CS 331
Computer Vision

11 / 10 / 2026

Instructor: Michael Eckmann

Topics

- Questions / Comments?
- Neural Networks
  - Finish the Larger example with hidden layers and how forward prop, backprop and weight updates for mini-batch gradient descent
- Start Convolutional Neural Networks

Choices for Gradient Descent

- Remind us of the choices for Gradient descent in the sense of how often we update the weights?

Choices for Gradient Descent

- Gradient descent can be done to alter the weights

a. once for each training example (Stochastic Gradient Descent (SGD)) – fast and low memory but noisy,

b. or once for the entire set of training examples (Batch Gradient Descent) – slow and high memory usage for large datasets

c. or once for each mini-batch of training examples (Mini-batch Gradient Descent) – practical middle-ground

How a Neural Network learns

- The steps are:

Initialize weights randomly

For many iterations

    For each batch #

        For each training example in the batch

            Forward Propagation to make prediction

            Backward Propagation to compute the gradient

            Accumulate the gradients

        Gradient Descent to alter the weights based on average of accumulated gradients over the batch

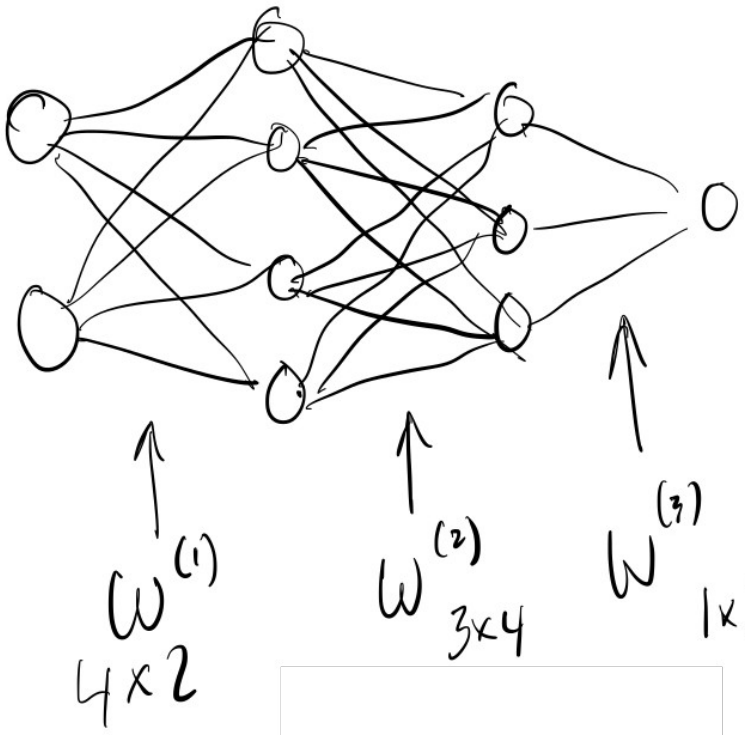
This describes Mini-batch Gradient Descent because the weights are updated once per mini-batch

Learning in a NN

again no bias units

$$x = \begin{bmatrix} x_1 \\ x_2 \end{bmatrix}$$

Sigmoid activation in all units of layers 2, 3, 4 totalling 8 units



$z^{(2)}$ are inputs to layer 2

$a^{(2)}$ are output of layer 2

Learning in a NN

- Let's go through a more complex example of a network with 2 input units, 2 hidden layers with 4 and 3 units and 1 output unit, activation function is sigmoid in all the units of the hidden layers and the output layer.
- And we'll have an arbitrary number of training examples divided into batches.
- One thing we'll need is the derivative of the sigmoid function.
- $g(z) = 1/(1+e^{-z})$
- $g'(z) = g(z) * (1 - g(z))$

Learning in a NN

- Because we have one output unit, we are doing binary classification.
- We use the cross-entropy loss function with sigmoid activation in the output unit.
- That's the name of the loss function I introduced for logistic regression:

$$\text{Loss} = -\left[\, y \ln \hat{y} + (1 - y) \ln(1 - \hat{y}) \,\right]$$

Learning in a NN

for all batches:

$\Delta^{(1)} \leftarrow 0$, $\Delta^{(2)} \leftarrow 0$ $\Delta^{(3)} \leftarrow 0$.

For all training examples in a batch:

forward prop

$a^{(1)} = X$

$z^{(2)} = W^{(1)} @ a^{(1)}$

$a^{(2)} = g(z^{(2)})$

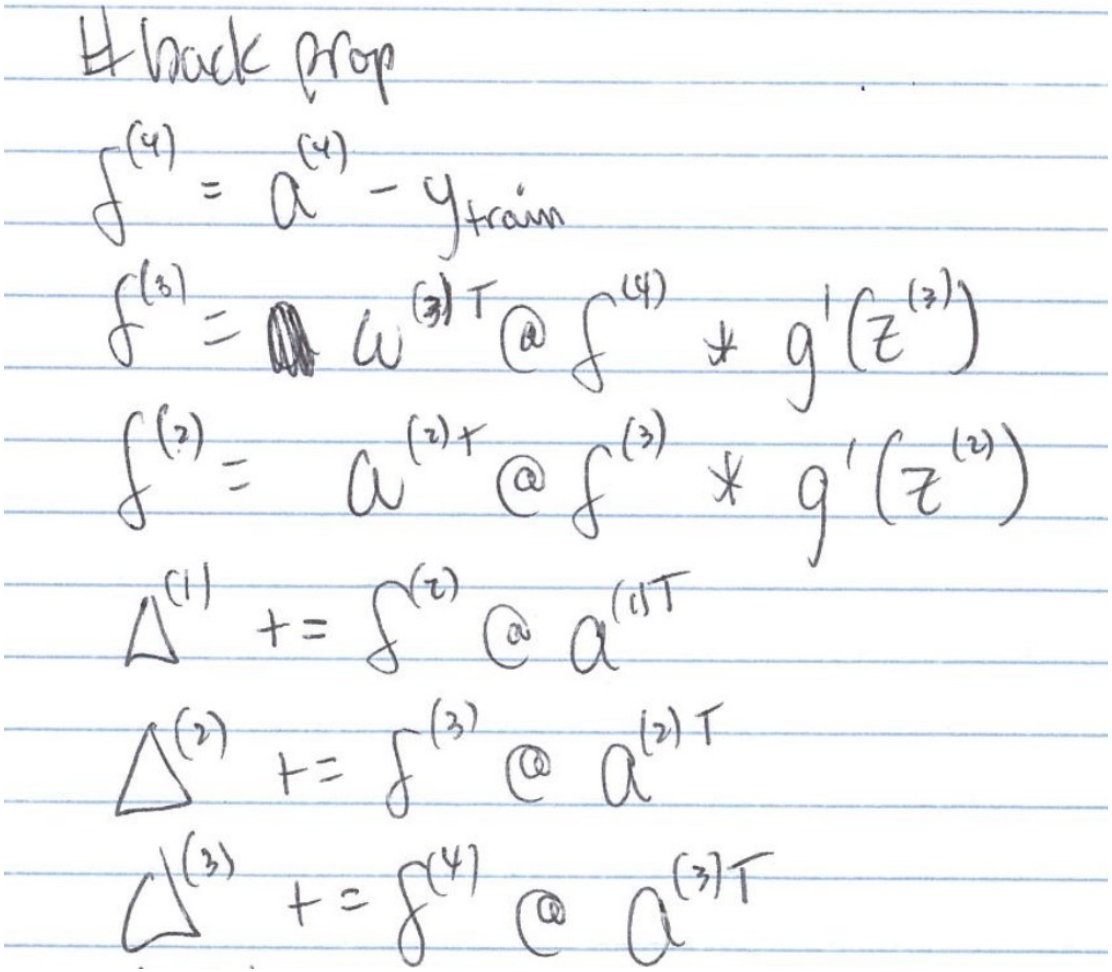
$z^{(3)} = W^{(2)} @ a^{(2)}$

$a^{(3)} = g(z^{(3)})$

$z^{(4)} = W^{(3)} @ a^{(3)}$

$a^{(4)} = g(z^{(4)})$

Learning in a NN



Learning in a NN

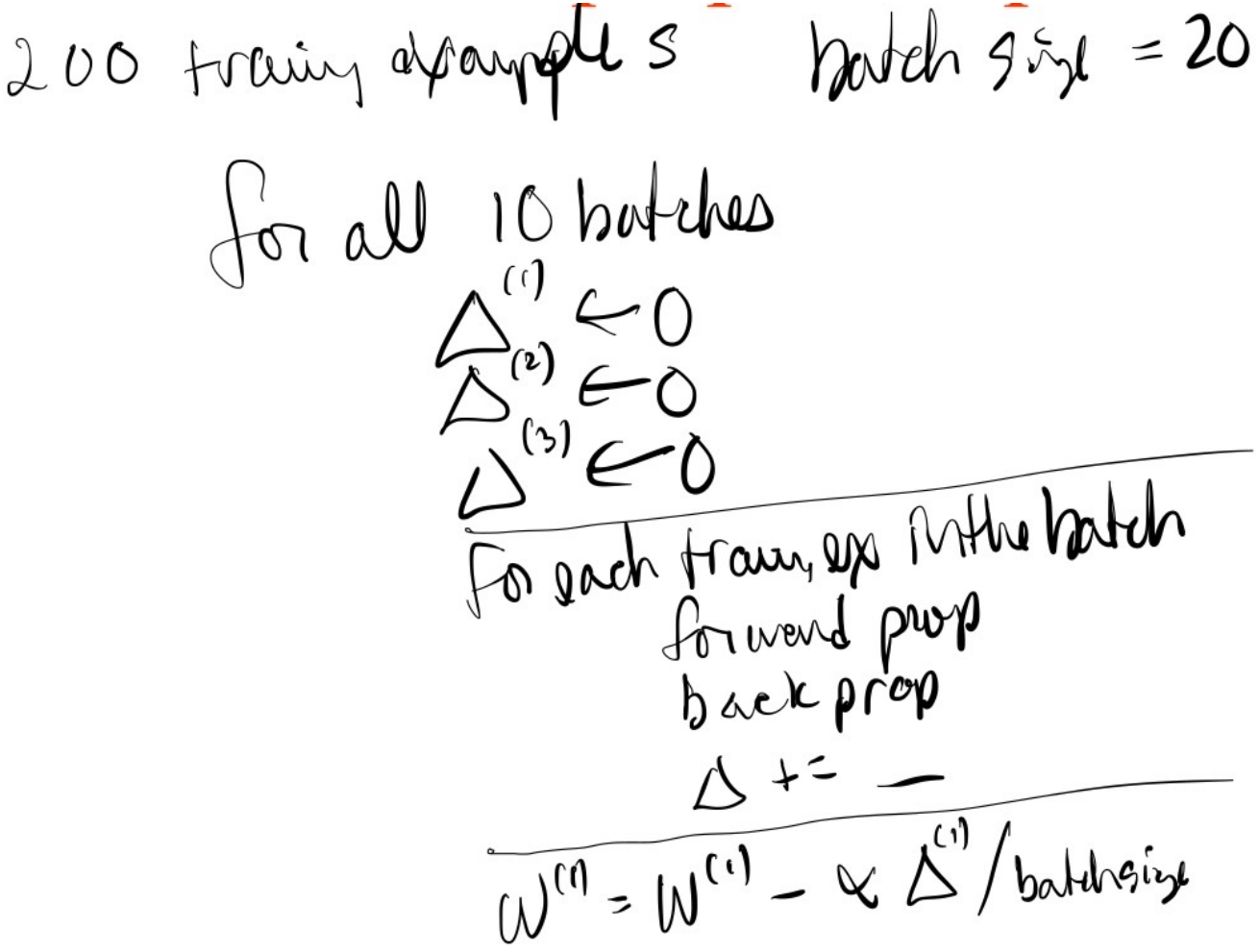
after a batch:

$$W^{(1)} = W^{(1)} - \alpha \left(\Delta^{(1)} / \text{batchsize} \right)$$

$$W^{(2)} = W^{(2)} - \alpha \left(\Delta^{(2)} / \text{batchsize} \right)$$

$$W^{(3)} = W^{(3)} - \alpha \left(\Delta^{(3)} / \text{batchsize} \right)$$

Learning in a NN



Learning in a NN

- If activation function is ReLU, we need its derivative
- ReLU(z) = max(0, z)
- ReLU’(z) = 0 if z <=0, 1 otherwise

Neural Networks

- So far we have learned how a Neural Network with fully connected layers functions. Another name for fully connected is dense.
- Fully connected means there are connections from each node in layer i to each node in layer i+1.
- If we were to try to do classification of images with the machinery we learned so far, we would need to take an image and stretch it out (aka flatten) into a 1d vector of inputs.
- e.g. a 50x50 3 color channel image has 2500 pixels with 3 color values each for a total of 7500 values. This 3-dimensional structure could be flattened into a 1-dimensional array of 7500 values.

Neural Networks

- 50x50 3 could be stretched out into a 1-dimensional array of 7500 values in the following way:
- Top row of 50 pixels' red channel concatenated to the next row of 50 pixels' red channel and so on so the first 2500 values in the 7500 array are the red channels. Then could follow that with top row of 50 pixels' green channels and so on.

Neural Networks

- So our Neural Network would have 7500 input nodes in layer 1 which would be fully connected to layer 2's nodes.
- But this removes the fact that pixels near each other (above, below, left, right) have meaning. That is, it removes the spatial aspect of the images and it stands to reason that the spatial aspect of images has an impact on learning to classify them.
- The reason the pixels don't retain their spatial neighbors in this style is because each connection learns its own weight (see next slide).

Neural Networks

- Suppose the next layer has 15000 units. The weight matrix for layer 1: $w^{(1)}$ will be 15000x7501 in size (7501 because of the bias unit).
- Notice: each input would have 15000 connections to the next layer.
- Because of that setup, the spatial nature of images is lost. A different weight can be learned for any of those in the weight matrix passing along some amount of any pixel to various units in the next layer.

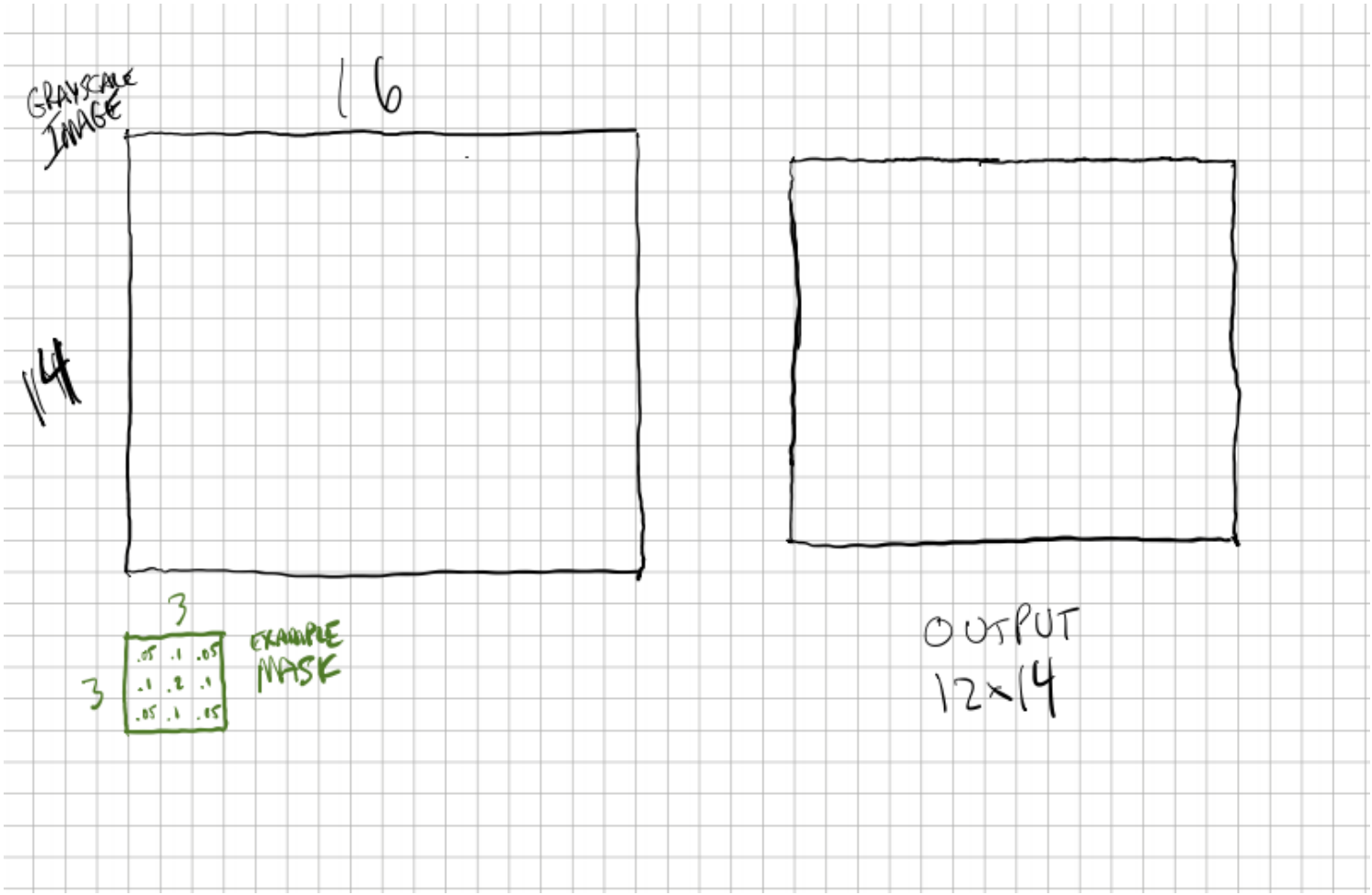
Neural Networks

- Think back to your nearest neighbor program using histograms. Did the histogram retain any spatial information?

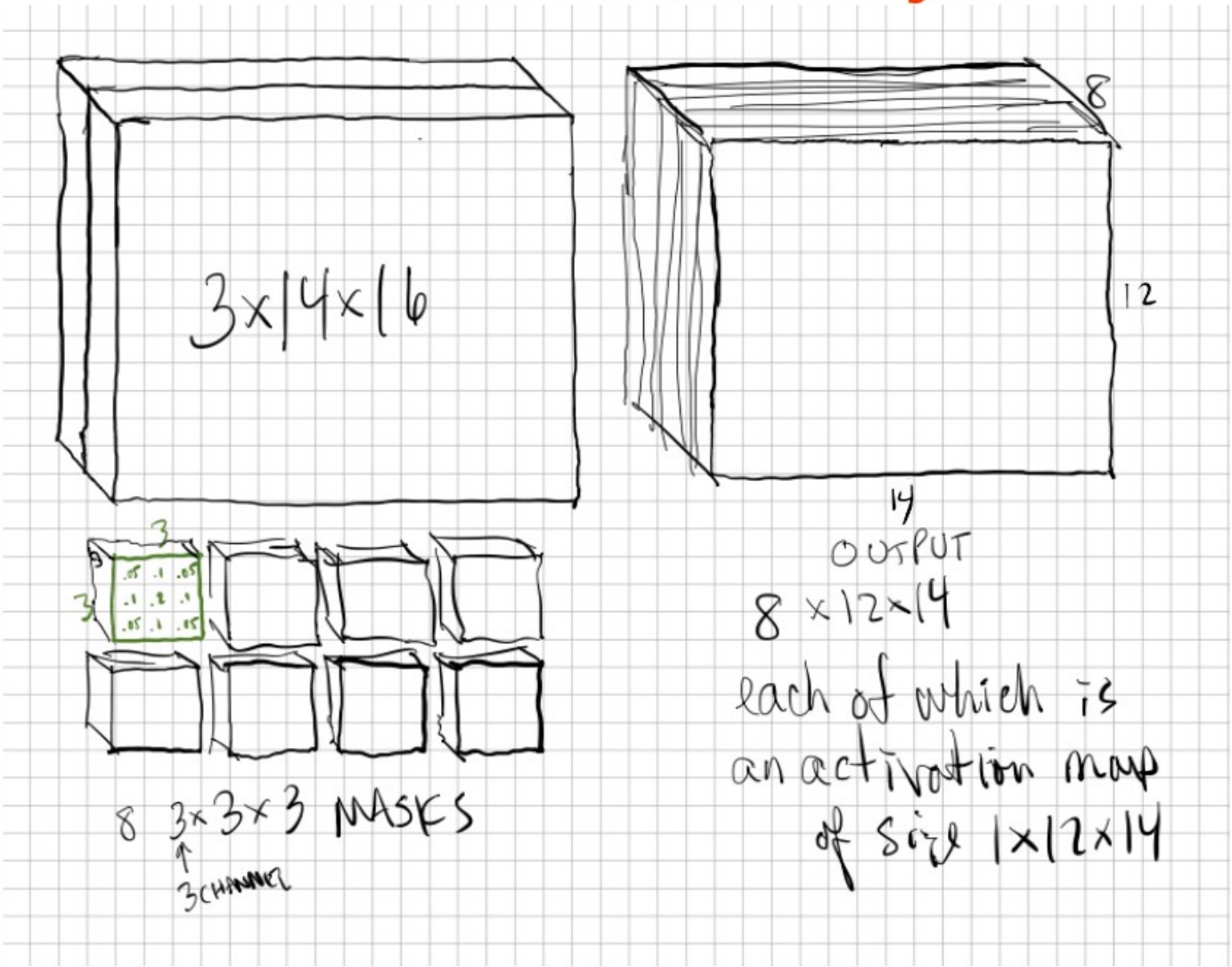
Convolutional Layer

- To fix that, a convolutional neural network's first layer is a convolutional layer.
  - Two key differences between a convolutional layer and a fully connected layer:
    - Input to a convolutional layer is spatial (not flattened)
    - Not fully connected to the next layer
- Convolutional Neural Networks are sometimes abbreviated ConvNets or CNNs.
- Recall the convolution operator. Usually we had a small, square mask (aka filter) of values that were multiplied by the pixels and summed up to get a result. The mask was moved around to get results at every possible place the mask fit.

Convolution



Convolutional Layer



Convolutional Layer

- In the example on last slide:
- Each of the 8 masks has 27 elements.
- Each 3x3x3 mask will be elementwise multiplied by a 3x3x3 chunk of the input image.
- One output value would be computed by doing 27 multiplies and 26 adds.

Another note: what I am referring to as a mask is often called a filter or a kernel in the literature. So I use these terms interchangeably.

Tensor

- Think of a tensor as a multidimensional array.
- A scalar is a 0-dimensional tensor
- A vector is a 1-dimensional tensor
- A matrix is a 2-dimensional tensor
- But for higher dimensions we usually call them tensors and the number of dimensions can be specified.

Convolutional Layer

- If there were 1 image with Cin channels, inRows rows and inCols columns the input is a 3 dimensional tensor of shape (Cin, inRows, inCols).
- Each mask will be of shape (Cin, maskRows, maskCols) and there could be Cout of them which could be stored in a 4-d tensor of shape (Cout, Cin, maskRows, maskCols).
- The output of the convolutional layer would produce a 3-dimensional tensor of shape (Cout, outRows, outCols).
- Last picture had: (Cin, inRows, inCols) = (3,14,16)
- (Cout, Cin, maskRows, maskCols) = (8,3,3,3)
- (Cout, outRows, outCols) = (8,12,14)
- These are specified channels-first style