THE UNIVERSITY of TENNESSEE KNOXVILLE

Lecture 32: Deep Convolutional Neural Networks

Instructor: Sergei V. Kalinin

This and that

Midterm 1:

- Some solutions are pretty impressive!
- Common problem – labelling!
- All problems – examples before
- Linear regression problem – solution in: https://github.com/SergeiVKalinin/MSE_Fall2023/blob/main/Module%204/13_PCA_CL.ipynb

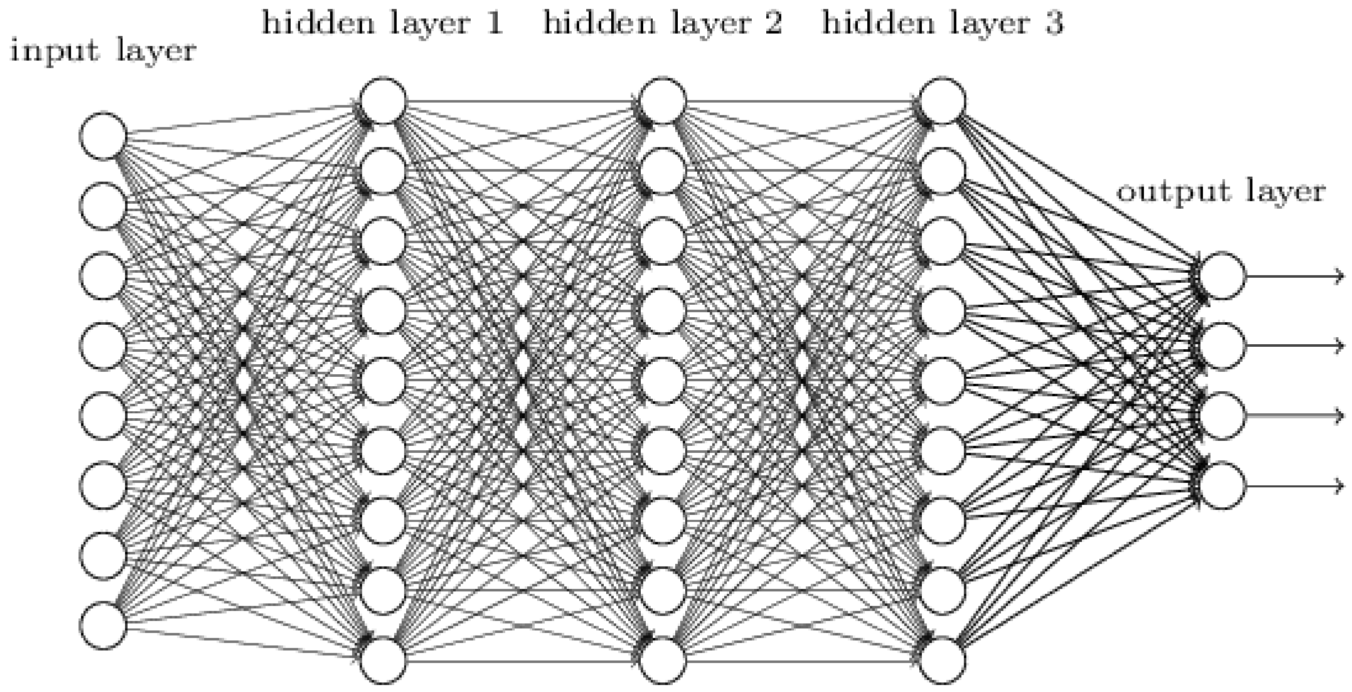
Midterm 2:

- Decision making
- Analysis of specific process/technology

Final project:

- Build neural network-based classifier, regressor, and segmentor, and quantify its performance
- Compare NN performance to linear methods
- Explore specific data set via VAE

Putting Neurons Together



- Linear neurons and activation functions
- Input dimensionality
- Output dimensionality
- Special attention: activation function of the last layer

- Loss function – how different is what we get from what we want (supervised ML)
- Backpropagation – how we adapt the neuron settings given the loss
- Metrics – how we monitor the training/performance

Loss functions for classifiers

1. Binary Cross-Entropy: Measures the performance of a classification model whose output is a probability between 0 and 1. It's suitable for models where each instance is expected to belong to one of two classes (Sigmoid)

2. Categorical Cross-Entropy: Used when there are two or more label classes. Assumes a single-label, multi-class classification problem where each class is mutually exclusive (Softmax)

3. Sparse Categorical Cross-Entropy: Similar to CCE but more efficient with a large number of classes. Used when the classes are mutually exclusive and the labels are integers (Softmax)

4. Hinge Loss: Commonly used with Support Vector Machine (SVM) classifiers but applicable to neural networks for binary classification tasks. It's designed to create a margin between data points (Linear or Tanh)

5. Focal Loss: An extension of cross-entropy loss, adding a focusing parameter to balance the importance of correctly classifying different classes, particularly useful in imbalanced datasets. Sigmoid (Binary), Softmax (Multi-class)

6. Kullback-Leibler Divergence: Measures how one probability distribution diverges from a second, expected probability distribution. Used in scenarios like learning a complex multi-class classification problem or training a model to output a probability distribution (Softmax)

Loss functions for regressors

1. Mean Squared Error (MSE): Standard Regression
Calculates the average squared difference between the estimated values and the actual value. Commonly used in regression problems and is sensitive to outliers (Linear).

2. Mean Absolute Error (MAE): Standard Regression
Computes the average of the absolute differences between predictions and actual values. Less sensitive to outliers compared to MSE (Linear)

3. Huber Loss: Robust Regression
Less sensitive to outliers than the MSE. It's quadratic for small errors and linear for large errors, combining the best properties of MSE and MAE (Linear)

4. Quantile Loss: Quantile Regression
Used in scenarios where the goal is to predict an interval instead of only point estimates. It allows the model to learn to predict a range that contains the true value with a certain probability (Linear).

7. Poisson Loss: Count Data Regression
Suitable for modeling count data and intensity rates where the output values are non-negative integers (Exponential)

Loss functions for image segmentation

1. Binary Cross-Entropy: Measures the pixel-wise binary classification loss. Commonly used in binary segmentation tasks (e.g., object vs. background). Ideal for problems where each pixel is classified independently into two classes.

2. Categorical Cross-Entropy: An extension of binary cross-entropy for multi-class segmentation tasks. It calculates the loss for each pixel and classifies it into one of the multiple classes. Used in scenarios with multiple classes (e.g., different types of objects in an image).

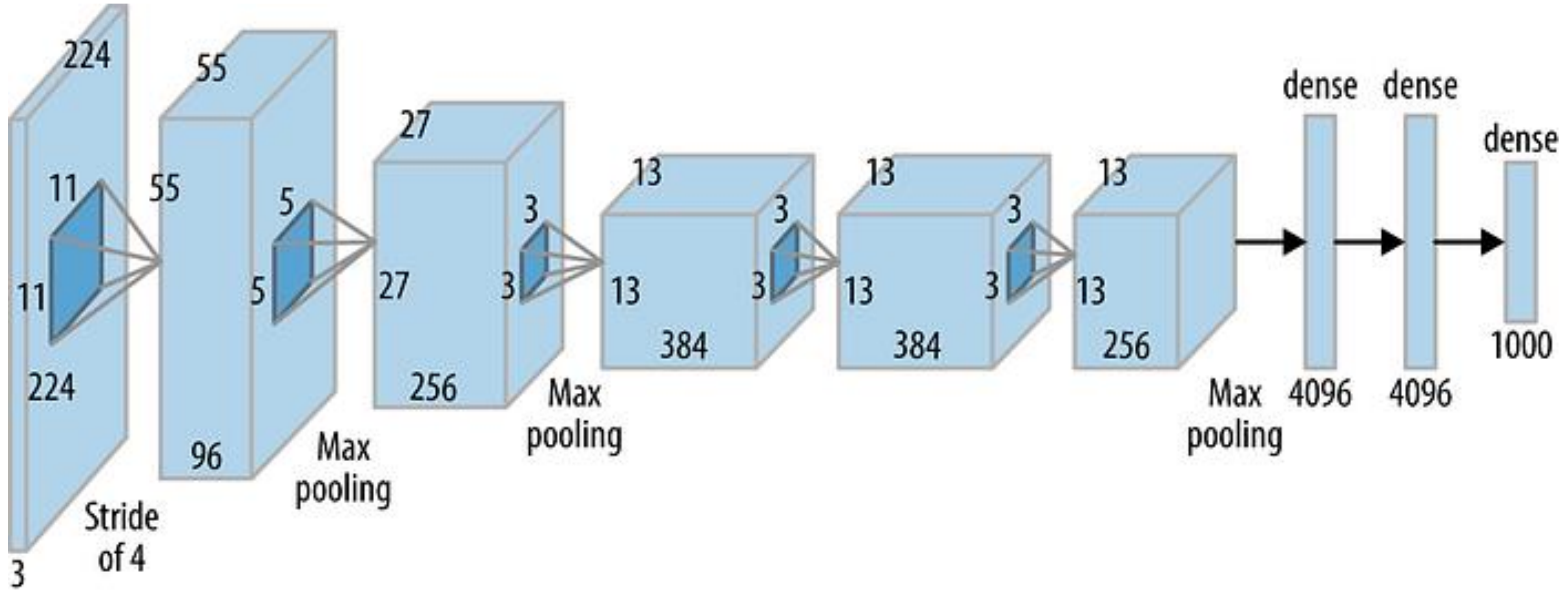
3. Dice Loss: Based on the Dice coefficient, which is a measure of overlap between two samples. Particularly effective for data with imbalanced classes, such as medical image segmentation.

4. Jaccard/Intersection Over Union (IoU) Loss. Based on Jaccard index. Useful in evaluating the similarity between the predicted segmentation and the ground truth, especially when the classes are imbalanced.

5. Weighted Cross-Entropy: A variant of cross-entropy loss where each class is given a specific weight, usually inversely proportional to the class frequency. Useful in cases of class imbalance where certain classes need more focus.

Deep Convolutional Neural Networks

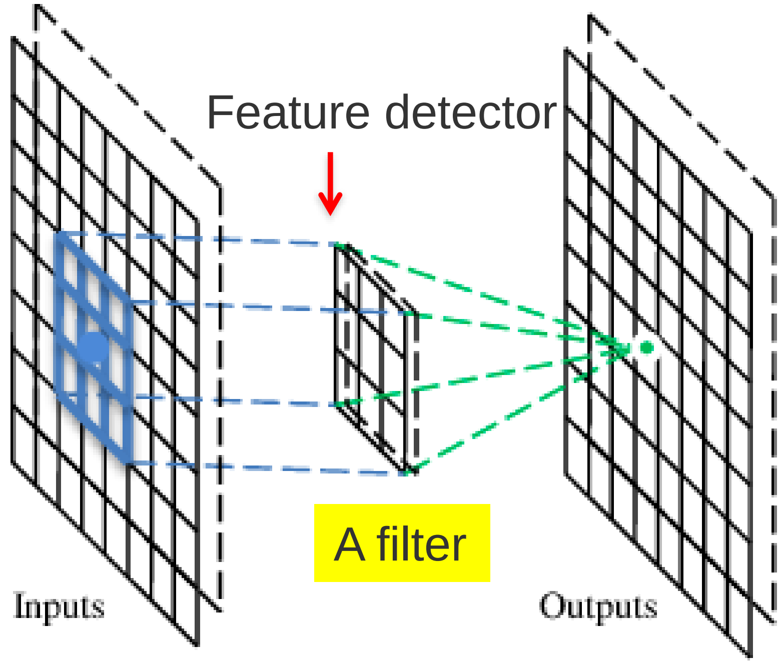
Structure of AlexNet



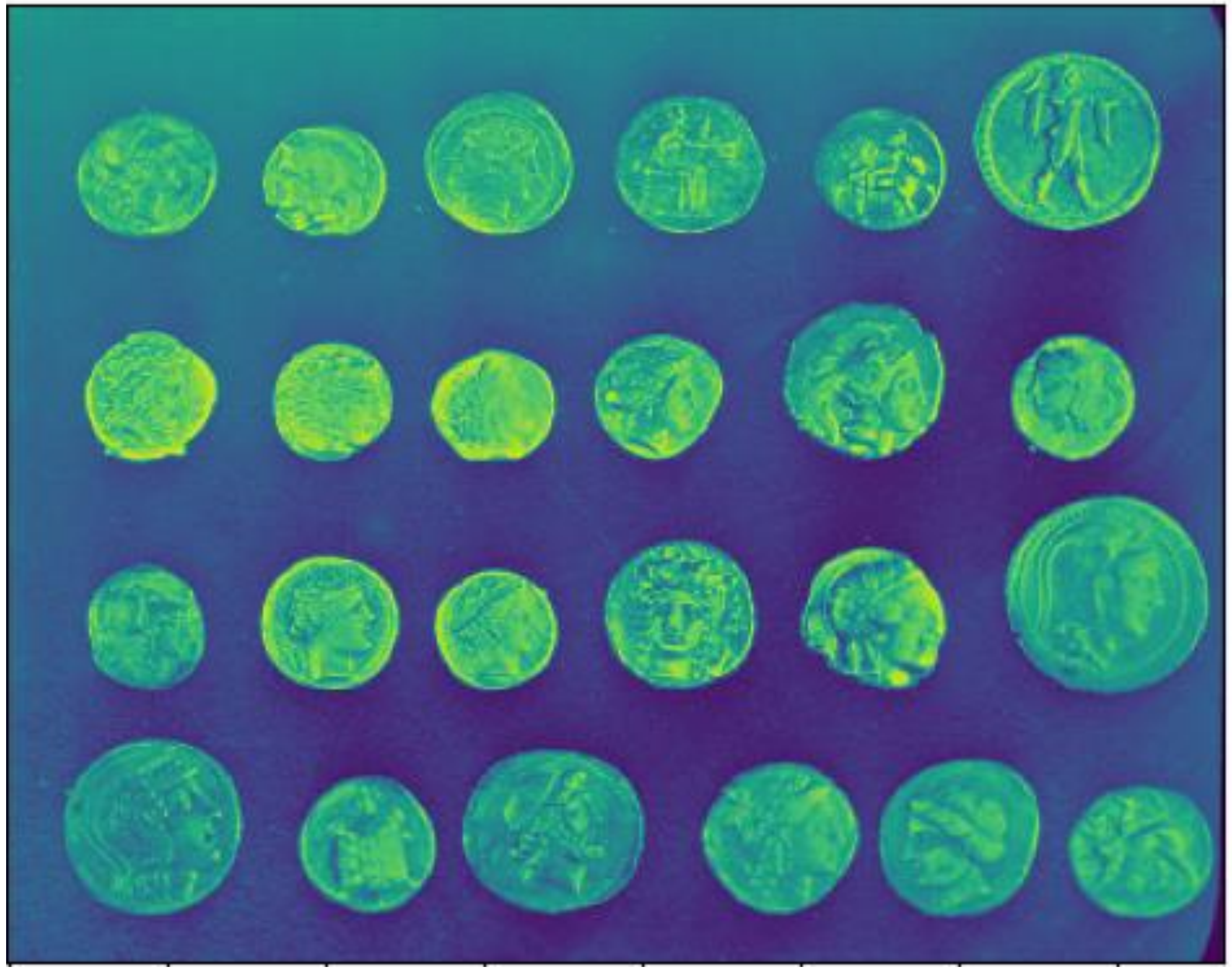
https://towardsdatascience.com/the-w3h-of-alexnet-vggnet-resnet-and-inception-7baaaecccc96

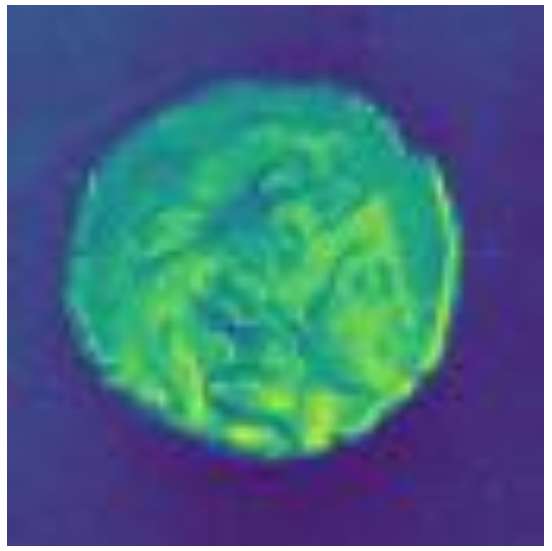
Finding features in images

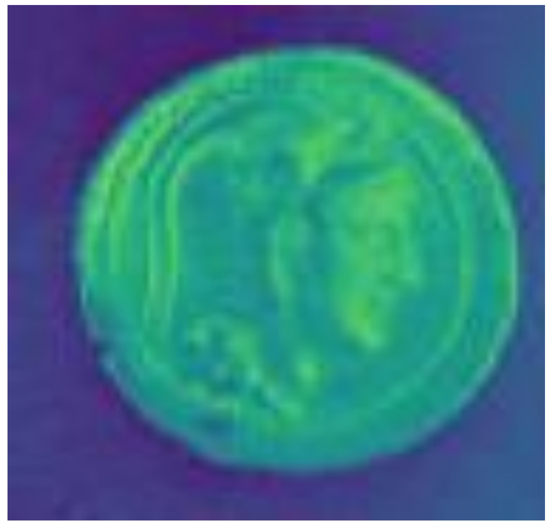
A CNN is a neural network with some convolutional layers (and some other layers). A convolutional layer has a number of filters that does convolutional operation.



Coin example (scikit-image)

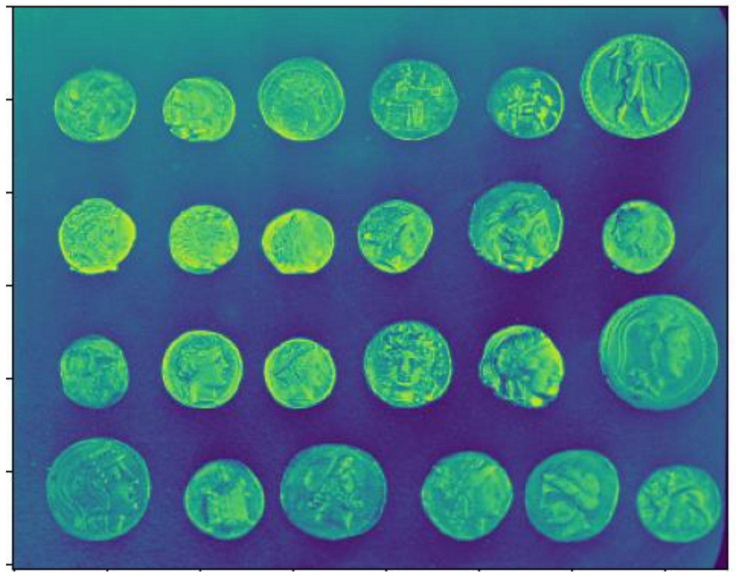


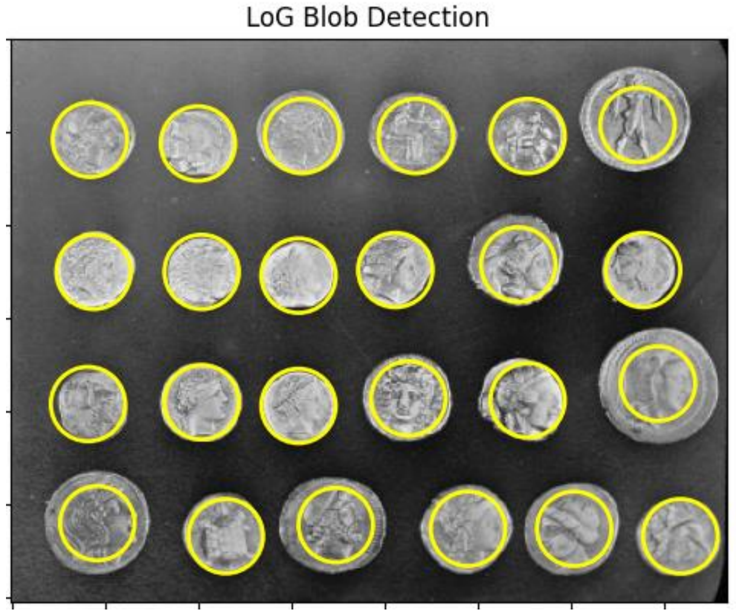


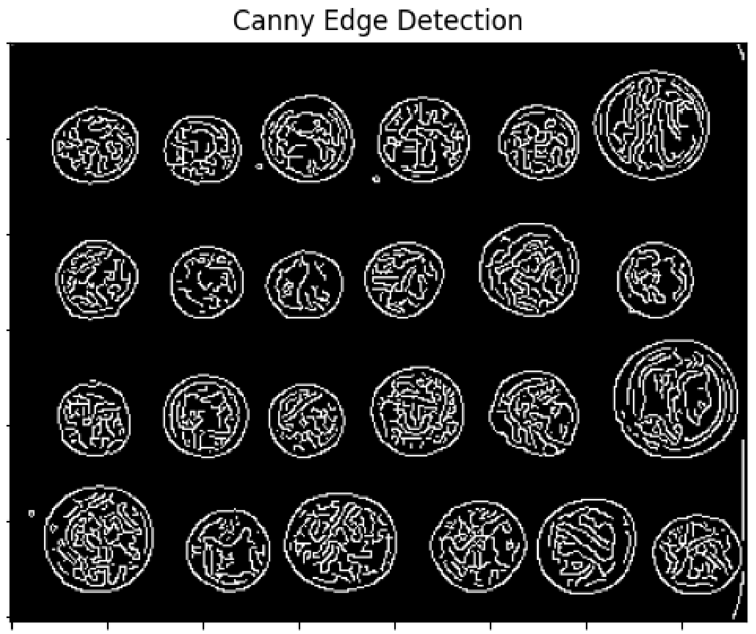


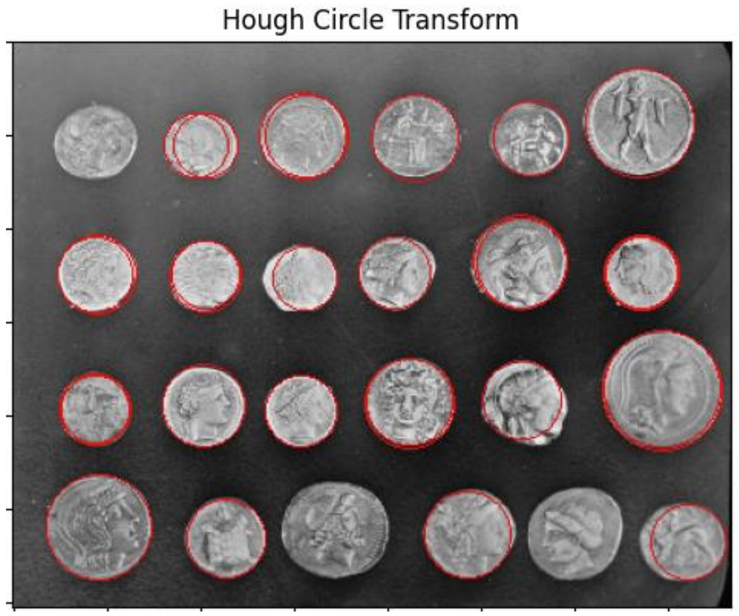
What are features in this image?

Feature finding in scikit-image

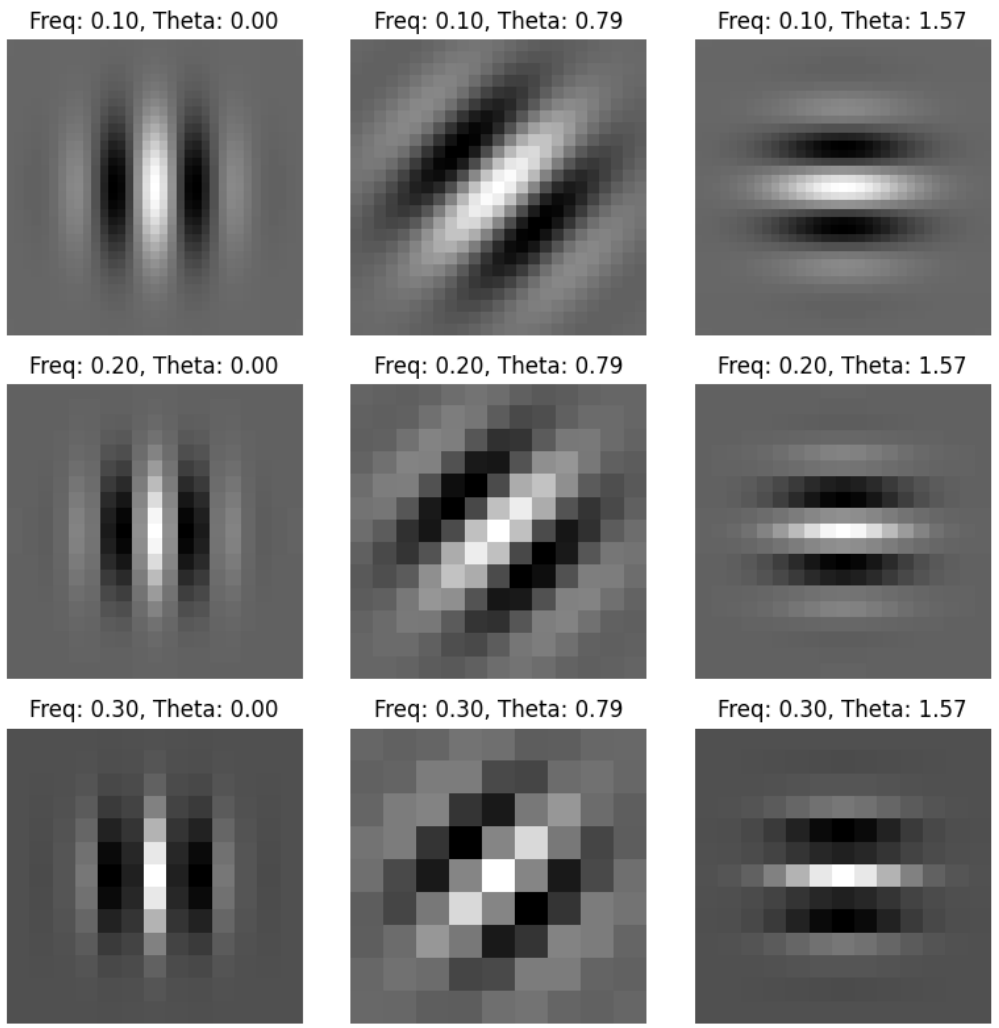


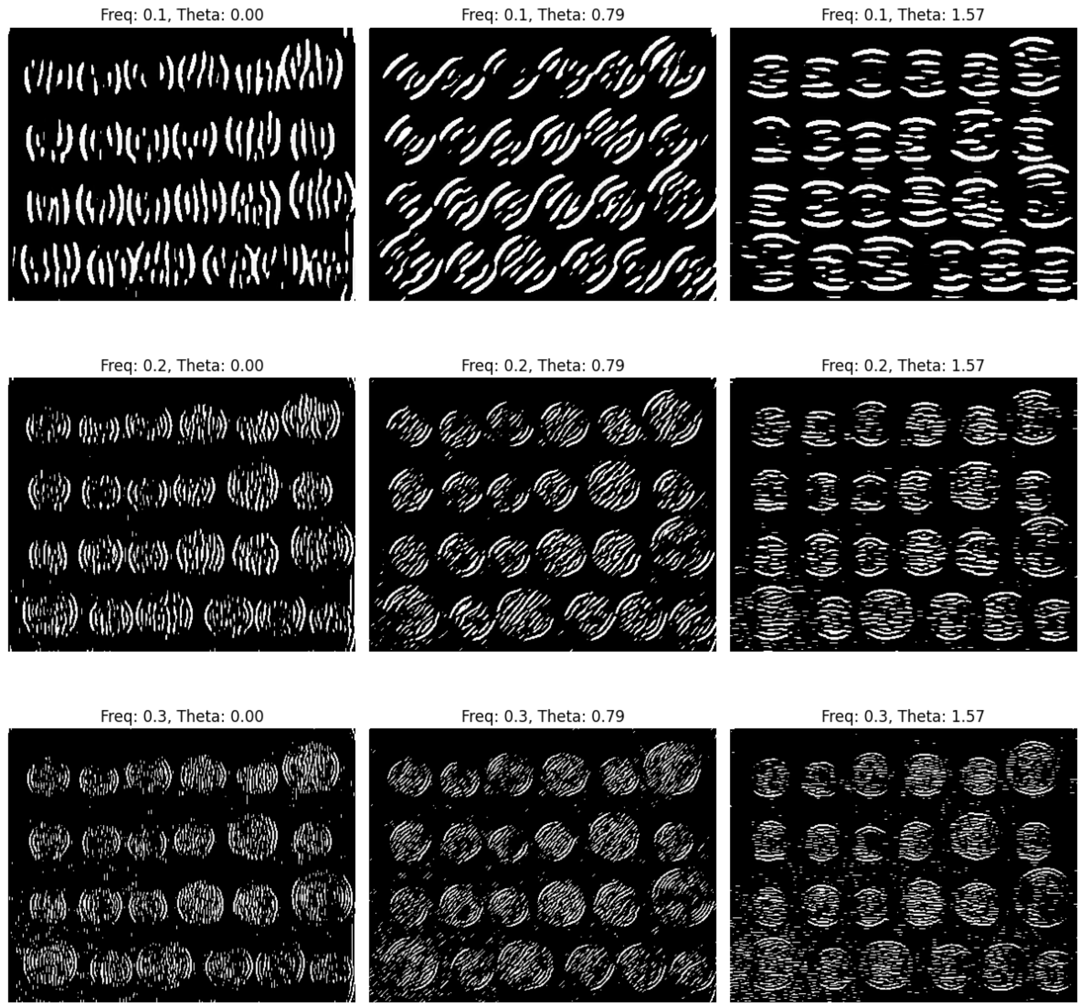






Gabor filters





What is convolution?

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

6 x 6 image

These are the network parameters to be learned.

1	-1	-1
-1	1	-1
-1	-1	1

Filter 1

-1	1	-1
-1	1	-1
-1	1	-1

Filter 2

⋮ ⋮

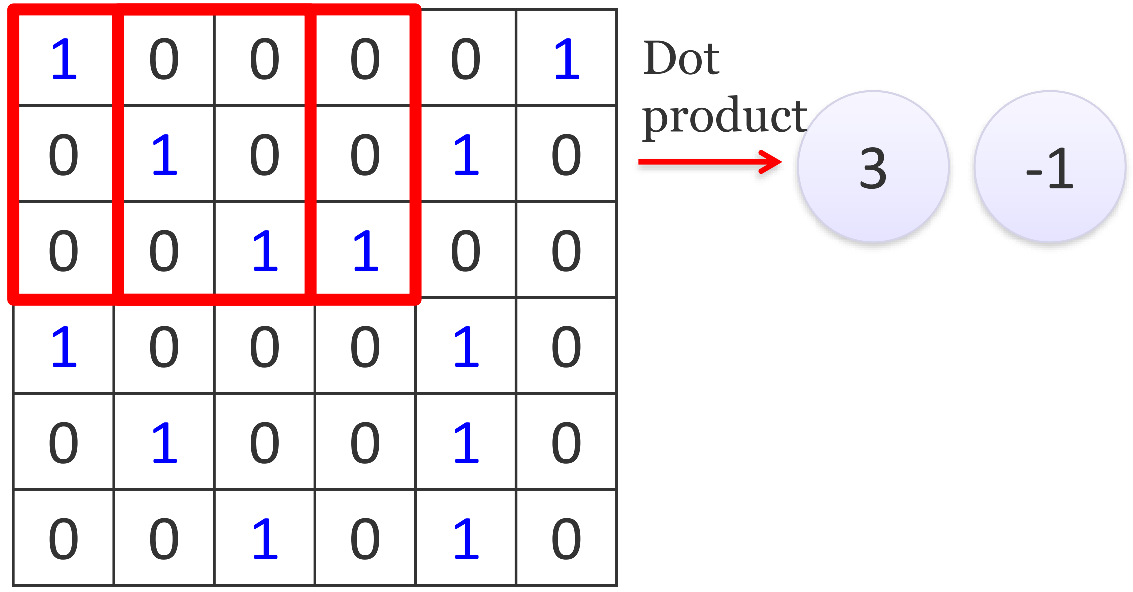
Each filter detects a small pattern (3 x 3).

What is convolution?

1	-1	-1
-1	1	-1
-1	-1	1

Filter 1

stride=1



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

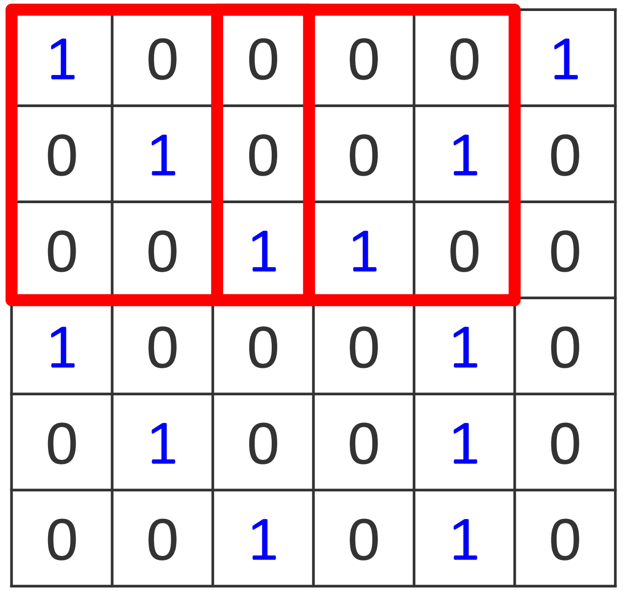
6 x 6 image

What is convolution?

1	-1	-1
-1	1	-1
-1	-1	1

Filter 1

If stride=2

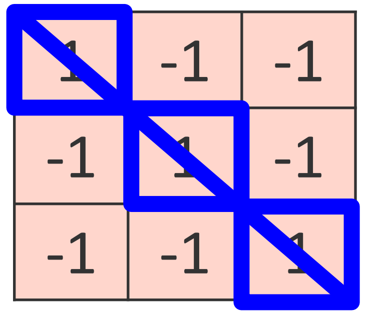


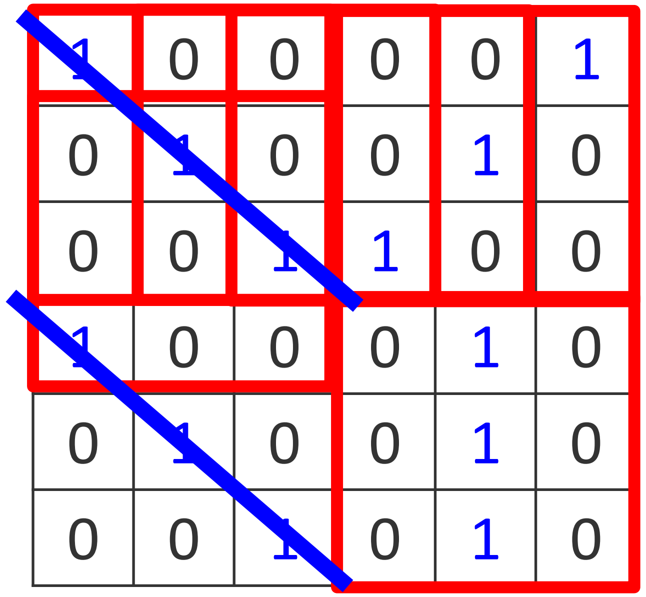
1	0	0	0	0	1
0	1	0	0	1	0
0	0	1	1	0	0
1	0	0	0	1	0
0	1	0	0	1	0
0	0	1	0	1	0

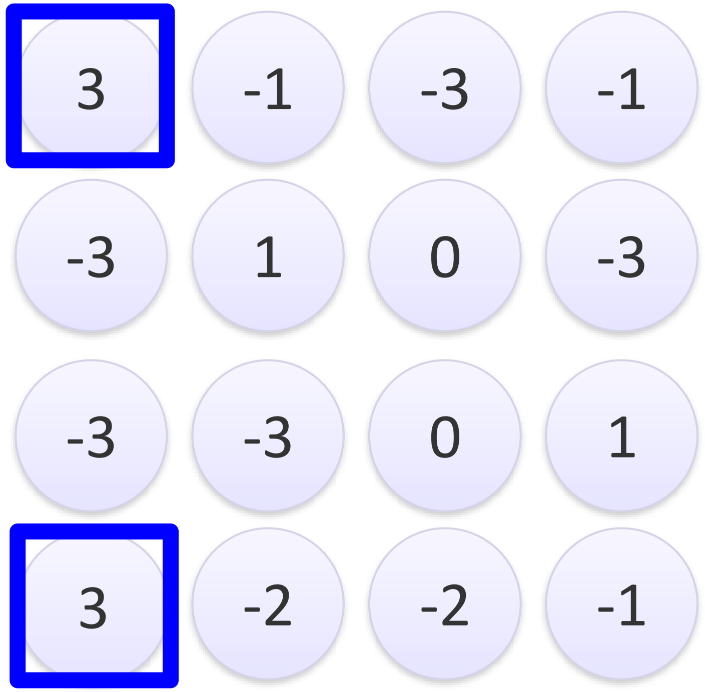
6 x 6 image

3 -3

What is convolution?







What is convolution?

-1	1	-1
-1	1	-1
-1	1	-1

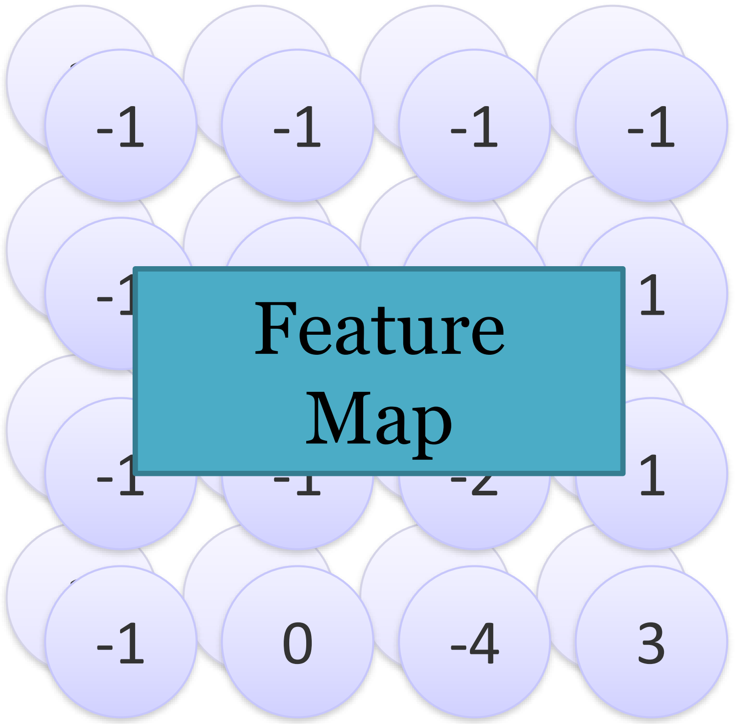
Filter 2

stride=1

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

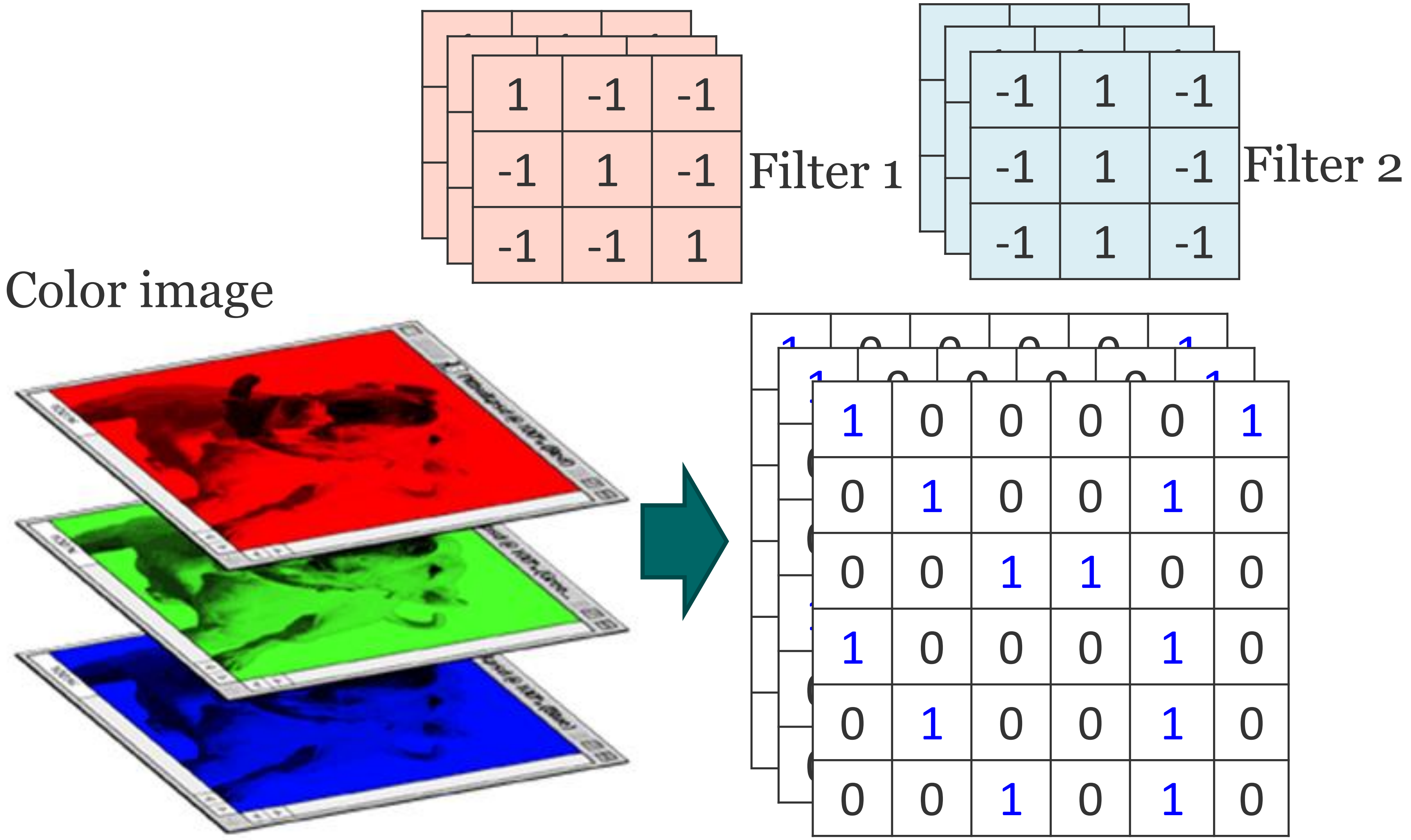
6 x 6 image

Repeat this for each filter



Two 4 x 4 images
Forming 2 x 4 x 4 matrix

But what about color?



Filter 1

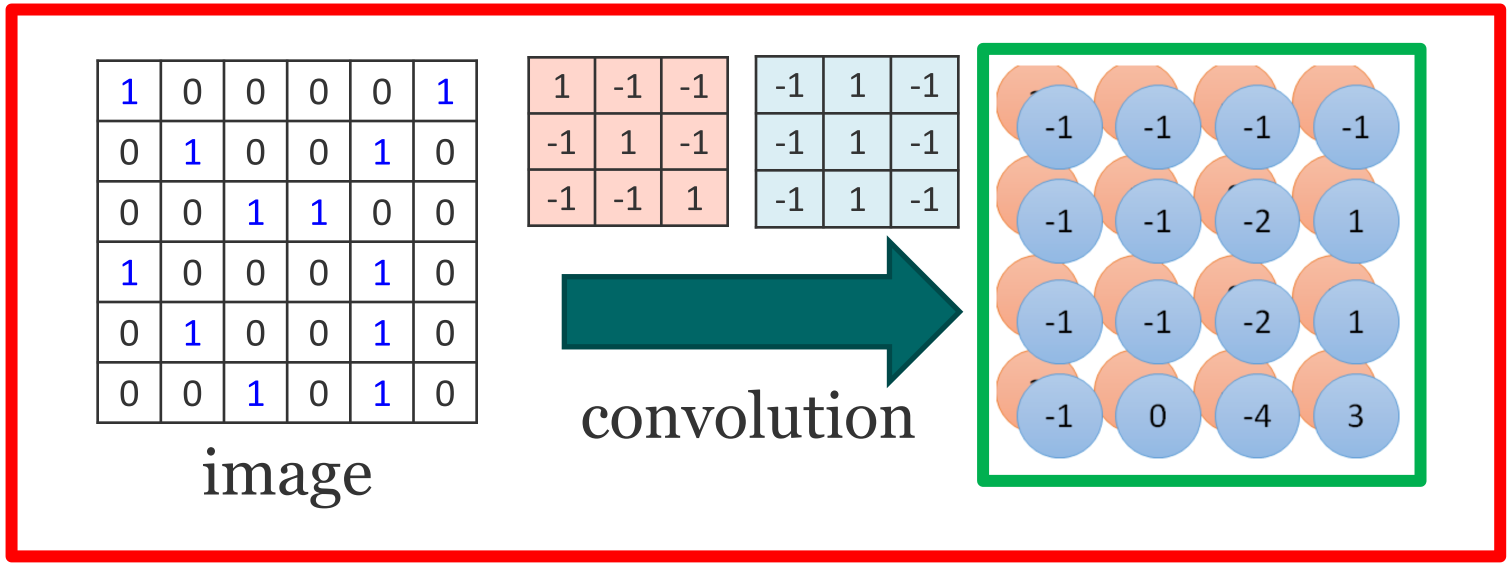
1	-1	-1
-1	1	-1
-1	-1	1

Filter 2

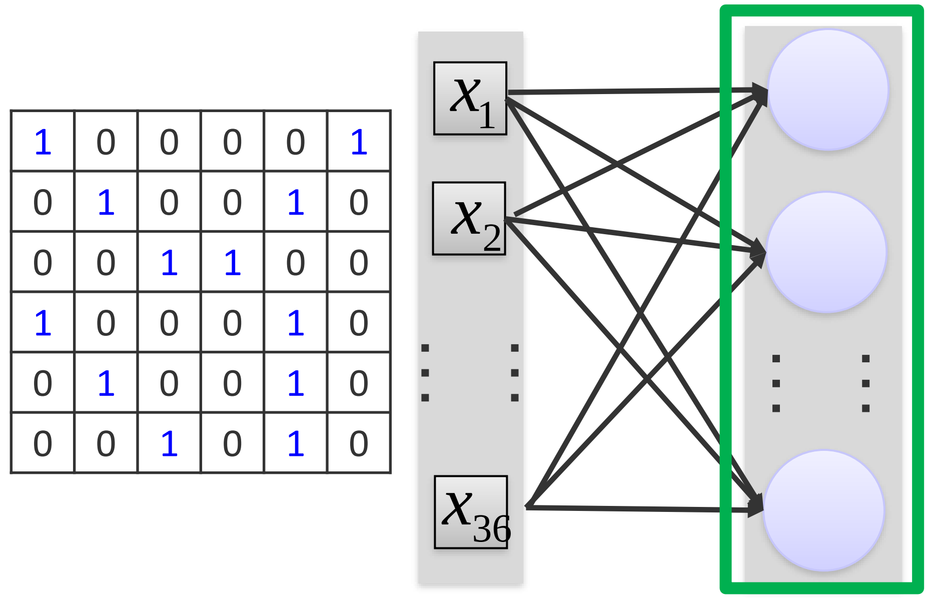
-1	1	-1
-1	1	-1
-1	1	-1

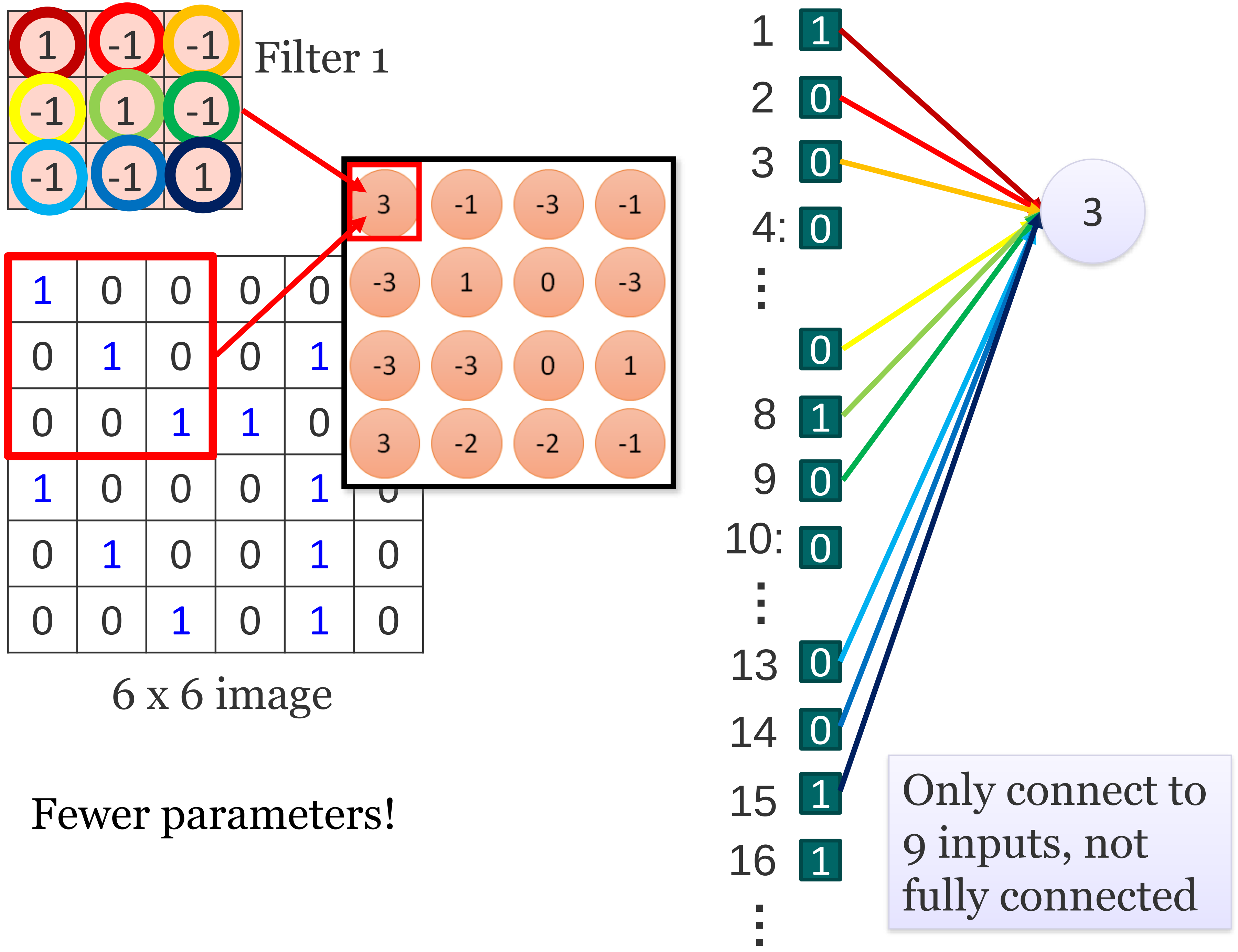
1	0	0	0	0	1
0	1	0	0	1	0
0	0	1	1	0	0
1	0	0	0	1	0
0	1	0	0	1	0
0	0	1	0	1	0

Convolution vs. fully connected



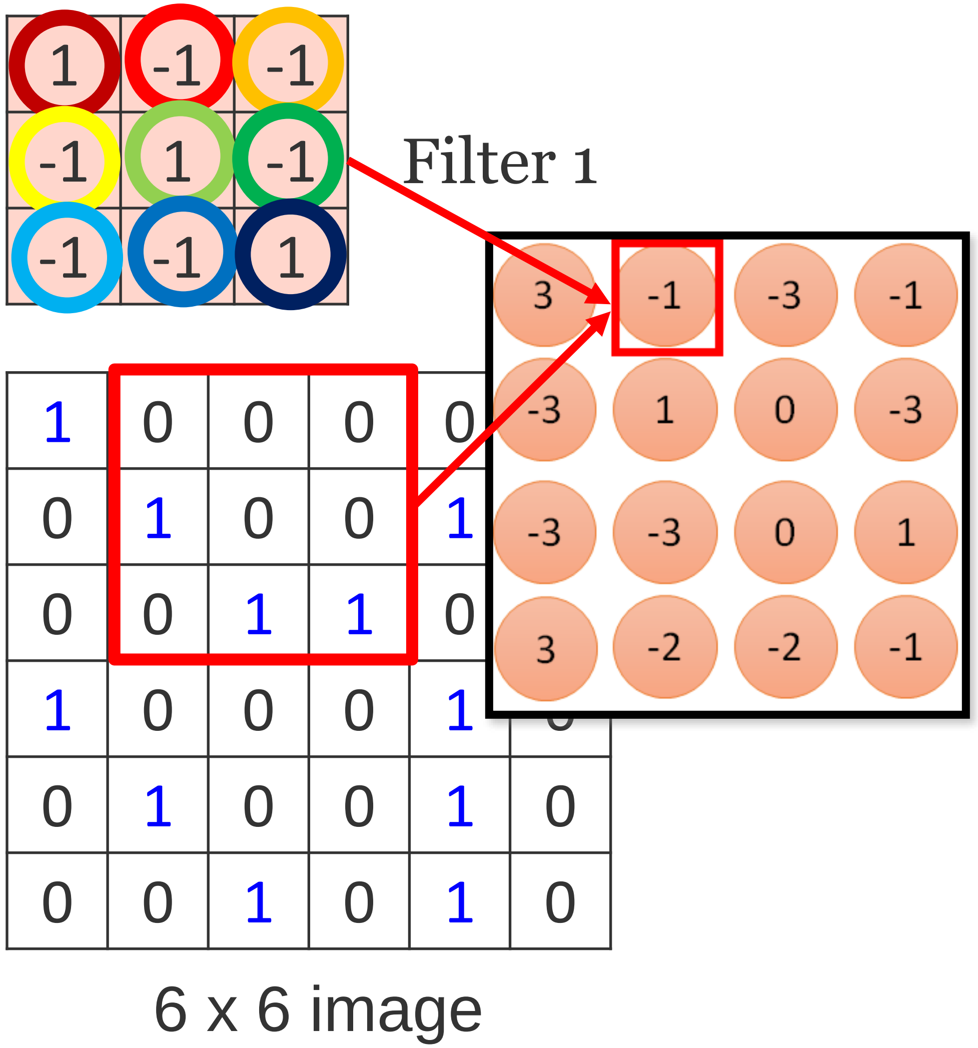
Fully-connected



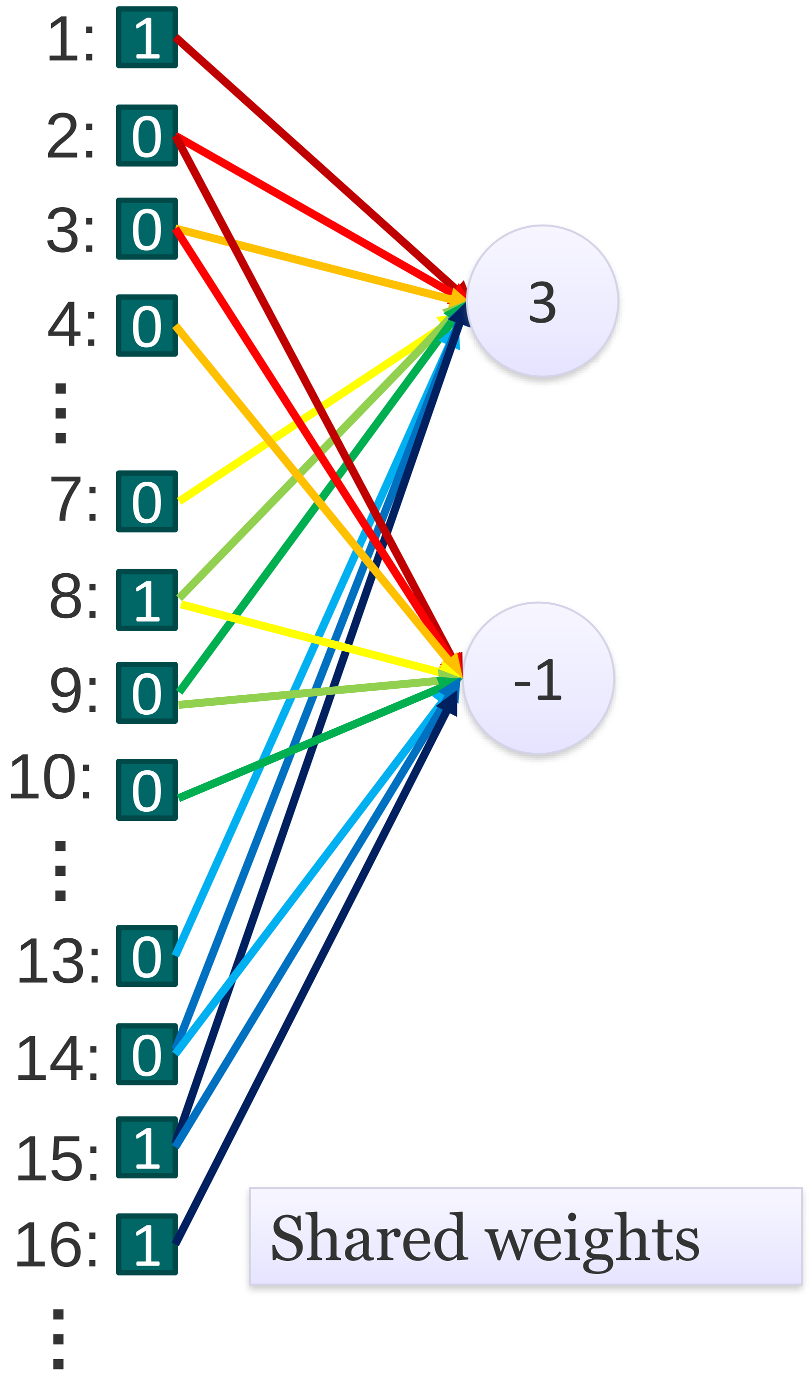


Only connect to 9 inputs, not fully connected

Fewer parameters!



Fewer parameters



Scale invariance: pooling

- Subsampling pixels will not change the object

bird

bird

We can subsample the pixels to make image smaller

→ fewer parameters to characterize the image

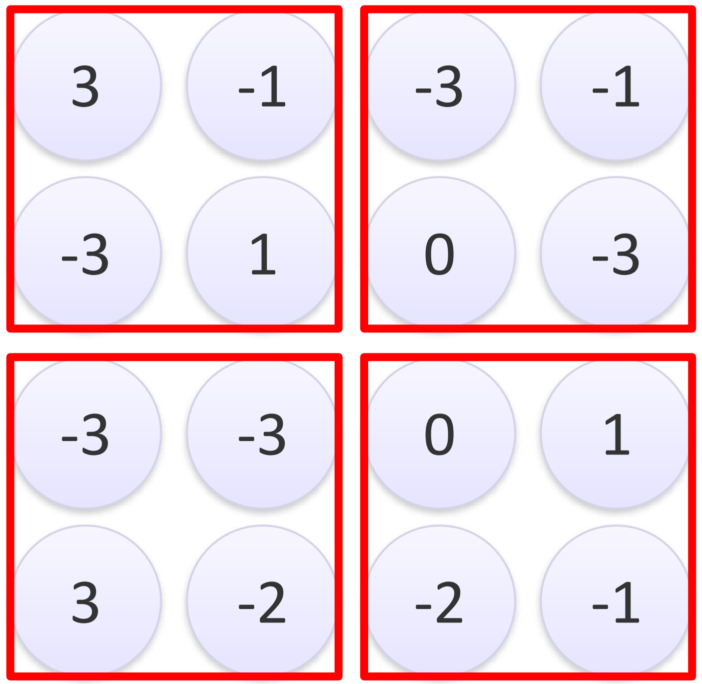
MaxPooling

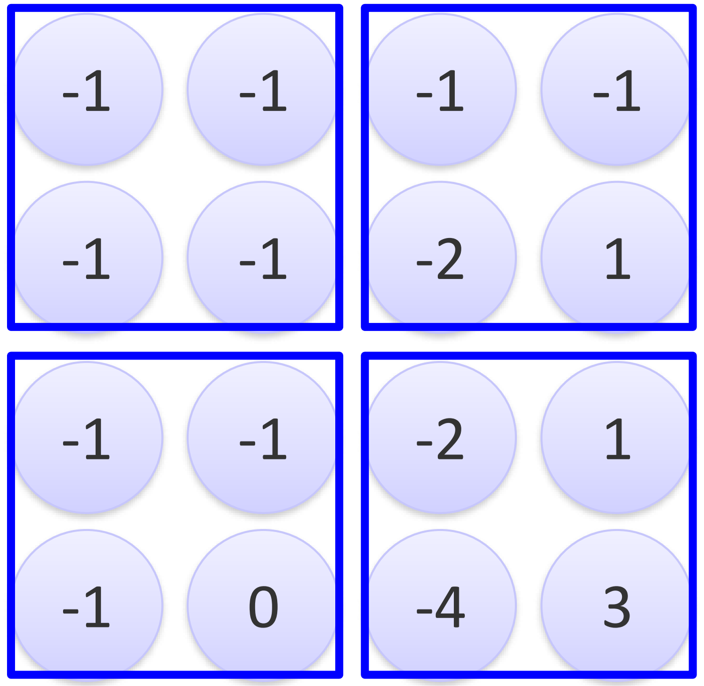
1	-1	-1
-1	1	-1
-1	-1	1

Filter 1

-1	1	-1
-1	1	-1
-1	1	-1

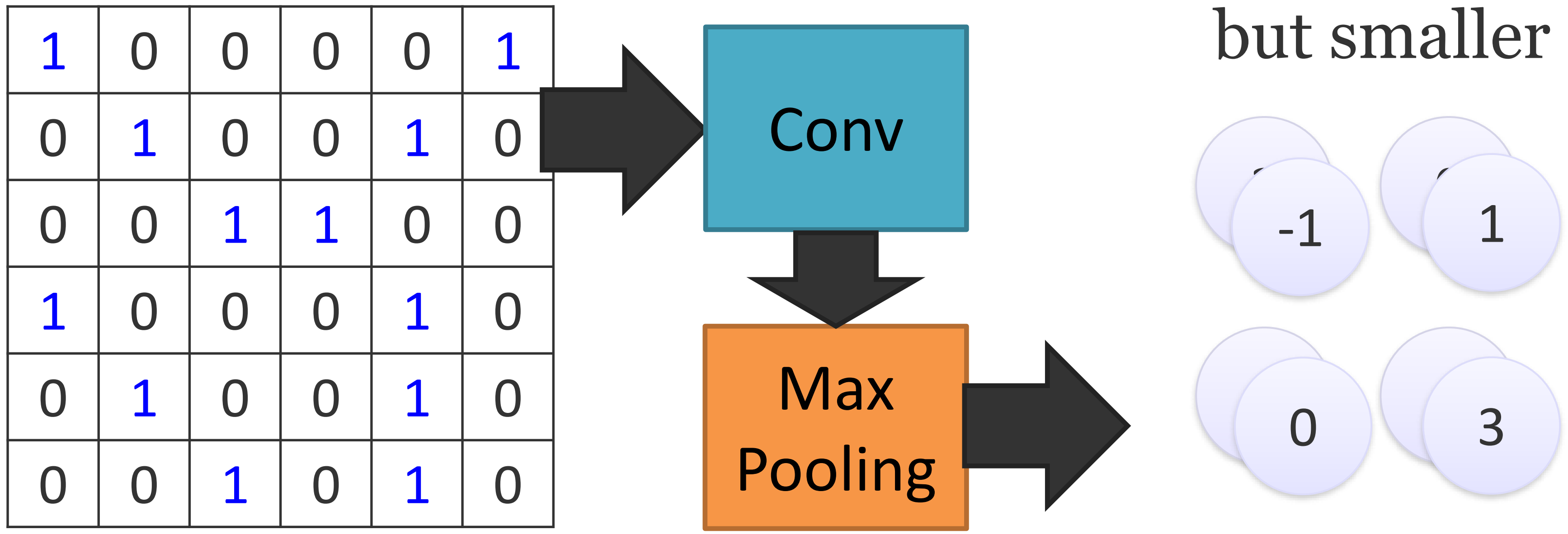
Filter 2





MaxPooling

New image
but smaller



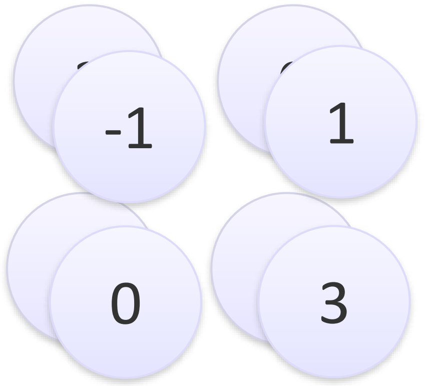
1	0	0	0	0	1
0	1	0	0	1	0
0	0	1	1	0	0
1	0	0	0	1	0
0	1	0	0	1	0
0	0	1	0	1	0

6 x 6 image

2 x 2 image

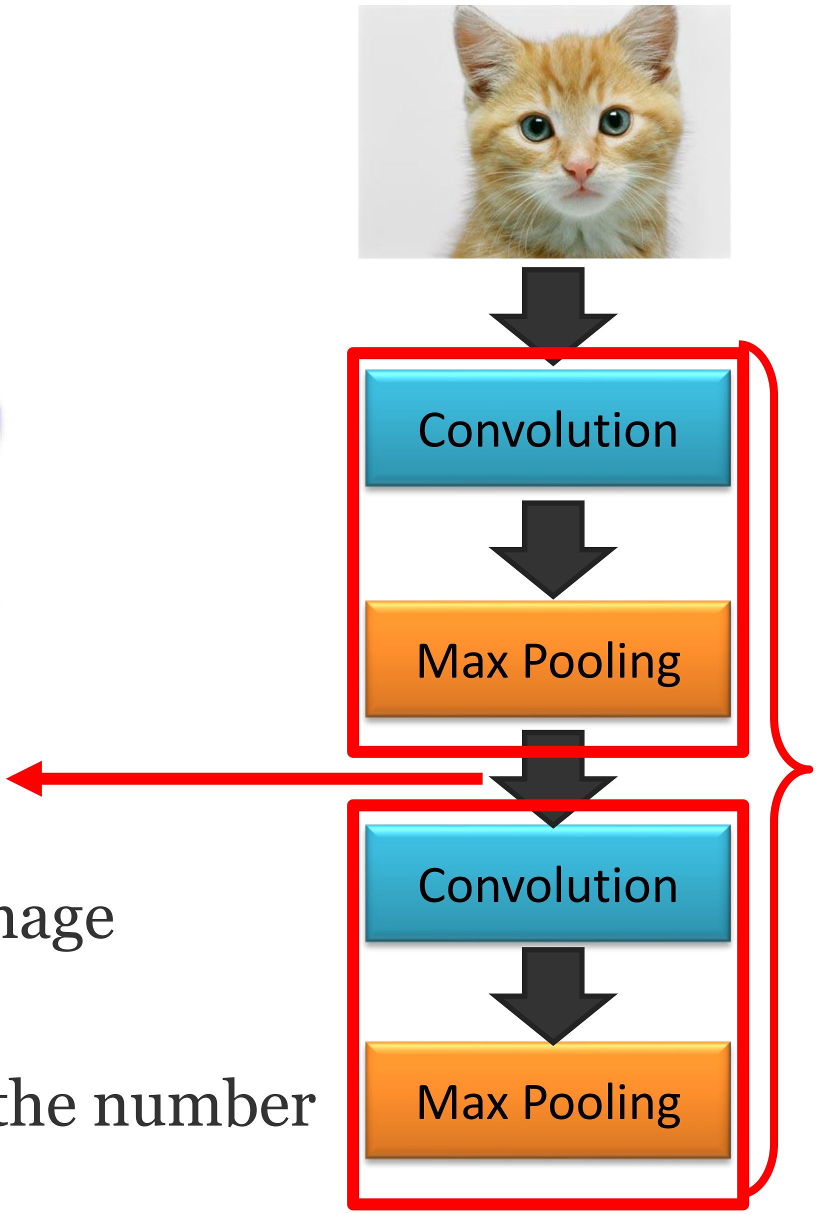
Each filter
is a channel

Making DCNN



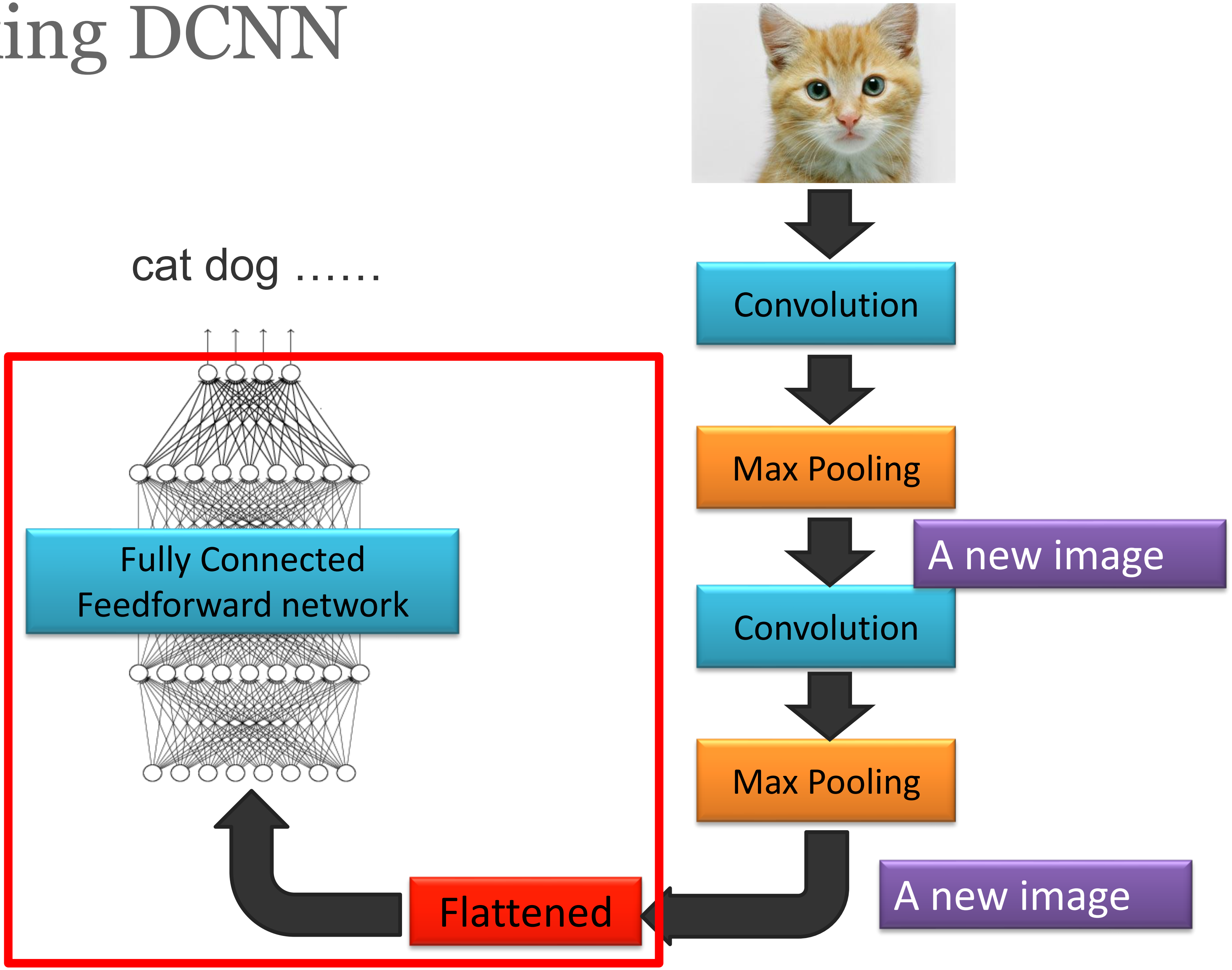
Smaller than the original image

The number of channels is the number of filters

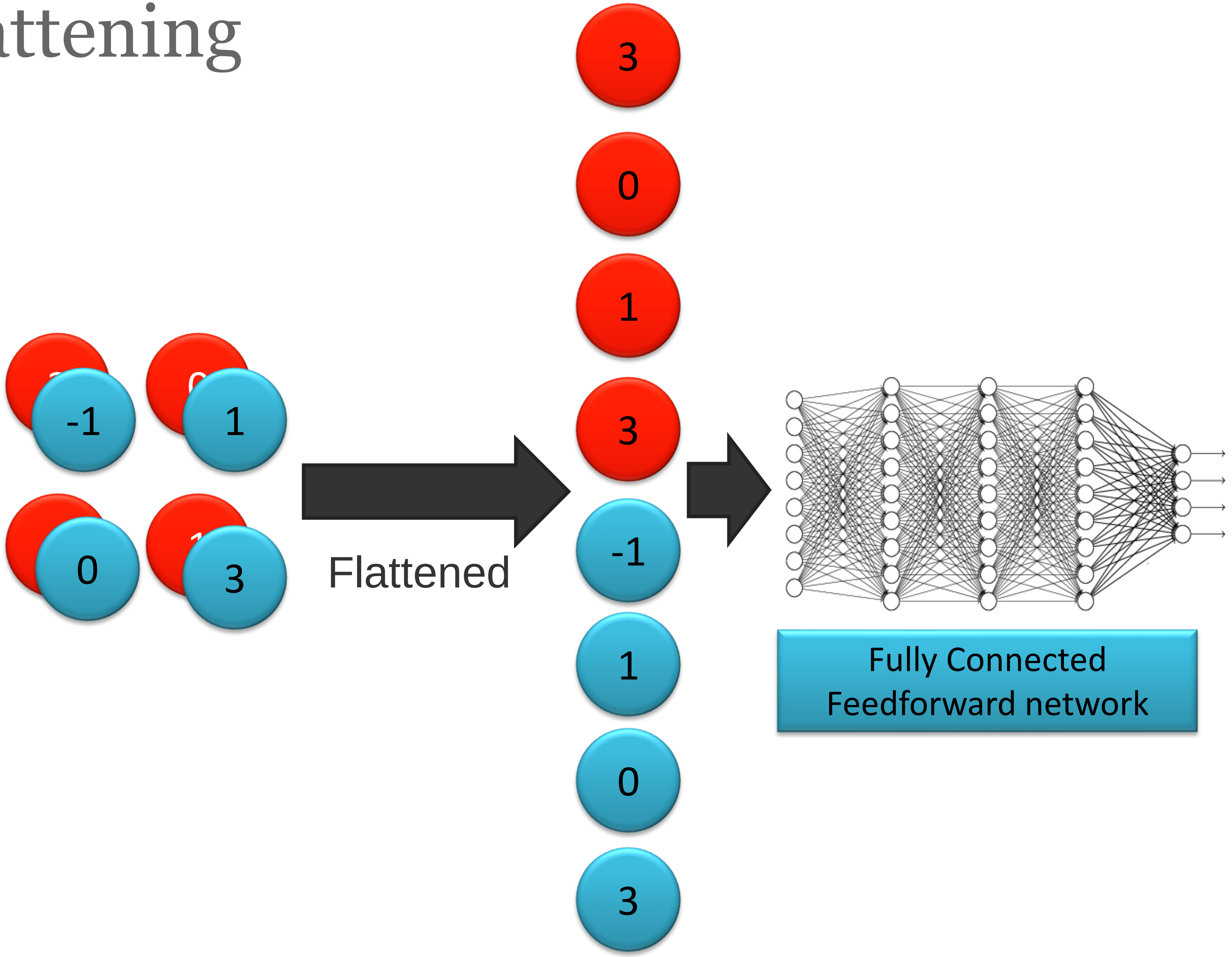


Can repeat many times

Making DCNN


cat dog ……
Fully Connected
Feedforward network
Flattened
Convolution
Max Pooling
A new image
Convolution
Max Pooling
A new image

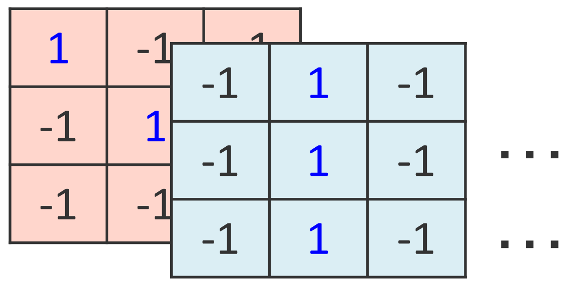
Flattening


-1
1
0
3
Flattened
3
0
1
3
-1
1
0
3
Fully Connected Feedforward network

Keras

Only modified the ***network structure*** and ***input format (vector -> 3-D tensor)***

```
model2.add( Convolution2D( 25,3,3,
             input_shape=(28,28,1)) )
```



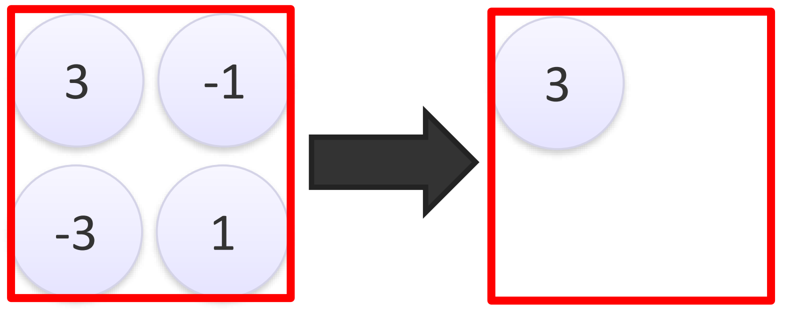
There are 25 3x3 filters.

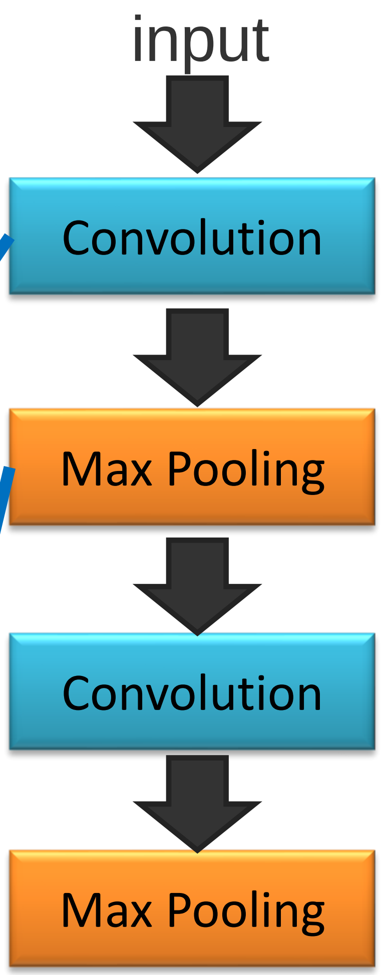
Input_shape = (28 , 28 , 1)

28 x 28 pixels

1: black/white, 3: RGB

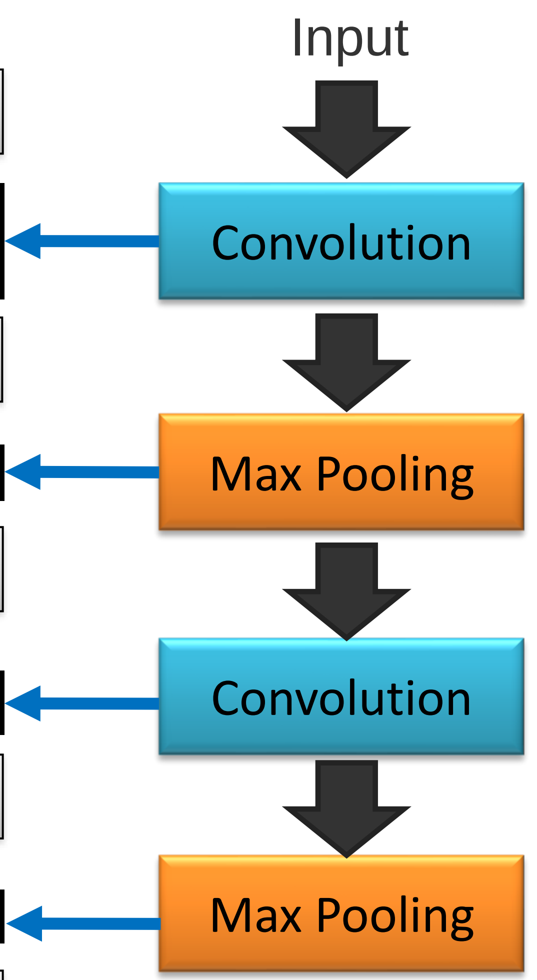
```
model2.add(MaxPooling2D((2,2)))
```





Keras

Only modified the ***network structure*** and ***input format (vector -> 3-D array)***



1 x 28 x 28

```
model2.add( Convolution2D( 25,3,3,
                  input_shape=(28,28,1)) )
```

25 x 26 x 26

```
model2.add(MaxPooling2D((2,2)))
```

25 x 13 x 13

```
model2.add(Convolution2D(50,3,3))
```

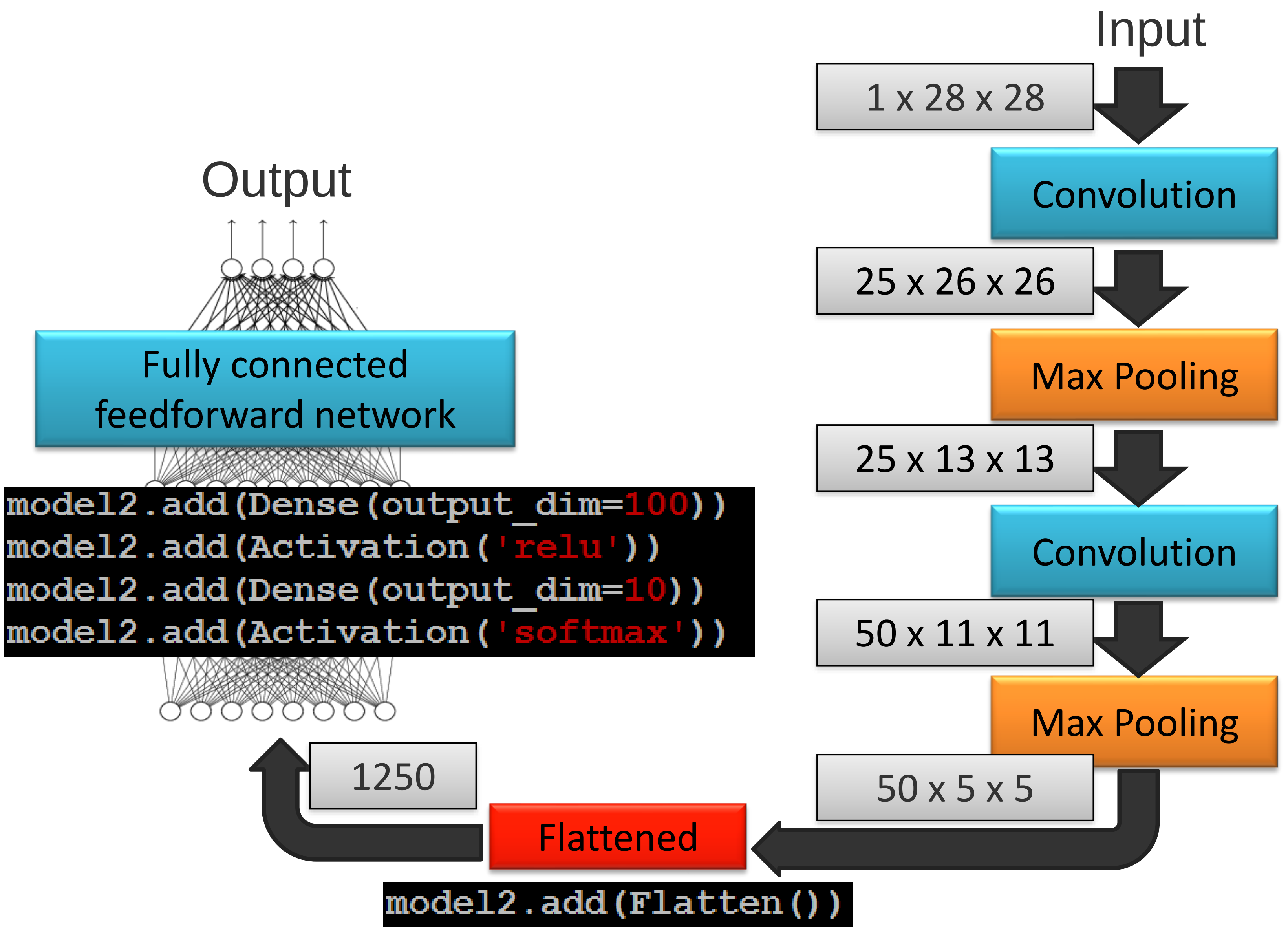
50 x 11 x 11

```
model2.add(MaxPooling2D((2,2)))
```

50 x 5 x 5

Keras

Only modified the ***network structure*** and ***input format (vector -> 3-D array)***



```
model2.add(Dense(output_dim=100))
model2.add(Activation('relu'))
model2.add(Dense(output_dim=10))
model2.add(Activation('softmax'))
```

```
model2.add(Flatten())
```

Layer types

1. **Convolutional Layer:** Extracts features from the input image through convolution operations. Key Parameters: Number of filters, kernel size, strides, padding.

2. **Activation Layer:** Introduces non-linearity to the model, allowing it to learn complex patterns. Common Types: ReLU (Rectified Linear Unit), Sigmoid, Tanh.

3. **Pooling Layer:** Reduces the spatial dimensions (height and width) of the input volume, making the model more computationally efficient and less sensitive to the exact location of features. Types: Max Pooling, Average Pooling.

4. **Batch Normalization Layer:** Normalizes the output of a previous layer by subtracting the batch mean and dividing by the batch standard deviation. Helps in speeding up training and reducing the sensitivity to network initialization.

5. **Dropout Layer:** Randomly sets a fraction of input units to zero at each update during training, helping to prevent overfitting.

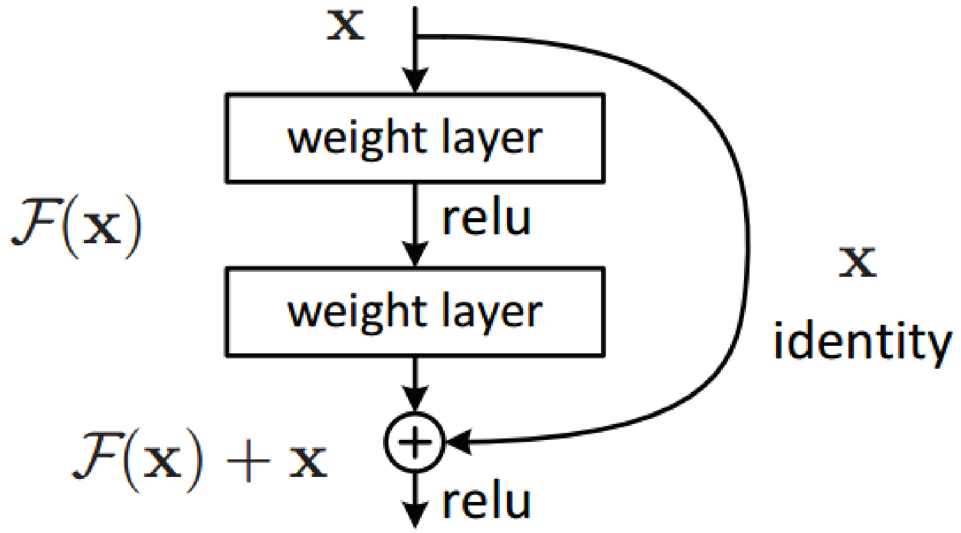
Layer types

6. **Fully Connected (Dense) Layer:** Neurons in a fully connected layer have connections to all activations in the previous layer, as seen in regular neural networks. Used to classify or regress based on features extracted by the convolutional layers.

7. **Flatten Layer:** Flattens the input from a multi-dimensional tensor to a 1D tensor, making it possible to connect convolutional and pooling layers with dense layers.

8. **Residual Connections** (as in ResNet):Allows the output of one layer to bypass one or more layers and be added to the output of a later layer, improving gradient flow through the network.

9. **Transposed Convolutional Layer (Deconvolution):** Used in generative models and some segmentation tasks; it upscales the input feature map.



DCNNs vs. MLP

1. Parameter Efficiency

- Fewer Parameters: CNNs require significantly fewer parameters than FCNs. They use shared weights and convolutional filters, reducing the total number of trainable parameters. This makes CNNs more efficient and less memory-intensive.
- Reduces Overfitting: With fewer parameters, CNNs are less prone to overfitting, especially with image data.

2. Exploitation of Spatial Structure

- Local Connectivity: CNNs exploit the spatial structure of the data by applying convolutional filters that capture local features (like edges, textures) in early layers and more complex features (like patterns or object parts) in deeper layers.
- Preservation of Spatial Relationships: Unlike FCNs that lose spatial relationships by flattening the input, CNNs maintain the spatial hierarchy and relationships between different parts of the input.

3. Translation Invariance

- Robust to Translation: Due to pooling layers and the nature of convolution operations, CNNs are inherently more robust to the translation of input data. This means that if an object shifts in an image, a CNN can still detect it effectively.