Convolution NN

- While you can use deep learning models for any kind of machine learning, they're particularly useful for dealing with data that consists of large arrays of numeric values - such as images. Machine learning models that work with images are the foundation for an area artificial intelligence called *computer vision*, and deep learning techniques have been responsible for driving amazing advances in this area over recent years.
- At the heart of deep learning's success in this area is a kind of model called a *convolutional neural network*, or *CNN*. A CNN typically works by extracting features from images, and then feeding those features into a fully connected neural network to generate a prediction. The feature extraction layers in the network have the effect of reducing the number of features from the potentially huge array of individual pixel values to a smaller feature set that supports label prediction.

Layers in a CNN

- CNNs consist of multiple layers, each performing a specific task in extracting features or predicting labels.

Convolution layers: One of the principal layer types is a *convolutional* layer that extracts important features in images. A convolutional layer works by applying a filter to images. The filter is defined by a *kernel* that consists of a matrix of weight values.

- For example, a 3x3 filter might be defined like this:

[1 -1 1

-1 0 -1

1 -1 1]

Convolution Layers

- An image is also just a matrix of pixel values. To apply the filter, you "overlay" it on an image and calculate a *weighted sum* of the corresponding image pixel values under the filter kernel. The result is then assigned to the center cell of an equivalent 3x3 patch in a new matrix of values that is the same size as the image. For example, suppose a 6 x 6 image has the following pixel values:

[255 255 255 255 255 255

255 255 100 255 255 255

255 100 100 100 255 255

100 100 100 100 100 255

255 255 255 255 255 255

255 255 255 255 255 255]

Applying the filter to the top-left 3x3 patch of the image would work like this:

```
255 255 255      1  -1   1    (255 x 1)+(255 x -1)+(255 x 1) +
255 255 100  x  -1   0  -1  = (255 x -1)+(255 x 0)+(100 x -1) +   = 155
255 100 100      1  -1   1    (255 x1 )+(100 x -1)+(100 x 1)
```

The result is assigned to the corresponding pixel value in the new matrix like this:

```
?   ?   ?   ?   ?   ?
?  155  ?   ?   ?   ?
?   ?   ?   ?   ?   ?
?   ?   ?   ?   ?   ?
?   ?   ?   ?   ?   ?
?   ?   ?   ?   ?   ?
```

Convolution Layers

- Now the filter is moved along (*convolved*), typically using a *step* size of 1 (so moving along one pixel to the right), and the value for the next pixel is calculated

```
255 255 255      1  -1   1    (255 x 1)+(255 x -1)+(255 x 1) +
255 100 255  x  -1   0  -1  = (255 x -1)+(100 x 0)+(255 x -1) +   = -155
100 100 100      1  -1   1    (100 x1 )+(100 x -1)+(100 x 1)
```

So now we can fill in the next value of the new matrix.

```
?   ?   ?   ?   ?   ?
 ?  155 -155 ?   ?   ?
 ?   ?   ?   ?   ?   ?
 ?   ?   ?   ?   ?   ?
 ?   ?   ?   ?   ?   ?
 ?   ?   ?   ?   ?   ?
```

```
[255 255 255 255 255 255
255 255 100 255 255 255
255 100 100 100 255 255
100 100 100 100 100 255
255 255 255 255 255 255
255 255 255 255 255 255]
```

Convolution Layers

The process repeats until we've applied the filter across all of the 3x3 patches of the image to produce a new matrix of values like this:

? ? ? ? ? ?

? 155 -155 155 -155 ?

? -155 310 -155 155 ?

? 310 155 310 0 ?

? -155 -155 -155 0 ?

? ? ? ? ? ?

Because of the size of the filter kernel, we can't calculate values for the pixels at the edge; so we typically just apply a *padding* value (often 0):

0 0 0 0 0 0

0 155 -155 155 -155 0

0 -155 310 -155 155 0

0 310 155 310 0 0

0 -155 -155 -155 0 0

0 0 0 0 0 0

The output of the convolution is typically passed to an activation function, which is often a *Rectified Linear Unit* (ReLU) function that ensures negative values are set to 0:

0 0 0 0 0 0

0 155 0 155 0 0

0 0 310 0 155 0

0 310 155 310 0 0

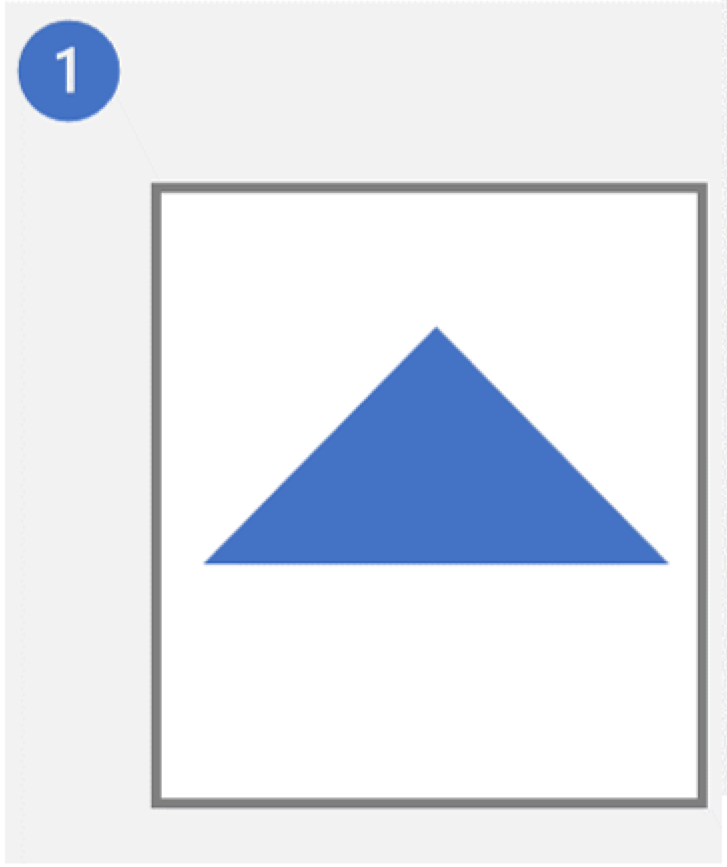
0 0 0 0 0 0

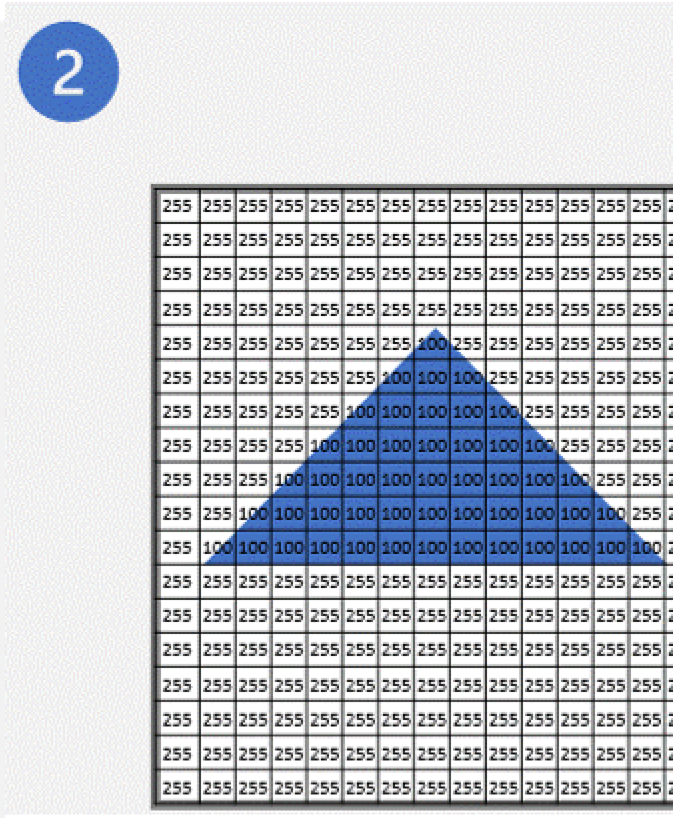
Convolution Layers..

The resulting matrix is a *feature map* of feature values that can be used to train a machine learning model.

The convolution process is follows:

1. An image is passed to the convolutional layer. In this case, the image is a simple geometric shape.
2. The image is composed of an array of pixels with values between 0 and 255 (for color images, this is usually a 3-dimensional array with values for red, green, and blue channels).



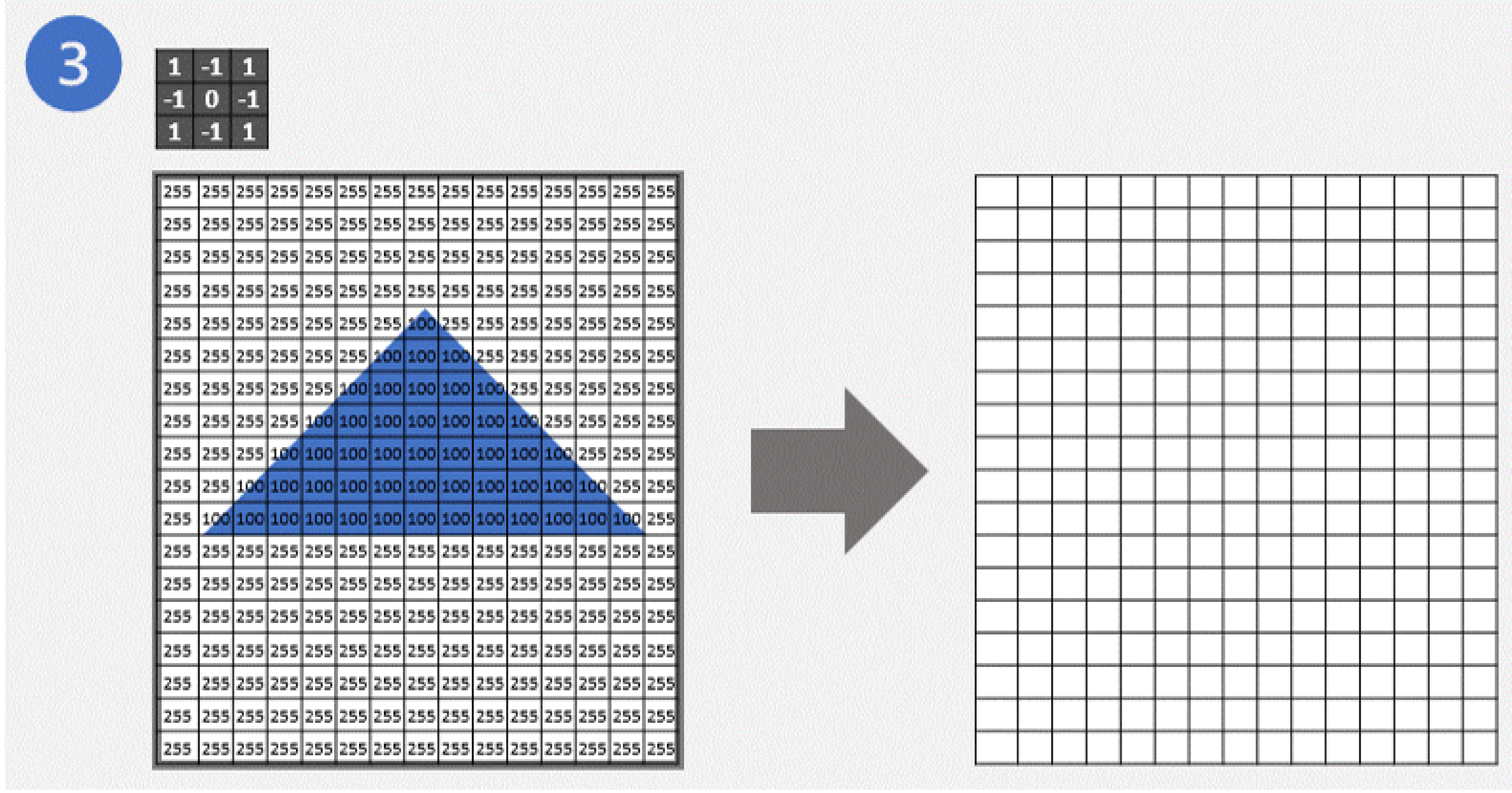


Convolution Layers..

The resulting matrix is a *feature map* of feature values that can be used to train a machine learning model.

The convolution process is follows:

1. ..
2. ..
3. A filter kernel is generally initialized with random weights (in this example, we've chosen values to highlight the effect that a filter might have on pixel values; but in a real CNN, the initial weights would typically be generated from a random Gaussian distribution). This filter will be used to extract a feature map from the image data.

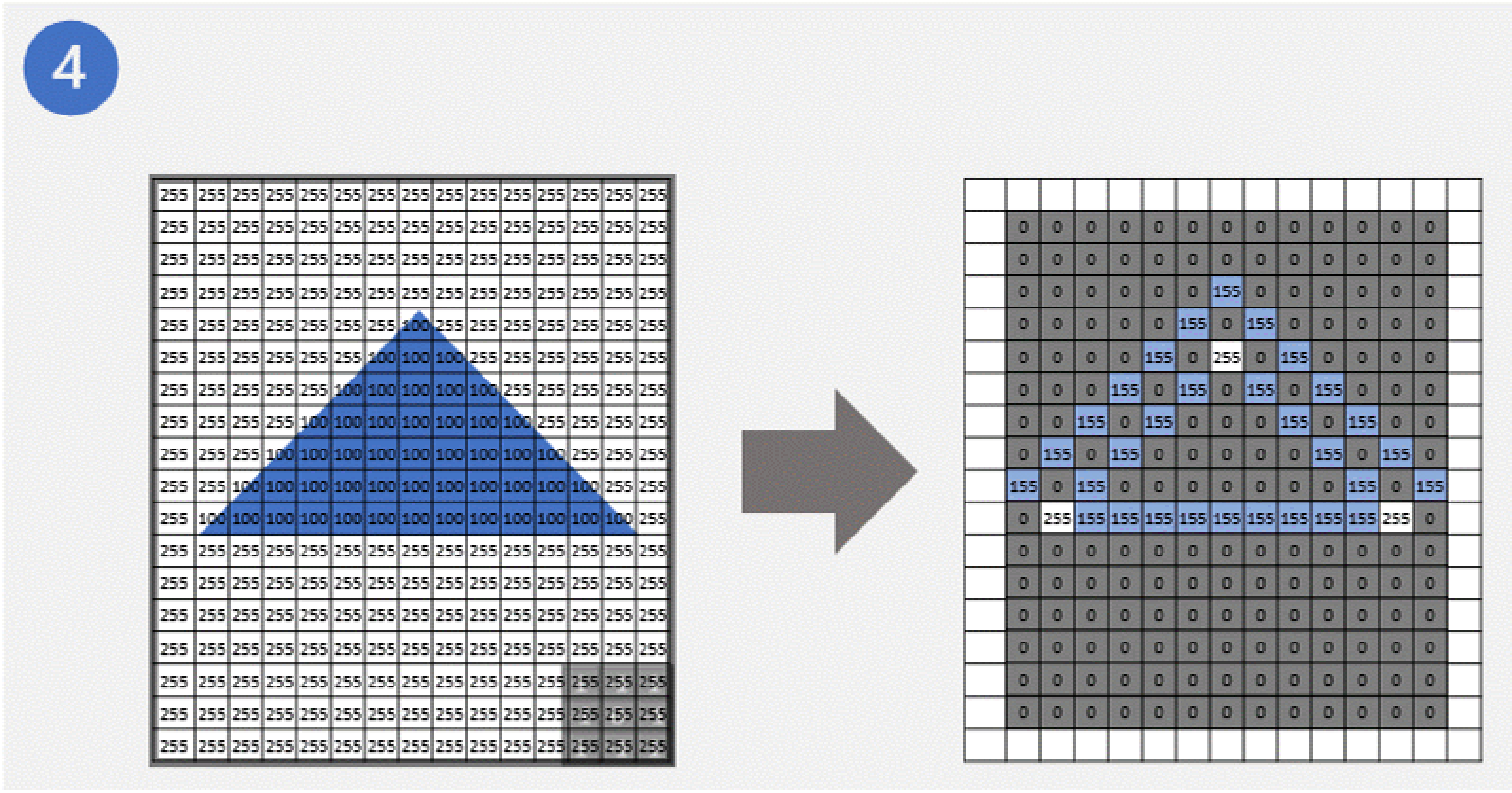


Convolution Layers..

The resulting matrix is a *feature map* of feature values that can be used to train a machine learning model.

The convolution process is follows:

1. ..
2. ..
3. ..
4. The filter is convolved across the image, calculating feature values by applying a sum of the weights multiplied by their corresponding pixel values in each position. A Rectified Linear Unit (ReLU) activation function is applied to ensure negative values are set to 0.

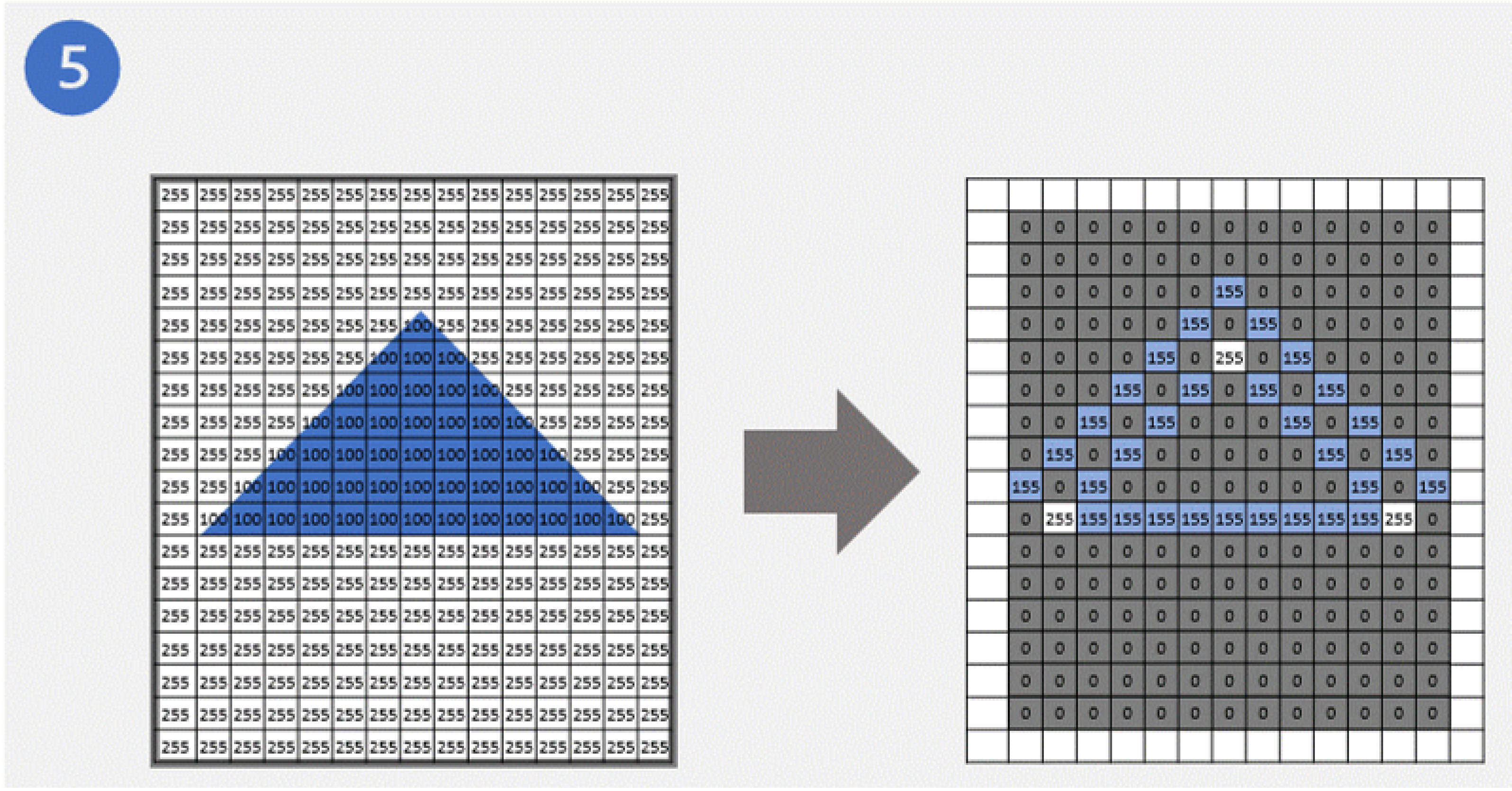


Convolution Layers..

The resulting matrix is a *feature map* of feature values that can be used to train a machine learning model.

The convolution process is follows:

1. ..
2. ..
3. ..
4. ..
5. After convolution, the feature map contains the extracted feature values, which often emphasize key visual attributes of the image. In this case, the feature map highlights the edges and corners of the triangle in the image.



Note: Typically, a convolutional layer applies multiple filter kernels. Each filter produces a different feature map, and all of the feature maps are passed onto the next layer of the network.

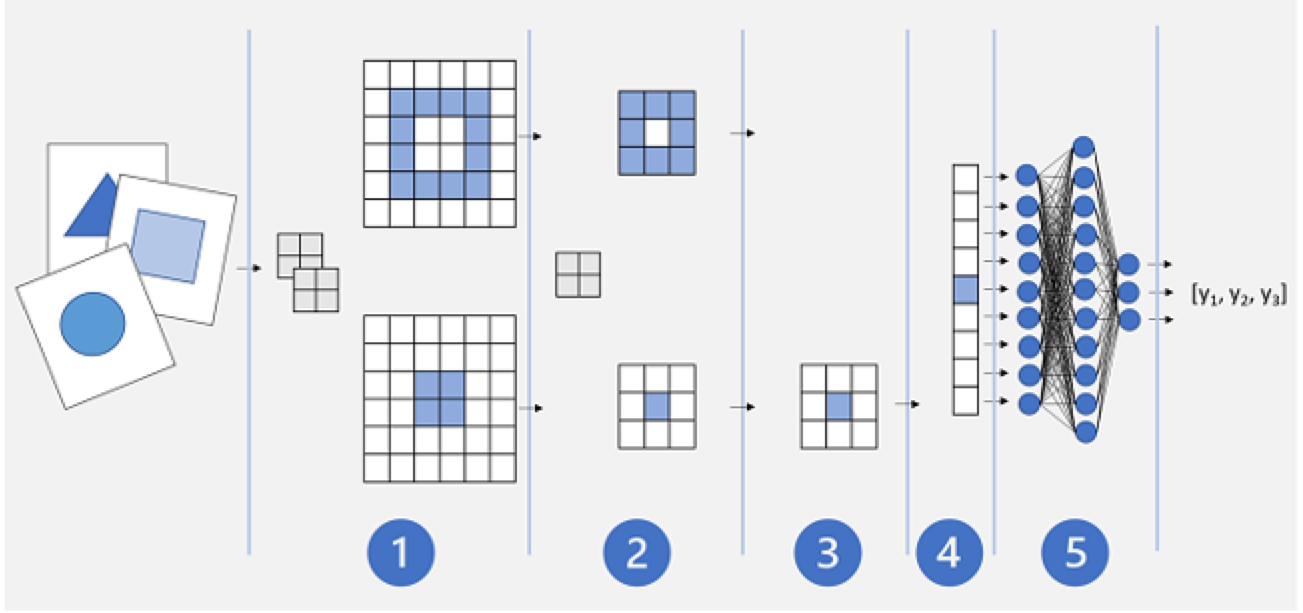
Pooling layers:

After extracting feature values from images, *pooling* (or *downsampling*) layers are used to reduce the number of feature values while retaining the key differentiating features that have been extracted

Flattening layer:

A flattening layer is used to flatten the feature maps into a vector of values that can be used as input to a fully connected layer.

The Fully connected CNN



1. Images are fed into a convolutional layer. In this case, there are two filters, so each image produces two feature maps.
2. The feature maps are passed to a pooling layer, where a 2x2 pooling kernel reduces the size of the feature maps.
3. A dropping layer randomly drops some of the feature maps to help prevent overfitting.
4. A flattening layer takes the remaining feature map arrays and flattens them into a vector.
5. The vector elements are fed into a fully connected network, which generates the predictions.