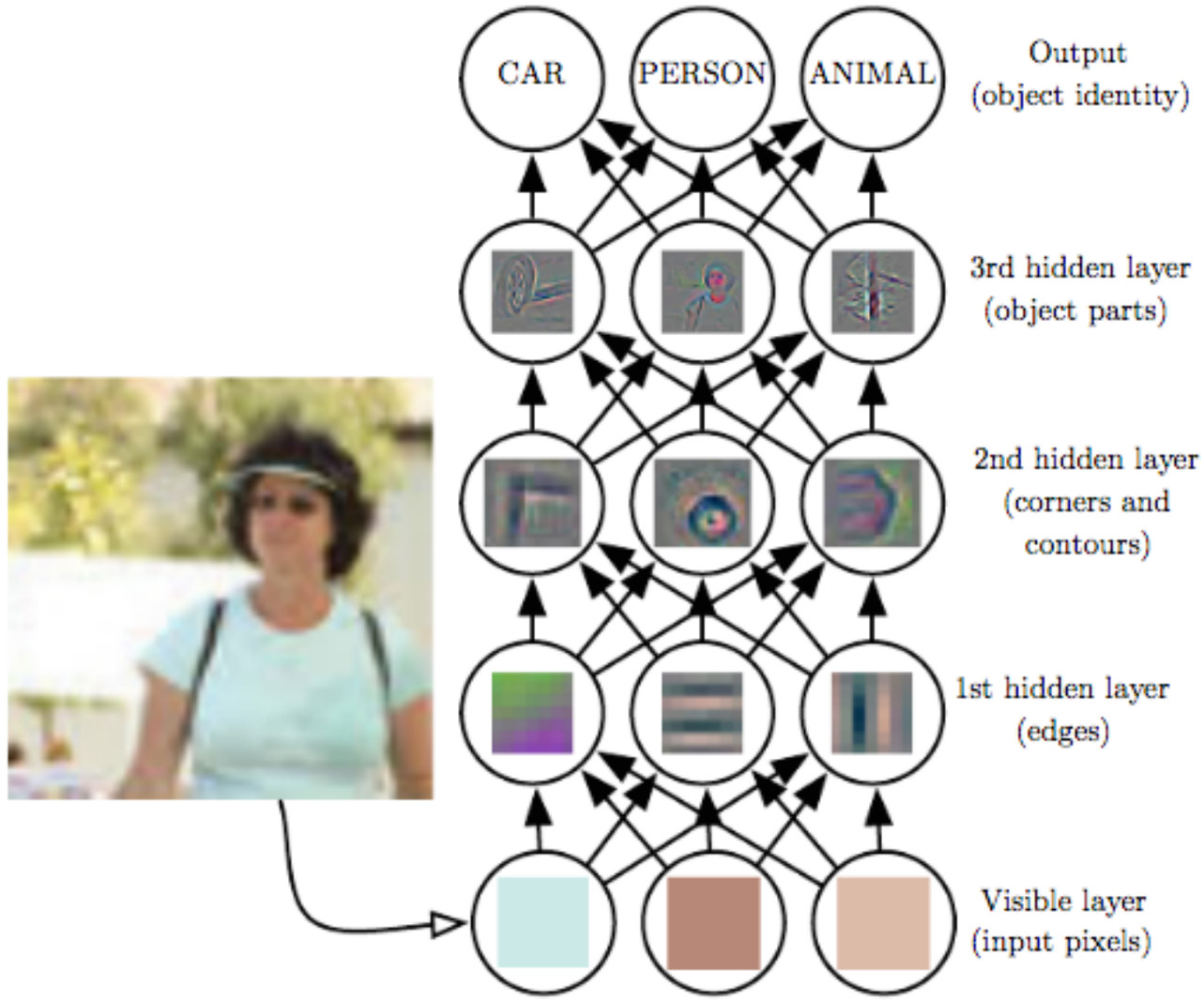
Supervised pre-training

- An alternative strategy to finding good feature representations is to borrow a NN from a related task.

- For instance, there now exist high-accuracy networks for recognizing 1000+ object categories from images (next slide).

- We can “borrow” the feature representation from one ML model and apply it to another application domain…

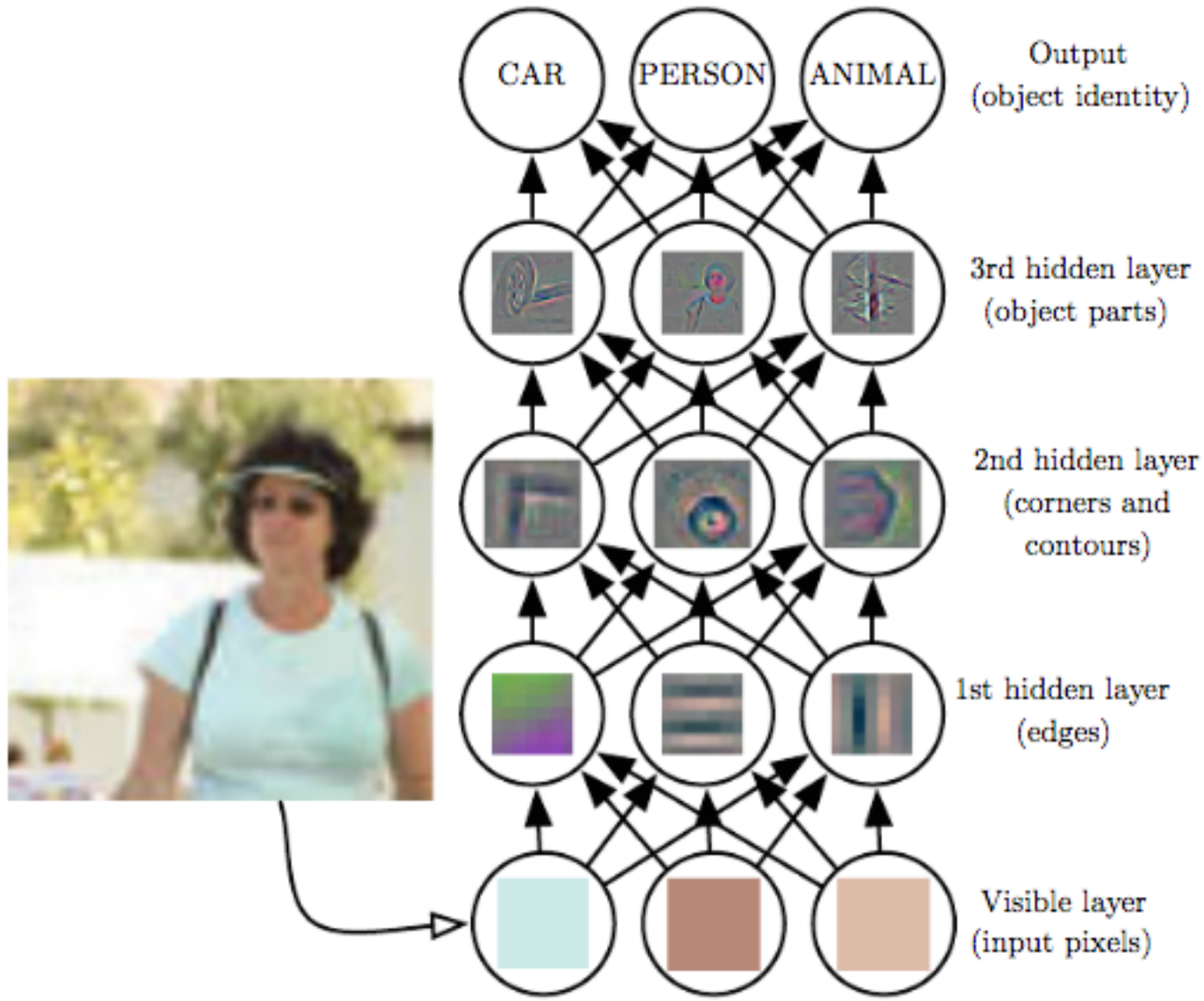
Learning representations



- The first feature representation looks vaguely like the representation learned by my MNIST network.

http://www.deeplearningbook.org/contents/intro.html

Learning representations



- Each layer of the network finds successively more abstract feature representations.
  - This was not “hard-coded” — it just turned out that these representations were useful for predicting the target labels.

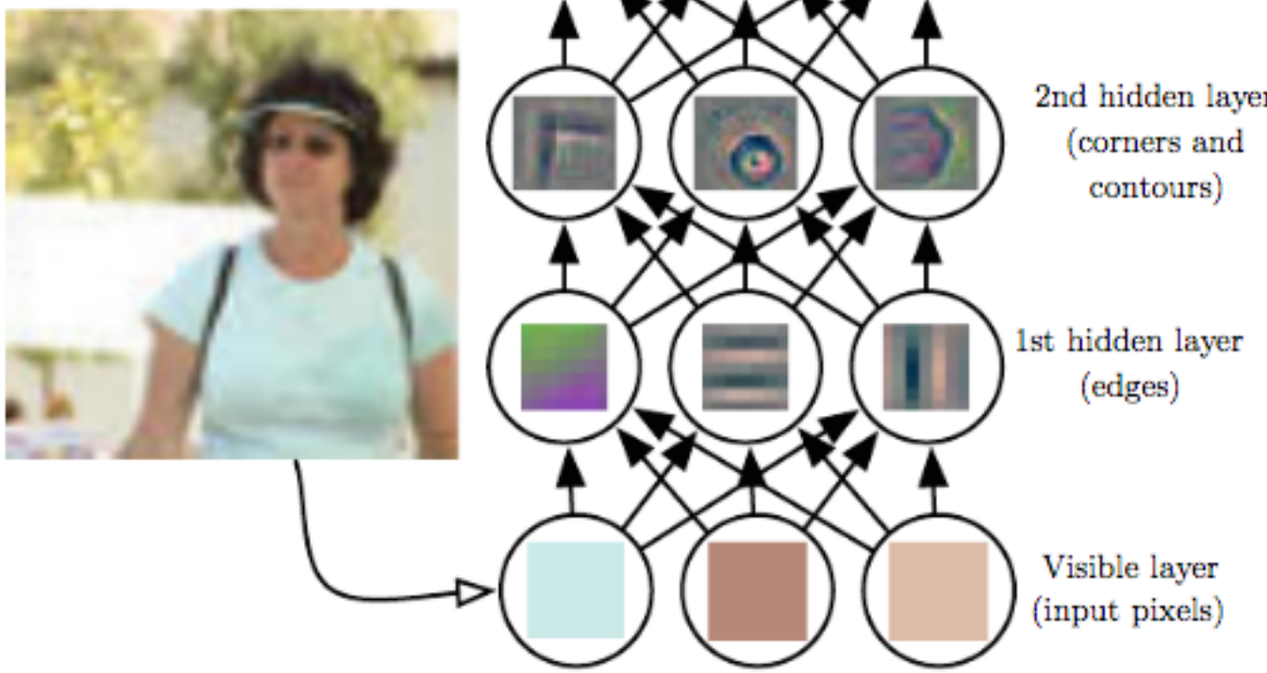
http://www.deeplearningbook.org/contents/intro.html

Supervised pre-training

- Might one (or more) of the feature representations from this NN do well on a different but related problem, e.g., smile detection or age estimation?
- Strategy:
  1. Pre-train a NN on a large dataset (e.g., ImageNet) for a general-purpose image recognition task.
  2. “Chop off” the final layer(s).
  3. Add a secondary network in place of the deleted layers, and train it for the new prediction task.
  4. Optional: fine-tune the rest of the NN.

Supervised pre-training

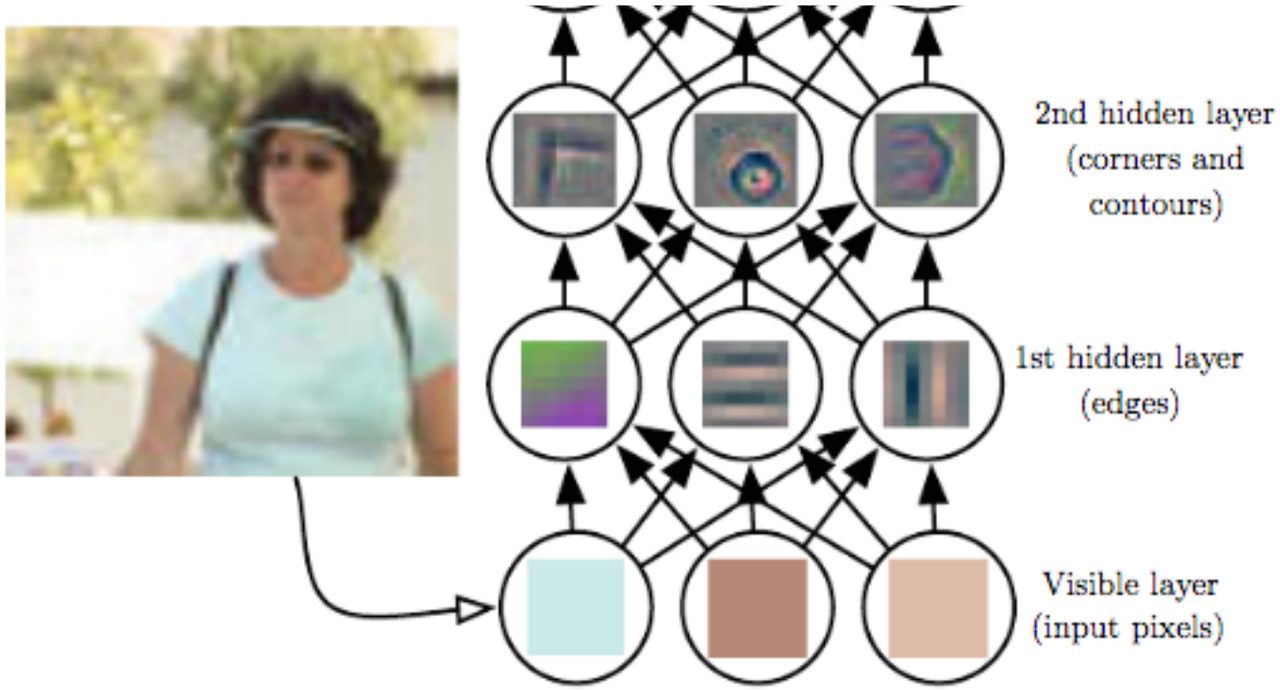
Replace with secondary network for new application domain.



http://www.deeplearningbook.org/contents/intro.html

Supervised pre-training

Chop off.



http://www.deeplearningbook.org/contents/intro.html

Supervised pre-training

- This strategy is known as **supervised pre-training** and can be highly effective for application domains for which only a small number of labeled data are available.

Multi-task learning (MTL)

Multi-task learning (MTL)

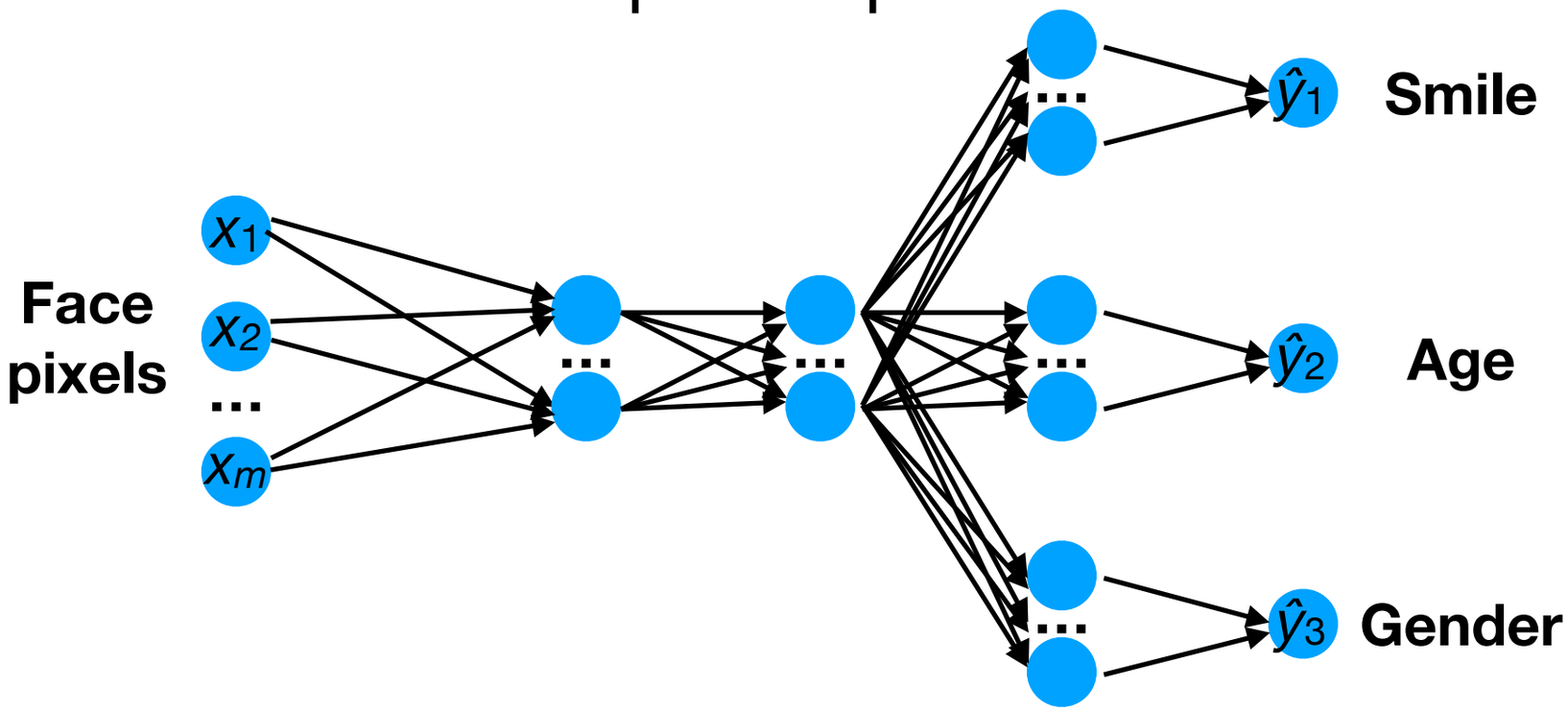
- A NN can generalize to unseen data when it computes a hidden representation that explains the data well.

- We can sometimes encourage the NN to learn a general hidden representation by training it to solve multiple related tasks.

- E.g., for automated face analysis:
  - Smile detection
  - Age estimation
  - Gender detection

Multi-task learning (MTL)

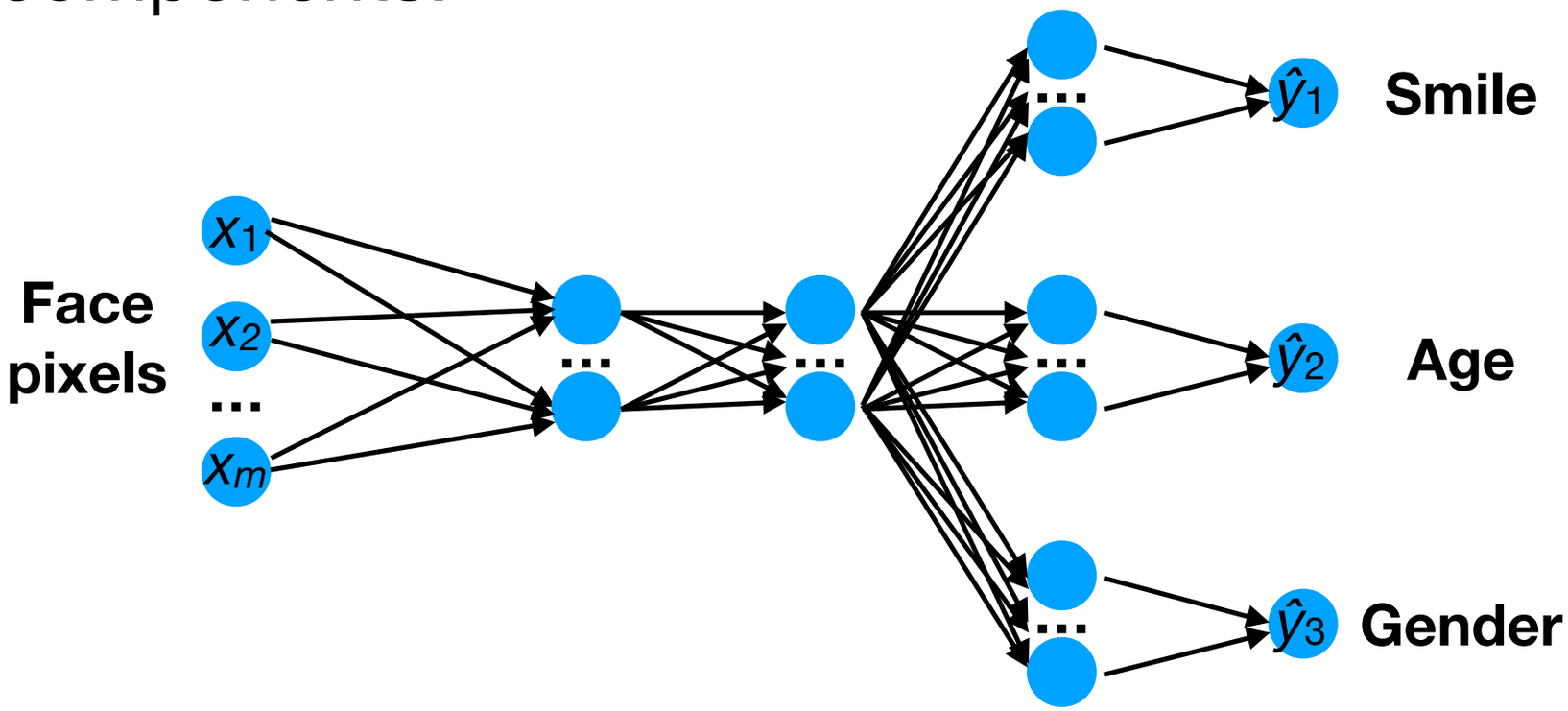
- If we have labeled data for all these tasks, we can train a network with multiple outputs:



- Note that the labels for the auxiliary tasks can be useful even if we only care about one task.

Multi-task learning (MTL)

- With MTL, our loss function consists of multiple components:



$$f_{\text{MTL}}(\mathbf{W}^{(1)}, \mathbf{b}^{(1)}, \ldots) = f_{\text{smile}}(\mathbf{W}^{(1)}, \mathbf{b}^{(1)}, \ldots) + f_{\text{age}}(\mathbf{W}^{(1)}, \mathbf{b}^{(1)}, \ldots) + f_{\text{gender}}(\mathbf{W}^{(1)}, \mathbf{b}^{(1)}, \ldots)$$

Fairness in ML

Fairness in ML

- Suppose we are training a classifier to perceive whether a person is smiling based on their face image.


vs.


- Suppose the joint probability distribution of our training labels is:

	Male	Female
Smile	0.47	0.02
Non-smile	0.45	0.06

Almost all data are male faces

Fairness in ML

- Suppose we are training a classifier to perceive whether a person is smiling based on their face image.


vs.


- Suppose the joint probability distribution of our training labels is:

	Male	Female
Smile	0.47	0.02
Non-smile	0.45	0.06

Almost all female faces are non-smiling

Fairness in ML

- To minimize prediction error, the training algorithm can harness the fact that *most women do not smile* ***in this dataset***.


vs.


	Male	Female
Smile	0.47	0.02
Non-smile	0.45	0.06

Fairness in ML

- At test time, that machine might implicitly infer that the face is female, and then use that to “downgrade” the estimate of the smile probability.


vs.


	Male	Female
Smile	0.47	0.02
Non-smile	0.45	0.06

Fairness in ML

- In contrast, a male face has a roughly equal chance of being classified as smile/ non-smile.

vs.

	Male	Female
Smile	0.47	0.02
Non-smile	0.45	0.06

Fairness in ML

- Consider the following definition of ML fairness:
  - For all subgroups (i.e., smiling males, non-smiling males, smiling females, non-smiling females), the prediction accuracy should be equal.

Fairness in ML

- Consider the following definition of ML fairness:
  - For all subgroups (i.e., smiling males, non-smiling males, smiling females, non-smiling females), the prediction accuracy should be equal.
- Then the classifier described above is unfair because female faces would likely be perceived less accurately compared to male faces.

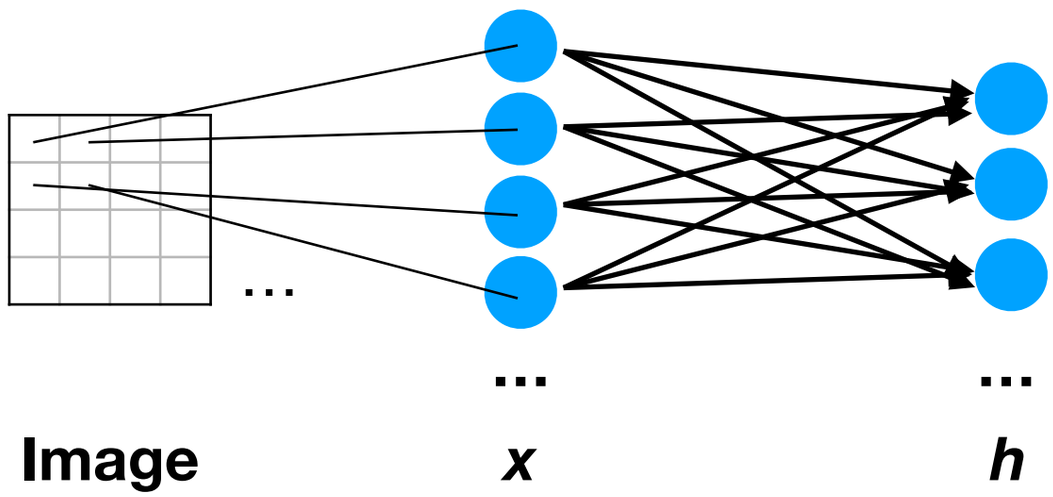
Fairness in ML

- Mitigating strategies:
    - Collect more training data for specific populations.
    - Train the classifier with structural constraints that prevent the exploitation of correlated but non-causal features:
        - In this dataset, gender is correlated with smile, but does not cause smile!

Image features

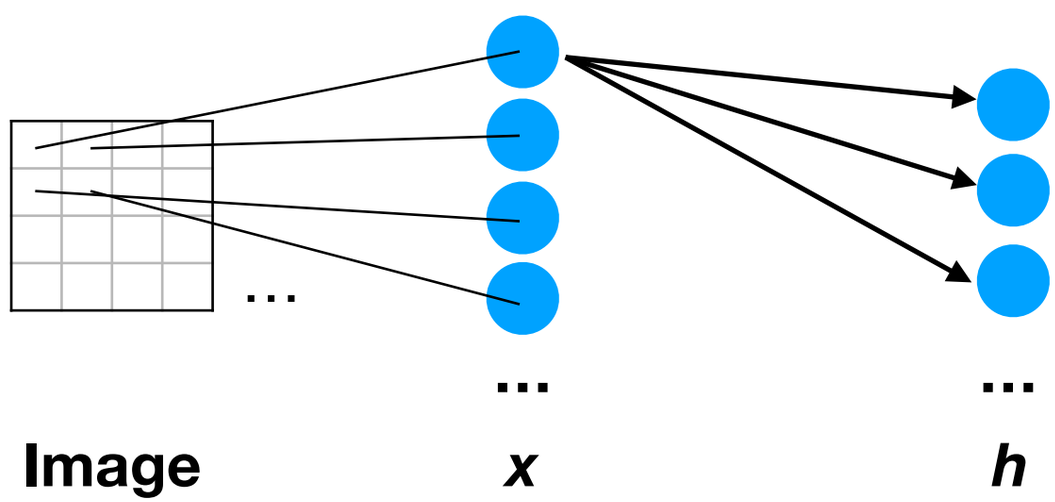
2-d spatial structure in images

- So far we have examined feed-forward neural networks that are fully connected, i.e., every neuron in layer *l* is connected to every neuron in layer *l*+1.



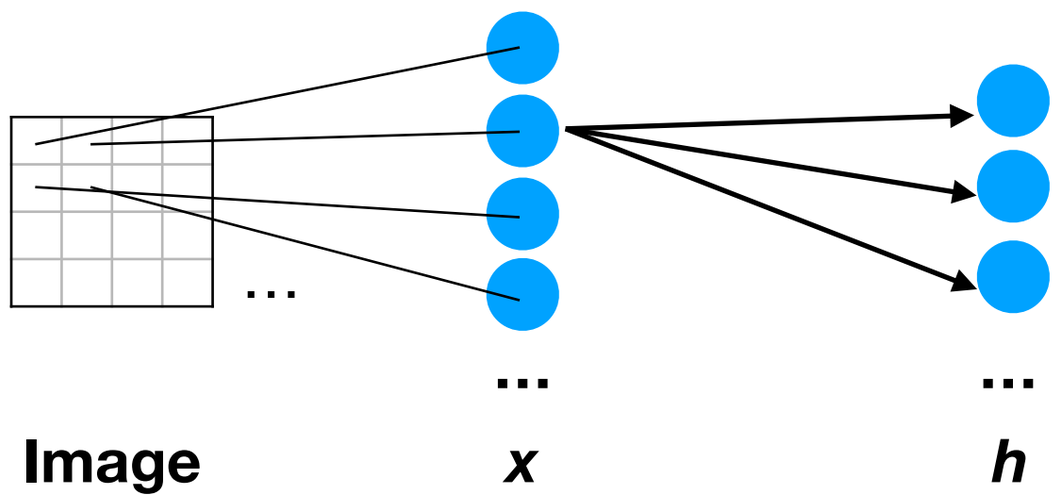
2-d spatial structure in images

- The weights from each neuron *i* in layer *l* are completely independent of the weights from every other neuron *i*'.



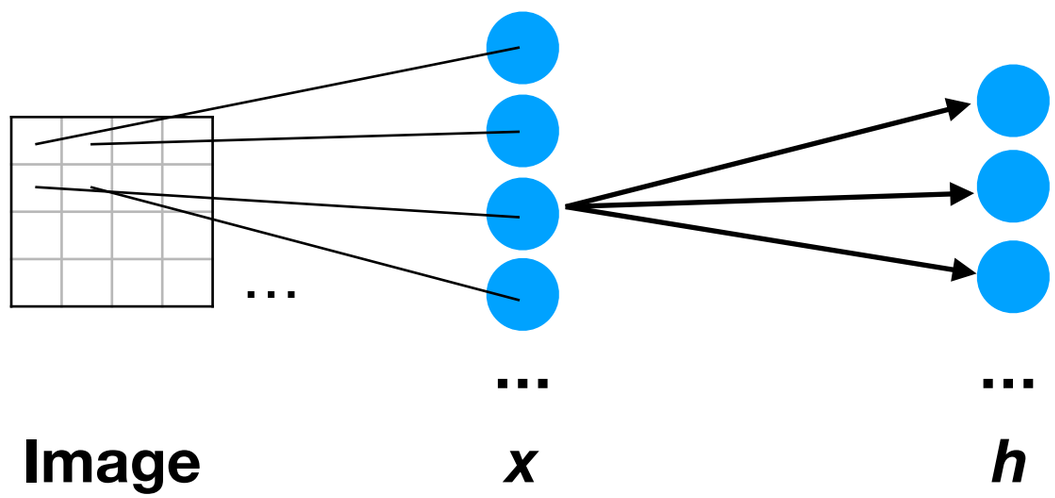
2-d spatial structure in images

- The weights from each neuron *i* in layer *l* are completely independent of the weights from every other neuron *i*'.



2-d spatial structure in images

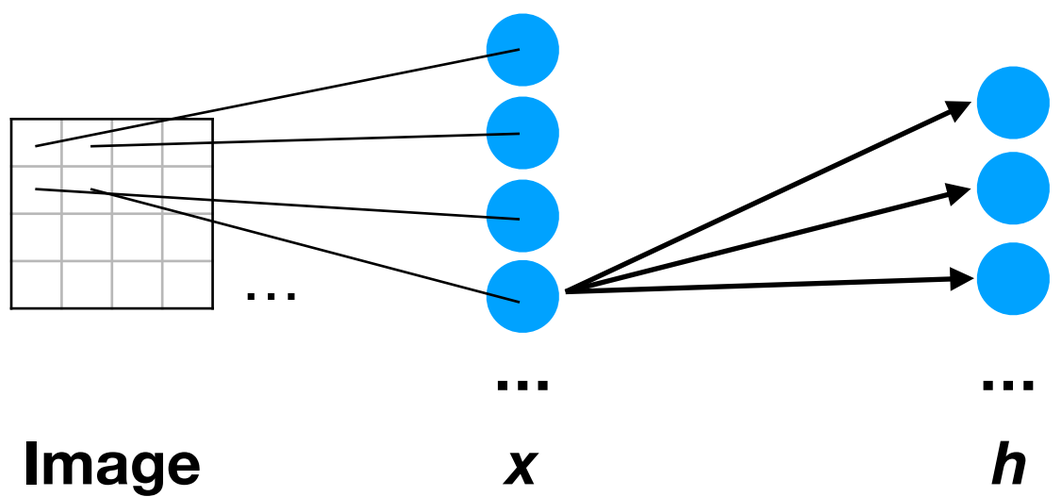
- The weights from each neuron *i* in layer *l* are completely independent of the weights from every other neuron *i*'.



2-d spatial structure in images

- The weights from each neuron *i* in layer *l* are completely independent of the weights from every other neuron *i*'.

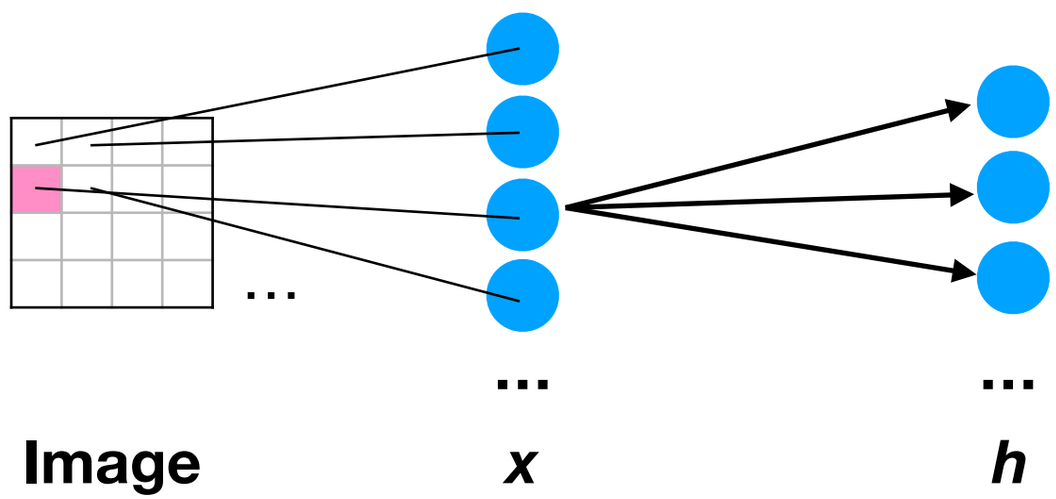
  - But does this make sense for images with a spatial structure?



2-d spatial structure in images

- The weights from each neuron *i* in layer *l* are completely independent of the weights from every other neuron *i*'.

  - A tiny shift in the input image can result in arbitrarily large change in the hidden unit activations.

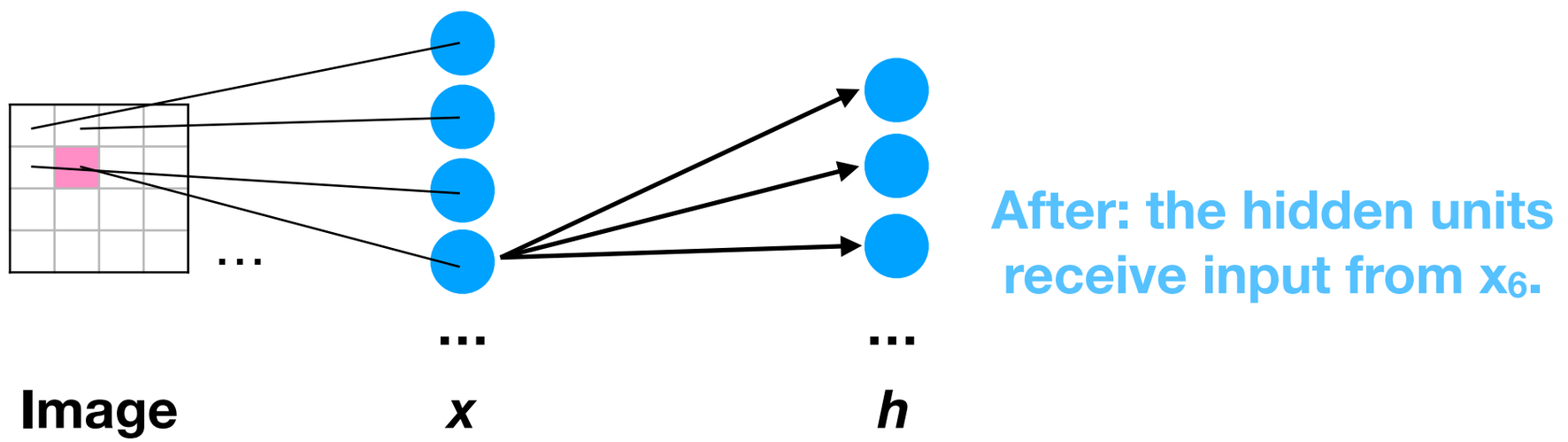


Before: the hidden units receive input from x_5.

2-d spatial structure in images

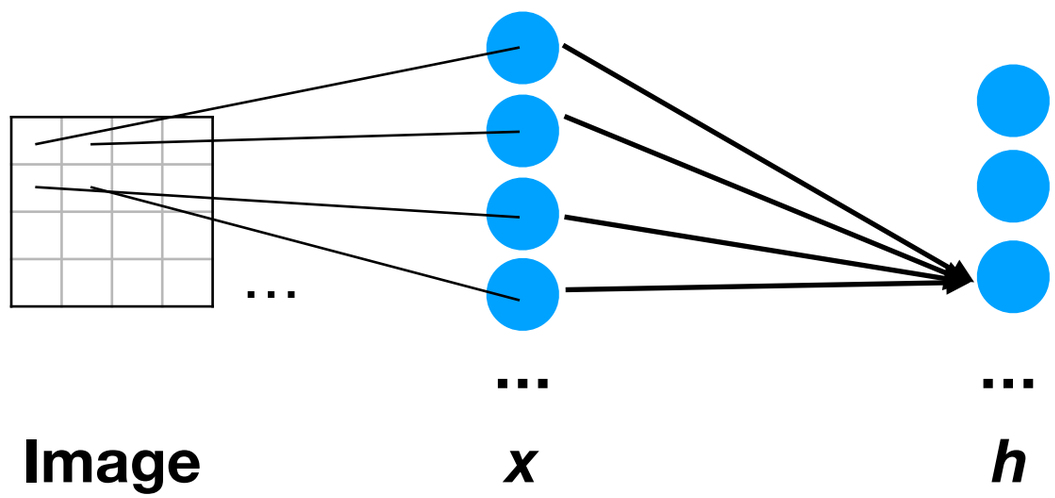
- The weights from each neuron *i* in layer *l* are completely independent of the weights from every other neuron *i*'.

  - A tiny shift in the input image can result in arbitrarily large change in the hidden unit activations.



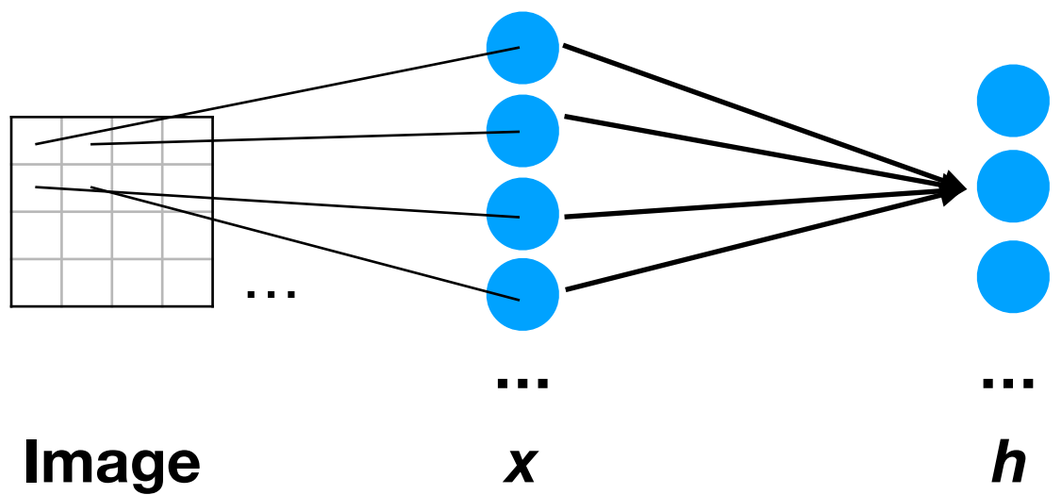
2-d spatial structure in images

- Also, the activation of each neuron *i* in layer *l*+1 is completely independent of the activation of every other neuron *i*'.



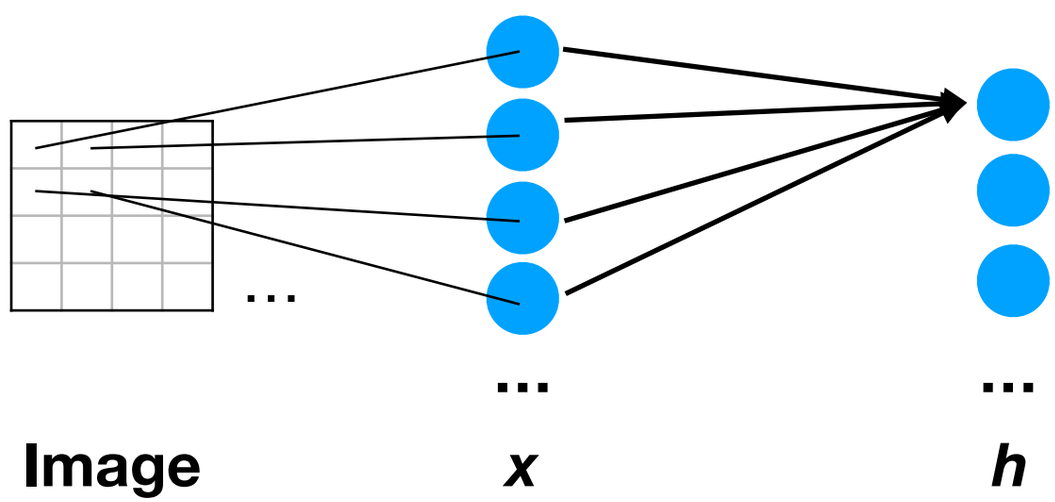
2-d spatial structure in images

- Also, the activation of each neuron *i* in layer *l*+1 is completely independent of the activation of every other neuron *i*'.



2-d spatial structure in images

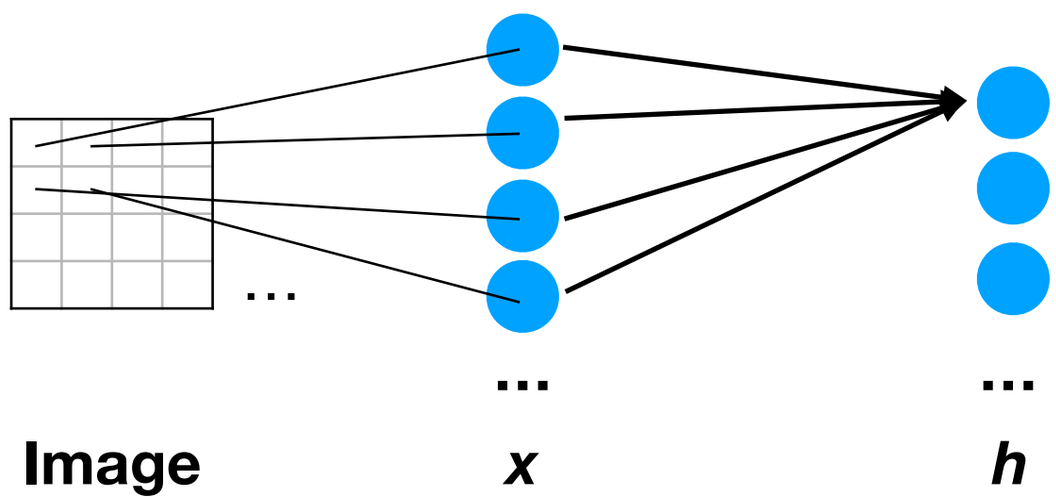
- Also, the activation of each neuron *i* in layer *l*+1 is completely independent of the activation of every other neuron *i*'.



2-d spatial structure in images

- Also, the activation of each neuron *i* in layer *l*+1 is completely independent of the activation of every other neuron *i*'.

- I.e., the hidden layer(s) have no spatial structure — the index *i* of each hidden unit is meaningless.



2-d spatial structure in images

- Throwing away the 2-d spatial structure is akin to trying to classify an image after applying some arbitrary (but fixed) permutation to it:

Pixel order:
1, 2, 3, …, 576

2-d spatial structure in images

- Throwing away the 2-d spatial structure is akin to trying to classify an image after applying some arbitrary (but fixed) permutation to it:

Pixel order:
1, 2, 3, …, 576

Pixel order:
25, 305, 213, …, 509

2-d spatial structure in images

- Images tend to contain the *same* visual features — lines and corners of different colors (see MNIST demo) — at *many* locations and scales.

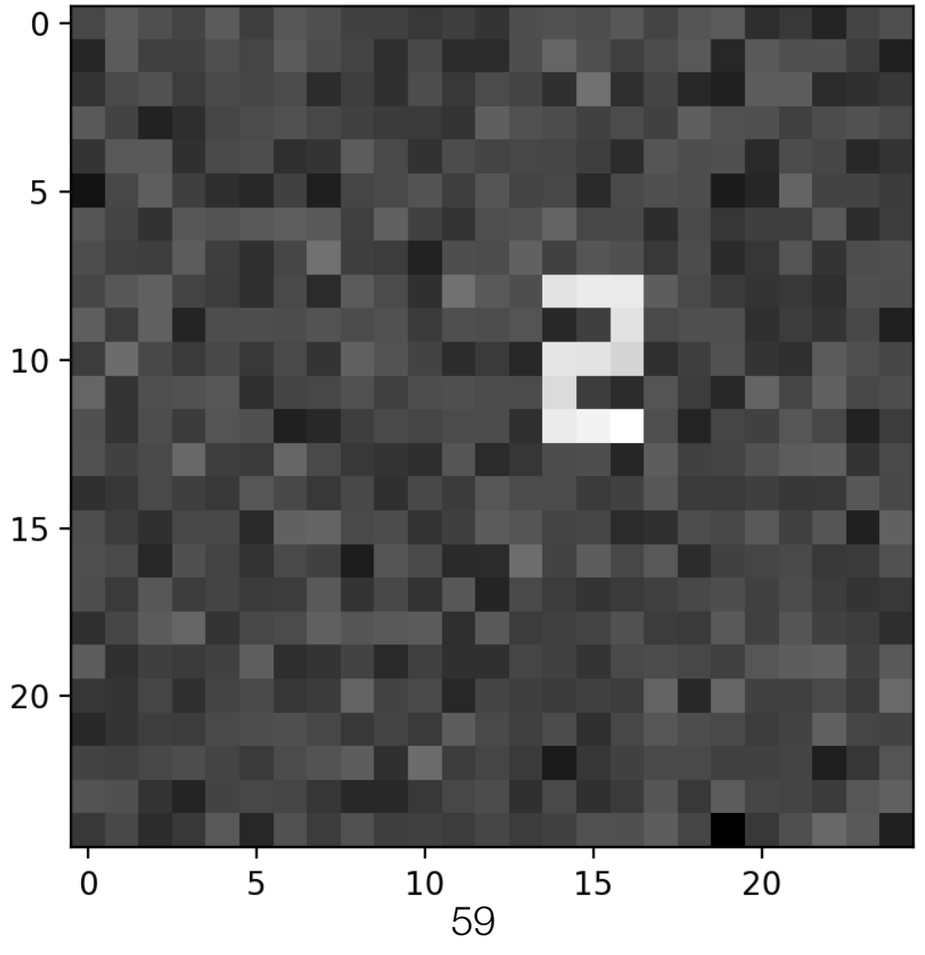
- By harnessing the spatial structure of images, we can train NNs with far fewer weights.

  - This is a powerful form of regularization.

Convolution

Motivation: object detection

- Suppose we wanted to classify an unknown digit of unknown location in an image.



Motivation: object detection

- One simple strategy is based on an old computer vision technique known as template matching:
  - We create a template image that equals one of the objects we want to detect.
  - We “slide” the template across every location in the image and see where the image & template best “match”.

Convolution

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

Image

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

Template

Output

- Without exceeding the image boundaries, how many 2-D locations are there at which the template can be superimposed with the image?

Convolution

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

Image

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

Template

Output

- Without exceeding the image boundaries, how many 2-D locations are there at which the template can be superimposed with the image?

4 rows, 6 columns

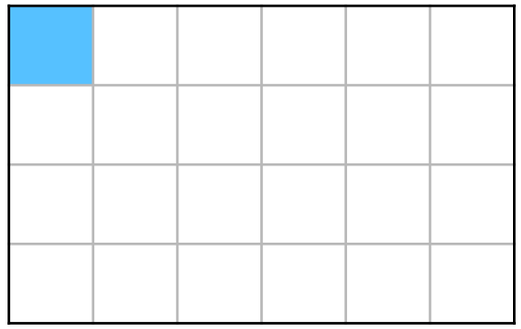
Convolution

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

Image

	-1	0	+1
-2	1	1	1
-1	0	0	1
0	1	1	1
+1	1	0	0
+2	1	1	1

Template



Output

- For simplicity of notation, let's renumber the indices of the elements of the template.

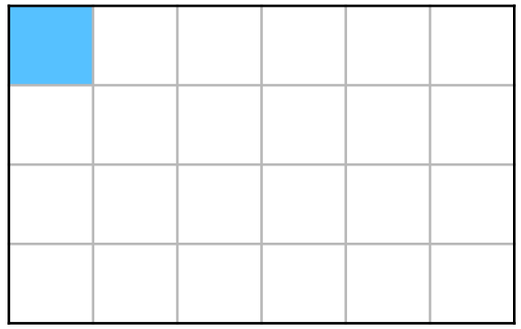
Convolution

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

Image

	-1	0	+1
-2	1	1	1
-1	0	0	1
0	1	1	1
+1	1	0	0
+2	1	1	1

Template



Output

- The output of the “template matching” at (*r*, *c*) is then the sum of products between each template pixel and the corresponding image pixel:

$$\mathrm{TM}(r,c) = \sum_{i=-t_h/2}^{+t_h/2} \sum_{j=-t_w/2}^{+t_w/2} \mathrm{im}[r+i, c+j]\mathrm{t}[i,j]$$

where t_h, t_w are the height and width of the template.

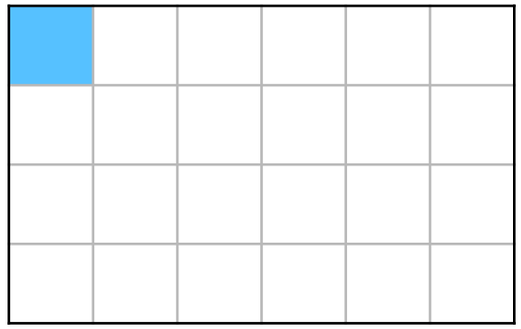
Convolution

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

Image

	-1	0	+1
-2	1	1	1
-1	0	0	1
0	1	1	1
+1	1	0	0
+2	1	1	1

Template



Output

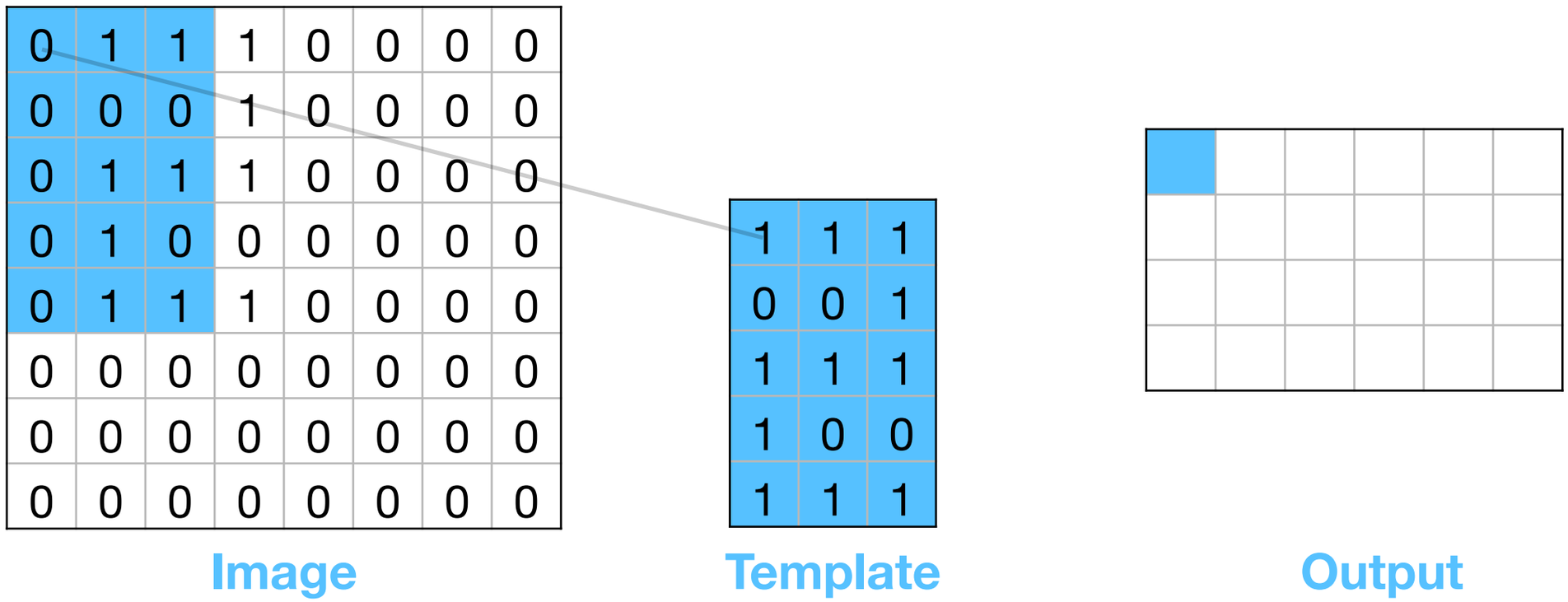
- At each (*r*, *c*), this can be thought of as a 2-D "dot product" between the image and the template. Over the whole image, it is also known as a 2-D **convolution**.*

$$\mathrm{TM}(r,c) = \sum_{i=-t_h/2}^{+t_h/2} \sum_{j=-t_w/2}^{+t_w/2} \mathrm{im}[r+i, c+j]\mathrm{t}[i,j]$$

where t_h, t_w are the height and width of the template.

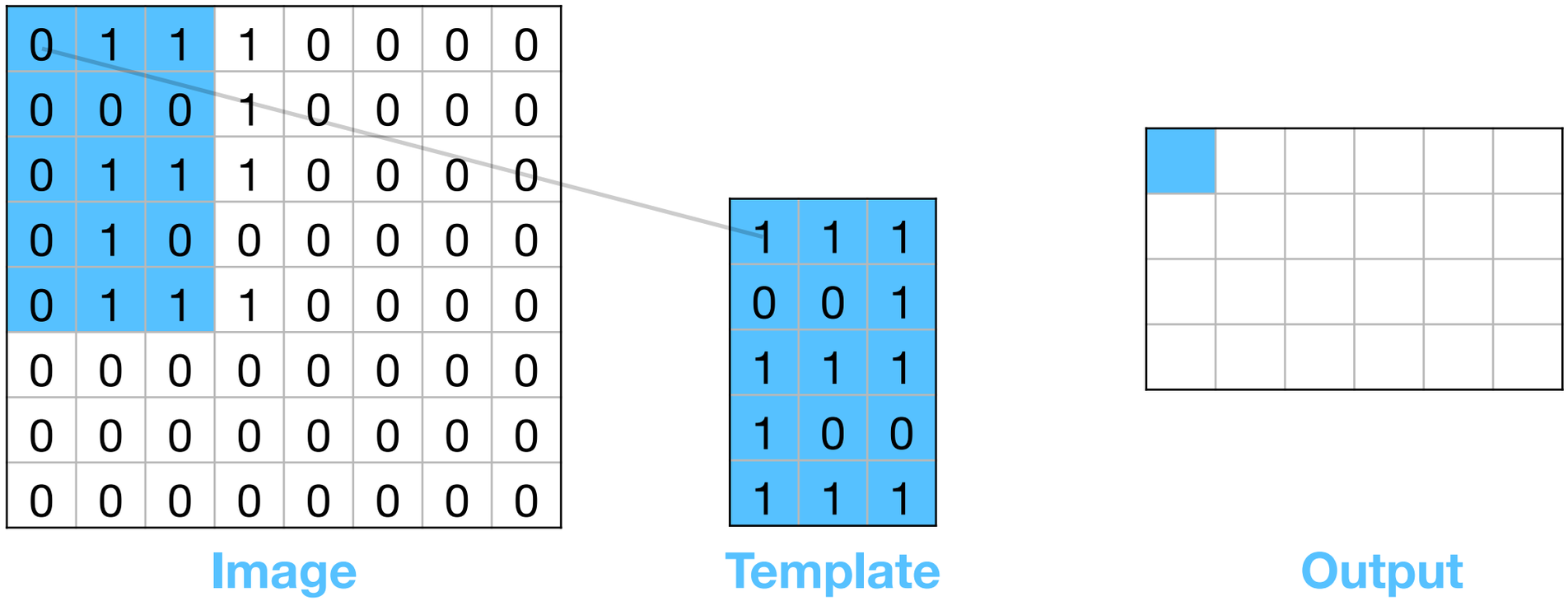
*Technically it is a cross-correlation; convolution would be *r-i*, *c-j*. But with NNs, the difference is not important.

Convolution



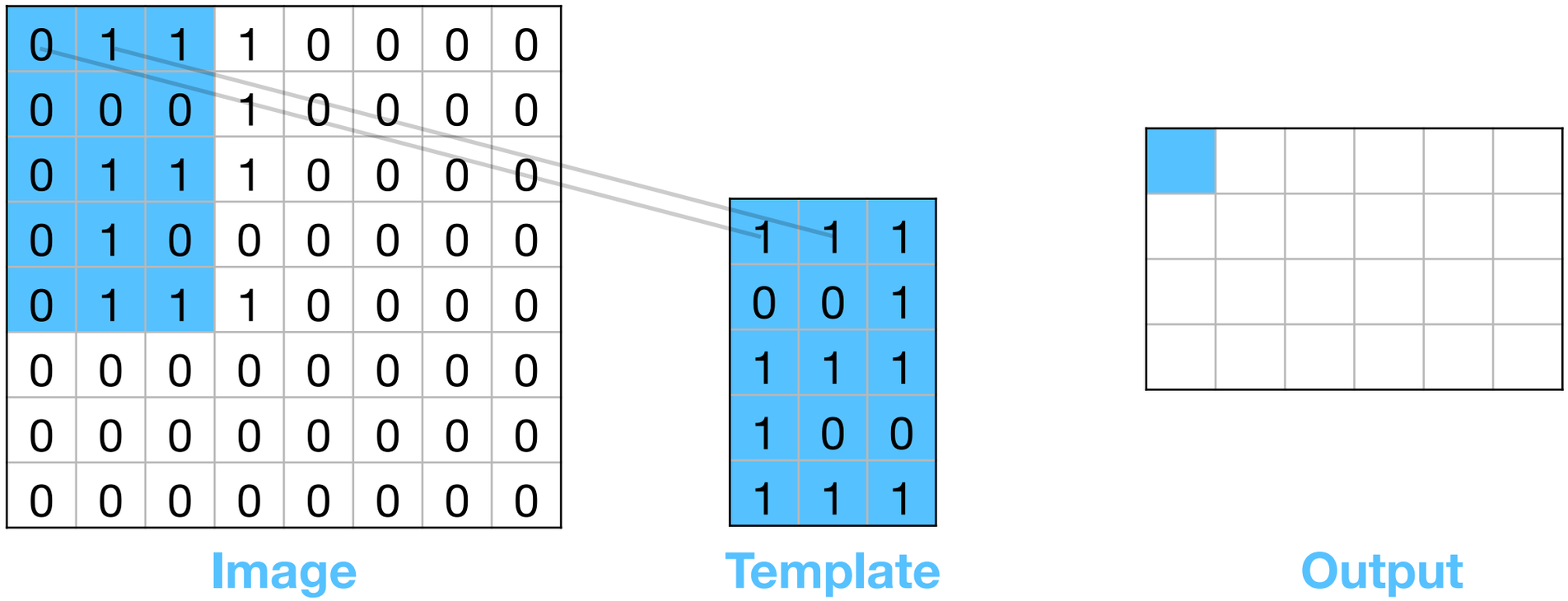
- Here's an illustration of how we construct the entire output image by applying a template to each location of the input image…

Convolution



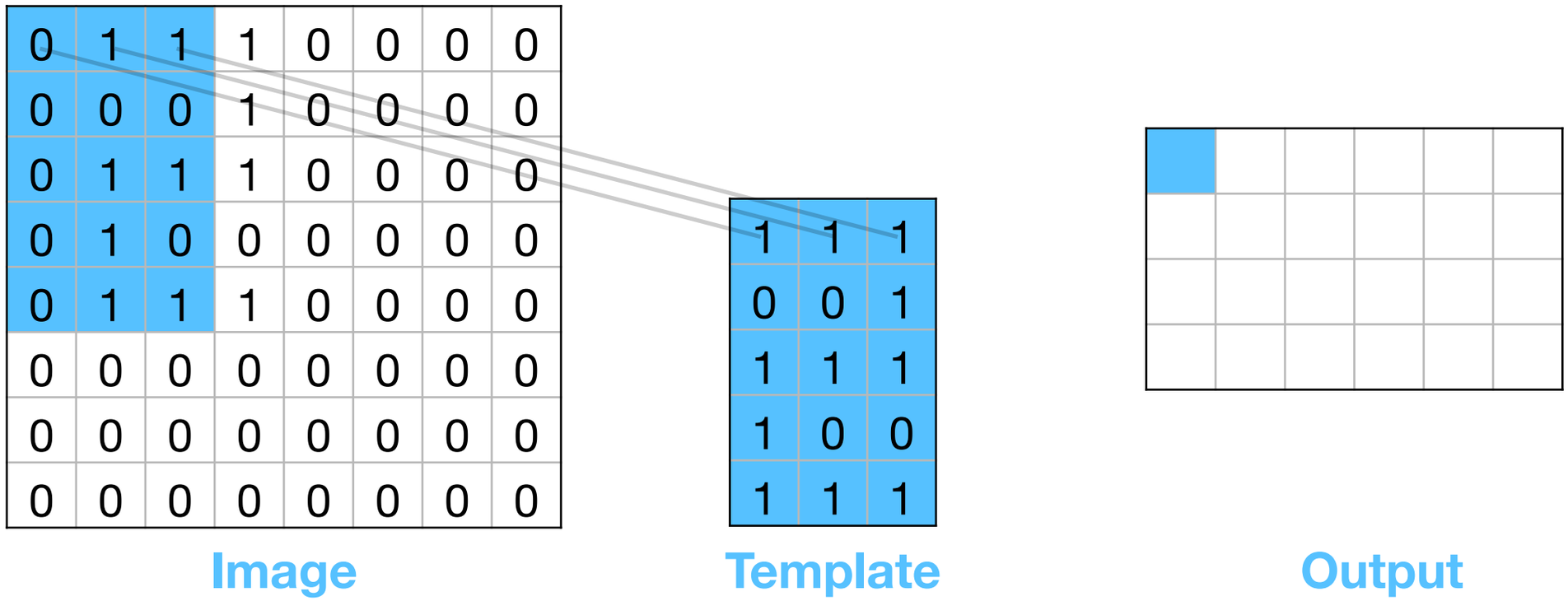
0*1

Convolution



0*1+1*1

Convolution



0*1+1*1+1*1

Convolution

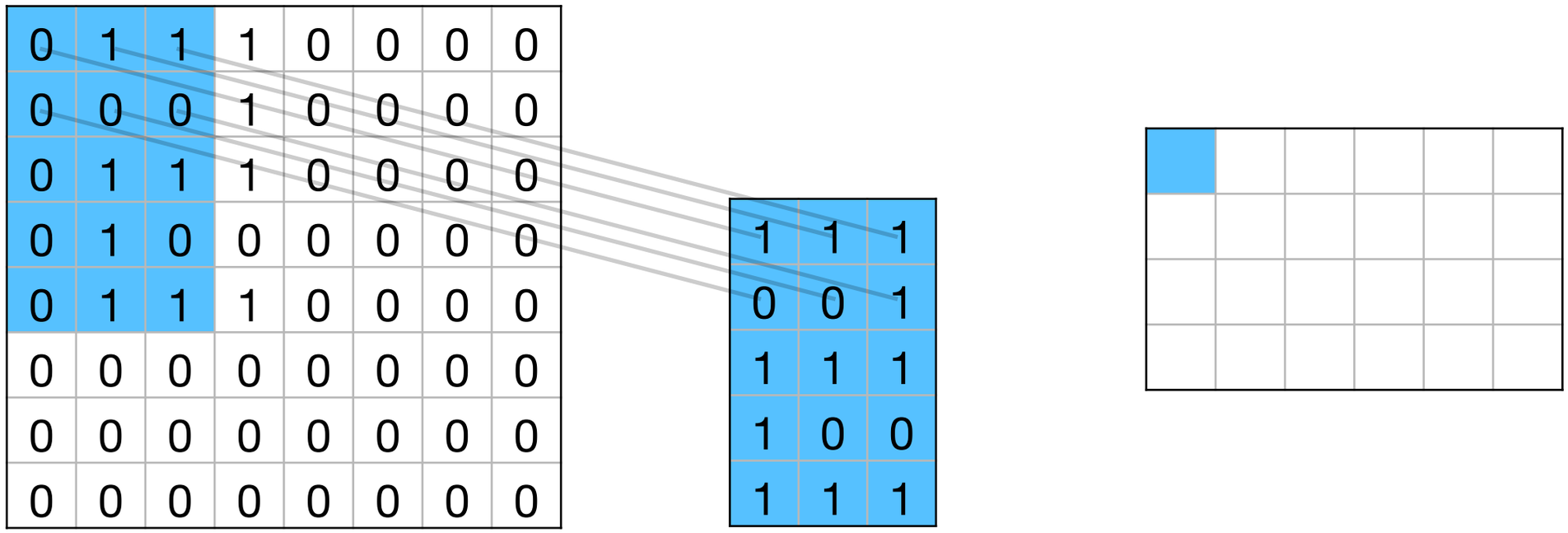
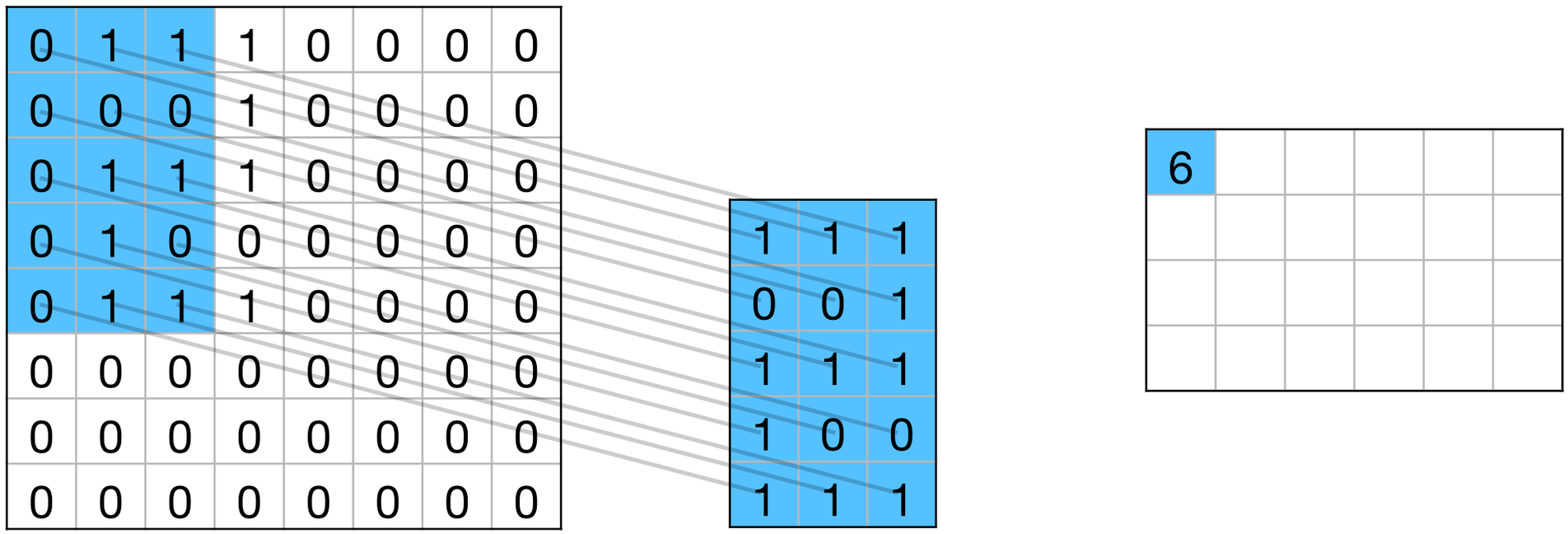


Image Template Output

0*1+1*1+1*1+
0*0+0*0+0*1

Convolution



0*1+1*1+1*1+
0*0+0*0+0*1+
0*1+1*1+1*1+
0*1+1*0+0*0+
0*1+1*1+1*1
=6

Convolution

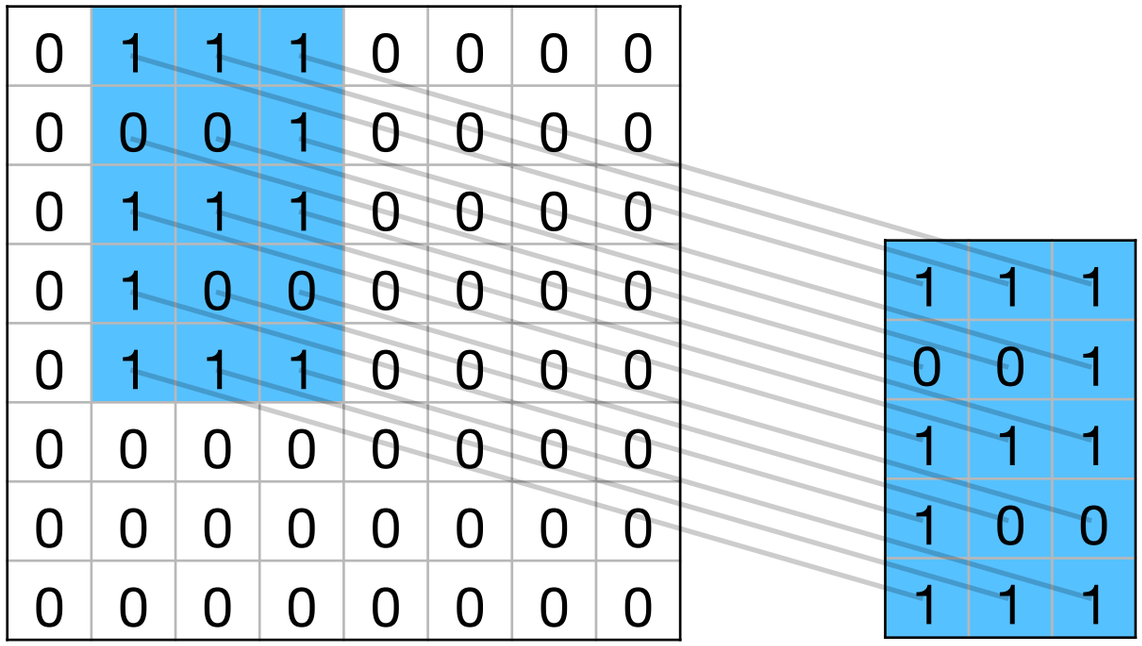


Image Template Output

1*1+1*1+1*1+
0*0+0*0+1*1+
1*1+1*1+1*1+
1*0+1*0+0*0+
1*1+1*1+1*1
=11

Convolution: exercise

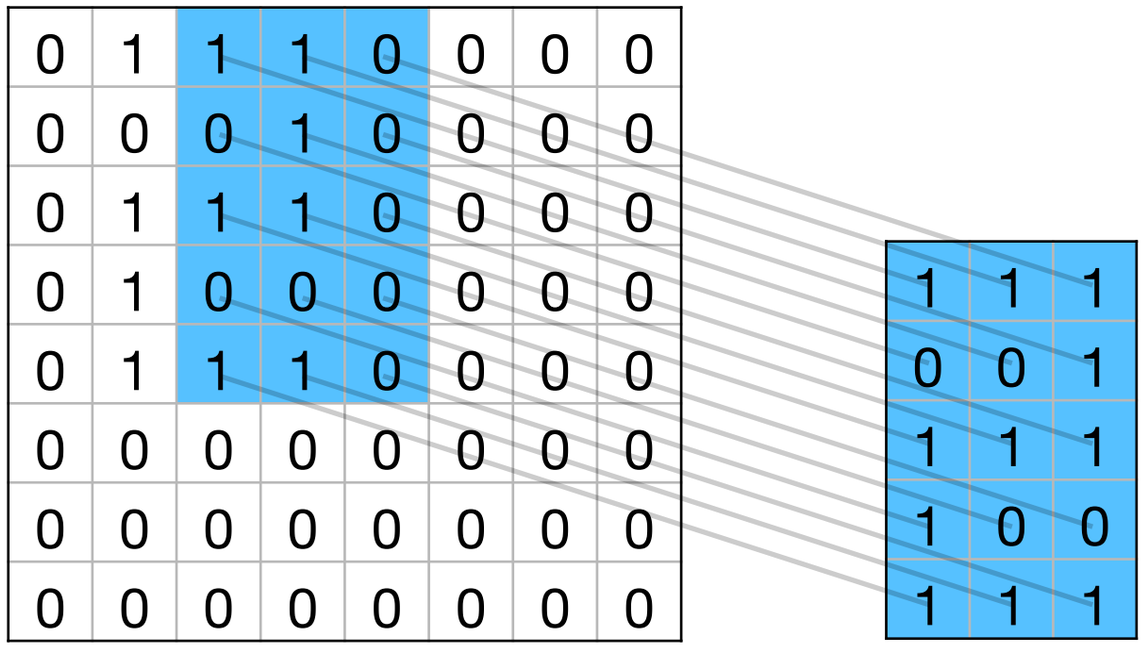


Image Template Output

Convolution: exercise

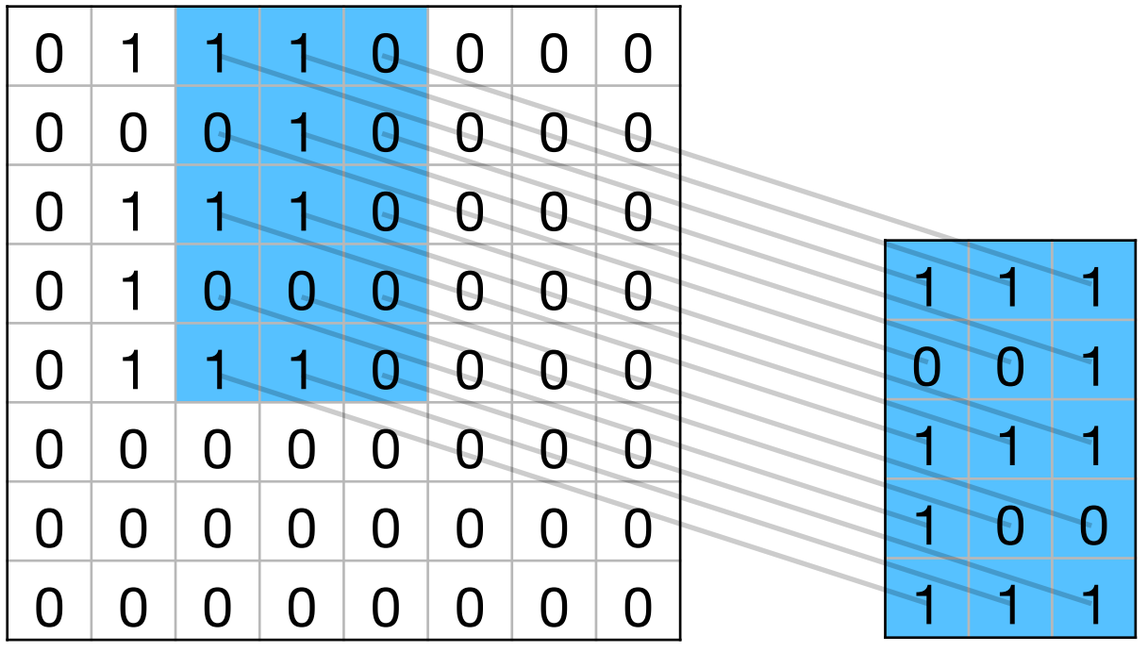


Image Template Output

1*1+1*1+0*1+
0*0+1*0+0*1+
1*1+1*1+0*1+
0*1+0*0+0*0+
1*1+1*1+0*1
=6

Convolution

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

Image

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

Template

6	11	6	…	0	0
…					0
				0	0
			…	0	0

Output

Pooling

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

Image

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

Template

6	11	6	…	0	0
…					0
				0	0
			…	0	0

Output

- The output image now expresses how much each location in the input image “looks like” the template.

Pooling

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

Image

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

Template

6	11	6	…	0	0
…					0
				0	0
			…	0	0

Output

11

Max-pool

- The output image now expresses how much each location in the input image “looks like” the template.
- To classify the object in the image (without caring where it is), we can just compute the *maximum* of the output.
- This is called a **maximum pool** (max-pool).

Convolution for classification

- Using this approach, we can build a simplistic translation-invariant object classifier for MNIST images:

- Construct a template for each digit class (0-9).

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

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

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

...

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

Convolution for classification

- Using this approach, we can build a simplistic translation-invariant object classifier for MNIST images:

- Convolve the image with each template.

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

Image

Class 0

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

Template

5	10	5	…	0	0
…					0
				0	0
			…	0	0

Output

Convolution for classification

- Using this approach, we can build a simplistic translation-invariant object classifier for MNIST images:

- Convolve the image with each template.

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

Image

Class 1

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

Template

4	3	4	…	0	0
…					0
				0	0
			…	0	0

Output

Convolution for classification

- Using this approach, we can build a simplistic translation-invariant object classifier for MNIST images:

- Convolve the image with each template.

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

Image

Class 2

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

Template

6	11	6	…	0	0
…					0
				0	0
			…	0	0

Output

Convolution for classification

- Using this approach, we can build a simplistic translation-invariant object classifier for MNIST images:

- Convolve the image with each template.

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

Image

...

Template

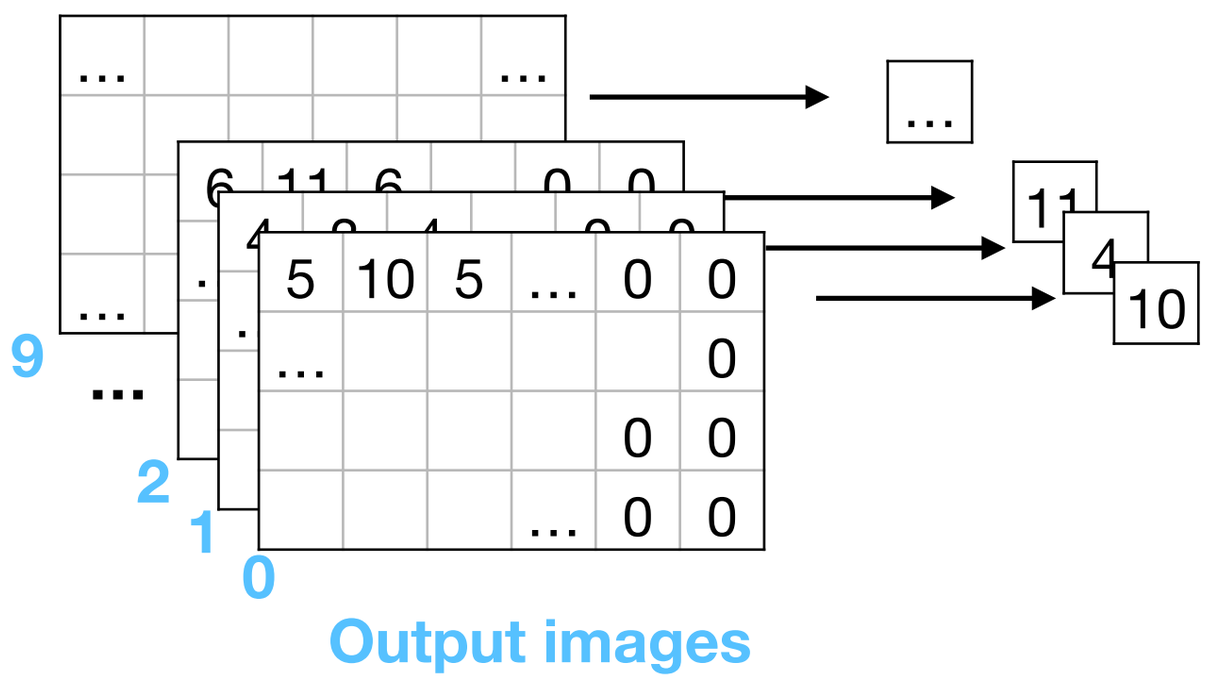
...					...
...					...

Output

Convolution for classification

- Using this approach, we can build a simplistic translation-invariant object classifier for MNIST images:

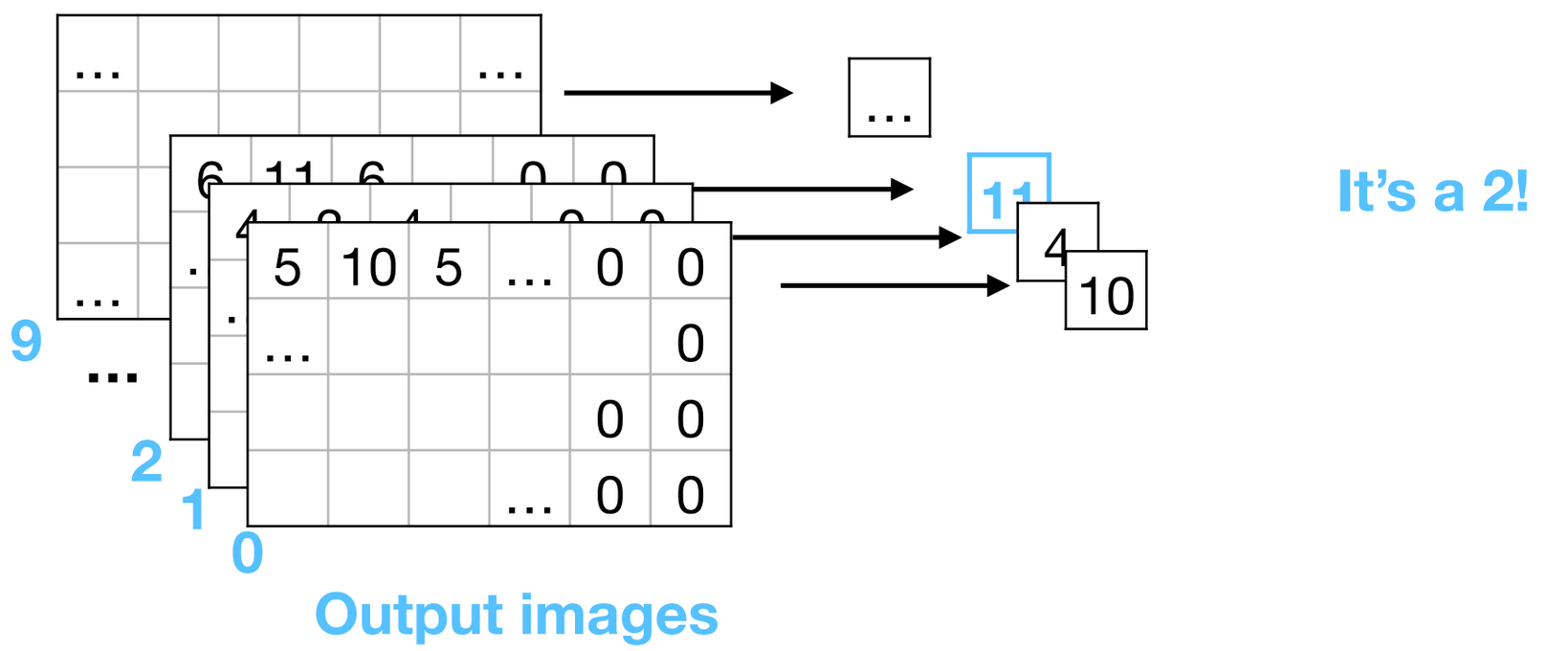
- Compute the maximum over each output image.



Convolution for classification

- Using this approach, we can build a simplistic translation-invariant object classifier for MNIST images:

- Predict the class whose max-pool value was largest.



Convolution: details

- The “templates” are more commonly known as **filters** or **kernels**.

- The output image is more commonly known as a **filter response** or a **feature map**.

8 x 8

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

Image

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

Kernel/ filter

4 x 6

Response/ Feature map

Convolution: details

- The output images in the examples above were always smaller than the input image.

8 x 8

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

Image

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

Kernel

4 x 6

Feature map

Convolution: details

- The output images in the examples above were always smaller than the input image.

- To preserve the image size, we can **pad** the input image:

12 x 10 (with padding)

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

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

Kernel

8 x 8

Feature map

Convolution: details

- The output images in the examples above were always smaller than the input image.

- Now we can apply the template at *every* image location.

12 x 10 (with padding)

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

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

Kernel

8 x 8

2							

Feature map

Convolution: details

- The output images in the examples above were always smaller than the input image.

- Now we can apply the template at *every* image location.

12 x 10 (with padding)

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

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

Kernel

8 x 8

2	4						

Feature map

Convolution: details

- The output images in the examples above were always smaller than the input image.

- Now we can apply the template at *every* image location.

12 x 10 (with padding)

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

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

Kernel

8 x 8

2	4	6					

Feature map

Convolution: details

- The output images in the examples above were always smaller than the input image.

- Now we can apply the template at *every* image location.

12 x 10 (with padding)

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

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

Kernel

8 x 8

2	4	6					
			…				
							0

Feature map

Convolution: details

- Sometimes we might want to *down-scale* the output image w.r.t. the input image.

- We can apply the template with a **stride** of *k* pixels:

12 x 10 (with padding)

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

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

Kernel

4 x 4 (with stride *k*=2)

2			

Feature map

Convolution: details

- Sometimes we might want to *down-scale* the output image w.r.t. the input image.

- We can apply the template with a **stride** of *k* pixels:

12 x 10 (with padding)

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

Kernel

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

4 x 4 (with stride *k*=2)

2	6		

Feature map

Convolution: details

- Sometimes we might want to *down-scale* the output image w.r.t. the input image.

- We can apply the template with a **stride** of *k* pixels:

12 x 10 (with padding)

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

Kernel

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

4 x 4 (with stride *k*=2)

2	6	3	

Feature map

Convolution: details

- Sometimes we might want to *down-scale* the output image w.r.t. the input image.

- We can apply the template with a **stride** of *k* pixels:

12 x 10 (with padding)

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

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

Kernel

4 x 4 (with stride *k*=2)

2	6	3	…
3			

Feature map

Convolution: details

- Sometimes we might want to *down-scale* the output image w.r.t. the input image.

- We can apply the template with a **stride** of *k* pixels:

12 x 10 (with padding)

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

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

Kernel

4 x 4 (with stride *k*=2)

2	6	3	…
3			
			…

Feature map

Dilated convolution

- To expand the spatial extent of the filter without increasing the number of parameters, we can dilate it by adding "spaces" (with dilation *k*=2) between filter elements.

1	9	5	-2	4
2	8	6	5	2
3	7	-5	-3	0
2	3	8	1	8
8	0	9	-4	2

Image

2	1
-3	9

Filter

1*2+5*1+3*-3+-5*9 = -47

-47		

Feature map

Dilated convolution

- To expand the spatial extent of the filter without increasing the number of parameters, we can dilate it by adding "spaces" (with dilation *k*=2) between filter elements.

1	9	5	-2	4
2	8	6	5	2
3	7	-5	-3	0
2	3	8	1	8
8	0	9	-4	2

Image

2	1
-3	9

Filter

9*2+-2*1+7*-3+-3*9 = -32

-47	-32	
	…	

Feature map

Dilated convolution

- Dilated convolution is sometimes called **à trous** convolution (“with holes”).

1	9	5	-2	4
2	8	6	5	2
3	7	-5	-3	0
2	3	8	1	8
8	0	9	-4	2

Image

2	1
-3	9

Filter

9*2+-2*1+7*-3+-3*9 = -32

-47	-32	
	…	

Feature map

Convolution in 3-D

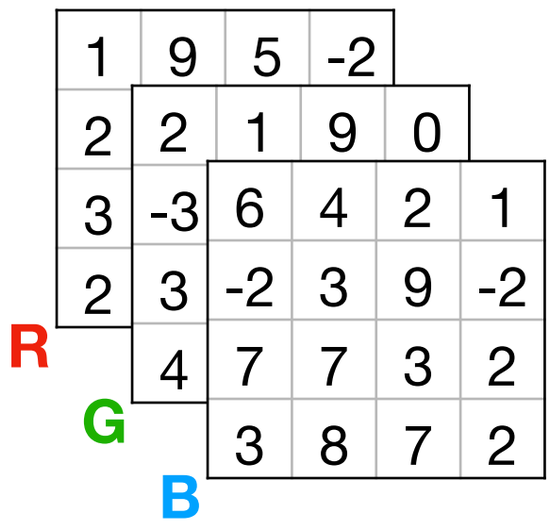
- It is possible to perform convolution in any number of dimensions.
- With neural networks, the usual formula of 3-D convolution is given by:

$$\mathrm{TM}(r,c) = \sum_{c=1}^{t_c} \sum_{i=-t_h/2}^{+t_h/2} \sum_{j=-t_w/2}^{+t_w/2} \mathrm{im}[r+i, c+j, c]\mathrm{t}[i,j,c]$$

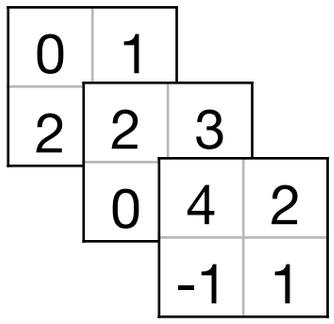
where t_c is the number of **channels** in the convolution kernel (and also the input image).

Convolution in 3-D: example

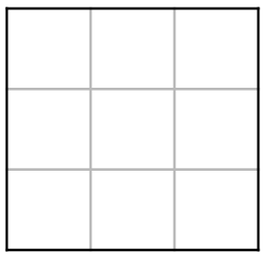
- Suppose our input image is RGB of size 4x4 — i.e., it is 4x4x3 — and we convolve it with a kernel of 2x2x3:



Image



Filter



Feature map

Convolution in 3-D: example

- To illustrate how this works, let's separate the different channels:

R

1	9	5	-2
2	8	6	5
3	7	-5	-3
2	3	8	1

0	1
2	-1

+

G

2	1	9	0
-3	0	-7	3
3	7	1	2
4	2	0	1

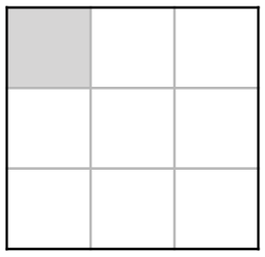
2	3
0	2

+

B

6	4	2	1
-2	3	9	-2
7	7	3	2
3	8	7	2

4	2
-1	1



Convolution in 3-D: example

- To illustrate how this works, let's separate the different channels:

R

1	9	5	-2
2	8	6	5
3	7	-5	-3
2	3	8	1

G

2	1	9	0
-3	0	-7	3
3	7	1	2
4	2	0	1

B

6	4	2	1
-2	3	9	-2
7	7	3	2
3	8	7	2

0	1
2	-1

1*0+9*1+2*2 +8*-1=5

+

2	3
0	2

2*2+1*3+-3*0 +0*2=7

+

4	2
-1	1

6*4+4*2+-2*-1 +3*1=37

49		

Convolution in 3-D: example

- To illustrate how this works, let's separate the different channels:

R

1	9	5	-2
2	8	6	5
3	7	-5	-3
2	3	8	1

0	1
2	-1

+

G

2	1	9	0
-3	0	-7	3
3	7	1	2
4	2	0	1

2	3
0	2

+

B

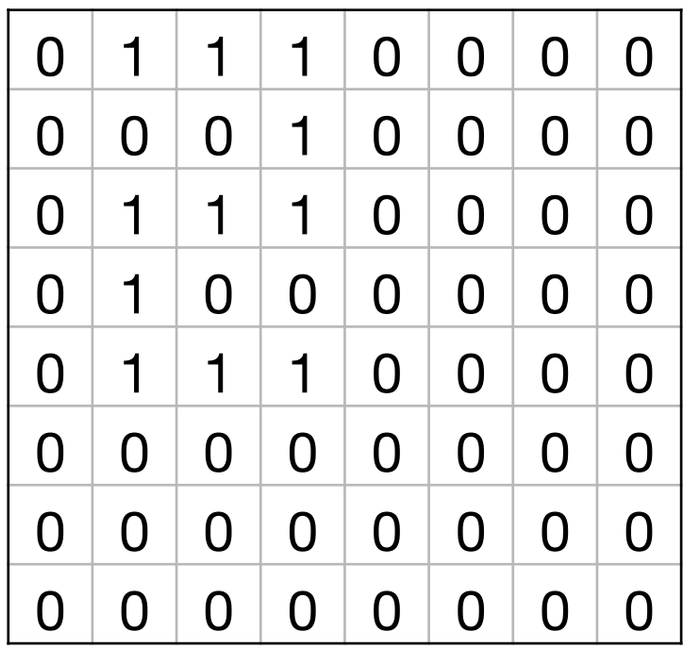
6	4	2	1
-2	3	9	-2
7	7	3	2
3	8	7	2

4	2
-1	1

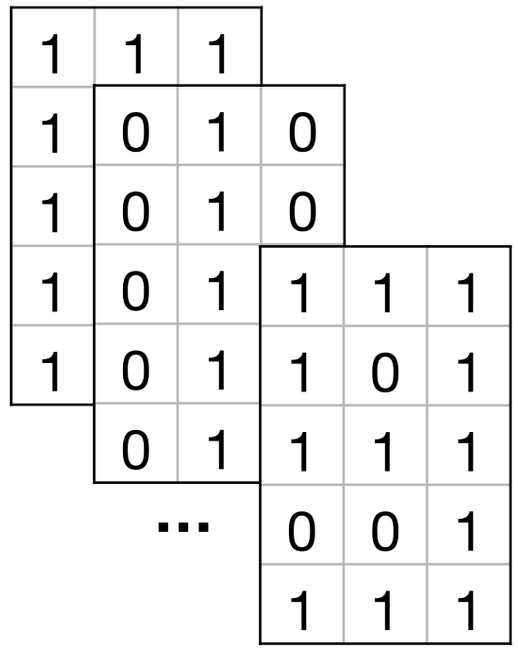
49	…	

Convolution: multiple output channels

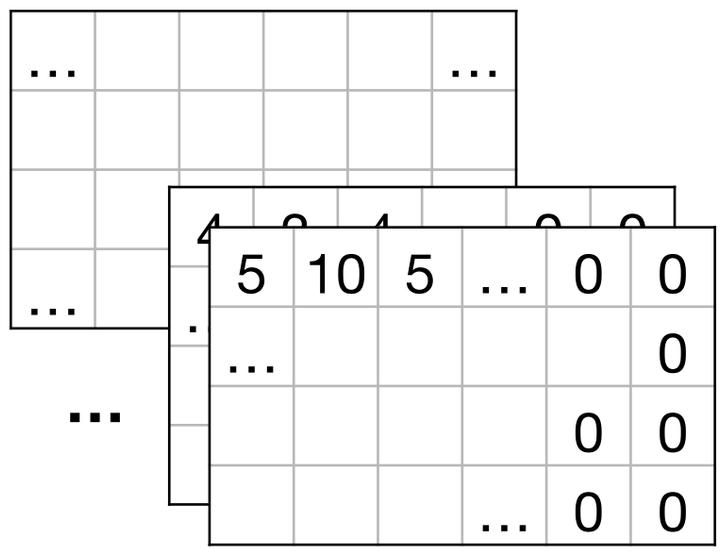
- By applying multiple convolution filters to each input image, we can produce multiple feature maps (recall the MNIST example):



Image



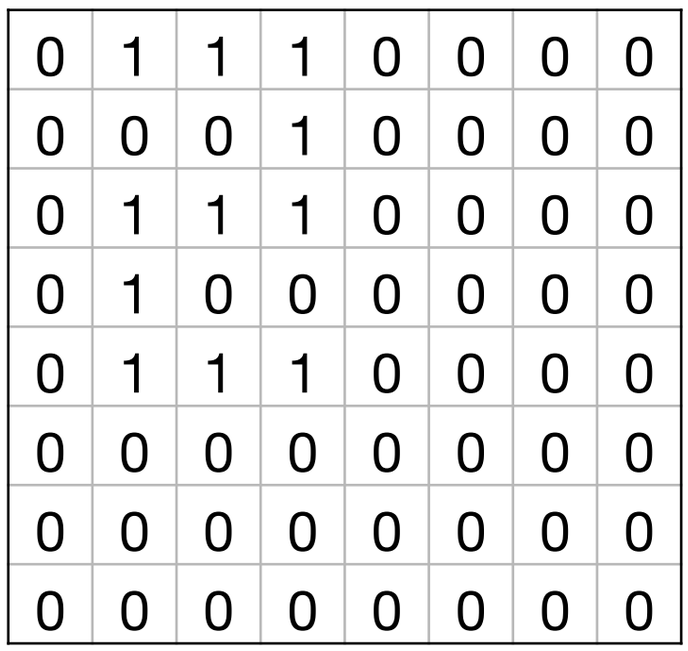
Multiple filters



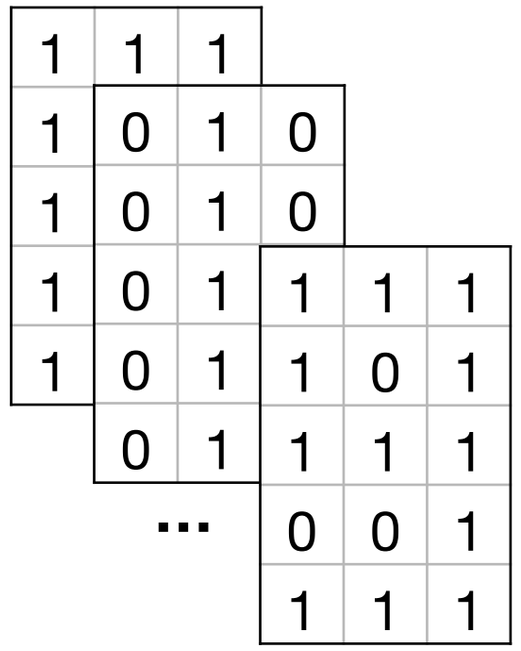
Multiple feature maps

Convolution: multiple output channels

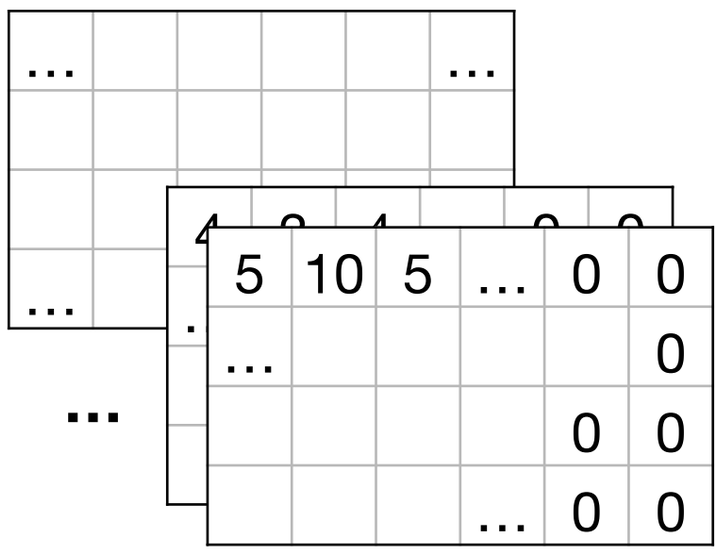
- Hence, convolution can take a multi-channel image as *input*, and also produce a multi-channel image as *output*.



Image



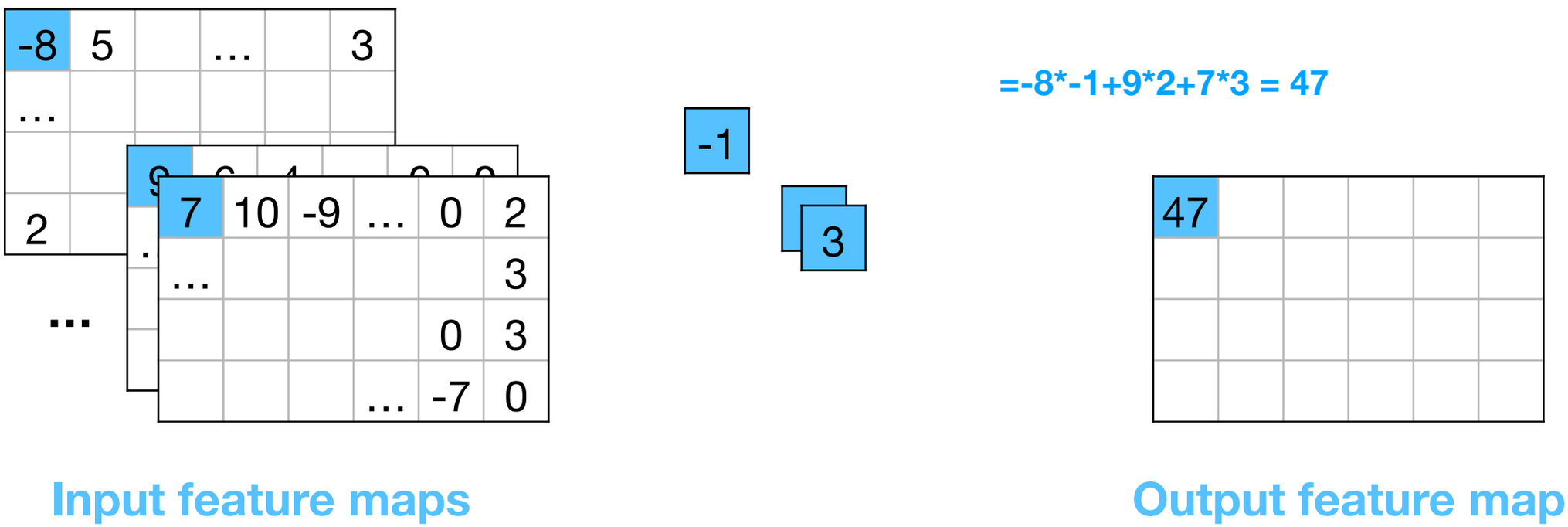
Multiple filters



Multiple feature maps

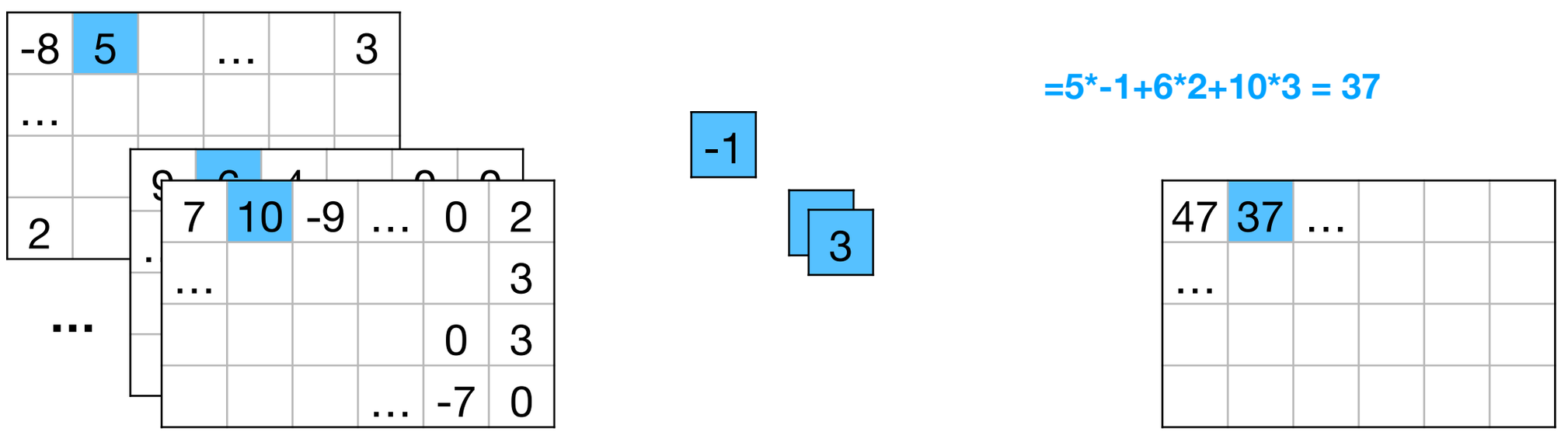
1-d convolution

- We can also perform “1-d convolution”.
- This is equivalent to a weighted sum among the input feature maps.
- It is useful to reduce the number of feature maps at the current NN layer.



1-d convolution

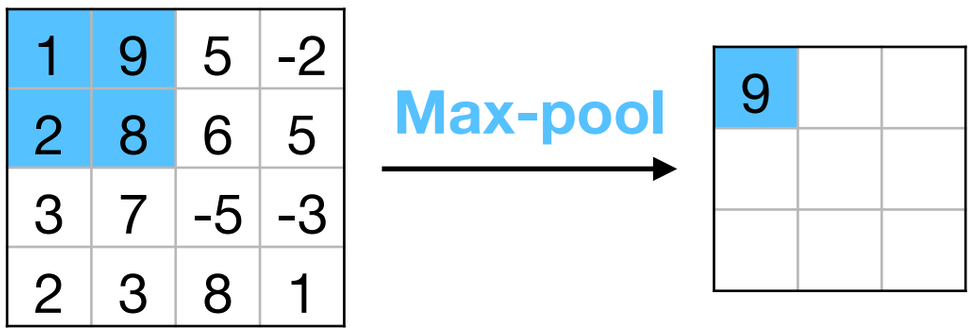
- We can also perform “1-d convolution”.
- This is equivalent to a weighted sum among the input feature maps.
- It is useful to reduce the number of feature maps at the current NN layer.



Pooling: details

- Instead of pooling over the whole image, we can pool over small sub-regions of size *w*. We can also use a stride of size *k*. (We could also use padding.)

- Example: *w*=2, *k*=1.



Pooling: details

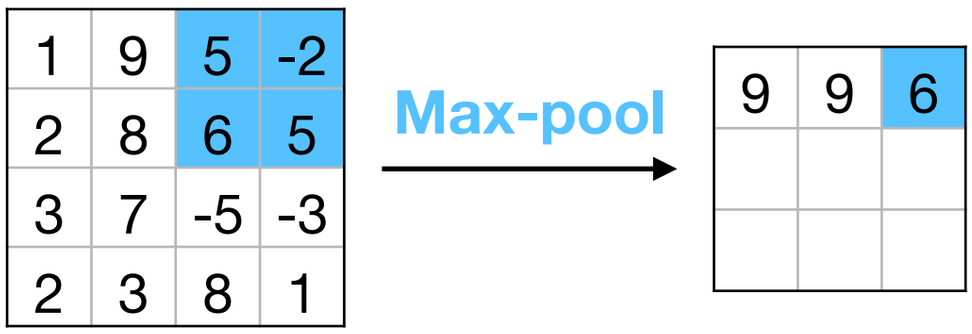
- Instead of pooling over the whole image, we can pool over small sub-regions of size *w*. We can also use a stride of size *k*. (We could also use padding.)

- Example: *w*=2, *k*=1.

Pooling: details

- Instead of pooling over the whole image, we can pool over small sub-regions of size *w*. We can also use a stride of size *k*. (We could also use padding.)

- Example: *w*=2, *k*=1.



Pooling: details

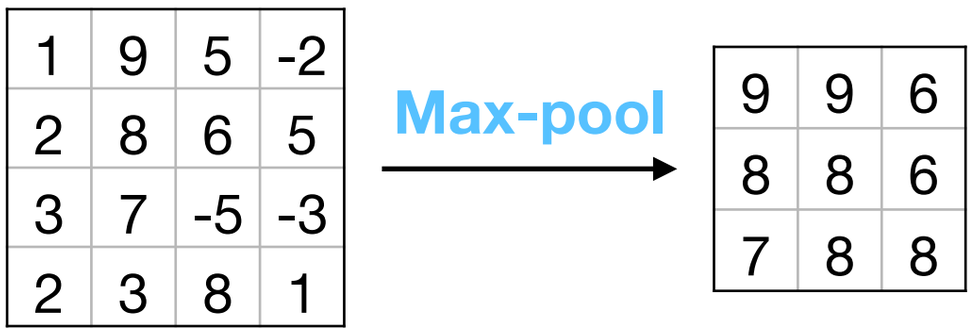
- Instead of pooling over the whole image, we can pool over small sub-regions of size *w*. We can also use a stride of size *k*. (We could also use padding.)

- Example: *w*=2, *k*=1.

Pooling: details

- Instead of pooling over the whole image, we can pool over small sub-regions of size *w*. We can also use a stride of size *k*. (We could also use padding.)

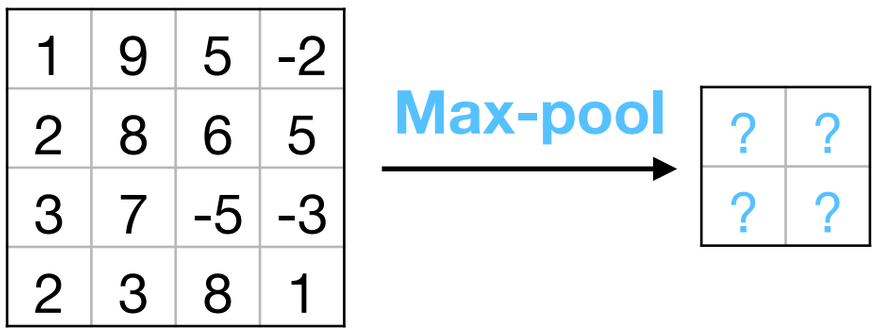
- Example: *w*=2, *k*=1.



Pooling: details

- Instead of pooling over the whole image, we can pool over small sub-regions of size *w*. We can also use a stride of size *k*. (We could also use padding.)

- Example: *w*=2, *k*=2.



Pooling: details

- Instead of pooling over the whole image, we can pool over small sub-regions of size *w*. We can also use a stride of size *k*. (We could also use padding.)

- Example: *w*=2, *k*=2.