Introduction to Convolutional Neural Networks

이영기
서울대학교 컴퓨터공학부

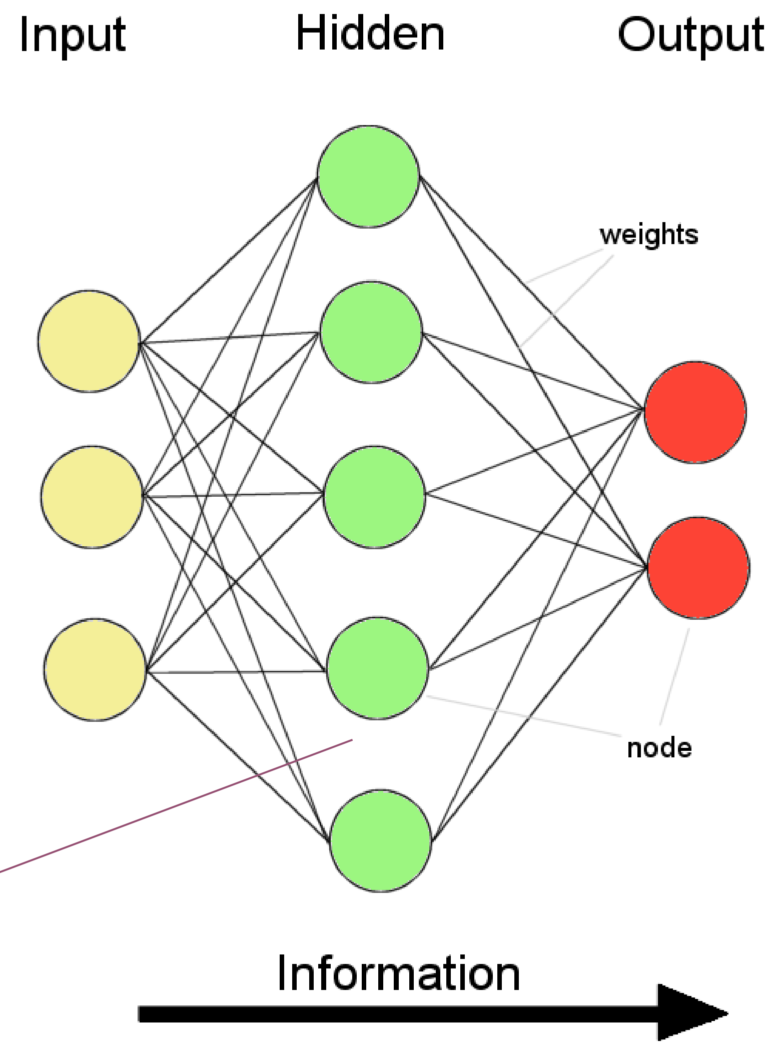
서 울 대 학 교
SEOUL NATIONAL UNIVERSITY

Convolutional Neural Network

- Convolutional Neural Networks are designed to recognize visual patterns directly from pixel images with minimal preprocessing.
- They can recognize patterns with extreme variability (such as handwritten characters).
- CNN is a feed-forward network that can extract topological properties from an image.
- Like almost every other neural networks they are trained with a version of the back-propagation algorithm.

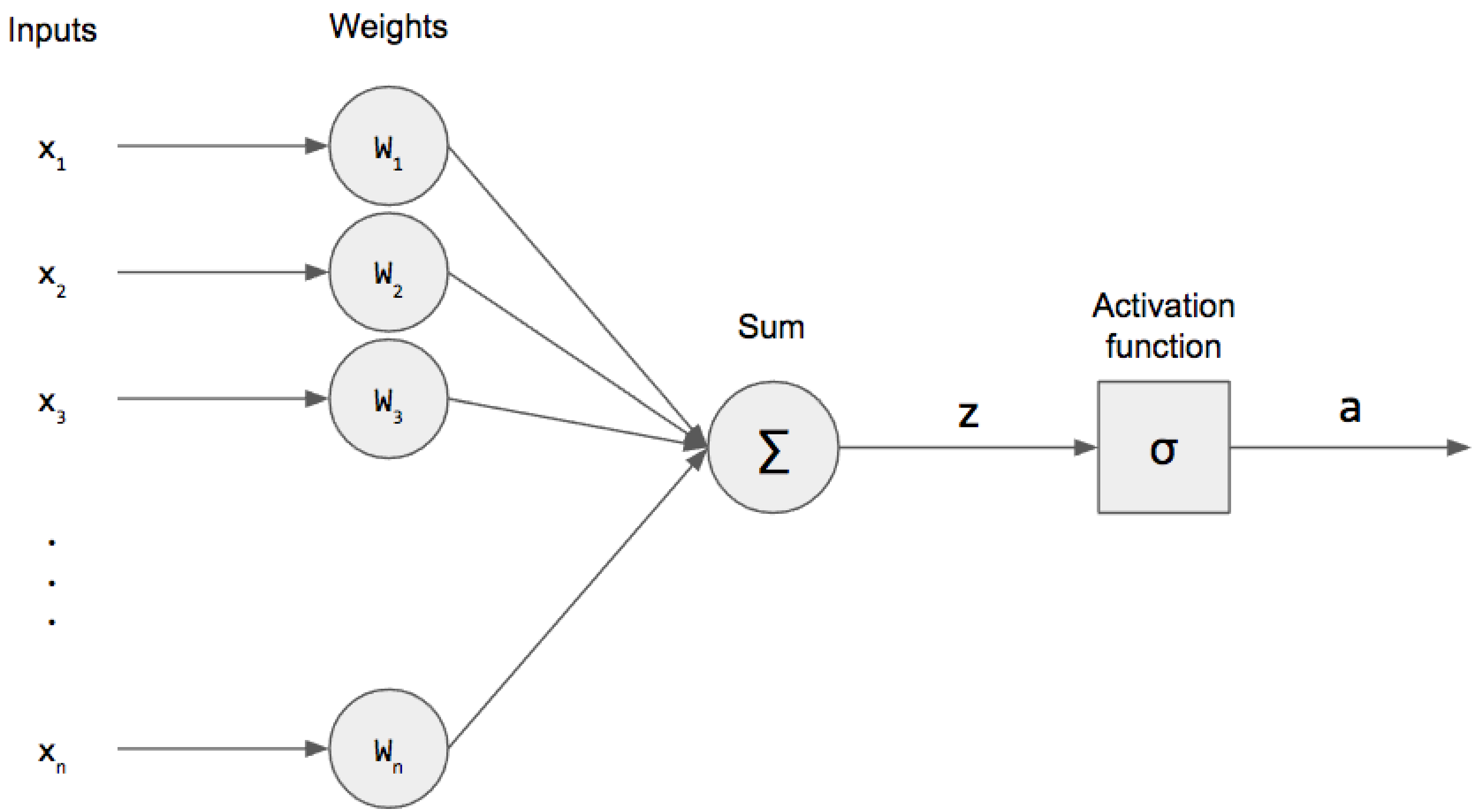
Feed-Forward Networks

- Information flow is unidirectional
  - Data is presented to *Input layer*
  - Passed on to *Hidden Layer*
  - Passed on to *Output layer*
- Information is distributed
- Information processing is parallel



Internal representation (interpretation) of data

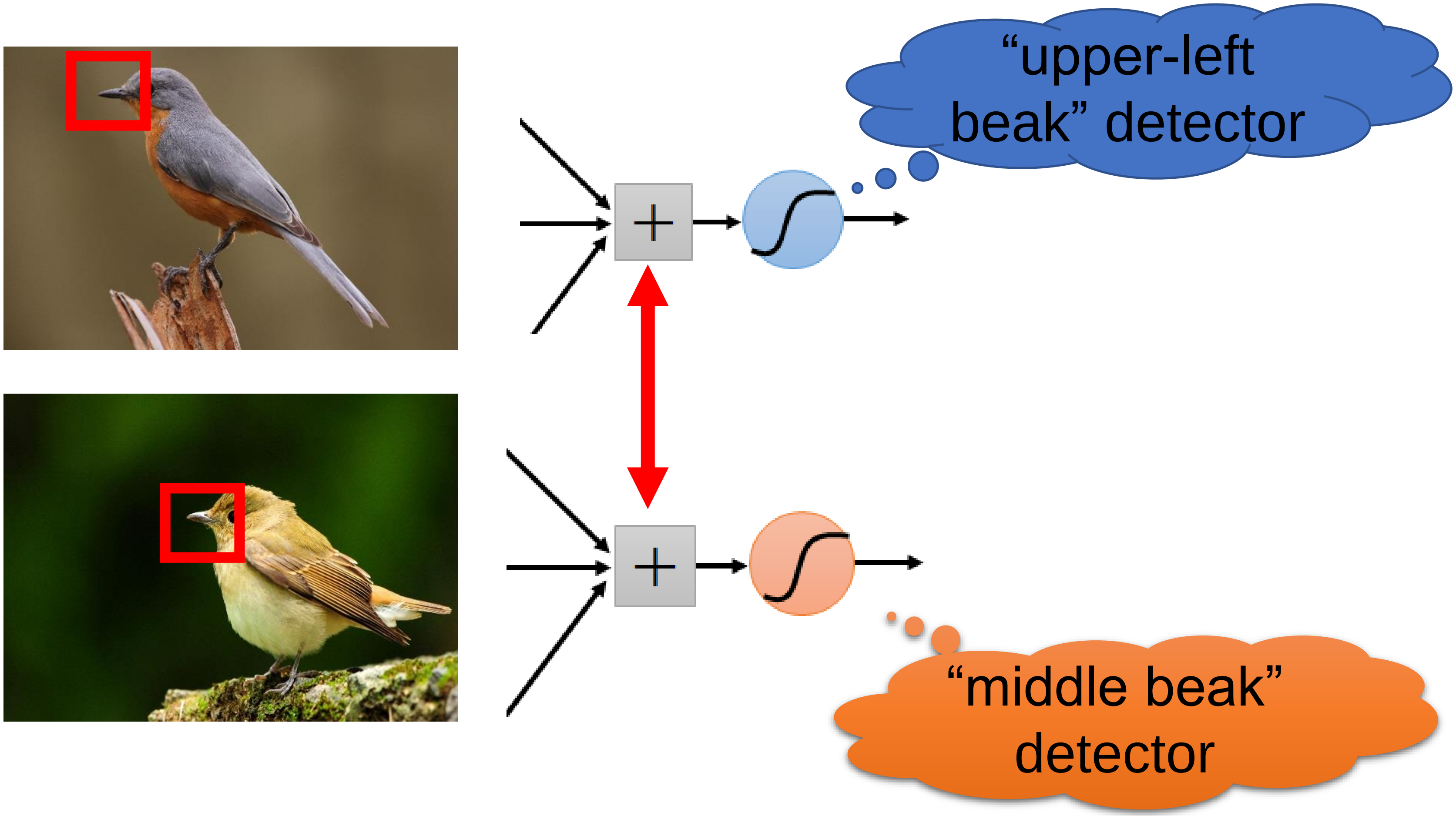
Each Node is a Perceptron!



Identifying a Bird in an Image

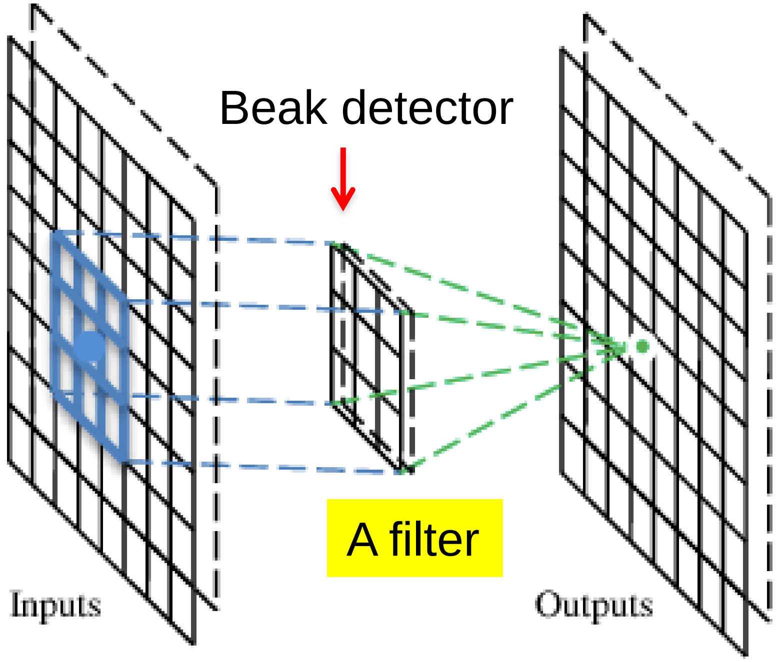
- Let's assume "beak" is unique to birds.
- "beak" exists in a small sub-region of an image.

“Beak” in Different Parts of Images



Convolutional Layer

A Convolutional Neural Network (CNN) is a neural network with “convolutional layers”, which has a number of filters that does convolutional operation.



Convolution

These are the network parameters to be learned.

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

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).

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

Dot product

3 -1

6 x 6 image

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

3 -3

6 x 6 image

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

3	-1	-3	-1
-3	1	0	-3
-3	-3	0	1
3	-2	-2	-1

Convolution

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

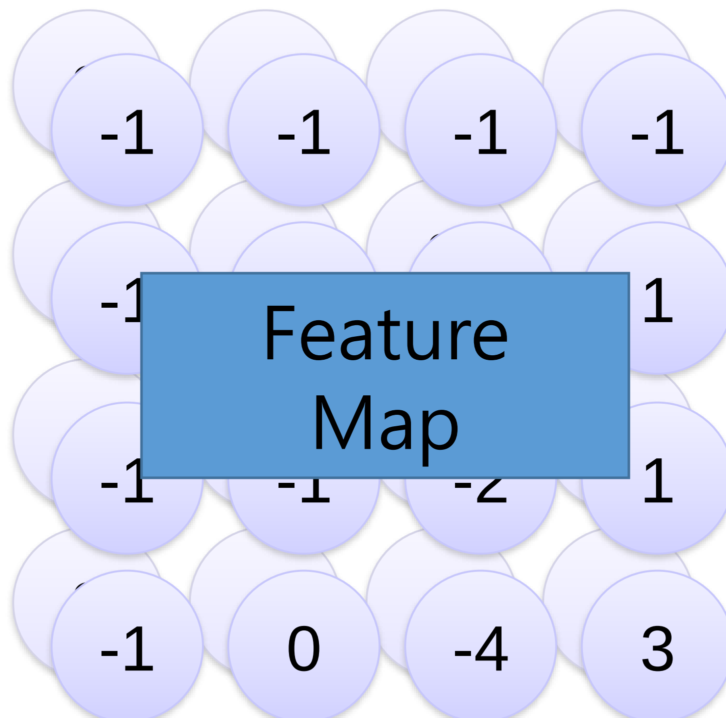
Filter 2

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

Color Image: RGB 3 Channels

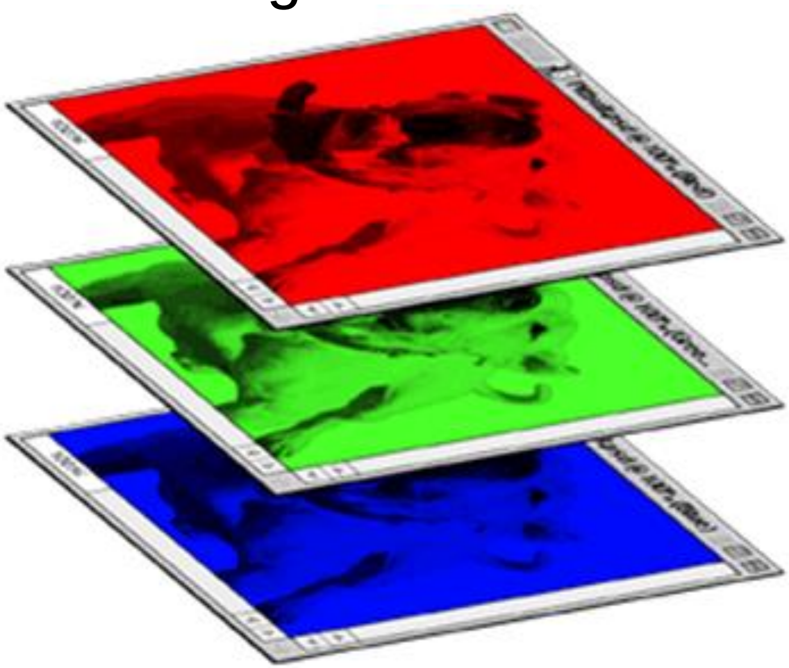
1	-1	-1
-1	1	-1
-1	-1	1

Filter 1

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

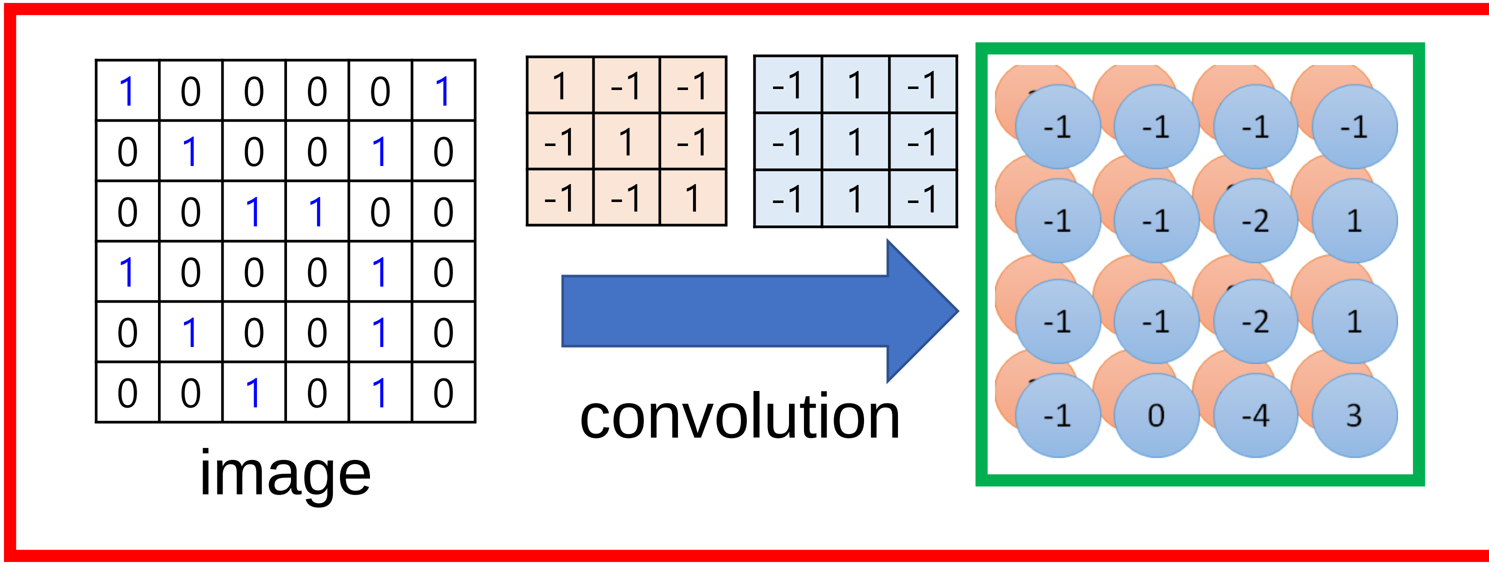
Filter 2

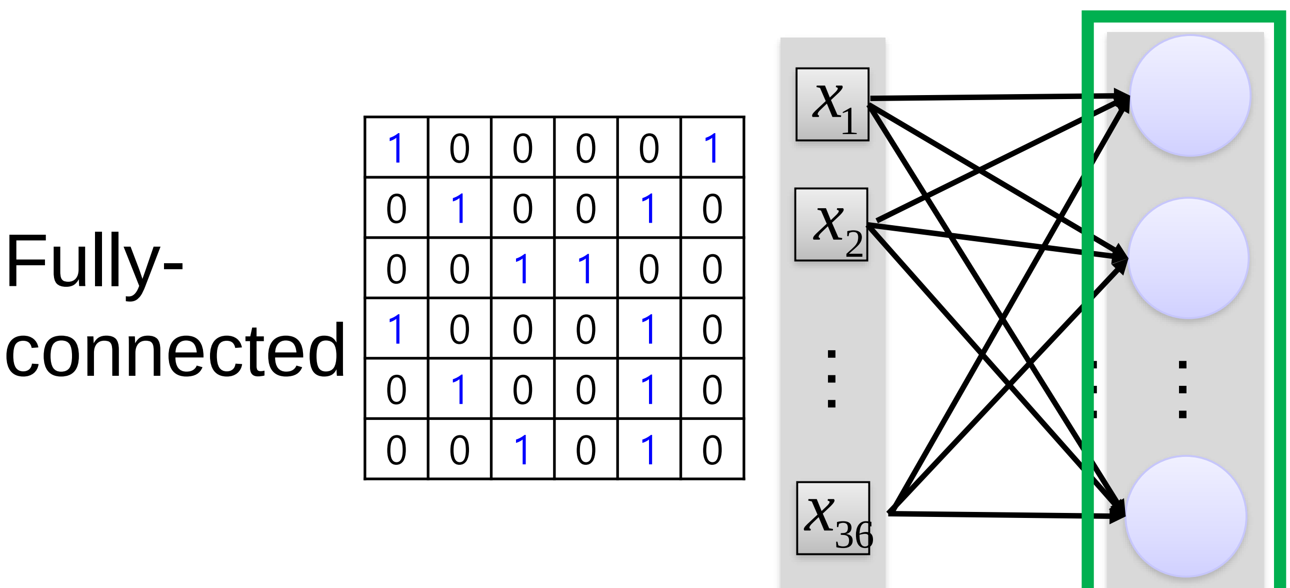
Color image

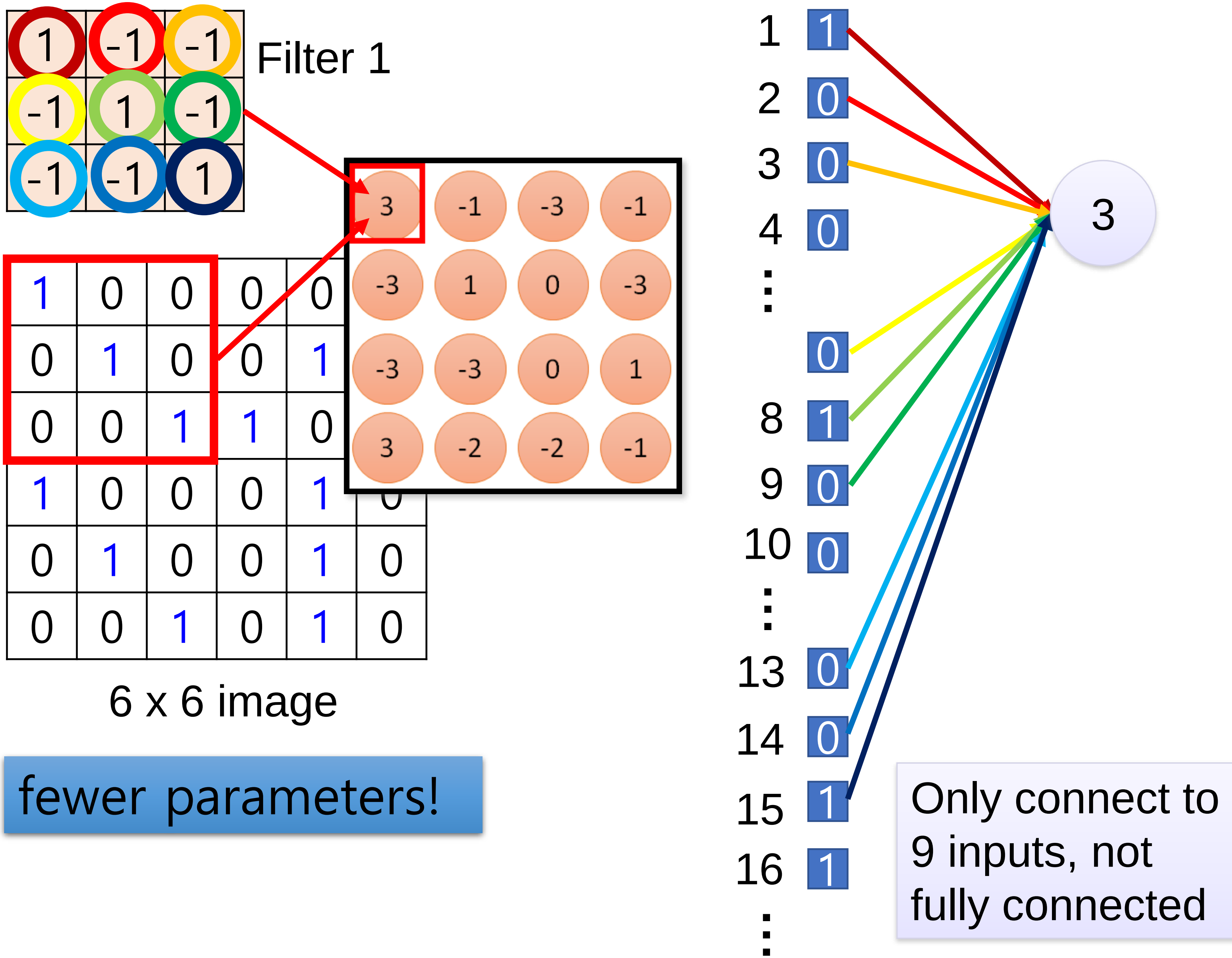


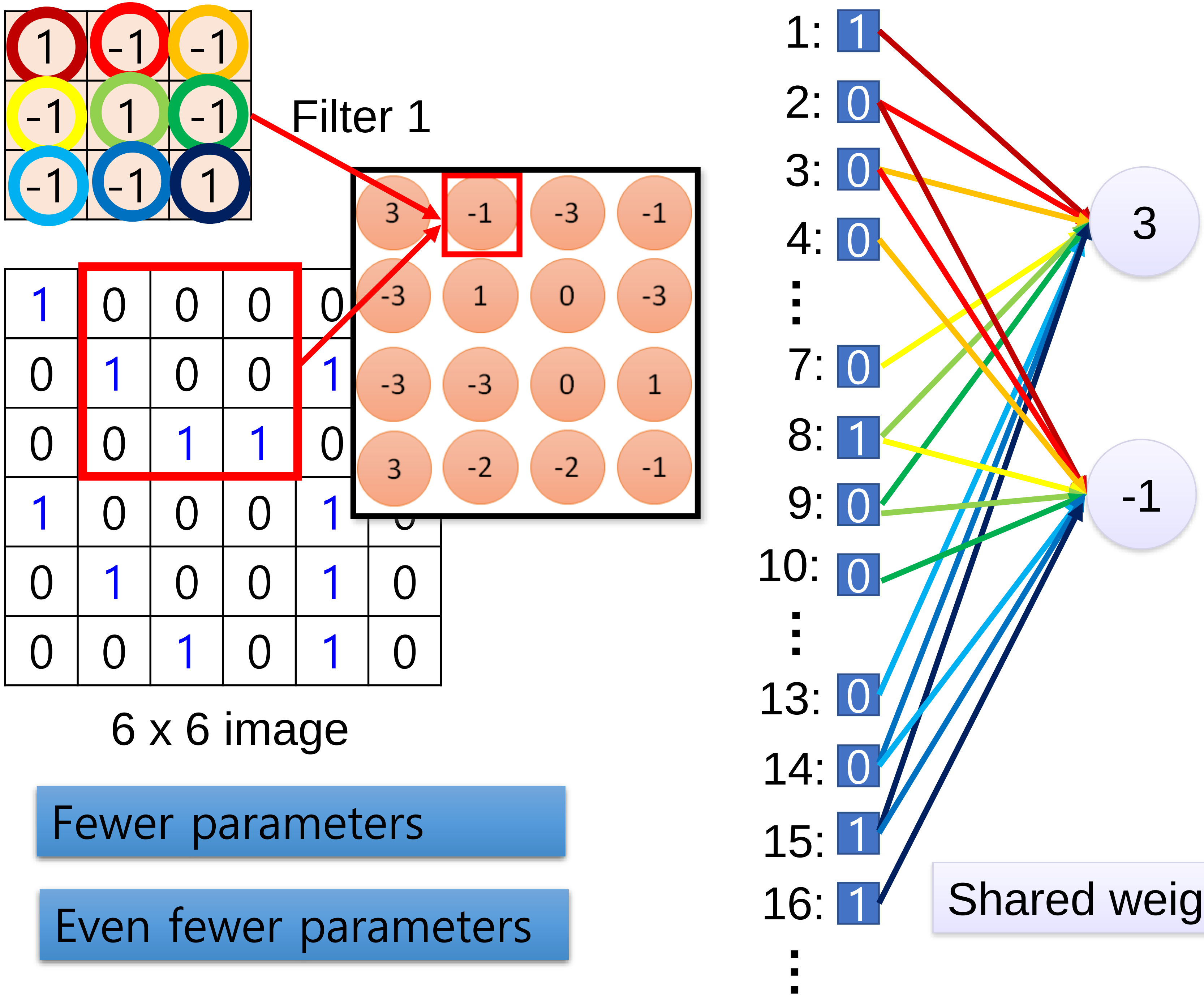
1	0	0	0	0	1
0	1	0	0	1	0
0	0	1	1	0	0
1	0	0	0	1	0
0	1	0	0	1	0
0	0	1	0	1	0

How to Form a Feed Forward Network






Filter 1
6 x 6 image
fewer parameters!
Only connect to 9 inputs, not fully connected

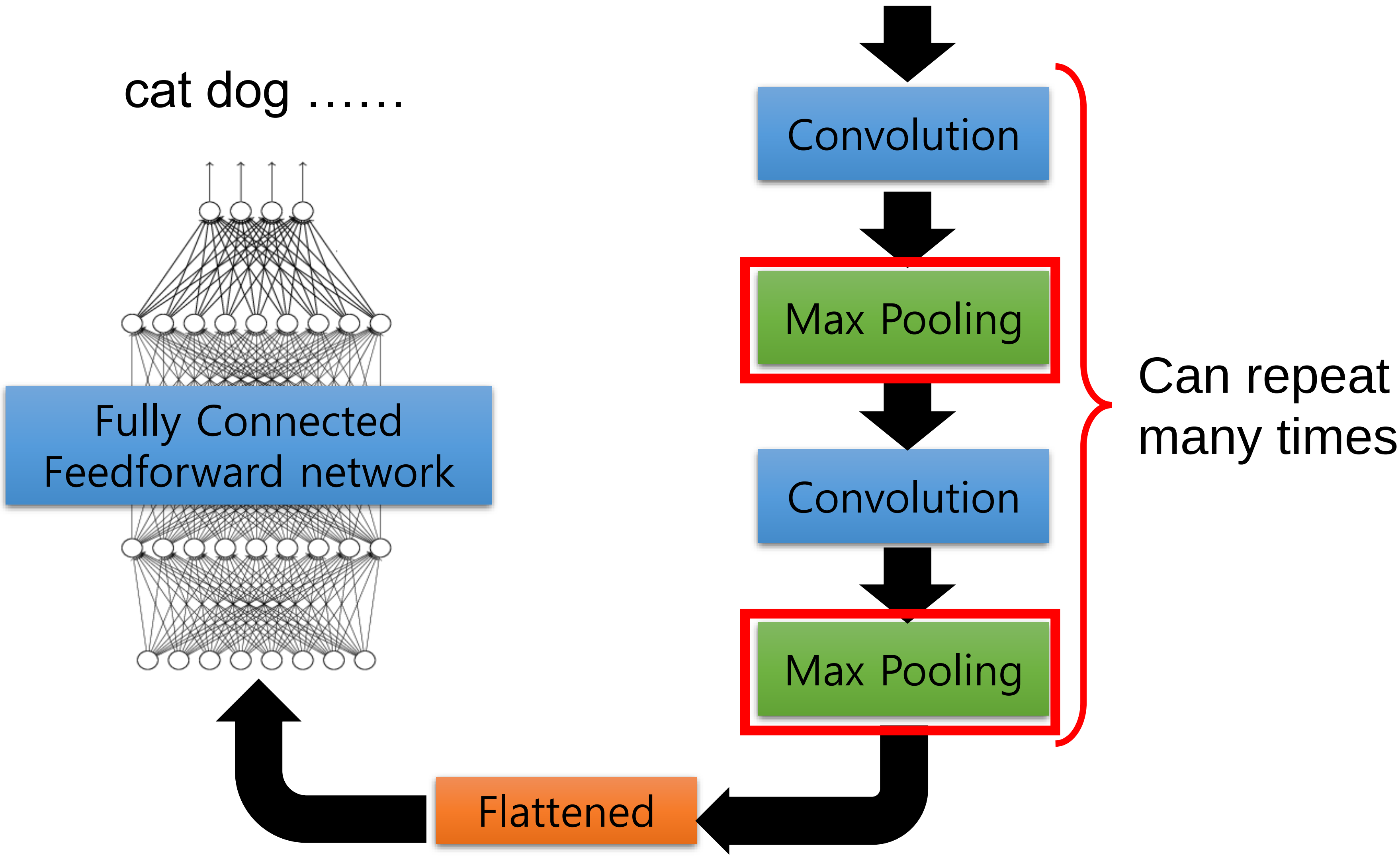


Fewer parameters

Even fewer parameters

Shared weights

The Whole CNN



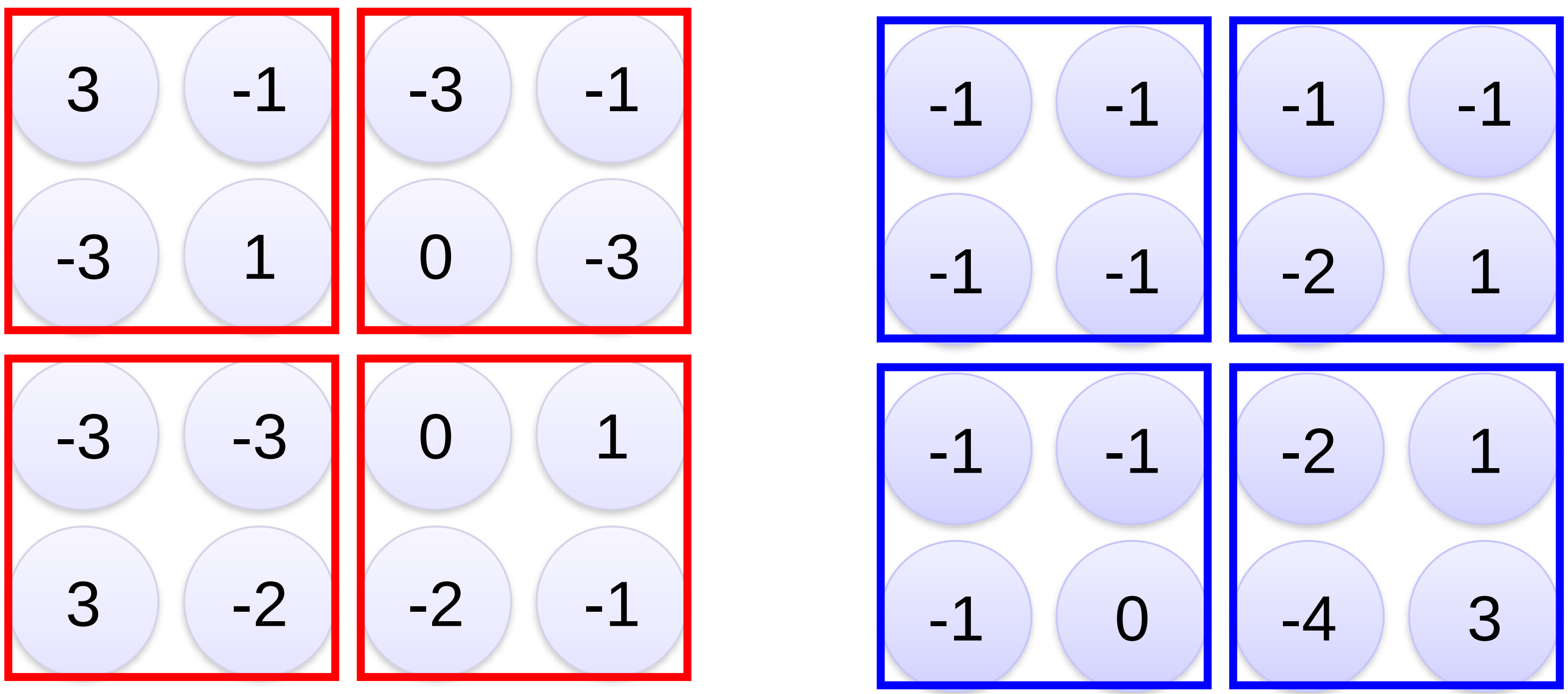
Max Pooling

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

Filter 1

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

Filter 2



Why Pooling

- Subsampling pixels will not change the object

bird

Subsampling

bird

We can subsample the pixels to make image smaller

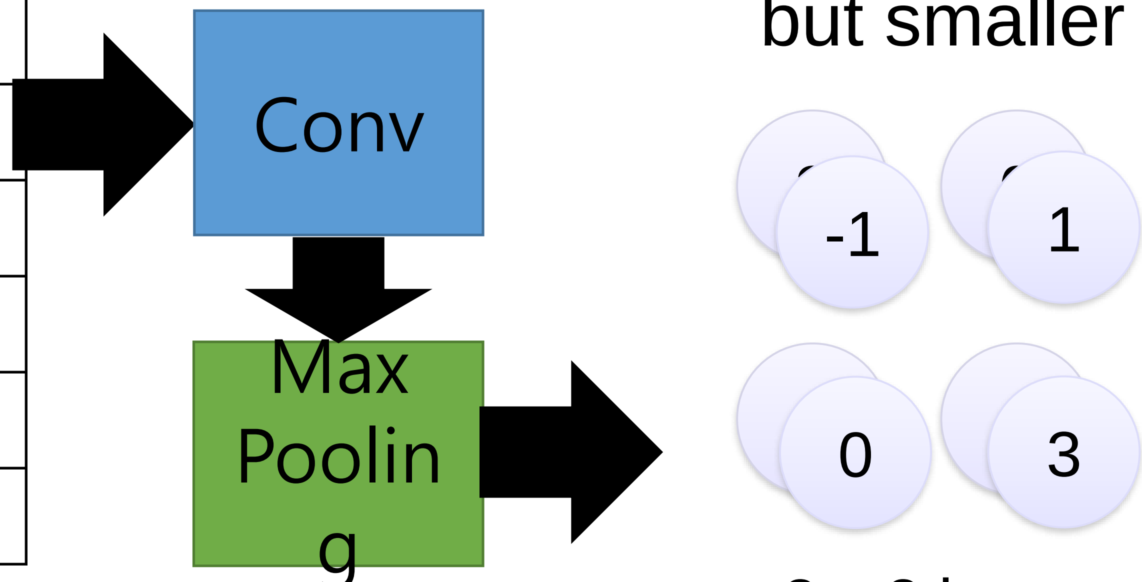
fewer parameters to characterize the image

Max Pooling

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

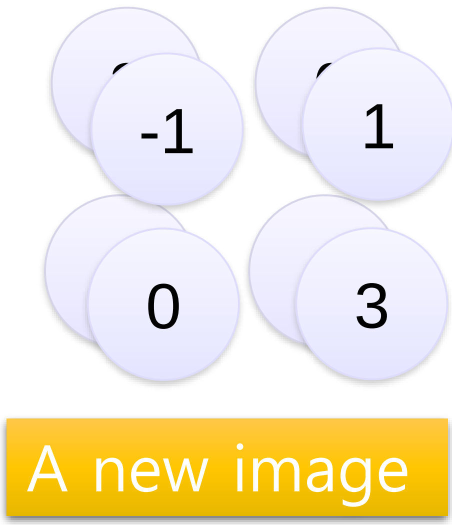
New image but smaller

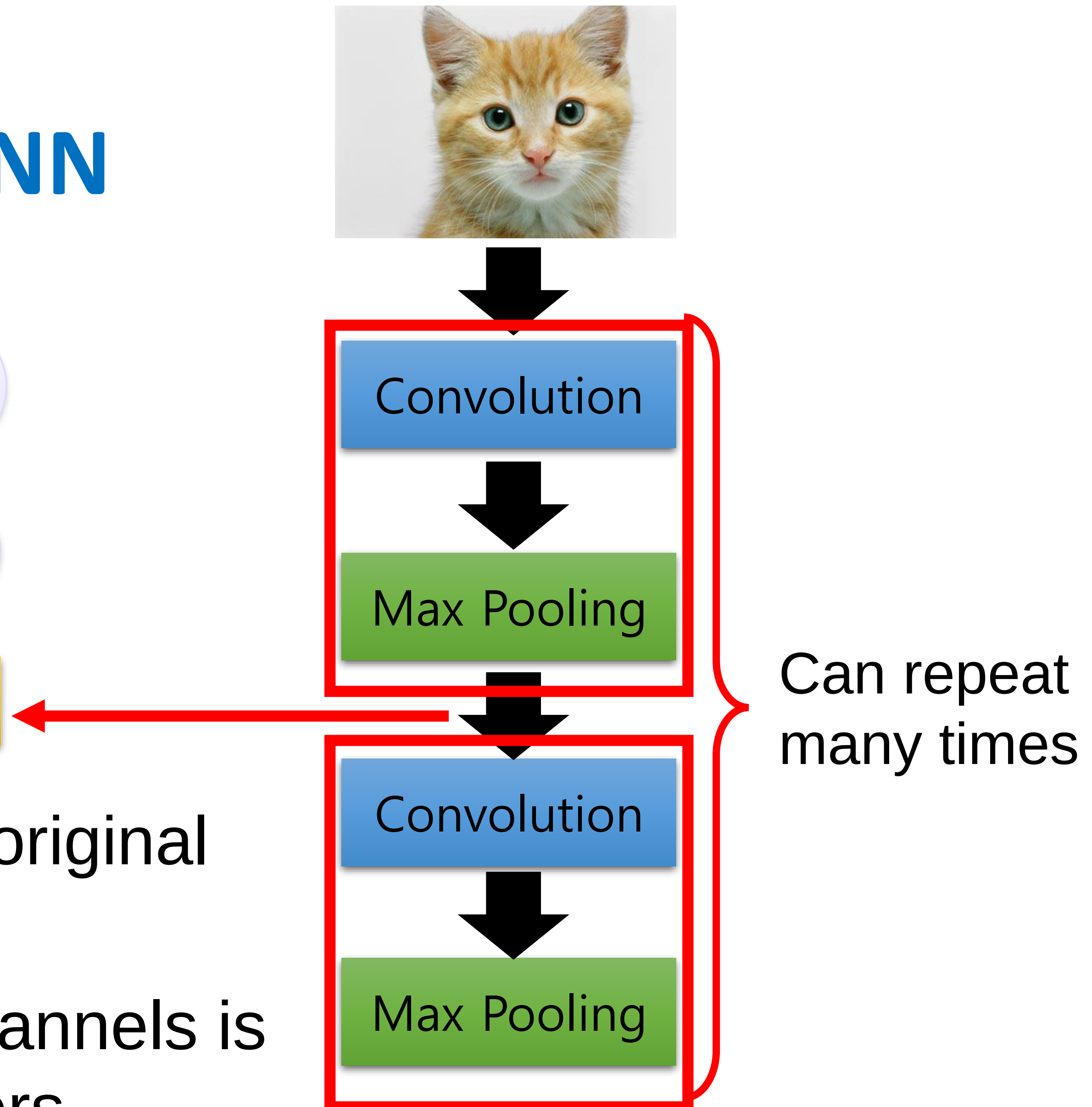


2 x 2 image

Each filter is a channel

The Whole CNN

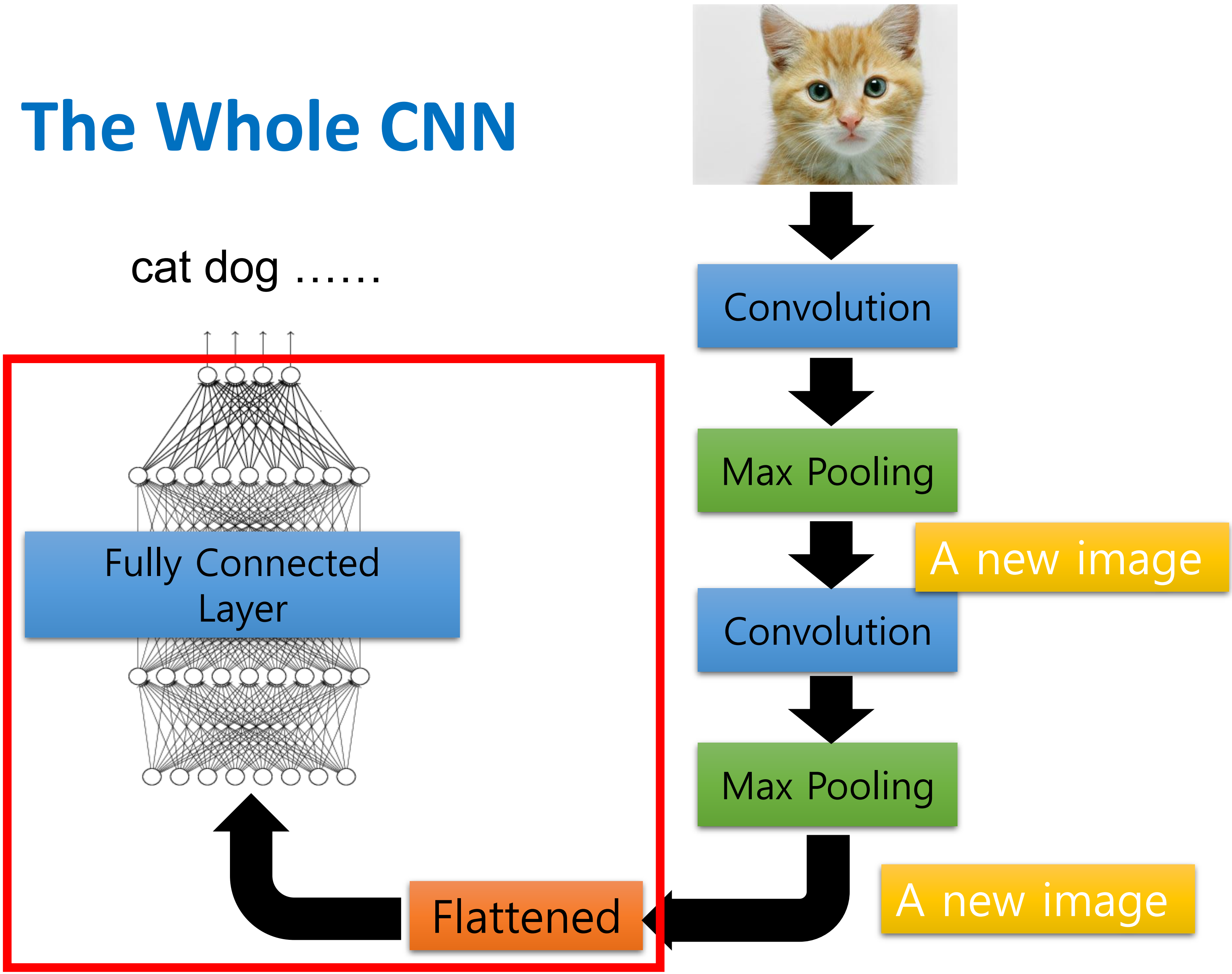




Smaller than the original image

The number of channels is the number of filters

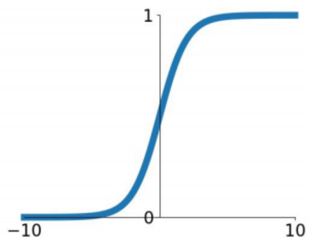
The Whole CNN



Activation Layer

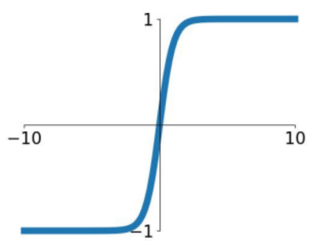
Sigmoid

$\sigma(x) = \frac{1}{1+e^{-x}}$



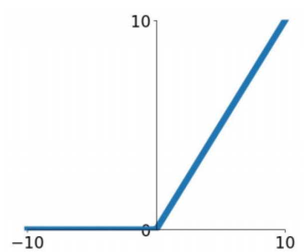
tanh

$\tanh(x)$



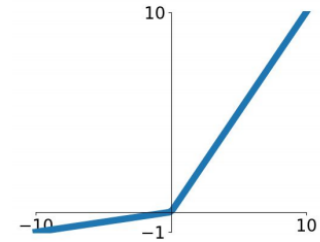
ReLU

$\max(0, x)$



Leaky ReLU

$\max(0.1x, x)$

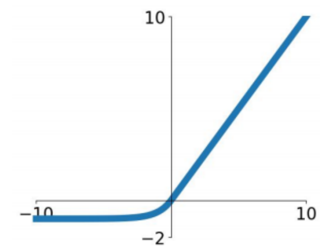


Maxout

$\max(w_1^T x + b_1, w_2^T x + b_2)$

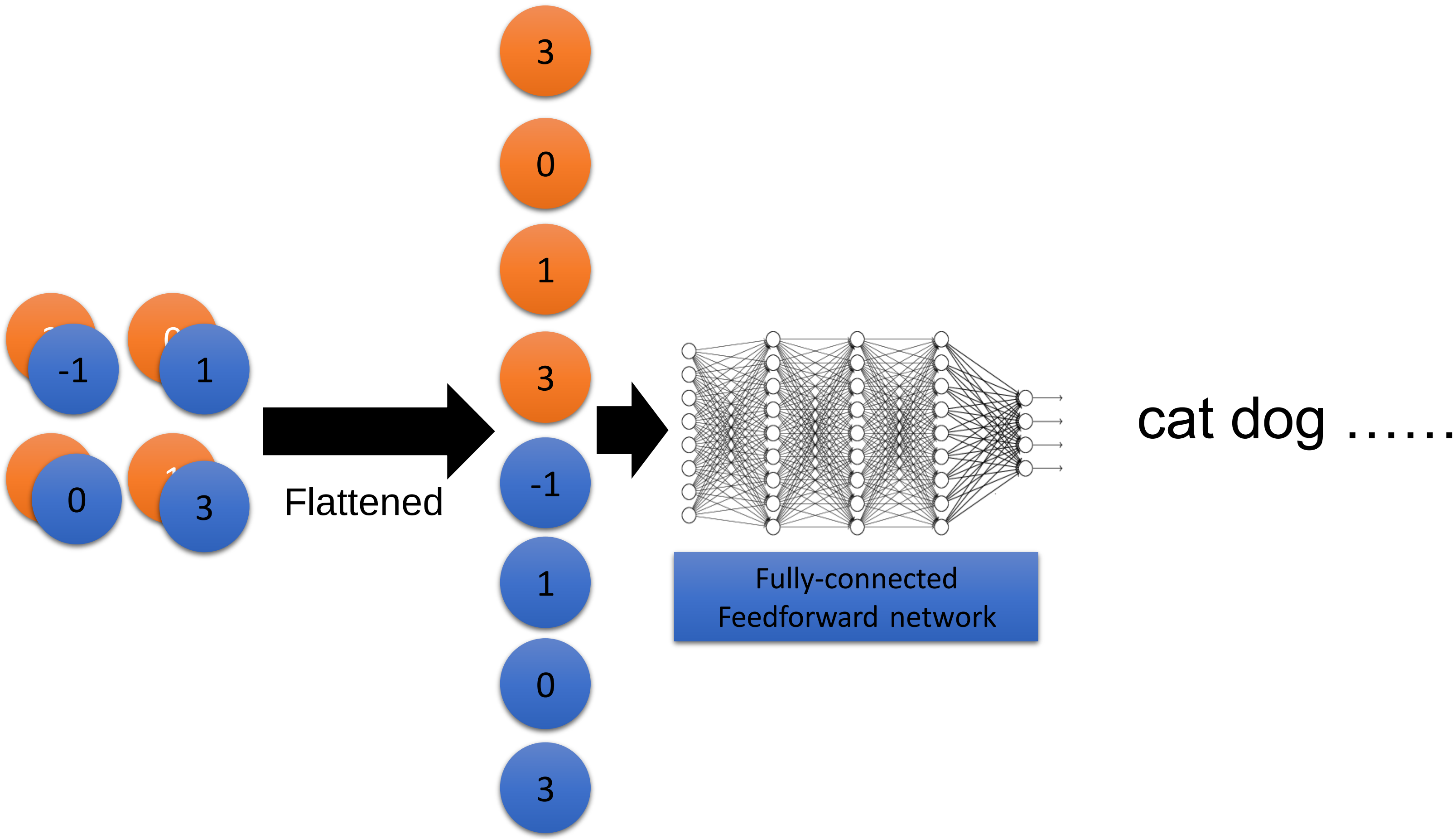
ELU

$$\begin{cases} x & x \geq 0 \\ \alpha(e^x - 1) & x < 0 \end{cases}$$



Fully Connected Layer

Conceptually, this can be understood as the voting process to see which input values contribute more to the output.



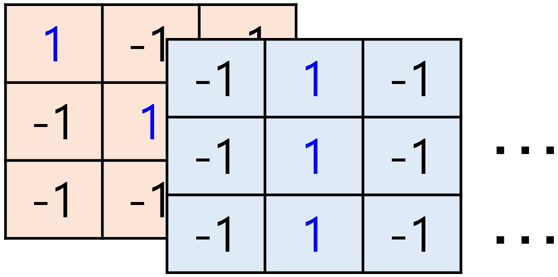
Tools and APIs

- Tensorflow (https://www.tensorflow.org/)
    - ✓Tensorflow light for Mobile and IoT
- PyTorch (https://pytorch.org)
- Caffe2 (https://caffe2.ai)
- Keras (https://keras.io/)

CNN in Keras

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

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



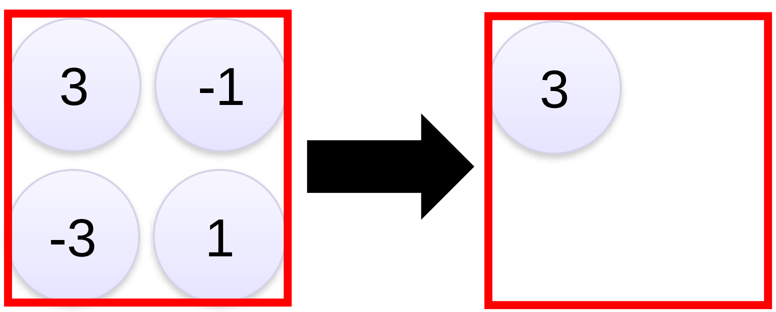
There are 25 3x3 filters.

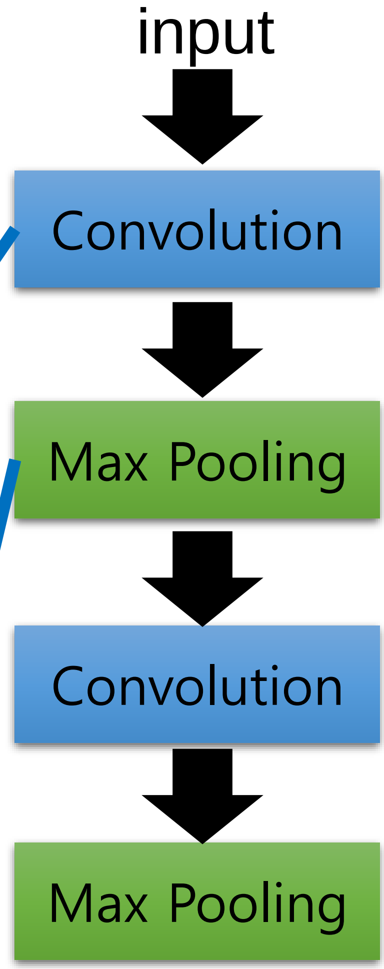
Input_shape = (28 , 28 , 1)

28 x 28 pixels

1: black/white, 3: RGB

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





CNN in Keras

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

Input

1 x 28 x 28

Convolution

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

How many parameters for each filter? **9**

25 x 26 x 26

Max Pooling

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

25 x 13 x 13

Convolution

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

How many parameters for each filter? **225= 25x9**

50 x 11 x 11

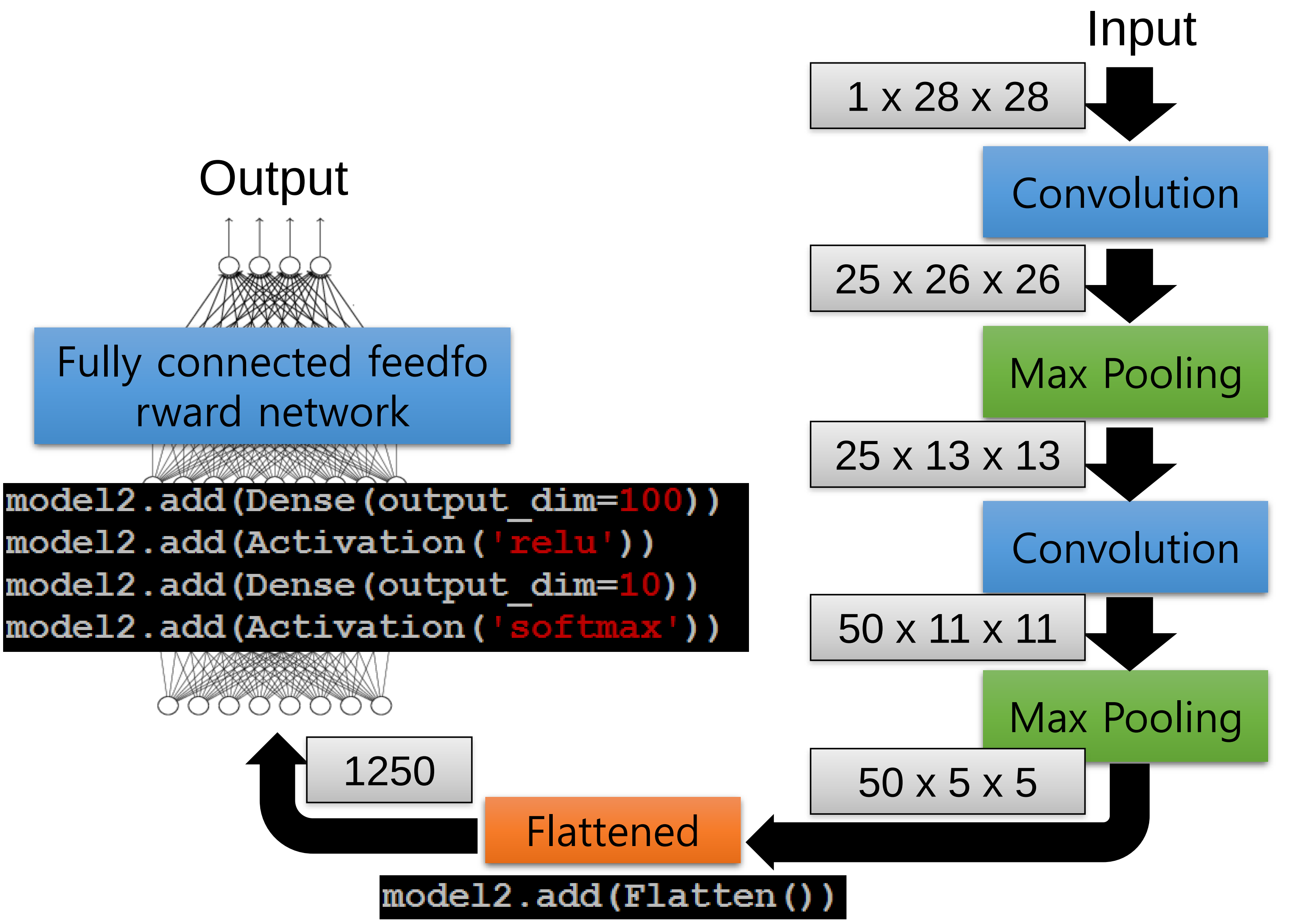
Max Pooling

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

50 x 5 x 5

CNN in Keras

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



AlphaGo

19 x 19 matrix

Black: 1

white: -1

none: 0

Neural Network

Next move

(19 x 19 positions)

Fully-connected feedforward network can be used

But CNN performs much better

AlphaGo's Policy Network

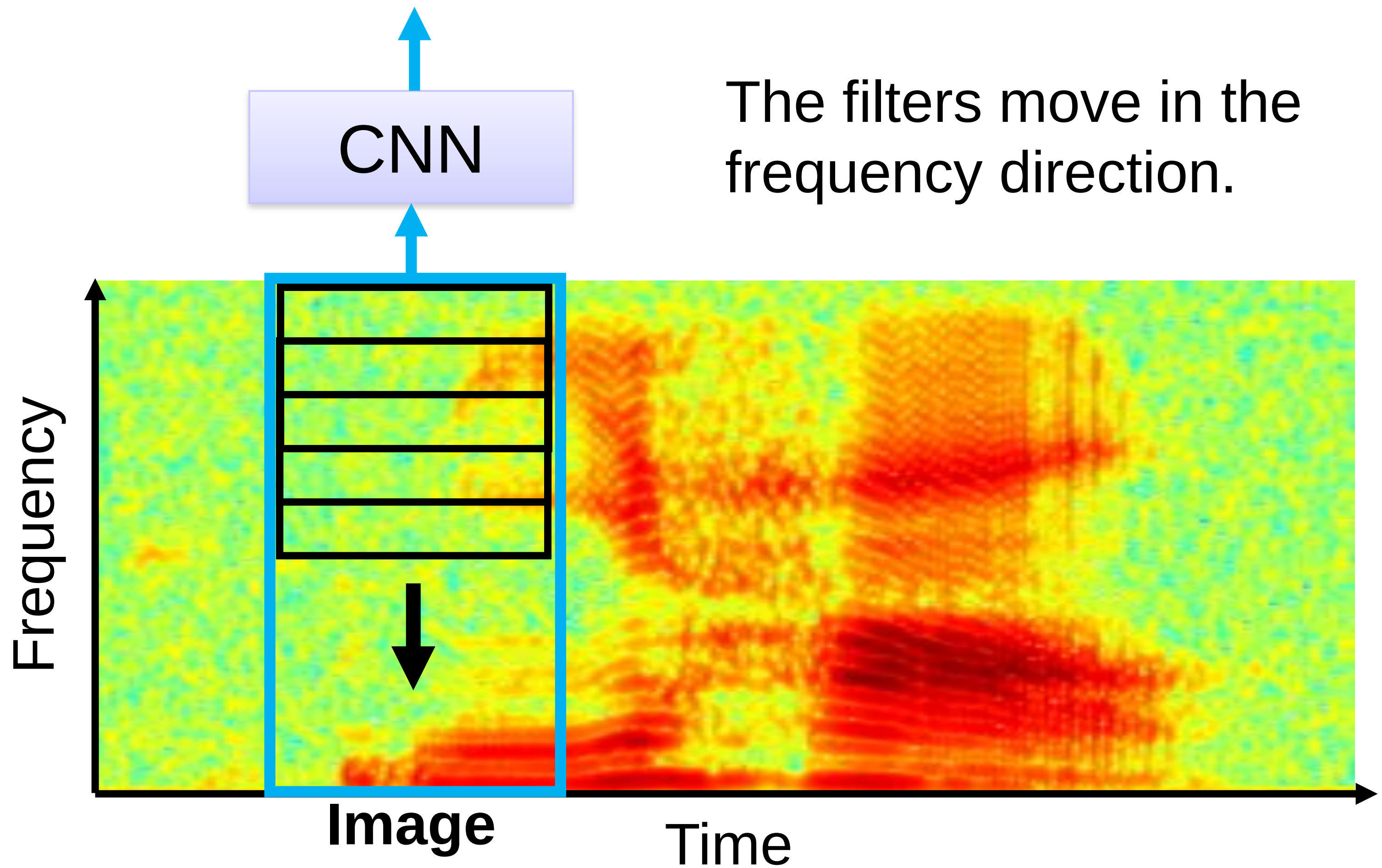
The following is quotation from their Nature article:

Note: AlphaGo does not use Max Pooling.

Neural network architecture. The input to the policy network is a $19 \times 19 \times 48$ image stack consisting of 48 feature planes. The first hidden layer zero pads the input into a 23×23 image, then convolves k filters of kernel size 5×5 with stride 1 with the input image and applies a rectifier nonlinearity. Each of the subsequent hidden layers 2 to 12 zero pads the respective previous hidden layer into a 21×21 image, then convolves k filters of kernel size 3×3 with stride 1, again followed by a rectifier nonlinearity. The final layer convolves 1 filter of kernel size 1×1 with stride 1, with a different bias for each position, and applies a softmax function. The match version of AlphaGo used $k = 192$ filters; Fig. 2b and Extended Data Table 3 additionally show the results of training with $k = 128$, 256 and 384 filters.

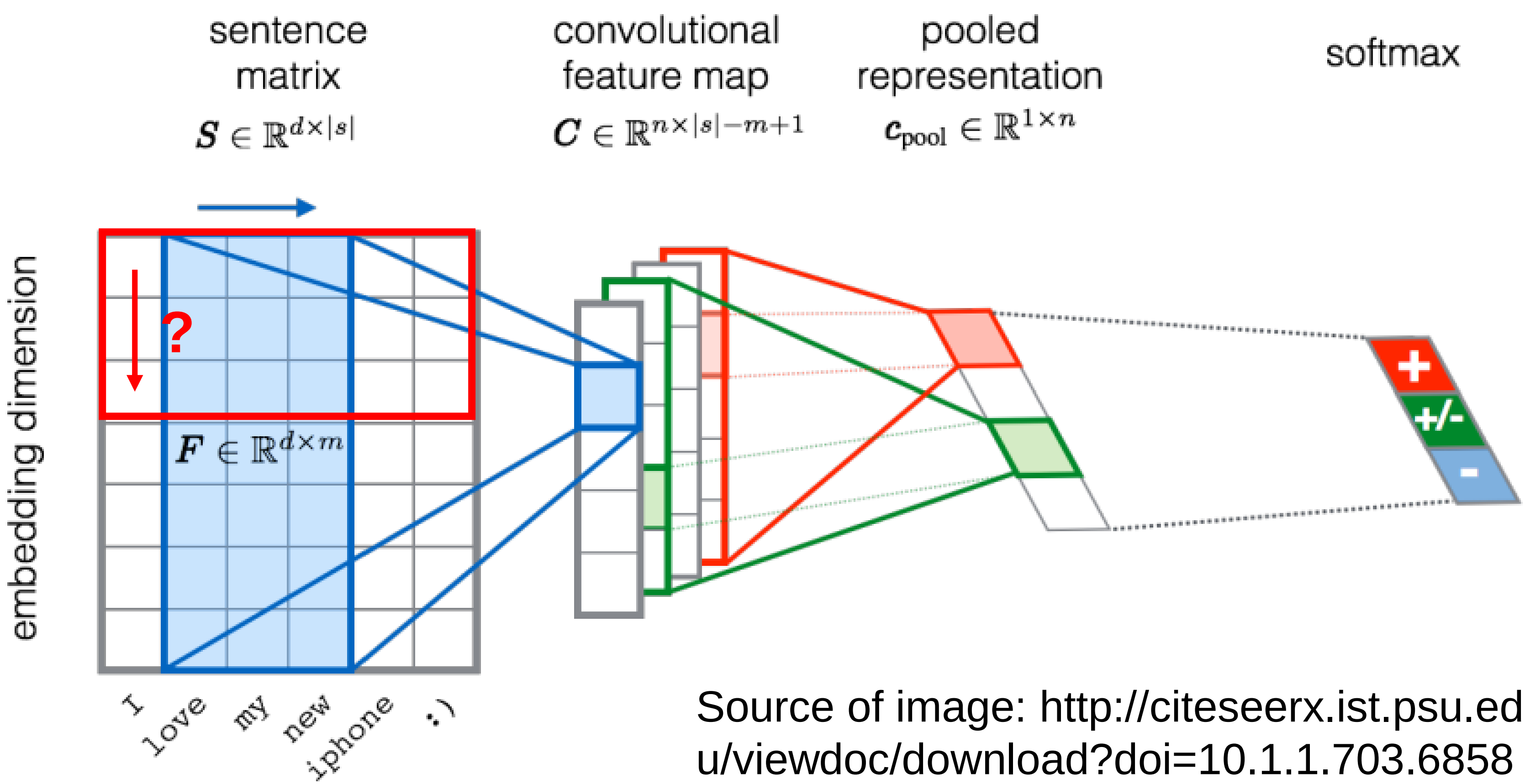
CNN in Speech Recognition

The filters move in the frequency direction.



Spectrogram

CNN in Text Classification



Source of image: http://citeseerx.ist.psu.edu/viewdoc/download?doi=10.1.1.703.6858&rep=rep1&type=pdf

CNN Optimization Techniques (Model Training)

이영기
서울대학교 컴퓨터공학부

서울대학교 SEOUL NATIONAL UNIVERSITY

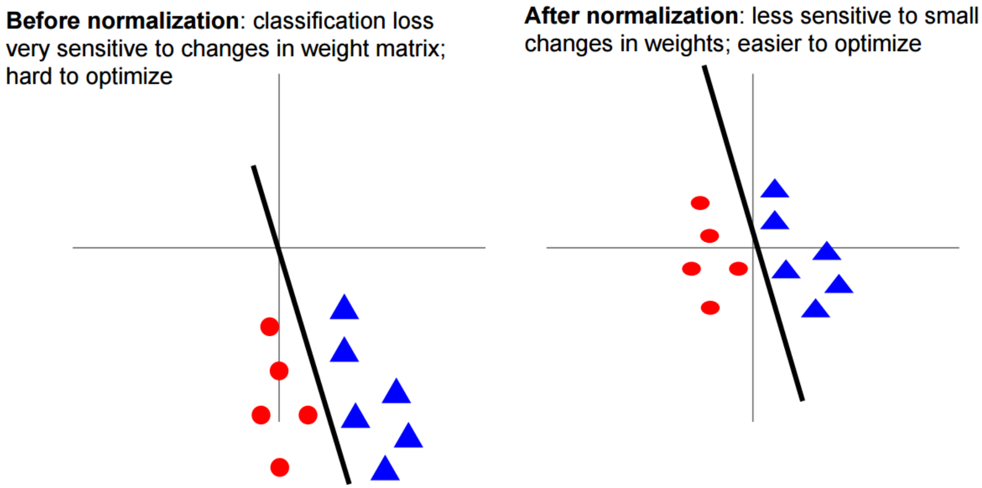
References

- Stanford CS231n Lecture Notes
  - ✓ Lecture 7 “Training Neural Networks, Part 1”
  - ✓ Lecture 8 “Training Neural Networks, Part 2”
- Ian Goodfellow, Yoshua Bengio, and Aaron Courville, “Deep Learning” MIT Press 2016.
- And many other papers and blogs in the Internet…

d)

Data Preprocessing

- **<u>Biased input degrades training performance</u>**
  - ✓ e.g., negative-biased input to ReLU activation "kills" the gradient, whereas positive-biased input makes ReLU meaningless
- Normalizing the data enhances training stability



- **<u>Common preprocessing methods (for images)</u>**
  - ✓ Subtract per-channel mean (e.g., $x - [123.68, 116.78, 103.94]$)
  - ✓ Normalize to [0,1] range (e.g., $(x - 127.5)/128$)

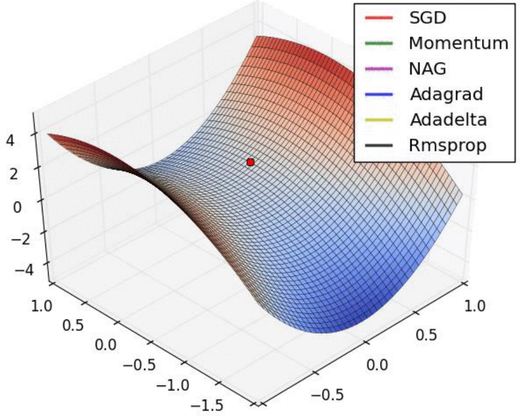
Weight Initialization

- **<u>"Where do we start climbing down the hill?"</u>**
- Why is it important?
  - ✓ **For deeper networks, all activations tends to collapse to zero**
  - ✓ When activations are all zero, model is not trained
  - ✓ We want "well-distributed" activations!
- **<u>Common initialization methods</u>**
  - ✓ Xavier initialization [1] (for **sigmoid** or **tanh** activation)
    - ➢ *tf.contrib.layers.xavier_initializer(), torch.nn.init.xavier_normal()*
  - ✓ He initialization [2] (for **ReLU** activation)
    - ➢ *tf.initializers.he_normal(), torch.nn.init.kaiming_normal()*

[1] X. Glorot and Y. Bengio, "Understanding the difficulty of training deep feedforward neural networks," AISTAT 2010.
[2] He et al, "Delving Deep into Rectifiers: Surpassing Human-Level Performance on ImageNet Classification", ICCV 2015

Optimizers & Learning Rate Decay

- **"How can we go down the hill efficiently?"**
  - ✓ Stepping policy (i.e., optimizer)
  - ✓ Step size (i.e., learning rate)
- Various methods have been proposed (refer to Stanford CS231n Lecture 8 for details)
  - ✓ Optimizers
    - ➢ SGD, Adagrad, RMSprop, Adam, … etc.
  - ✓ Learning rate decay
    - ➢ Step, cosine, linear, … etc.



- **Common practice**
  - ✓ Use Adam/RMSprop
  - ✓ Start with learning rate between 1e-3 ~ 1e-4
  - ✓ Use step decay (period heavily depends on task & loss function)

Batch Normalization

- We want "well-distributed" activations …
  → **Why don't we intentionally make them so?**
- **<u>For a batch of activations at a layer, make each dimension close to zero-mean, unit variance</u>**

Input: Values of x over a mini-batch: $\mathcal{B} = \{x_{1...m}\}$;
Parameters to be learned: γ, β
Output: $\{y_i = \text{BN}_{\gamma,\beta}(x_i)\}$

$$\mu_\mathcal{B} \leftarrow \frac{1}{m}\sum_{i=1}^{m} x_i \qquad \text{// mini-batch mean}$$

$$\sigma_\mathcal{B}^2 \leftarrow \frac{1}{m}\sum_{i=1}^{m}(x_i - \mu_\mathcal{B})^2 \qquad \text{// mini-batch variance}$$

$$\widehat{x}_i \leftarrow \frac{x_i - \mu_\mathcal{B}}{\sqrt{\sigma_\mathcal{B}^2 + \epsilon}} \qquad \text{// normalize}$$

$$y_i \leftarrow \gamma\widehat{x}_i + \beta \equiv \text{BN}_{\gamma,\beta}(x_i) \qquad \text{// scale and shift}$$

[3] S. Ioffe and C. Szegedy, "Batch Normalization: Accelerating Deep Network Training by Reducing Internal Covariate Shift," arXiv 2015.

Batch Normalization

- We want "well-distributed" activations …
  → **Why don't we intentionally make them so?**
- **For a batch of activations at a layer, make each dimension close to zero-mean, unit variance**

Input: Values of x over a mini-batch: $\mathcal{B} = \{x_{1\ldots m}\}$;
Parameters to be learned: γ, β
Output: $\{y_i = \mathrm{BN}_{\gamma,\beta}(x_i)\}$

$$\boxed{\begin{aligned} \mu_\mathcal{B} &\leftarrow \frac{1}{m}\sum_{i=1}^{m} x_i && \text{// mini-batch mean} \\ \sigma_\mathcal{B}^2 &\leftarrow \frac{1}{m}\sum_{i=1}^{m} (x_i - \mu_\mathcal{B})^2 && \text{// mini-batch variance} \end{aligned}}$$

$$\widehat{x}_i \leftarrow \frac{x_i - \mu_\mathcal{B}}{\sqrt{\sigma_\mathcal{B}^2 + \epsilon}} \qquad \text{// normalize}$$

$$y_i \leftarrow \gamma \widehat{x}_i + \beta \equiv \mathrm{BN}_{\gamma,\beta}(x_i) \qquad \text{// scale and shift}$$

At test time, use (running) average of values seen during training

c.f.) be careful for this in implementation!
tf.layers.batch_normalization(training=True)

[3] S. Ioffe and C. Szegedy, "Batch Normalization: Accelerating Deep Network Training by Reducing Internal Covariate Shift," arXiv 2015.

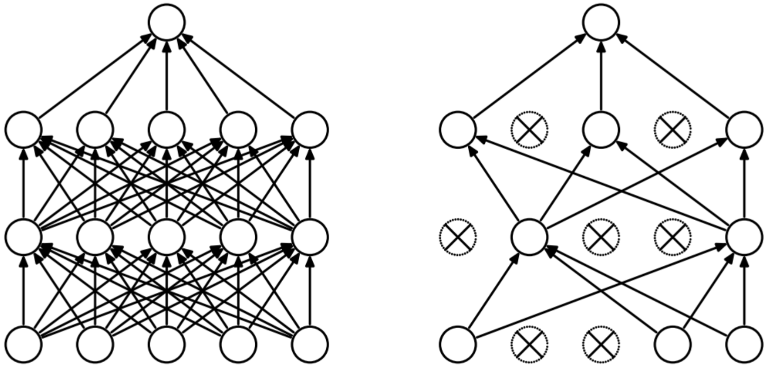
Batch Normalization

- We want "well-distributed" activations …
  → **Why don't we intentionally make them so?**
- **For a batch of activations at a layer, make each dimension close to zero-mean, unit variance**
- **Benefits**
  - ✓ Normalizing activations prevent small changes to the parameters from amplifying into larger and sub-optimal changes
    → **Enables higher learning rates (and thereby training speed)**
  - ✓ Each training sample is seen in conjunction with others in the same mini-batch
    → **Prevents model overfitting**

[3] S. Ioffe and C. Szegedy, "Batch Normalization: Accelerating Deep Network Training by Reducing Internal Covariate Shift," arXiv 2015.

Regularization

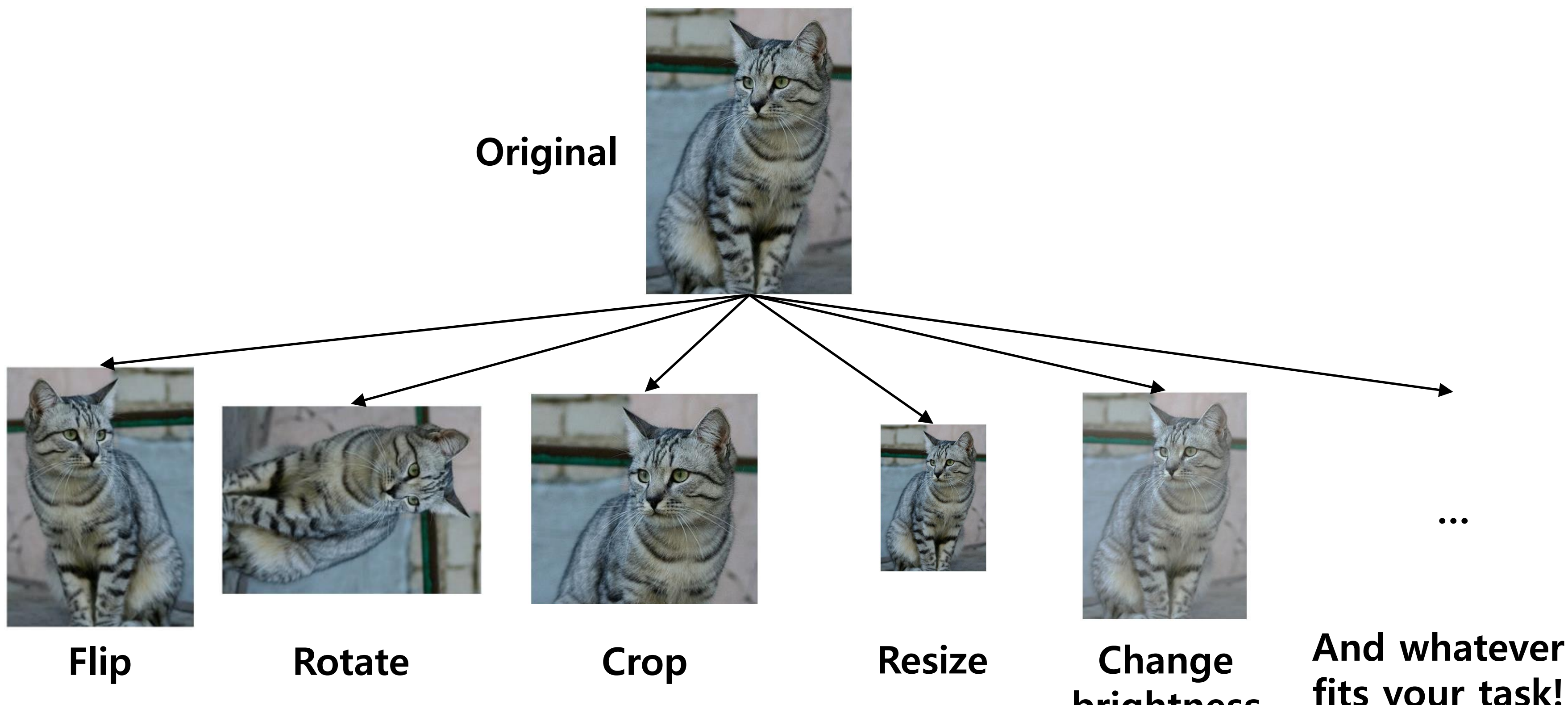
- **<u>Goal: prevent model from overfitting to training data</u>**
- How?
  - ✓ Add additional loss term $\tilde{L} = L + \lambda \boxed{R(W)}$
    - ➢ L2 regularization (also known as weight decay) $R(W) = \sum\sum W_{k,l}^2$
    - ➢ L1 regularization $R(W) = \sum\sum |W_{k,l}|$
  - ✓ Dropout [4]
    - ➢ Randomly set some neurons to zero at training time (e.g., with prob. 0.5)
    - ➢ Effect of training a large number of models (ensemble)



[4] Srivastava et al, "Dropout: A simple way to prevent neural networks from overfitting", JMLR 2014

Data Augmentation

- **<u>How can we make the model more robust?</u>**
  → Add various changes to the data
- Mimicking “real-world” variations remain a challenging question!



Model Ensemble

- <u>**Use multiple models to enhance accuracy!**</u>
  (e.g., by averaging the results at test time)
  - ✓ Multiple "snapshots" of a same model during training
  - ✓ Multiple independent model
- Very common for challenges and competitions

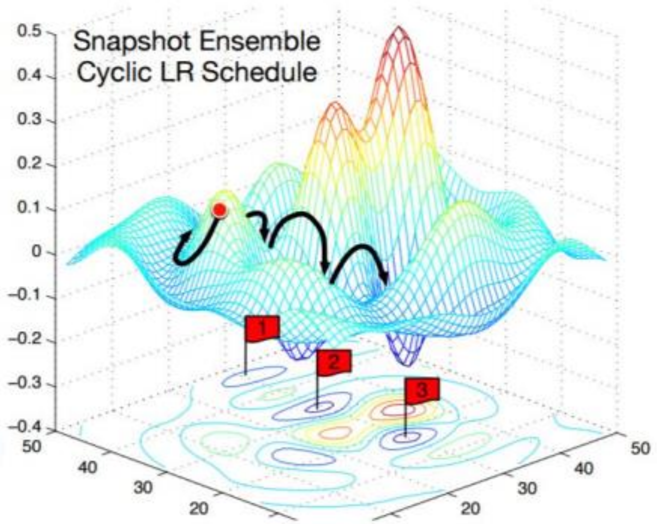
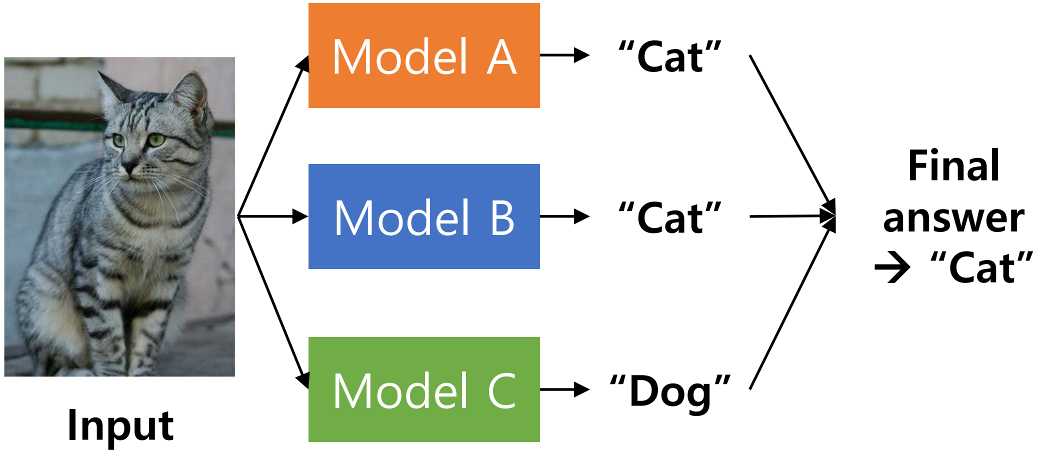


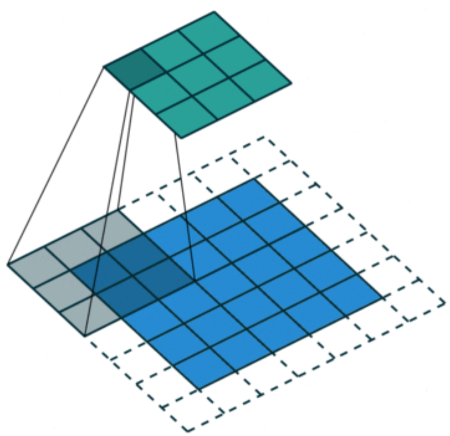
Figure from *G. Huang et al., "SNAPSHOT ENSEMBLES: TRAIN 1, GET M FOR FREE," ICLR 2017.*



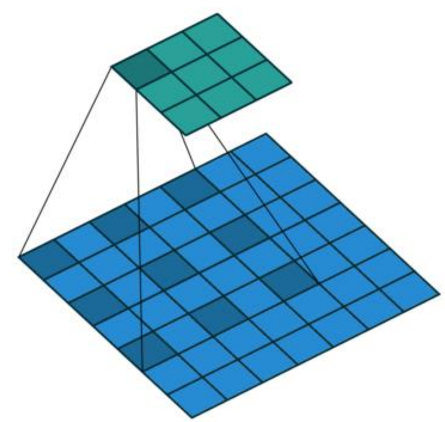
Efficient Convolutional Layers

- **Dilated convolution** [5]
  - ✓ Efficient for tasks that require a "global view" of the image (e.g., image segmentation)

Standard convolution



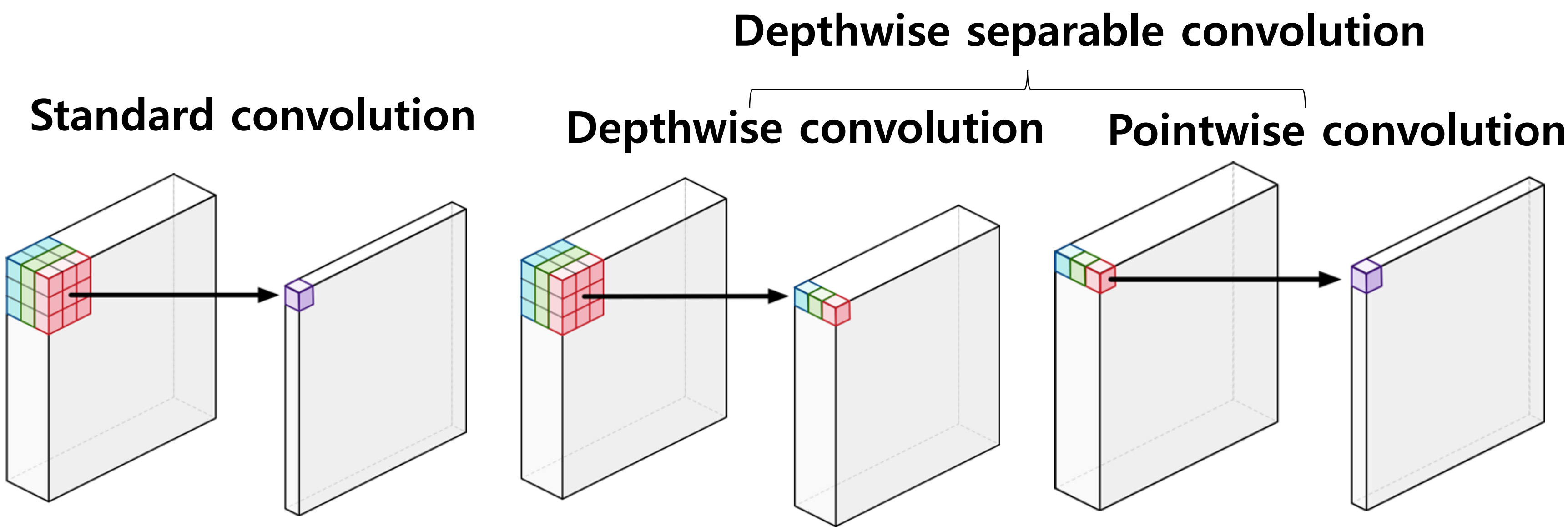
Dilated convolution



[5] F. Yu et al., "MULTI-SCALE CONTEXT AGGREGATION BY DILATED CONVOLUTIONS," ICLR 2016.

Efficient Convolutional Layers

- **<u>Dilated convolution</u>** [5]
  - ✓ Efficient for tasks that require a "global view" of the image (e.g., image segmentation)
- **<u>Depthwise separable convolution</u>** [6]
  - ✓ Commonly used in lightweight CNNs (e.g., MobileNet, Xception)

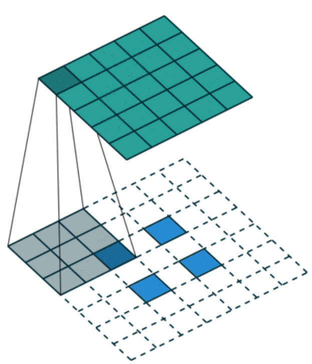


[6] A. Howard et al., "MobileNets: Efficient Convolutional Neural Networks for Mobile Vision ," arXiv 2017.

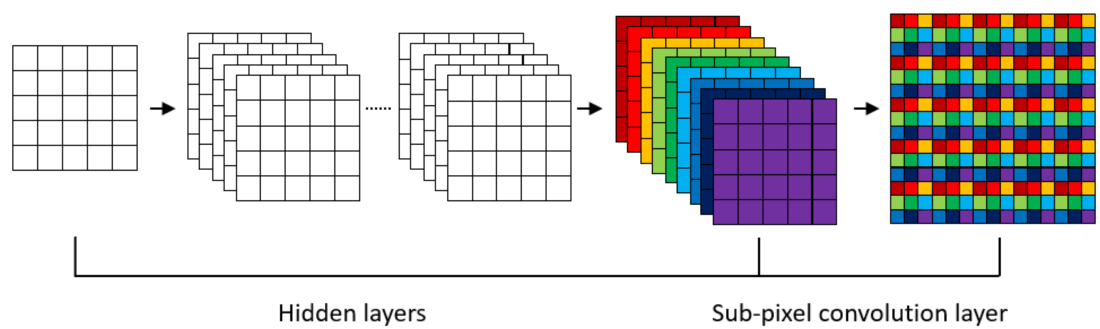
Efficient Convolutional Layers

- **Dilated convolution** [5]
  - ✓ Efficient for tasks that require a "global view" of the image (e.g., image segmentation)
- **Depthwise separable convolution** [6]
  - ✓ Commonly used in lightweight CNNs (e.g., MobileNet, Xception)
- **Sub-pixel convolution** [7]
  - ✓ Computationally efficient alternative of transposed convolution

Transposed convolution



Sub-pixel convolution



[7] *W. Shi et al., "Real-Time Single Image and Video Super-Resolution Using an Efficient Sub-Pixel Convolutional Neural Network," CVPR 2016.*