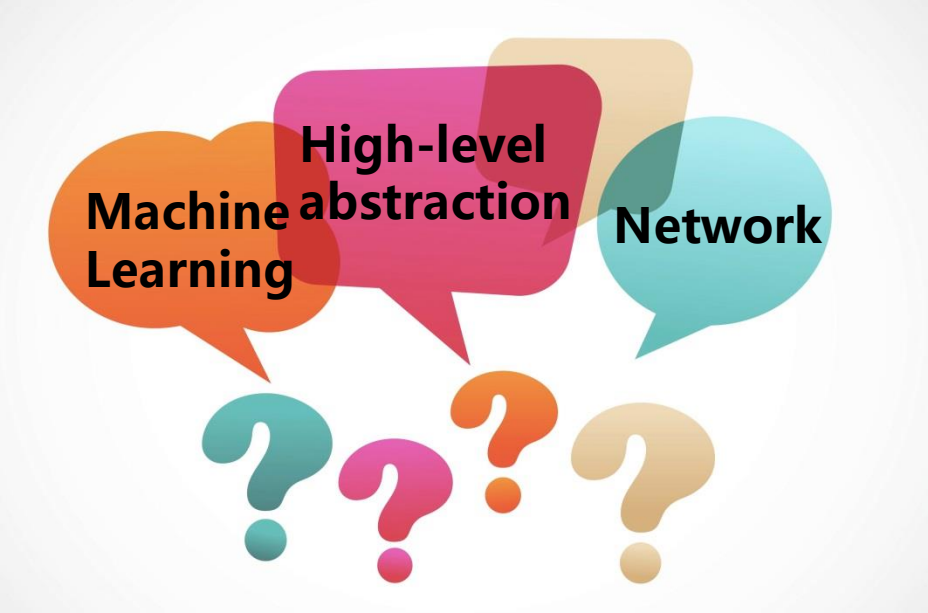
Introduction to deep learning

Sungjoon Choi
(sungjoon.choi@cpslab.snu.ac.kr)

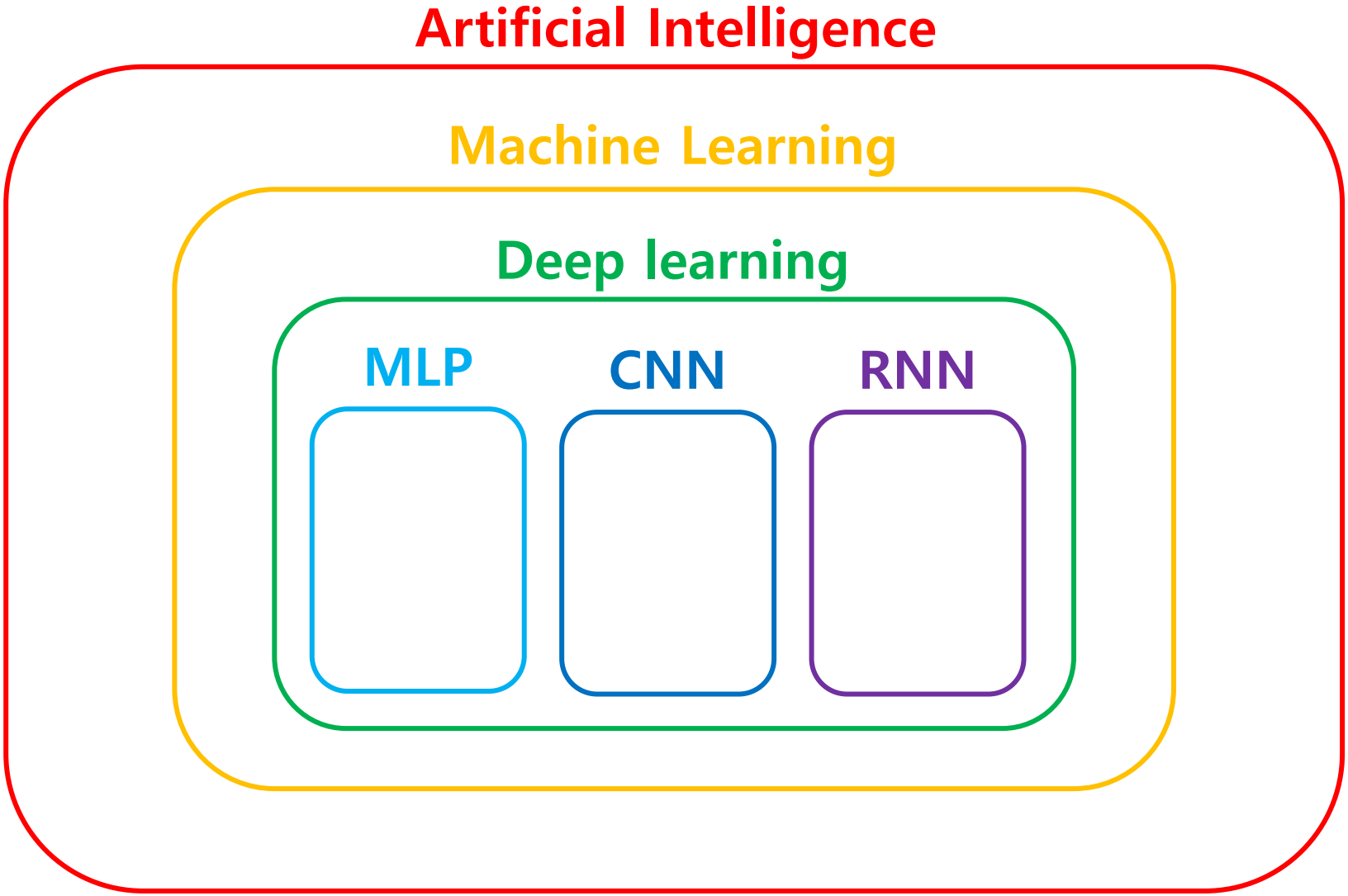
Deep learning

Wikipedia says:

"**Deep learning** is a branch of machine learning based on a set of algorithms that attempt to model high-level abstractions in data by using multiple processing layers, with complex structures or otherwise, composed of multiple non-linear transformations."



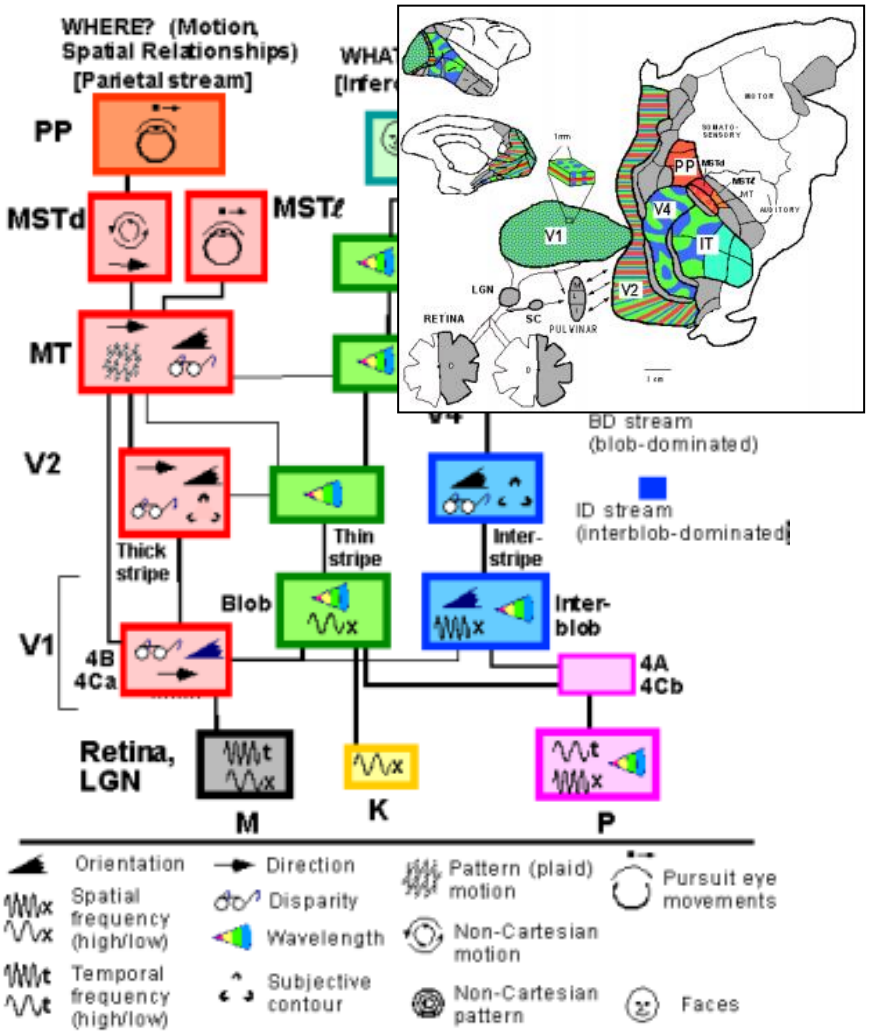
Deep learning



Is it brand new?

Neural Nets	McCulloch & Pitt 1943
Perceptron	Rosenblatt 1958
RNN	Grossberg 1973
CNN	Fukushima 1979
RBM	Hinton 1999
DBN	Hinton 2006
D-AE	Vincent 2008
AlexNet	Alex 2012
GoogLeNet	Szegedy 2015

Inspired by nature?



Sure there is some resemblance!

L'Avion III de Clement Ader, 1897

But, do not go TOO FAR.

Slide credit: Yann LeCun

Theory

Neural Networks, Vol. 2, pp. 359–366, 1989
Printed in the USA. All rights reserved.

0893-6080/89 $3.00 + .00

English

There is a single hidden layer feedforward network that approximates any measurable function to any desired degree of accuracy on some compact set K.

Math

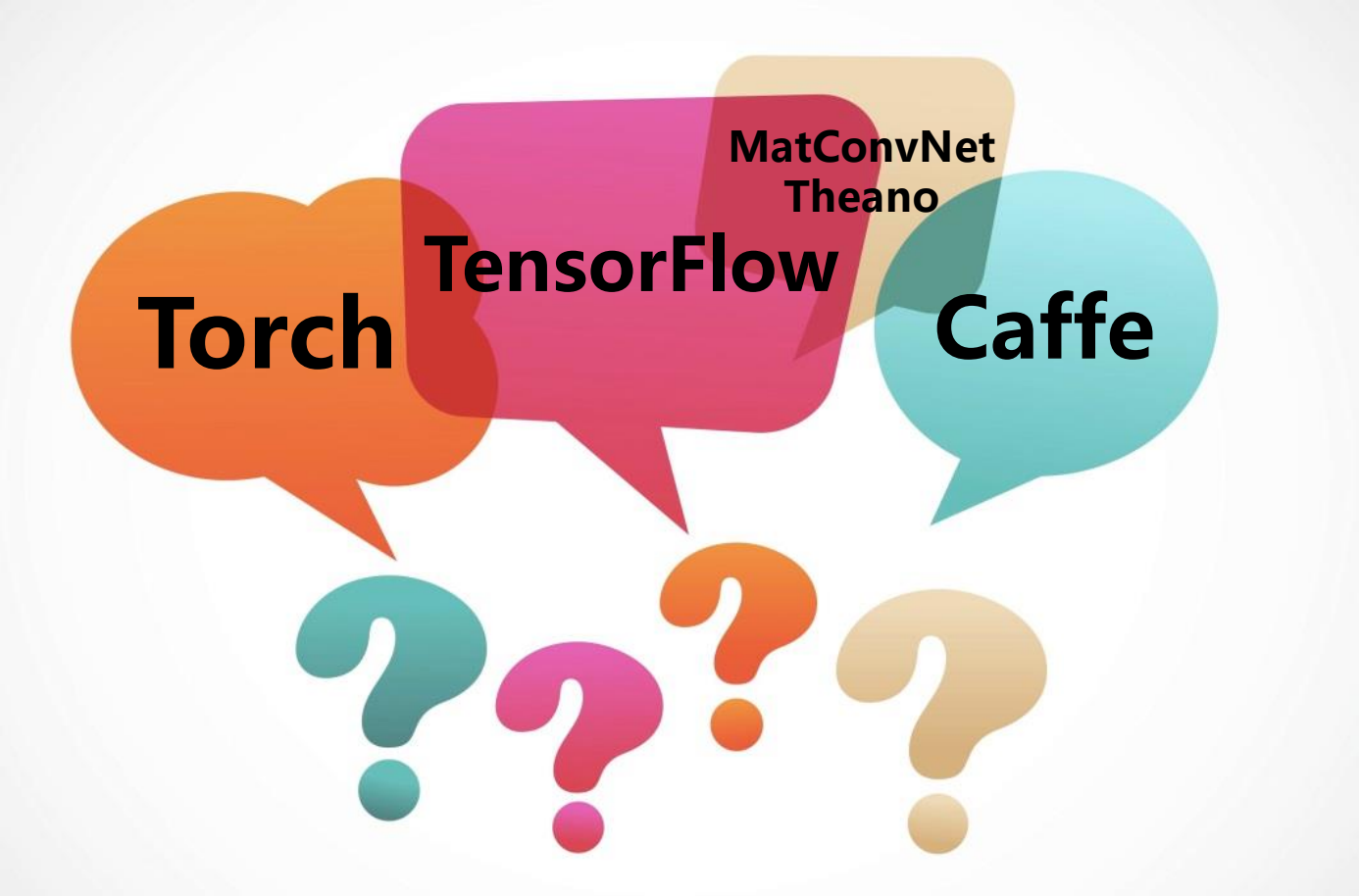
For every function g in M^r there is a compact subset K of R^r and an $f \in \sum^r(\Psi)$ such that for any $\epsilon > 0$ we have $\mu(K) < 1 - \epsilon$ and for every $X \in K$ we have $|f(x) - g(x)| < \epsilon$, regardless of Ψ, r, or μ.

hidden units are available. In this sense, multilayer feedforward networks are a class of universal approximators.

Keywords—Feedforward networks, Universal approximation, Mapping networks, Network representation capability, Stone-Weierstrass Theorem, Squashing functions, Sigma-Pi networks, Back-propagation networks.

Cited > 12,000

Deep learning tools?



White to black

Course Contents

Sungjoon Choi
(sungjoon.choi@cpslab.snu.ac.kr)

Convolutional neural network

AlexNet

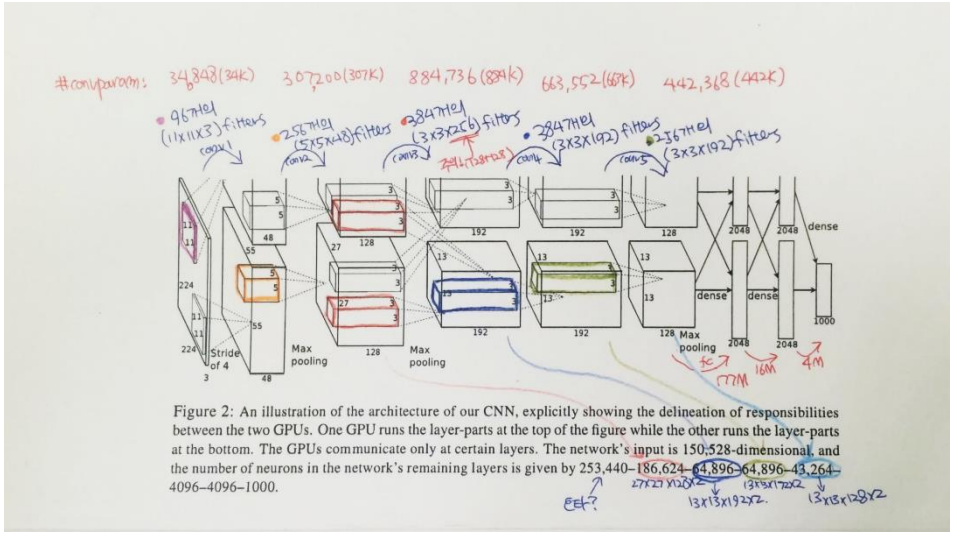
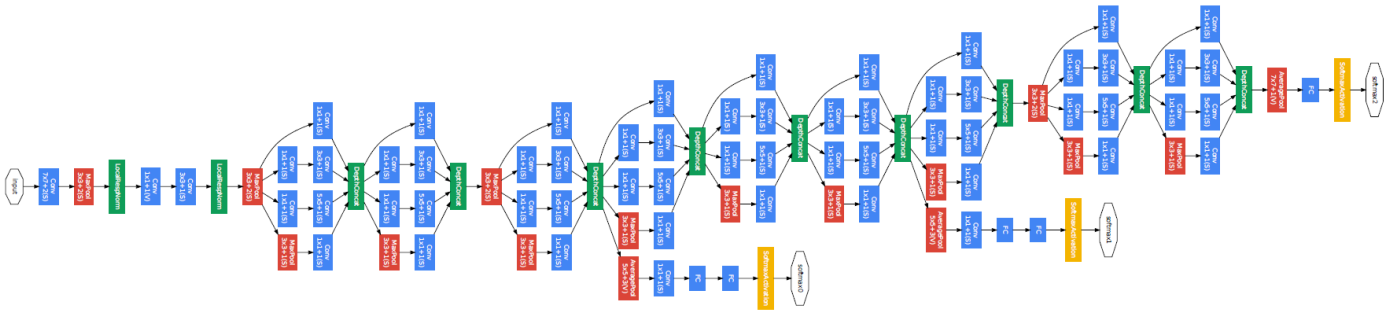
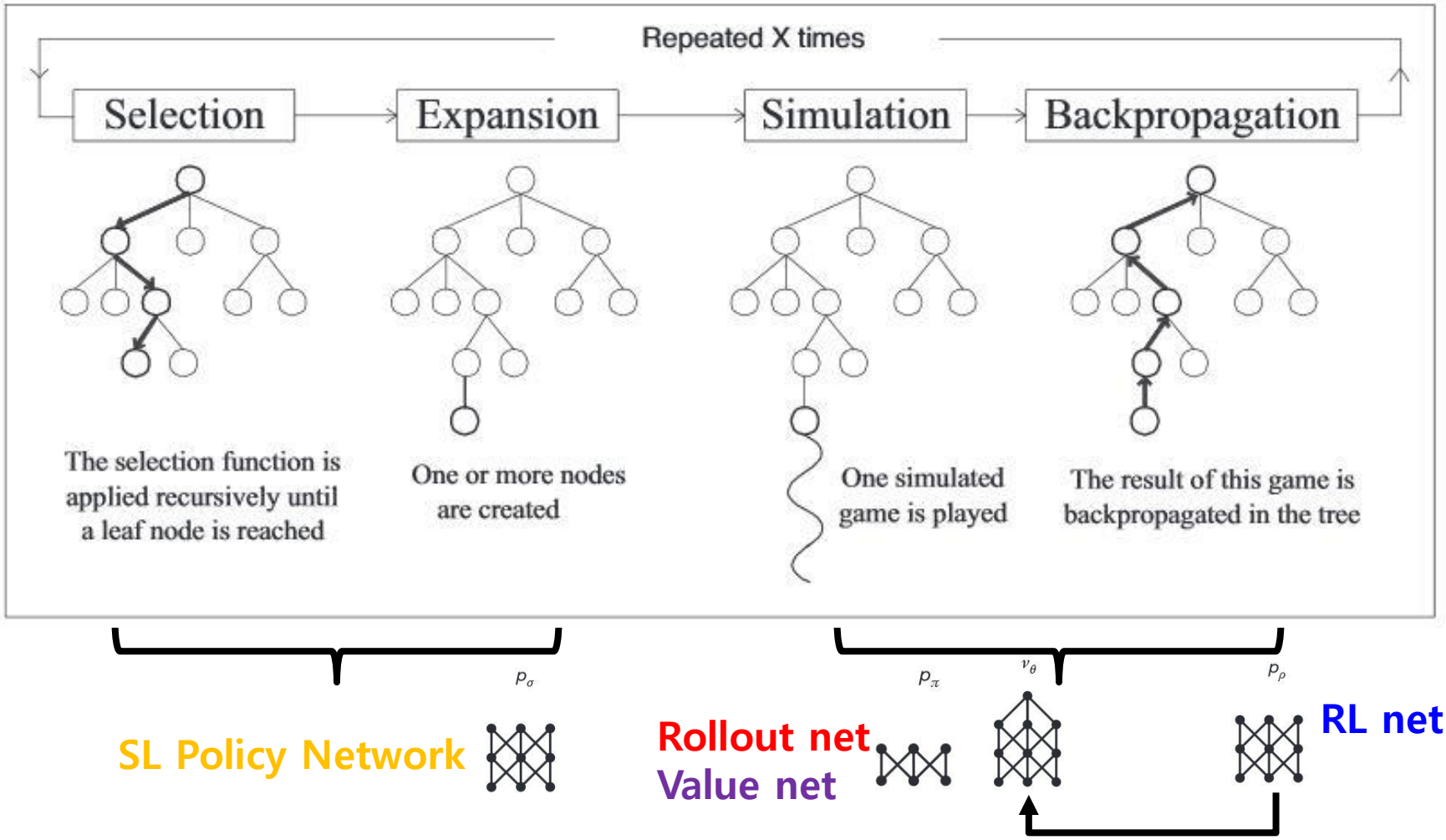


Figure 2: An illustration of the architecture of our CNN, explicitly showing the delineation of responsibilities between the two GPUs. One GPU runs the layer-parts at the top of the figure while the other runs the layer-parts at the bottom. The GPUs communicate only at certain layers. The network's input is 150,528-dimensional, and the number of neurons in the network's remaining layers is given by 253,440–186,624–64,896–64,896–43,264–4096–4096–1000.

GoogleLeNet



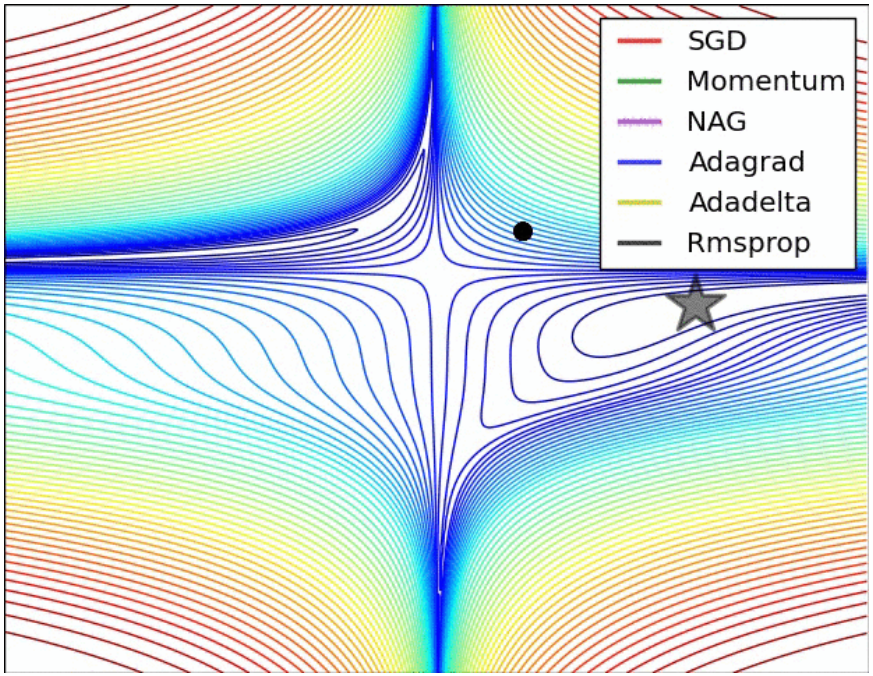
Algorithms behind AlphaGo

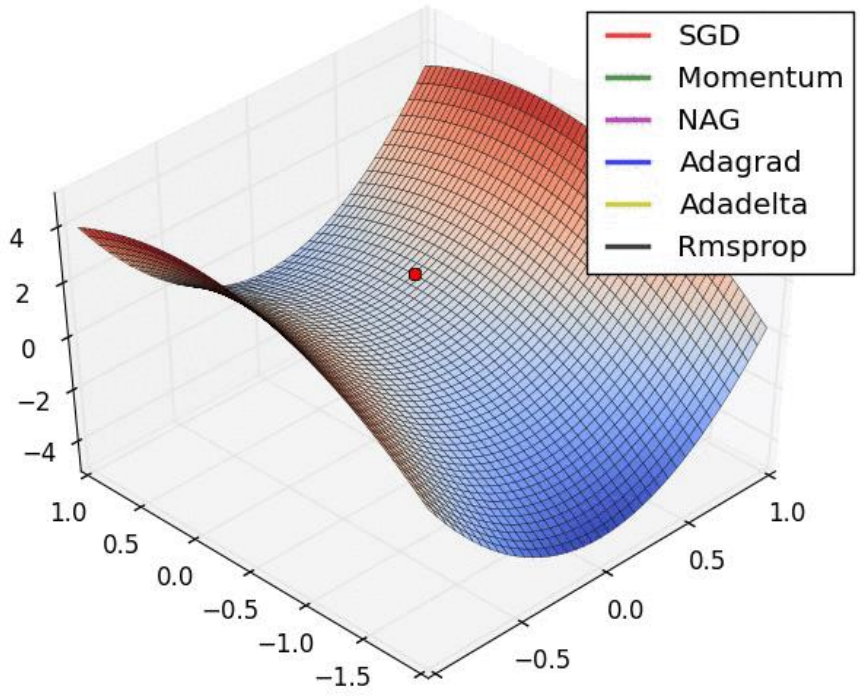


Regularization

Main purpose is to avoid "**OverFitting**"

Optimization Methods





Restricted Boltzmann machine

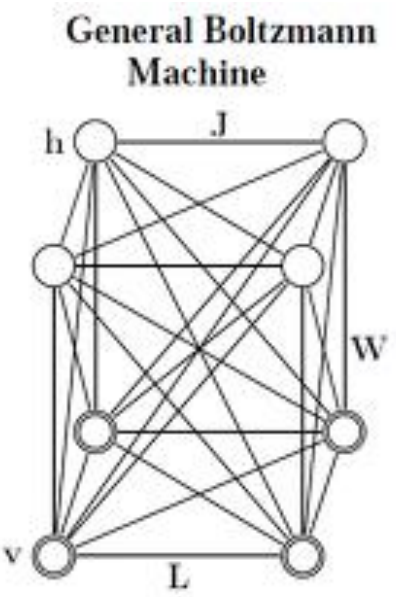
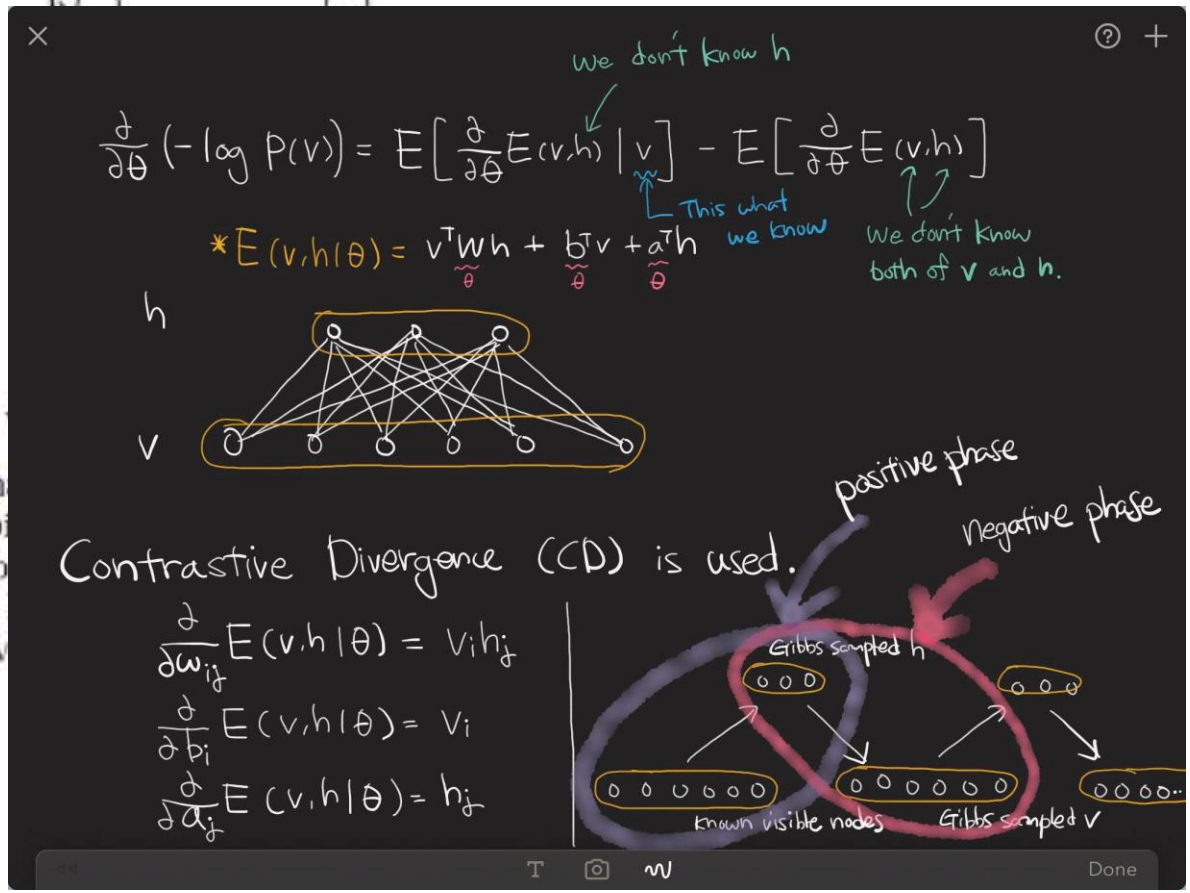


Figure 1: **Left**: A general Boltzm
represents a vector of stochastic bi
the bottom layer represents a vecto
ble" variables. **Right**: A restricted
hidden-to-hidden and no visible-to-v



Semantic Segmentation

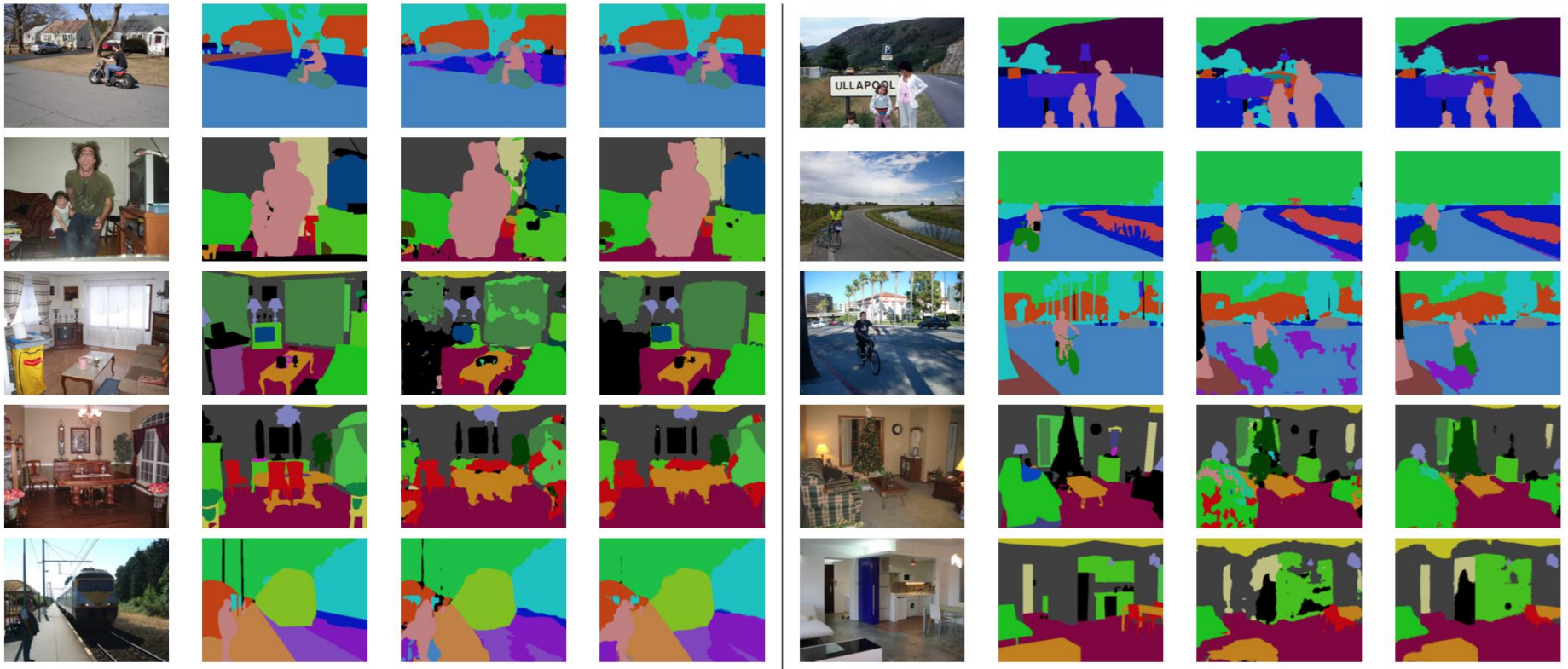
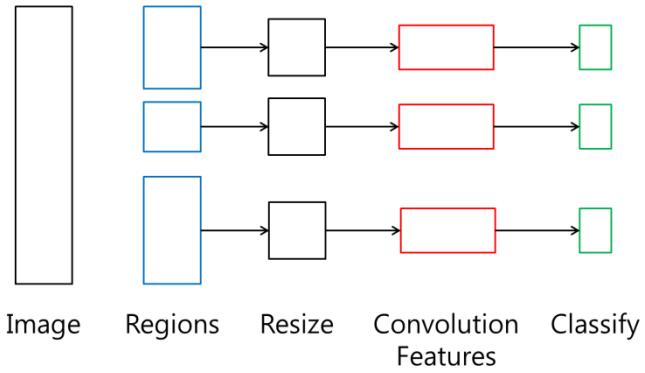
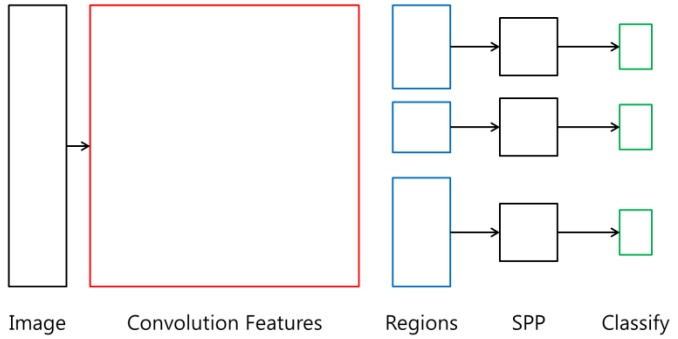


Fig. 11: PASCAL-Context results. Input image, ground-truth, and our DeepLab results before/after CRF.

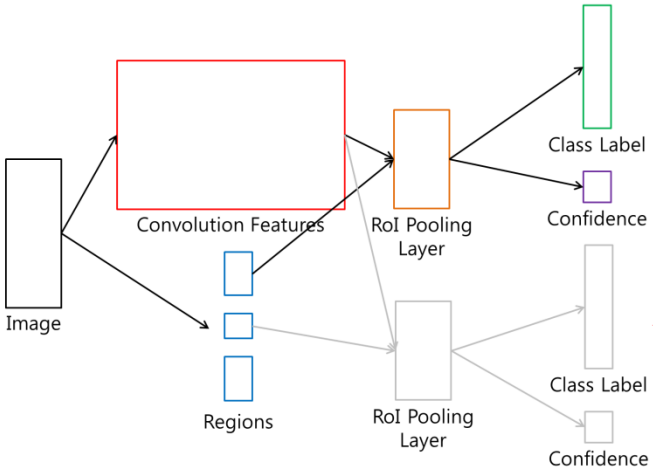
Detection Methods



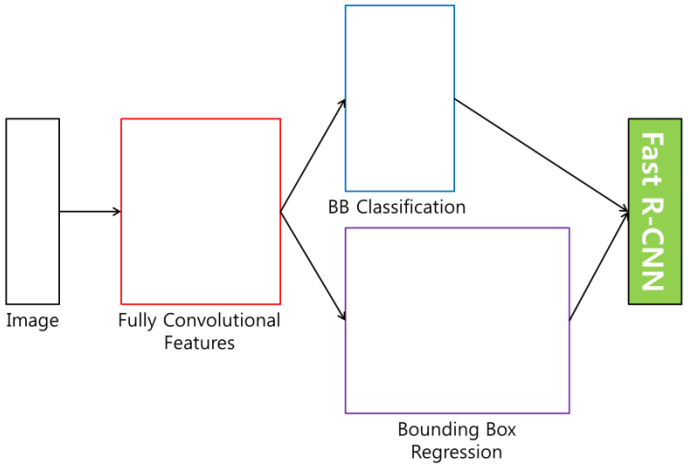
R-CNN



SPP net

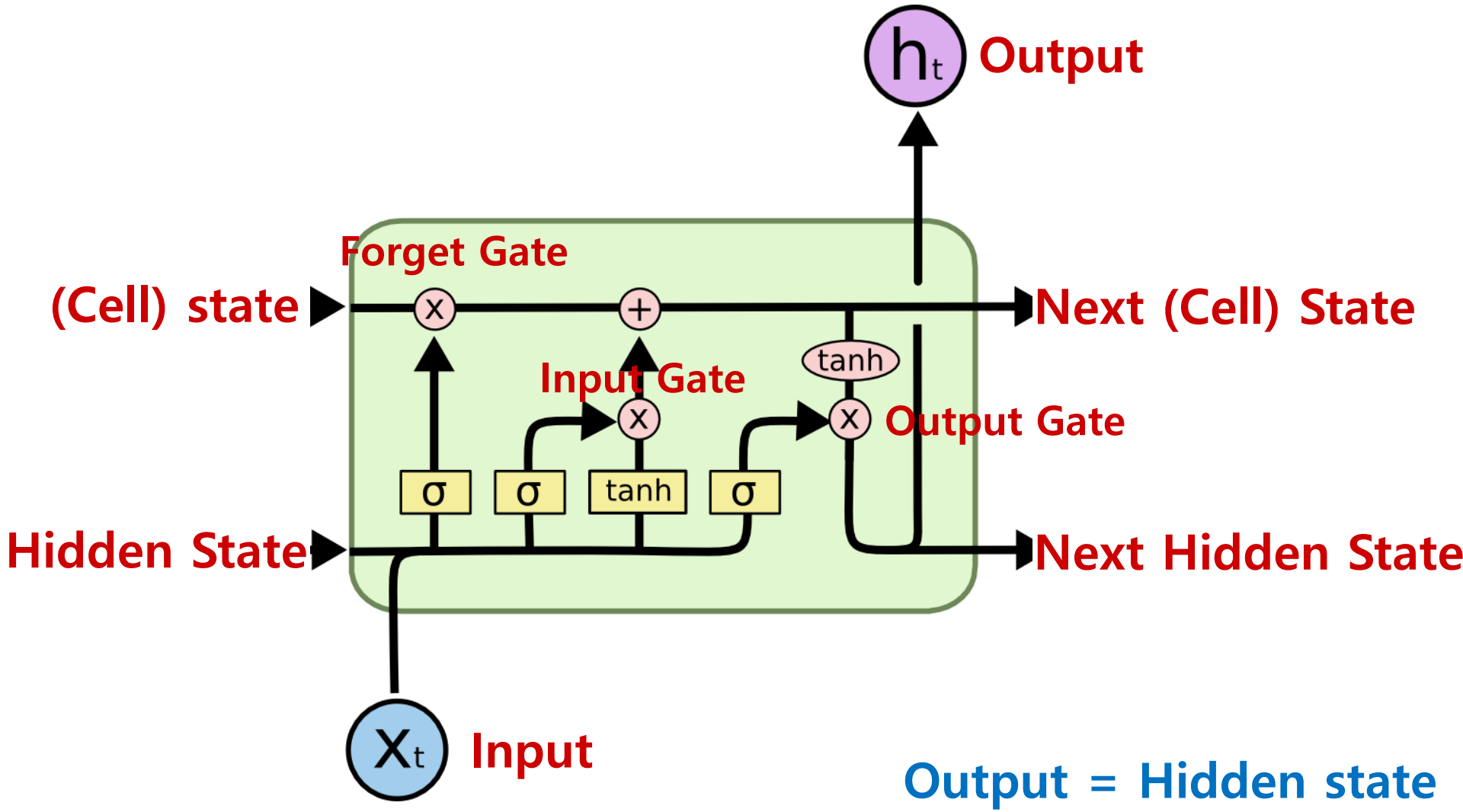


Fast R-CNN



Faster R-CNN

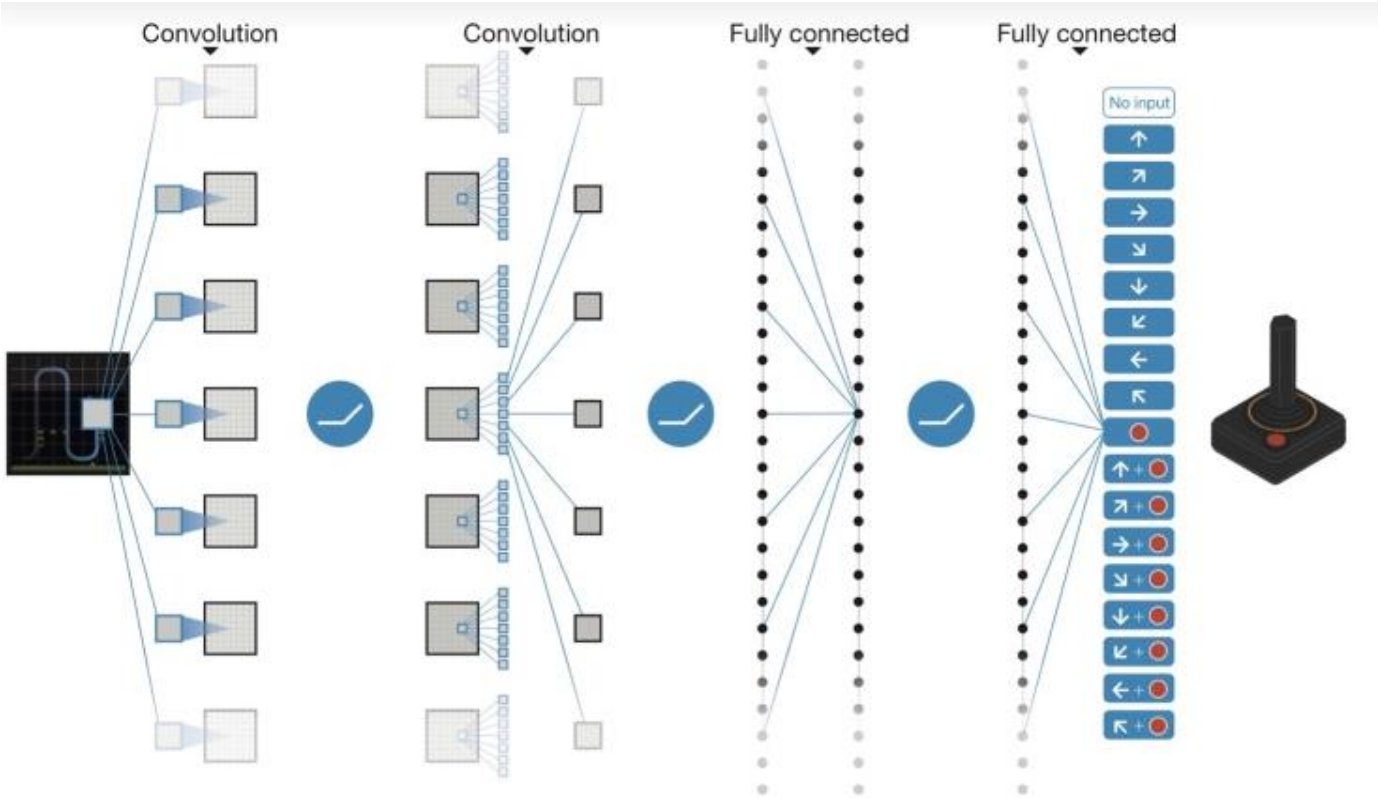
RNN - LSTM



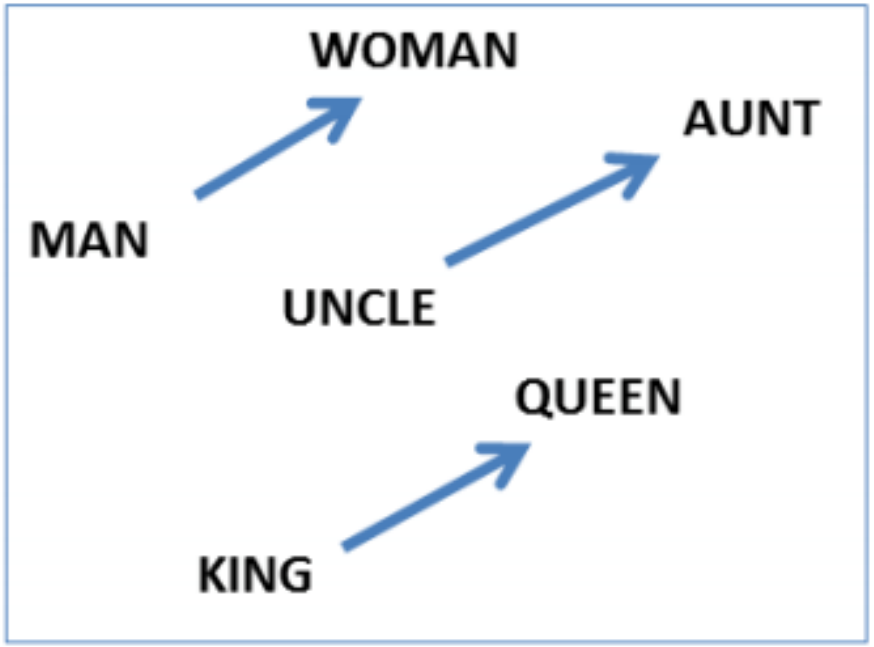
Output = Hidden state

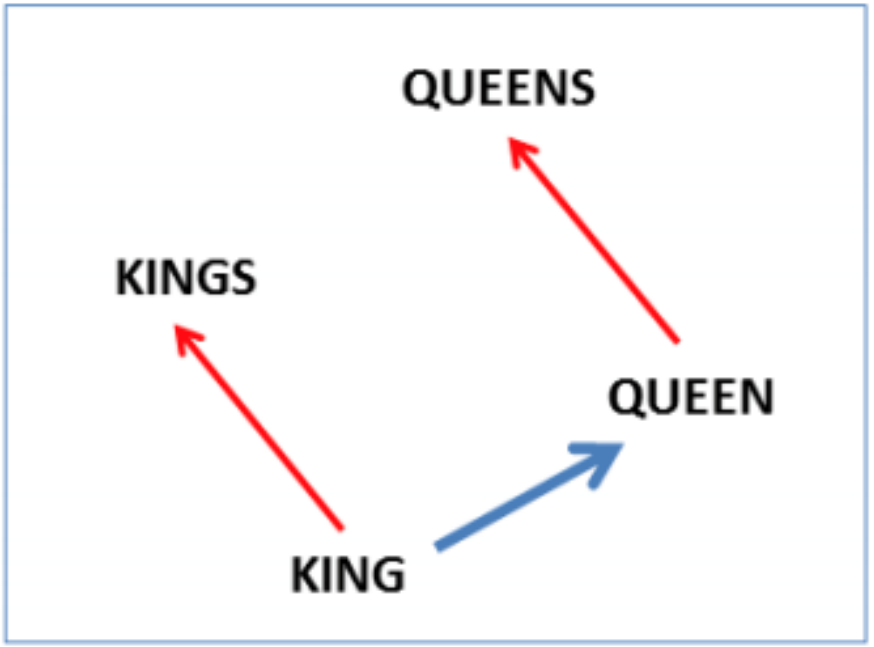
http://colah.github.io/posts/2015-08-Understanding-LSTMs/

Deep Reinforcement Learning



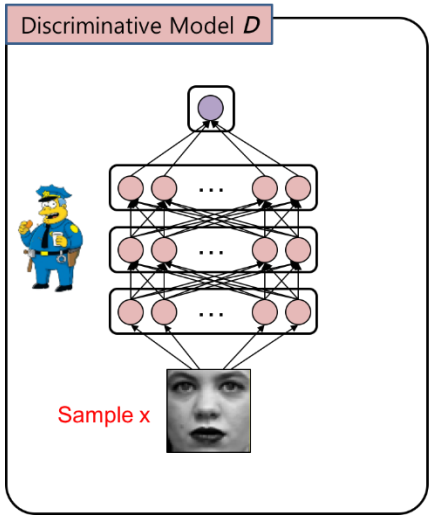
Word Embedding

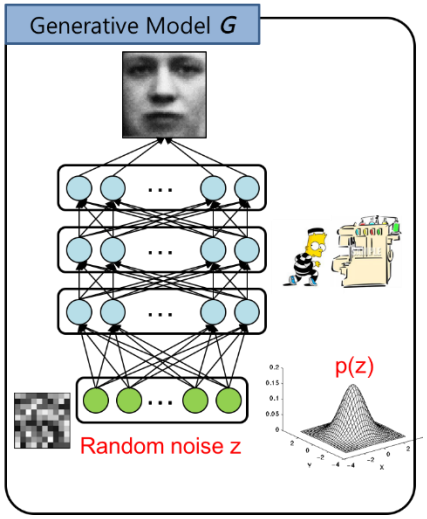


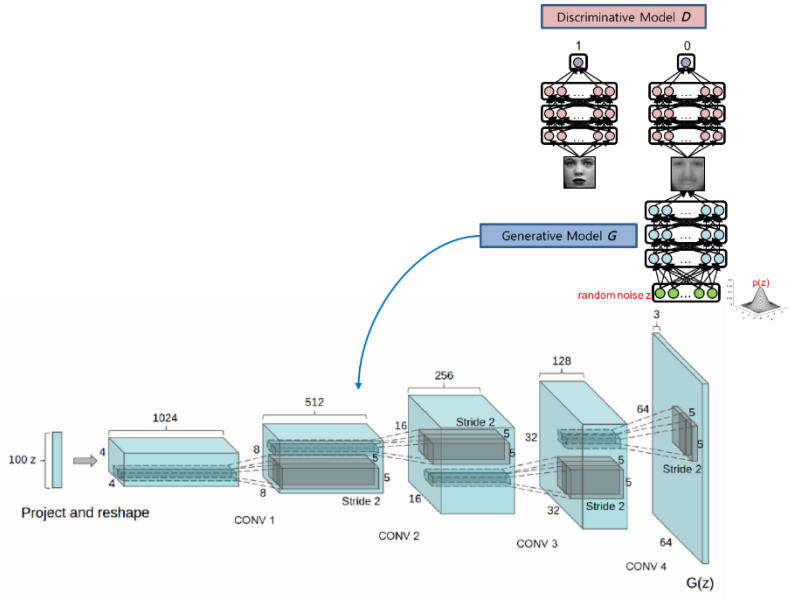


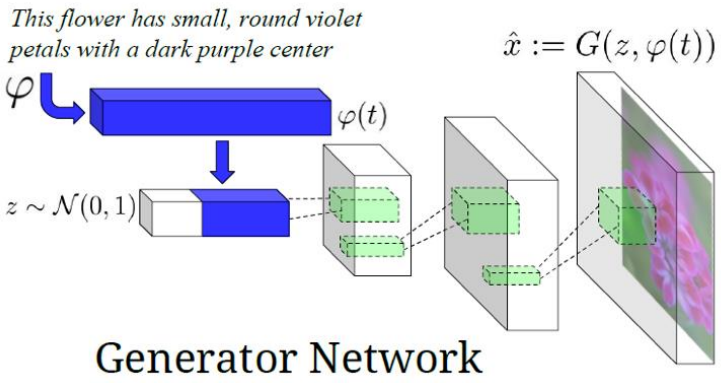
(Mikolov et al., NAACL HLT, 2013)

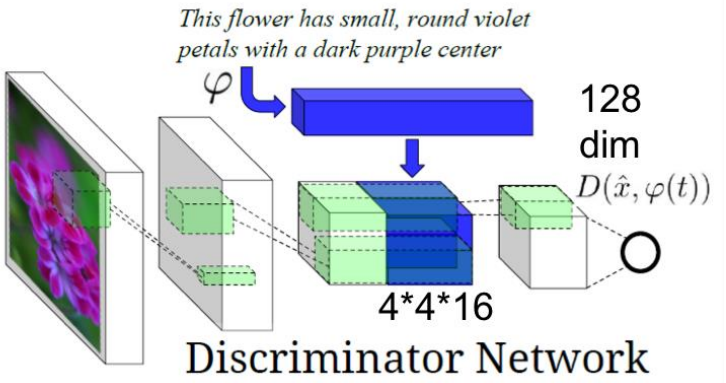
Generative Adversarial Networks









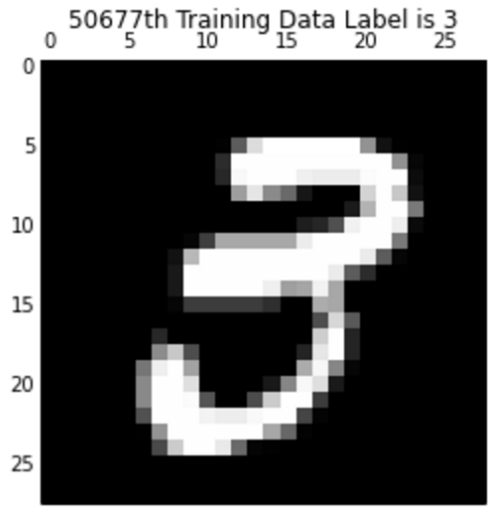


+ Alpha

Using MNIST

```
# How does the training data look like?
print ("How does the data look like?")
nsample = 1
randidx = np.random.randint(trainimg.shape[0], size=nsample)

for i in randidx:
    curr_img   = np.reshape(trainimg[i, :], (28, 28)) # 28 by 28 matrix
    curr_label = np.argmax(trainlabel[i, :] ) # Label
    plt.matshow(curr_img, cmap=plt.get_cmap('gray'))
    plt.title("" + str(i) + "th Training Data "
              + "Label is " + str(curr_label))
    print ("" + str(i) + "th Training Data "
           + "Label is " + str(curr_label))
```

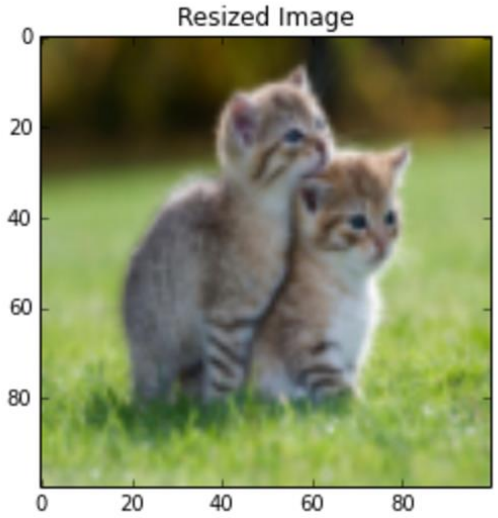
```
How does the data look like?
50677th Training Data Label is 3
```



Basic Image Handling

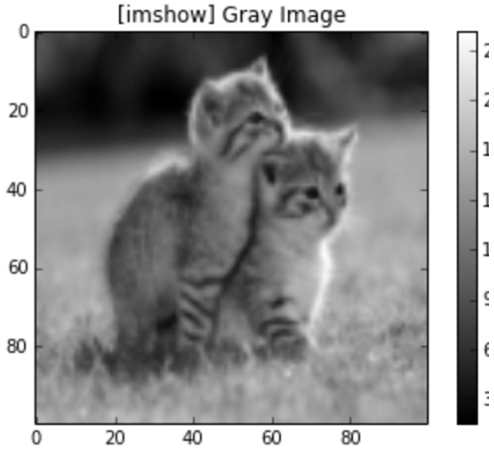
```
# Resize
catsmall = imresize(cat, [100, 100, 3])
print_typeshape(catsmall)
# Plot
plt.figure(1)
plt.imshow(catsmall)
plt.title("Resized Image")
plt.draw()
```

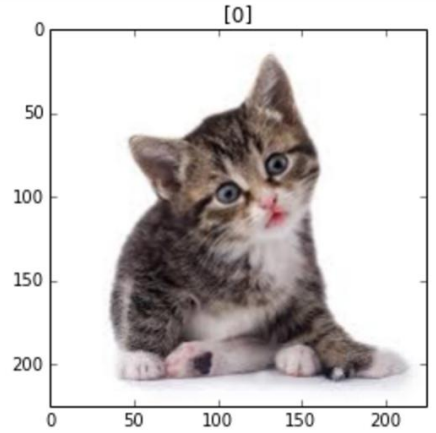
```
Type is <type 'numpy.ndarray'>
Shape is (100, 100, 3)
```

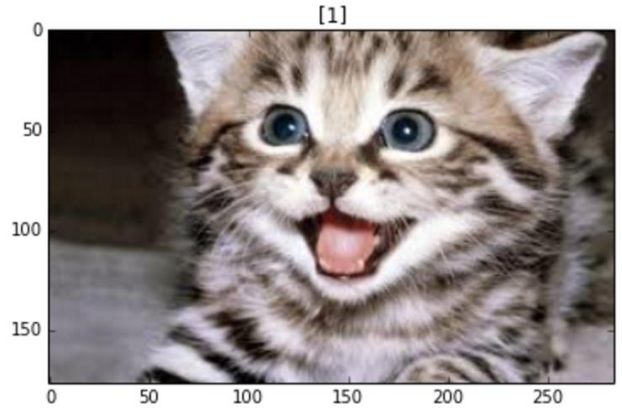



```
# Grayscale
def rgb2gray(rgb):
    if len(rgb.shape) is 3:
        return np.dot(rgb[...,:3]
    else:
        print ("Current Image if (
        return rgb
catsmallgray = rgb2gray(catsmall)

print ("size of catsmallgray is %s
print ("type of catsmallgray is",

plt.figure(2)
plt.imshow(catsmallgray, cmap=plt.
plt.title("[imshow] Gray Image")
plt.colorbar()
plt.draw()
```

```
size of catsmallgray is (100, 100)
('type of catsmallgray is', <type
```

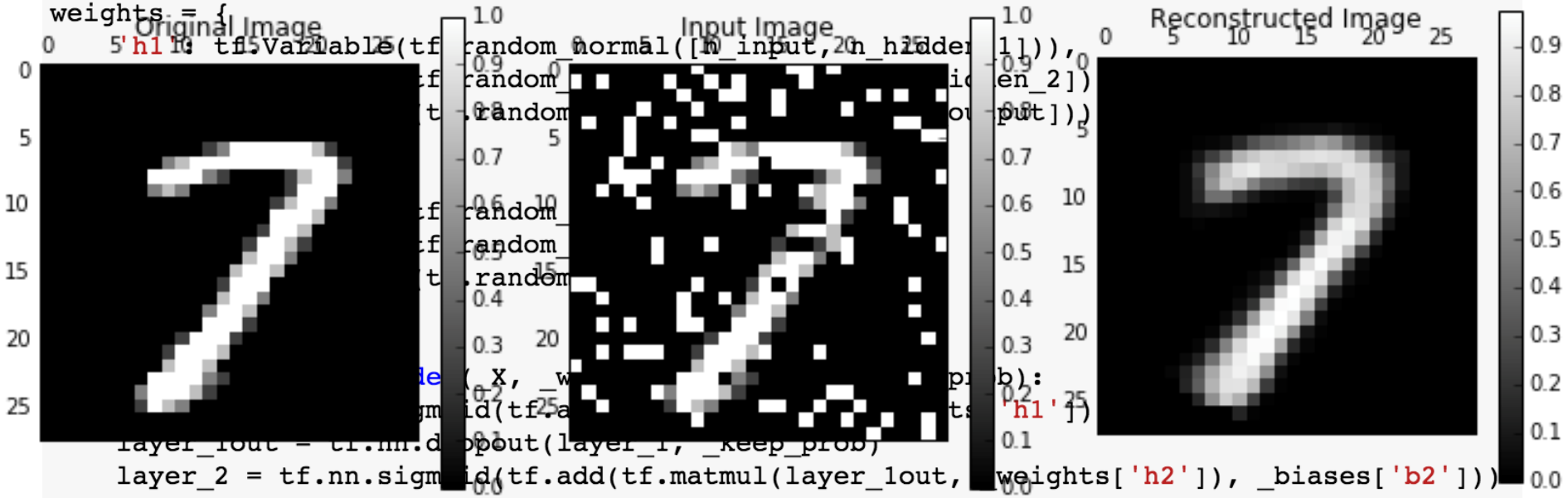






Denoising Auto-Encoder

```
# tf Graph input
x = tf.placeholder("float", [None, n_input])
y = tf.placeholder("float", [None, n_output])
dropout_keep_prob = tf.placeholder("float")
# Store layers weight & bias
weights = {
    'h1': tf.Variable(tf.random_normal([n_input, n_hidden_1])),
    'h2': tf.Variable(tf.random_normal([n_hidden_1, n_hidden_2])),
    'out': tf.Variable(tf.random_normal([n_hidden_2, n_output]))
}
biases = {
    'b1': tf.Variable(tf.random_normal([n_hidden_1])),
    'b2': tf.Variable(tf.random_normal([n_hidden_2])),
    'out': tf.Variable(tf.random_normal([n_output]))
}

def multilayer_perceptron(_X, _weights, _biases, _keep_prob):
    layer_1 = tf.nn.sigmoid(tf.add(tf.matmul(_X, _weights['h1']), _biases['b1'])) #Hidden layer with sigmoid activation
    layer_1out = tf.nn.dropout(layer_1, _keep_prob)
    layer_2 = tf.nn.sigmoid(tf.add(tf.matmul(layer_1out, _weights['h2']), _biases['b2']))
    layer_2out = tf.nn.dropout(layer_2, _keep_prob)
    return tf.nn.sigmoid(tf.matmul(layer_2out, _weights['out']) + _biases['out'])
```



Convolutional Neural Network

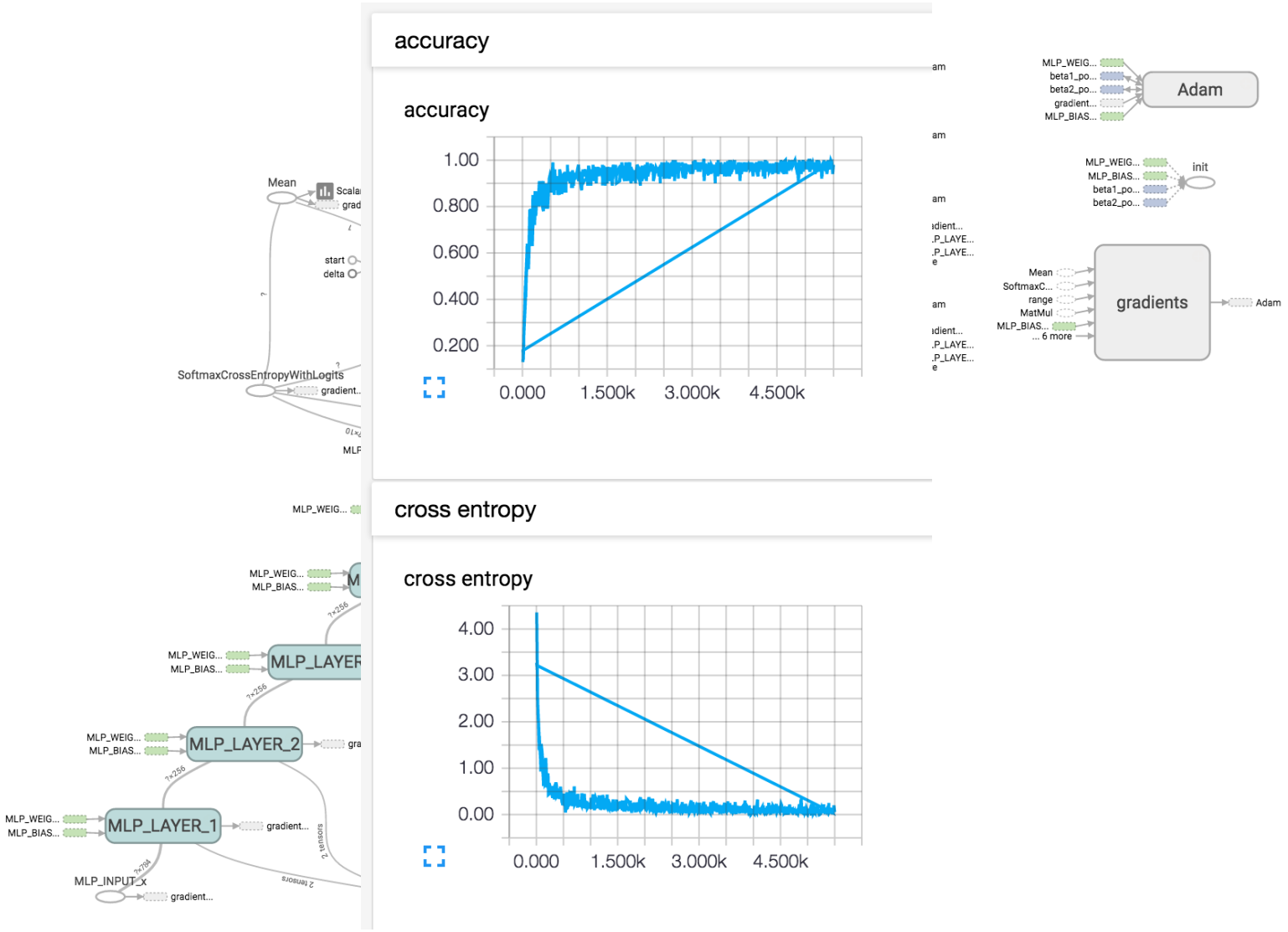
```
weights  = {
    'wc1': tf.Variable(tf.random_normal([3, 3, 1, 64], stddev=0.1)),
    'wc2': tf.Variable(tf.random_normal([3, 3, 64, 128], stddev=0.1)),
    'wd1': tf.Variable(tf.random_normal([7*7*128, 1024], stddev=0.1)),
    'wd2': tf.Variable(tf.random_normal([1024, n_output], stddev=0.1))
}
biases   = {
    'bc1': tf.Variable(tf.random_normal([64], stddev=0.1)),
    'bc2': tf.Variable(tf.random_normal([128], stddev=0.1)),
    'bd1': tf.Variable(tf.random_normal([1024], stddev=0.1)),
    'bd2': tf.Variable(tf.random_normal([n_output], stddev=0.1))
}
def conv_basic(_input, _w, _b, _keepratio):
    # Input
    _input_r = tf.reshape(_input, shape=[-1, 28, 28, 1])
    # Conv1
```

```
# Compute test accuracy
test_acc = sess.run(accr, feed_dict={x: testimg, y: testlabel, keepratio:1.})
print (" Test accuracy: %.3f" % (test_acc))
```

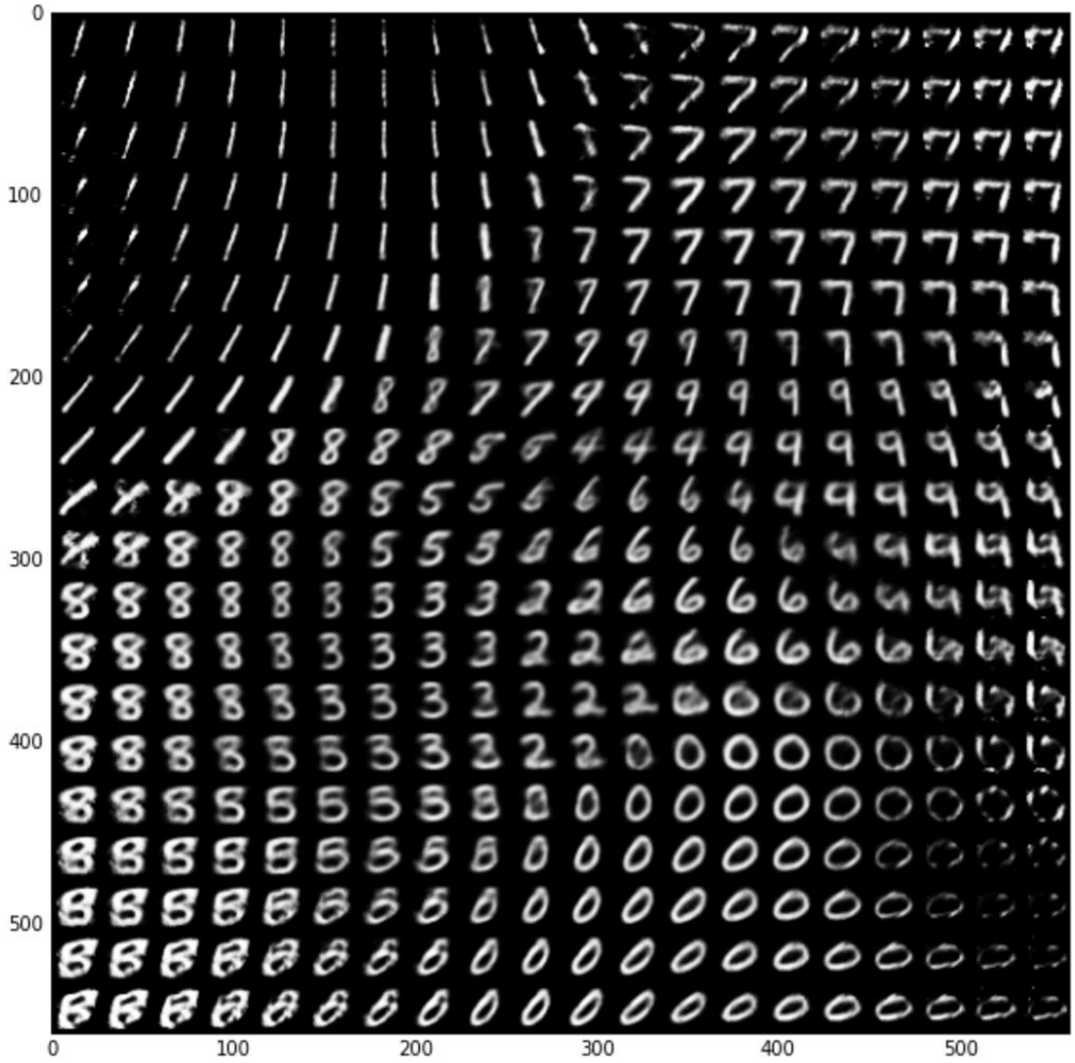
```
Test accuracy: 0.994
```

```
    # Vectorize
    _dense1 = tf.reshape(_pool_dr2, [-1, _w['wd1'].get_shape().as_list()[0]])
    # Fc1
    _fc1 = tf.nn.relu(tf.add(tf.matmul(_dense1, _w['wd1']), _b['bd1']))
    _fc_dr1 = tf.nn.dropout(_fc1, _keepratio)
    # Fc2
    _out = tf.add(tf.matmul(_fc_dr1, _w['wd2']), _b['bd2'])
    # Return everything
    out = {
        'input_r': _input_r,
        'conv1': _conv1,
        'pool1': _pool1,
        'pool1_dr1': _pool_dr1,
        'conv2': _conv2,
        'pool2': _pool2,
        'pool_dr2': _pool_dr2,
        'dense1': _dense1,
        'fc1': _fc1,
        'fc_dr1': _fc_dr1,
        'out': _out
    }
    return out
```

Tensor Board



Variational Autoencoder (VAE)

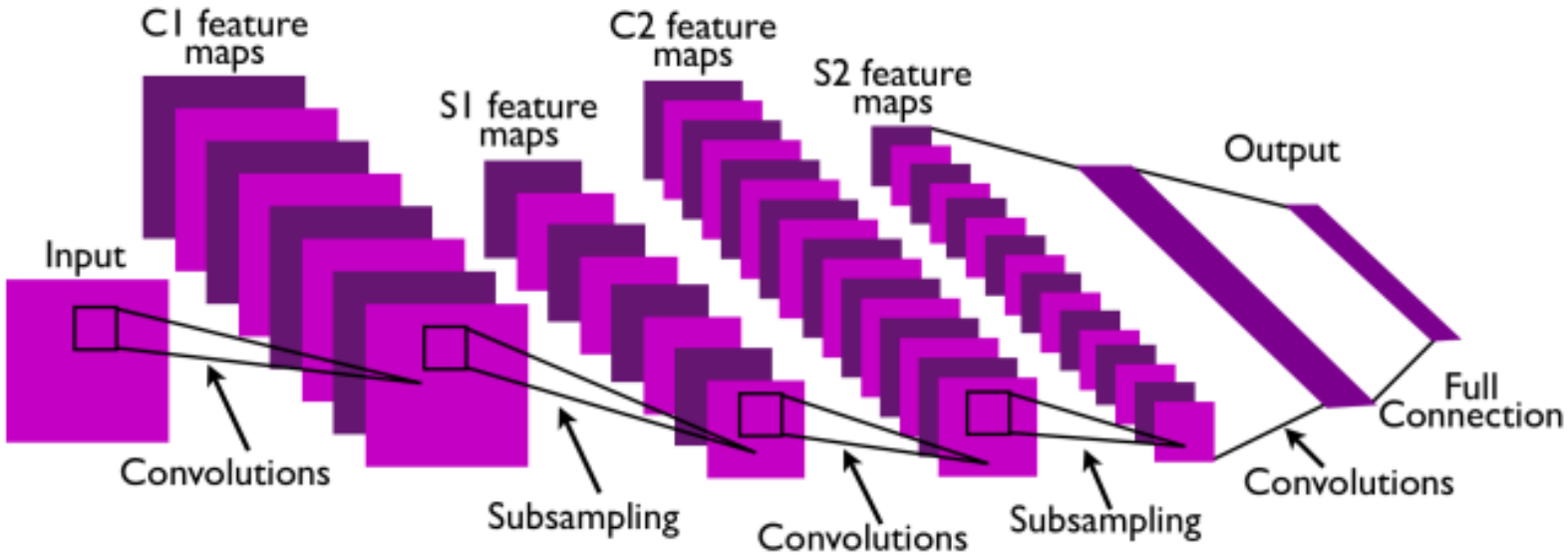


Introduction to CNNs

Sungjoon Choi
(sungjoon.choi@cpslab.snu.ac.kr)

CNN

Convolutional **N**eural **N**etwork



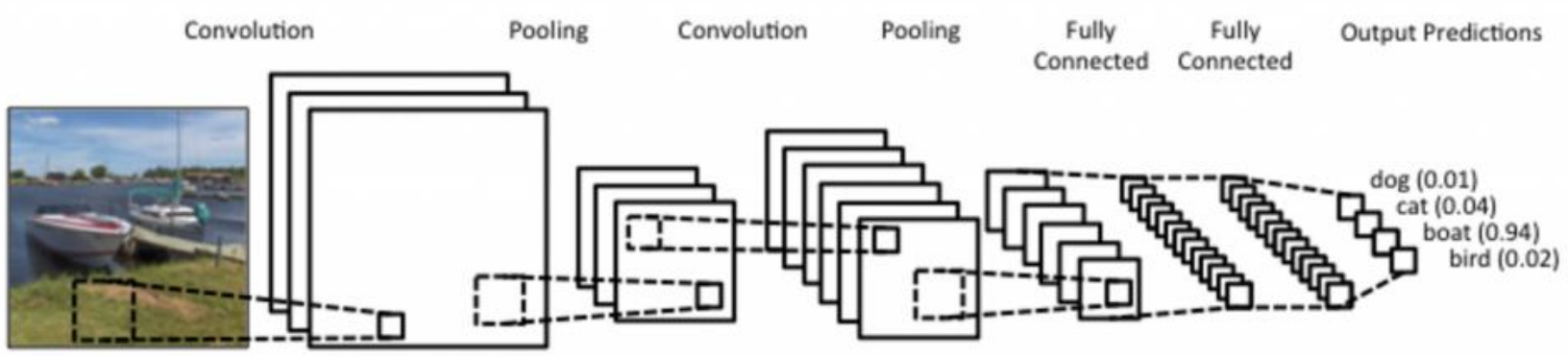
This is pretty much **everything** about the convolutional neural network.

Convolution + Subsampling + Full Connection

CNN

CNNs are basically layers of **convolutions** followed by **subsampling** and **fully connected layers.**

Intuitively speaking, **convolutions** and **subsampling** layers works as feature extraction layers while a **fully connected layer** classifies which category current input belongs to using extracted features.



Why so powerful?

Local Invariance

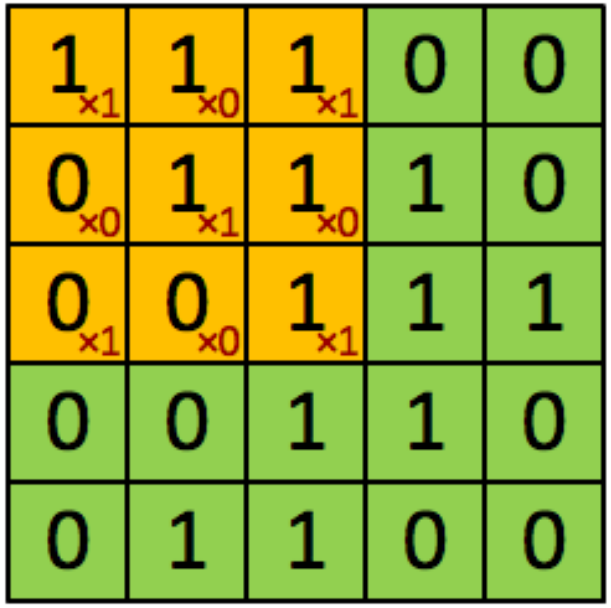
Loosely speaking, as the convolution filters are '**sliding**' over the input image, the exact location of the object we want to find does not matter much.

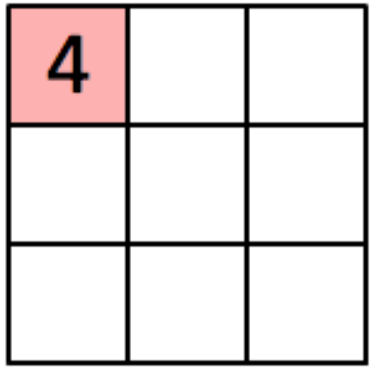
Compositionality

There is a hierarchy in CNNs. It is GOOD!

https://starwarsanon.wordpress.com/tag/darth-sidious-vs-yoda/

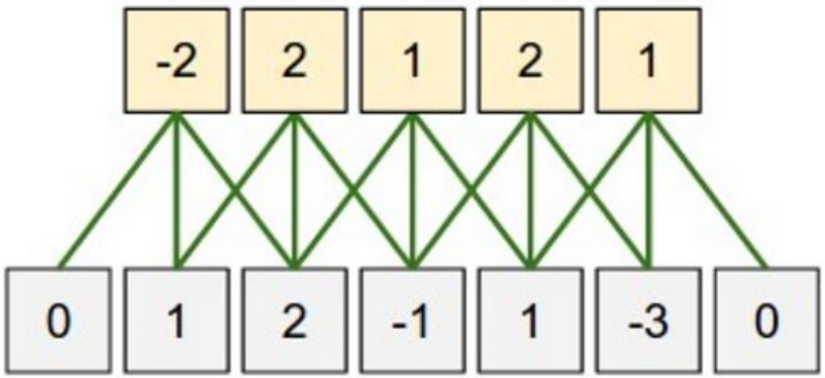
Convolution





http://deeplearning.stanford.edu/wiki/index.php/Feature_extraction_using_convolution

Zero-padding



What is the size of the input? $n_{in} = 5$

What is the size of the output? $n_{out} = 5$

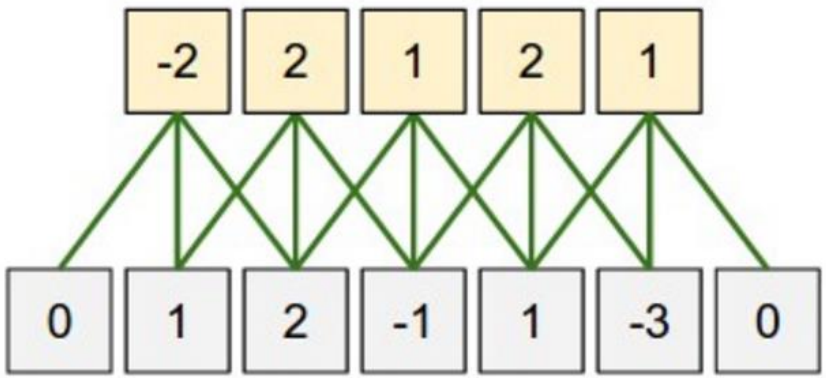
What is the size of the filter? $n_{filter} = 3$

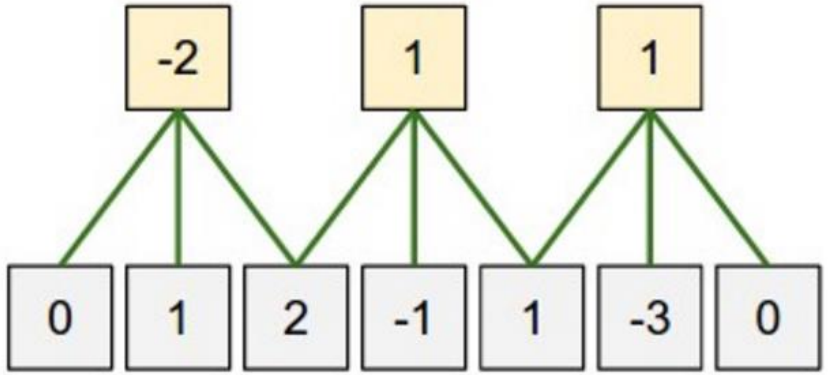
What is the size of the zero-padding? $n_{padding} = 1$

$n_{out} = (n_{in} + 2 * n_{padding} - n_{filter}) + 1$ $5 = (5 + 2 * 1 - 3) + 1$

Stride

Stride





(Left) Stride size: 1

(Right) Stride size: 2

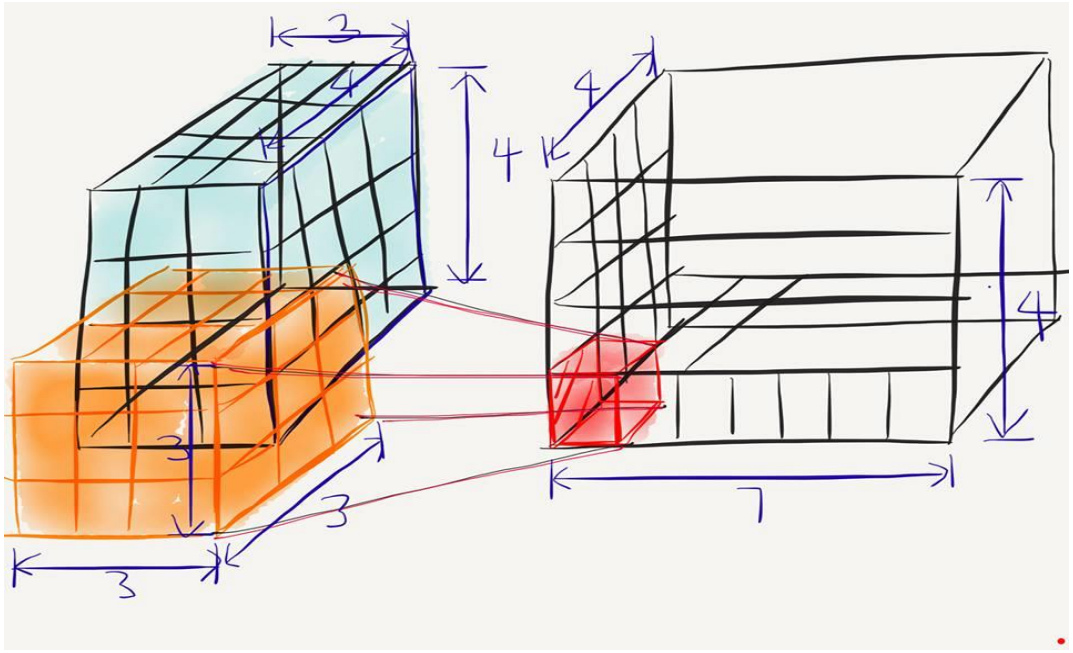
If stride size equals the filter size, there will be **no overlapping.**

Conv2D

```
tf.nn.conv2d(input, filter, strides, padding,
use_cudnn_on_gpu=None, name=None)
```

Computes a 2-D convolution given 4-D `input` and `filter` tensors.

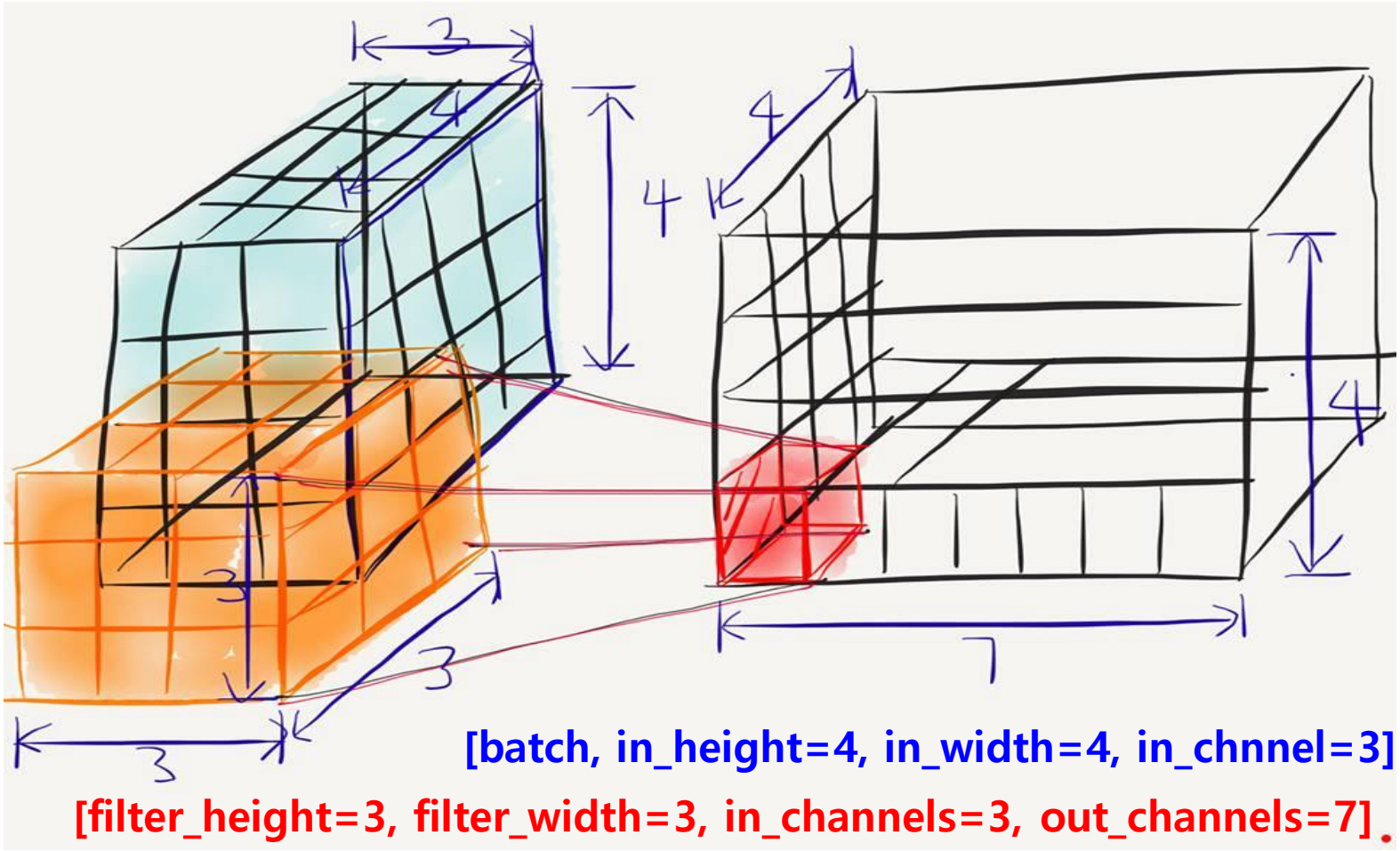
Given an input tensor of shape **[batch, in_height, in_width, in_chnnel]** and a filter / kernel tensor of shape **[filter_height, filter_width, in_channels, out_channels]** this op performs the following:



[batch, in_height=4, in_width=4, in_chnnel=3]

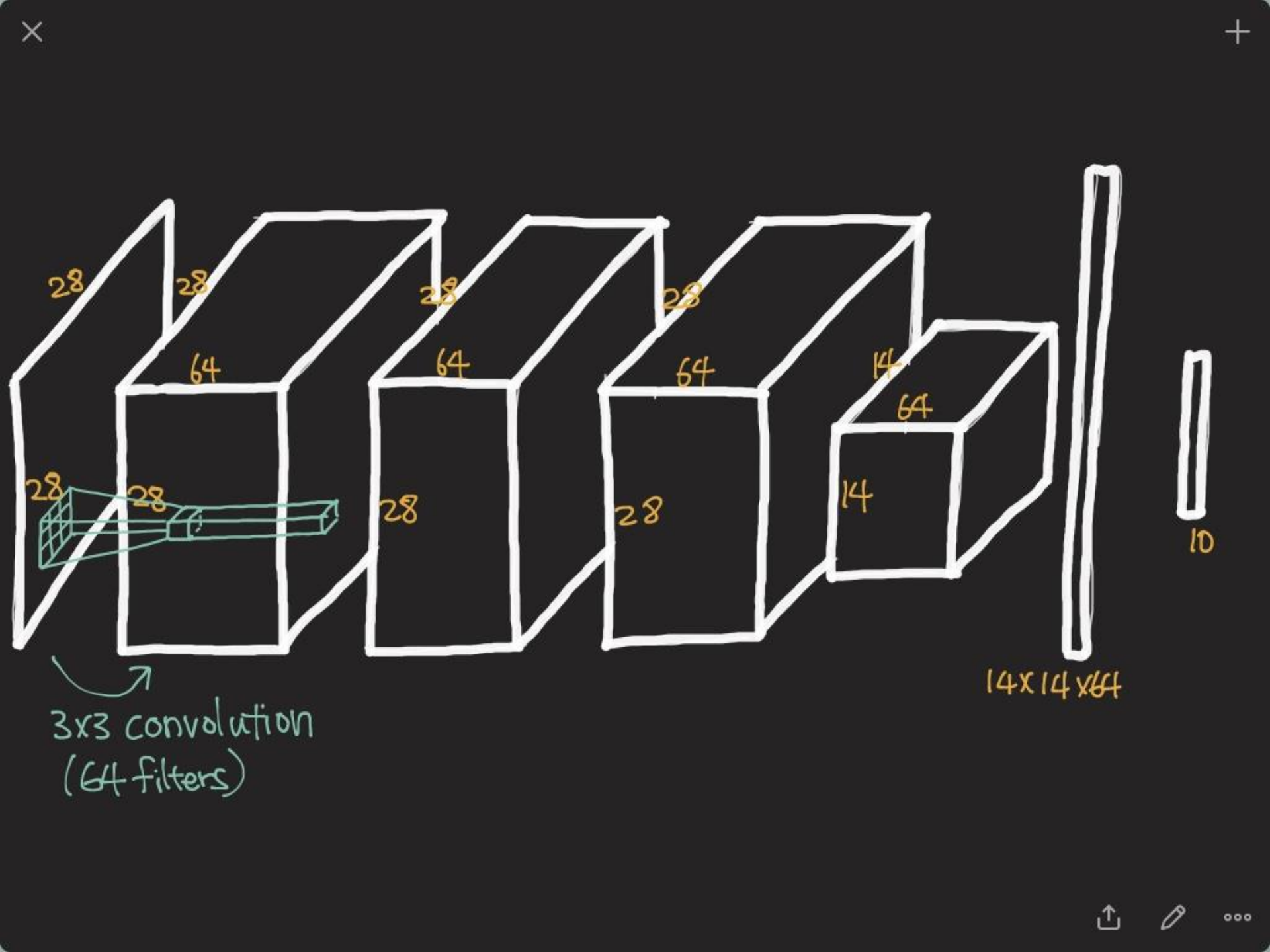
[filter_height=3, filter_width=3, in_channels=3, out_channels=7]

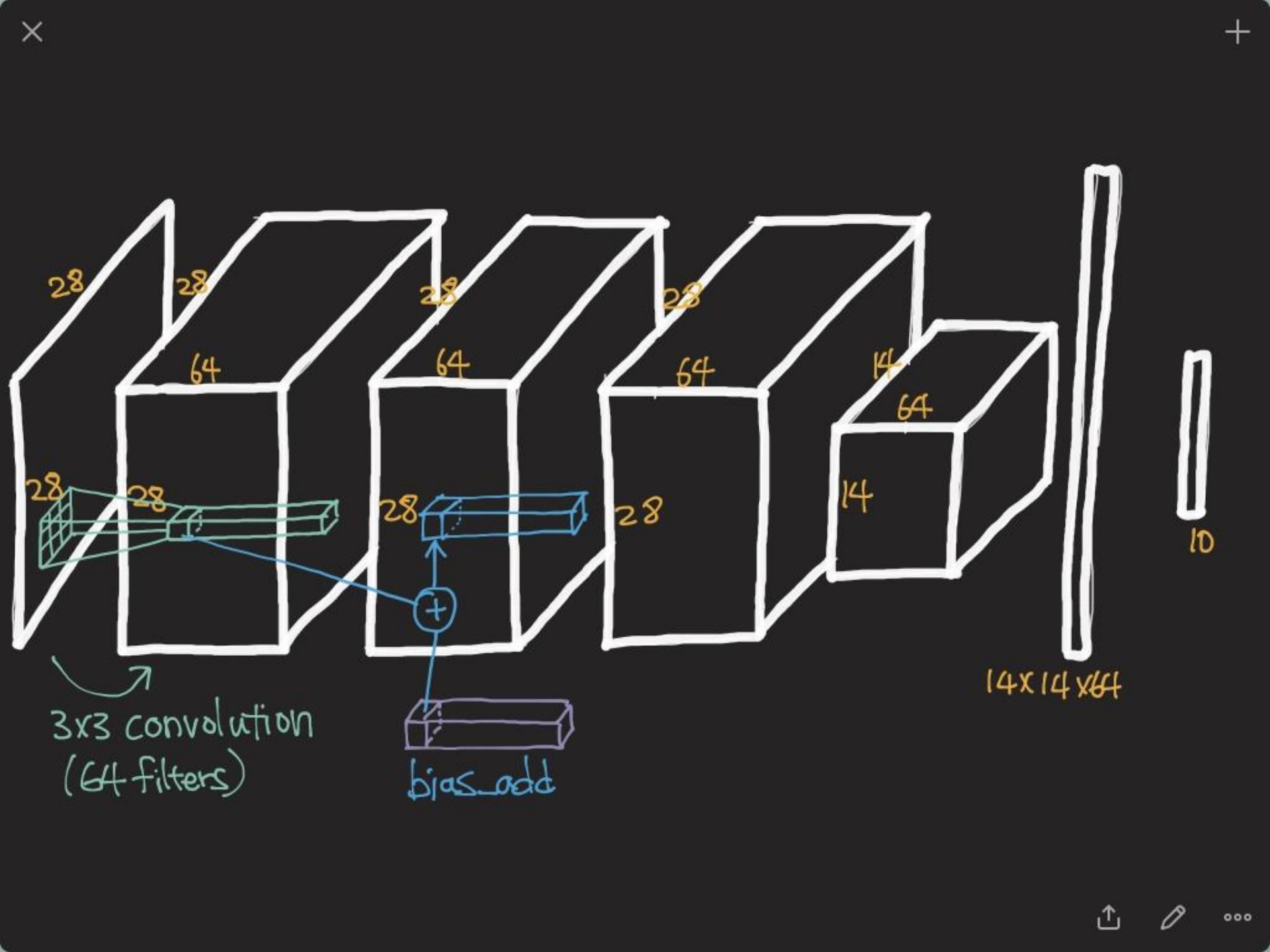
Conv2D

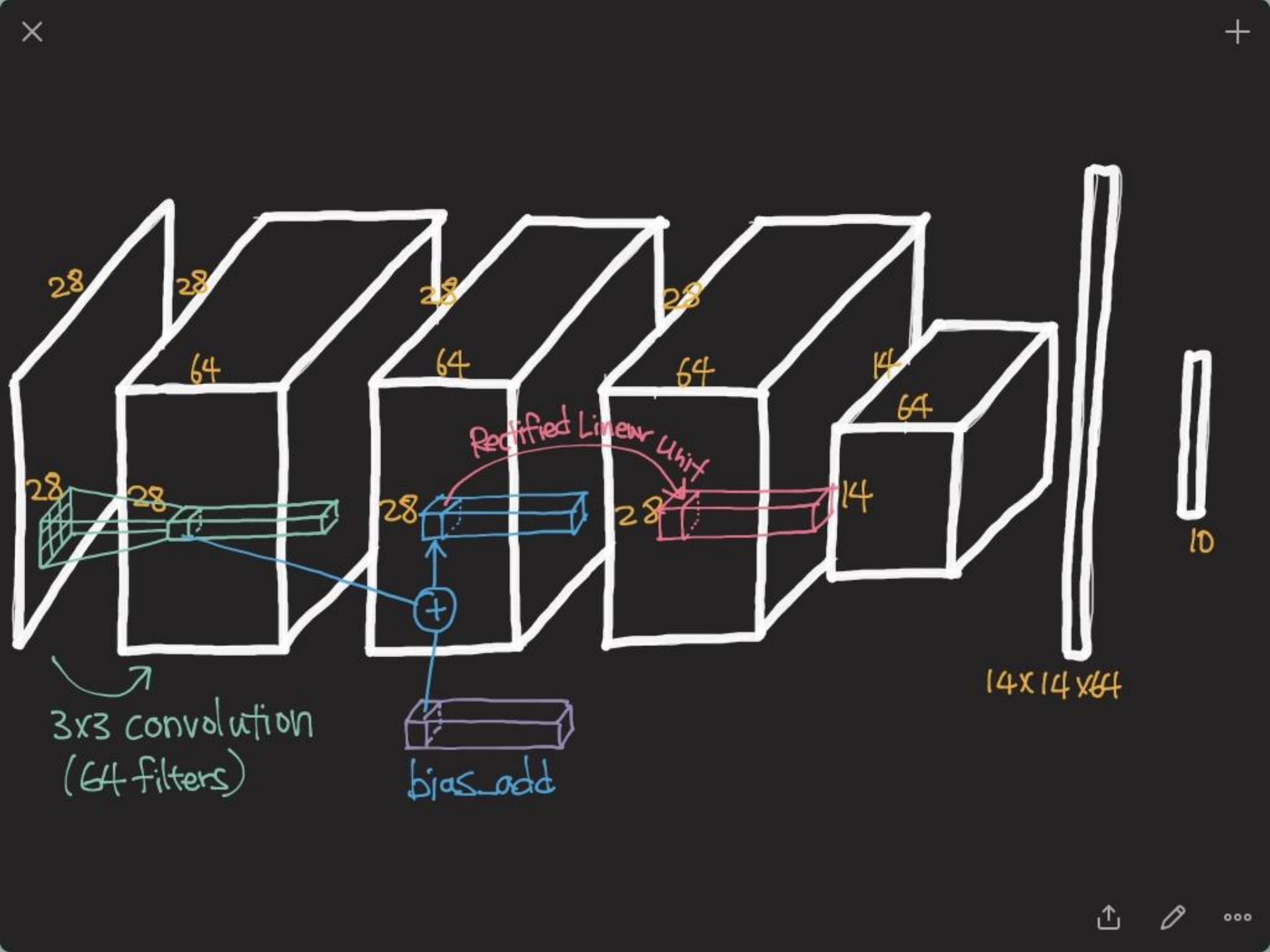


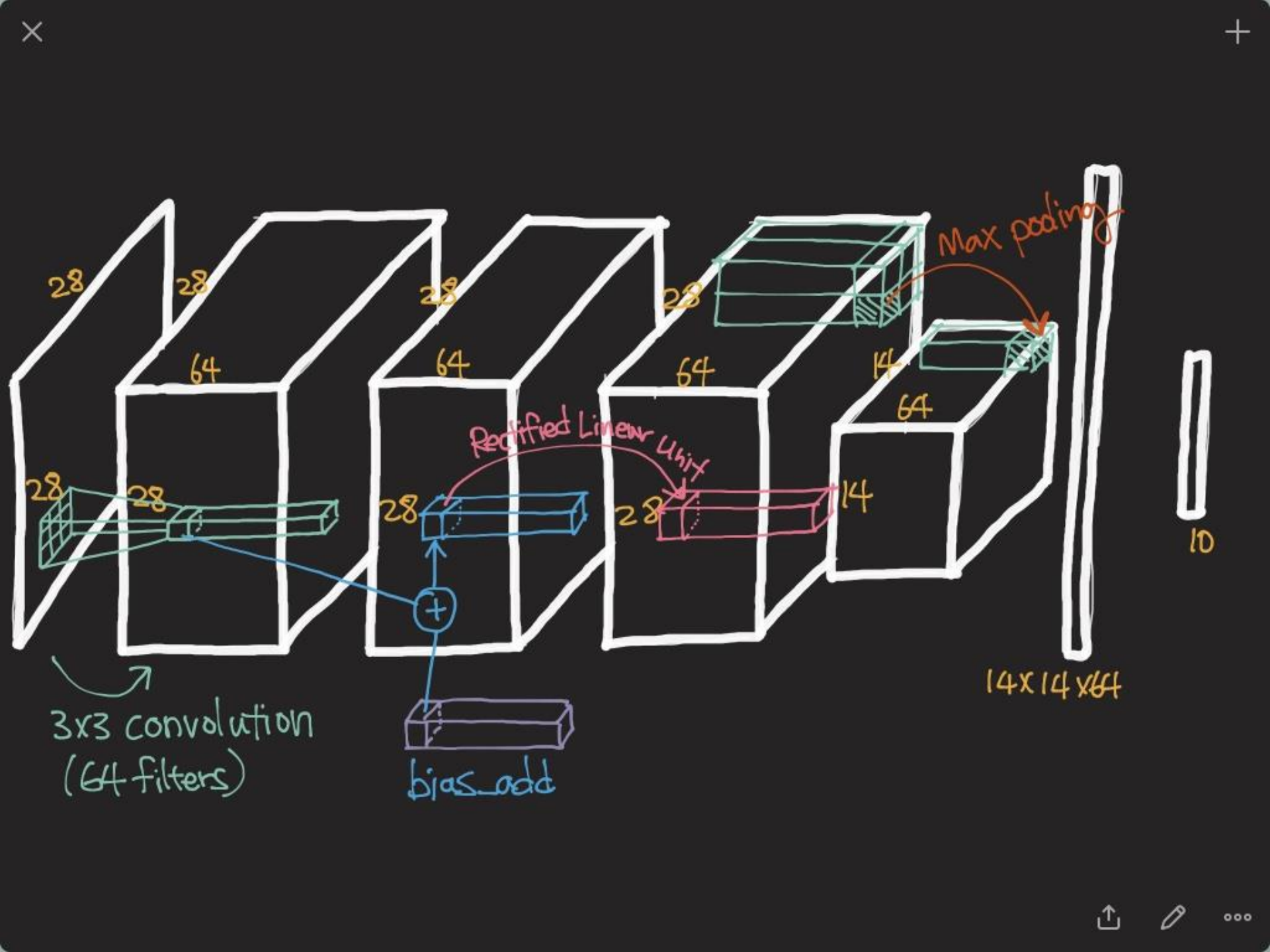
What is the **number of parameters** in this convolution layer?

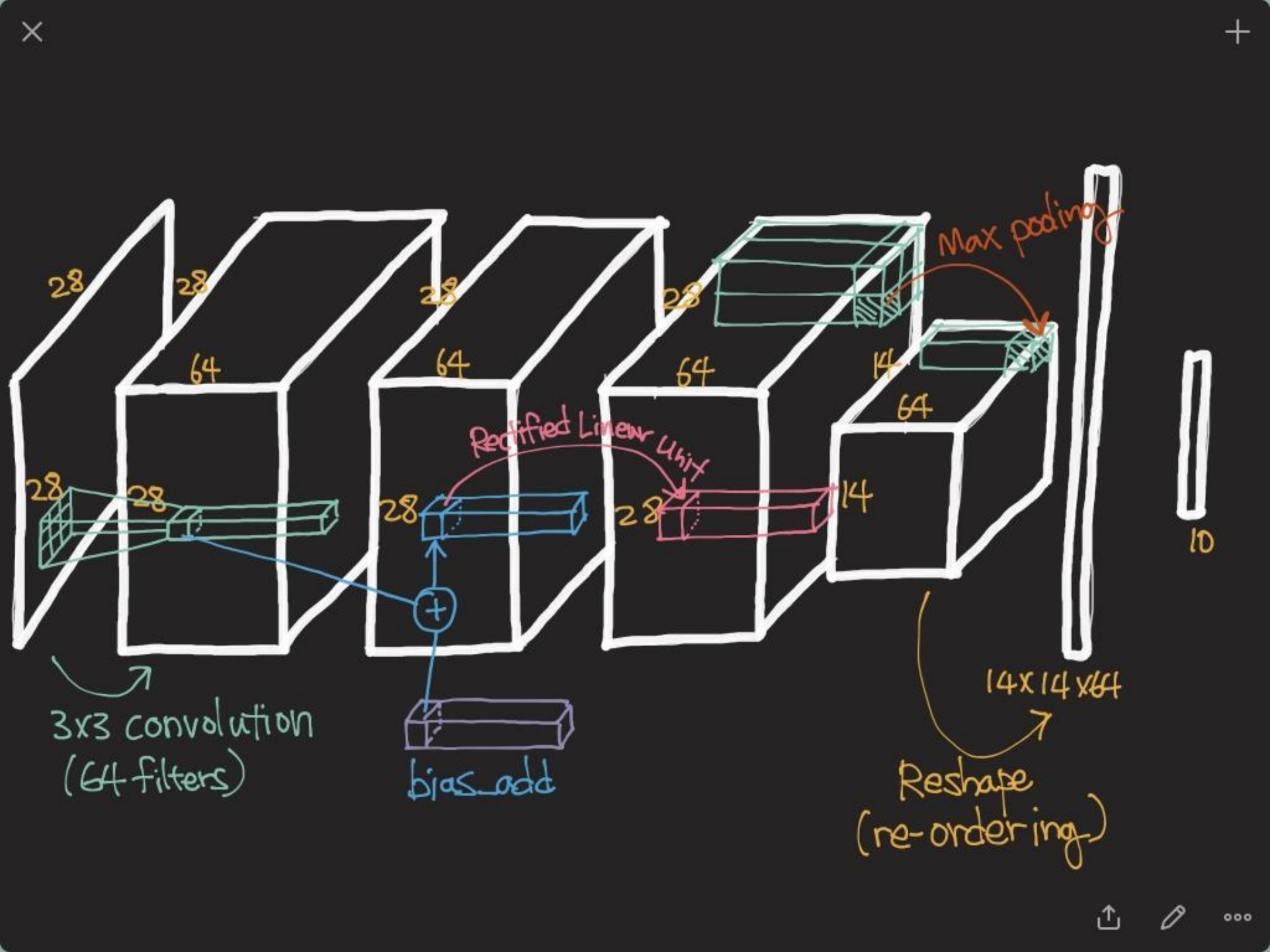
→ $189 = 3 * 3 * 3 * 7$

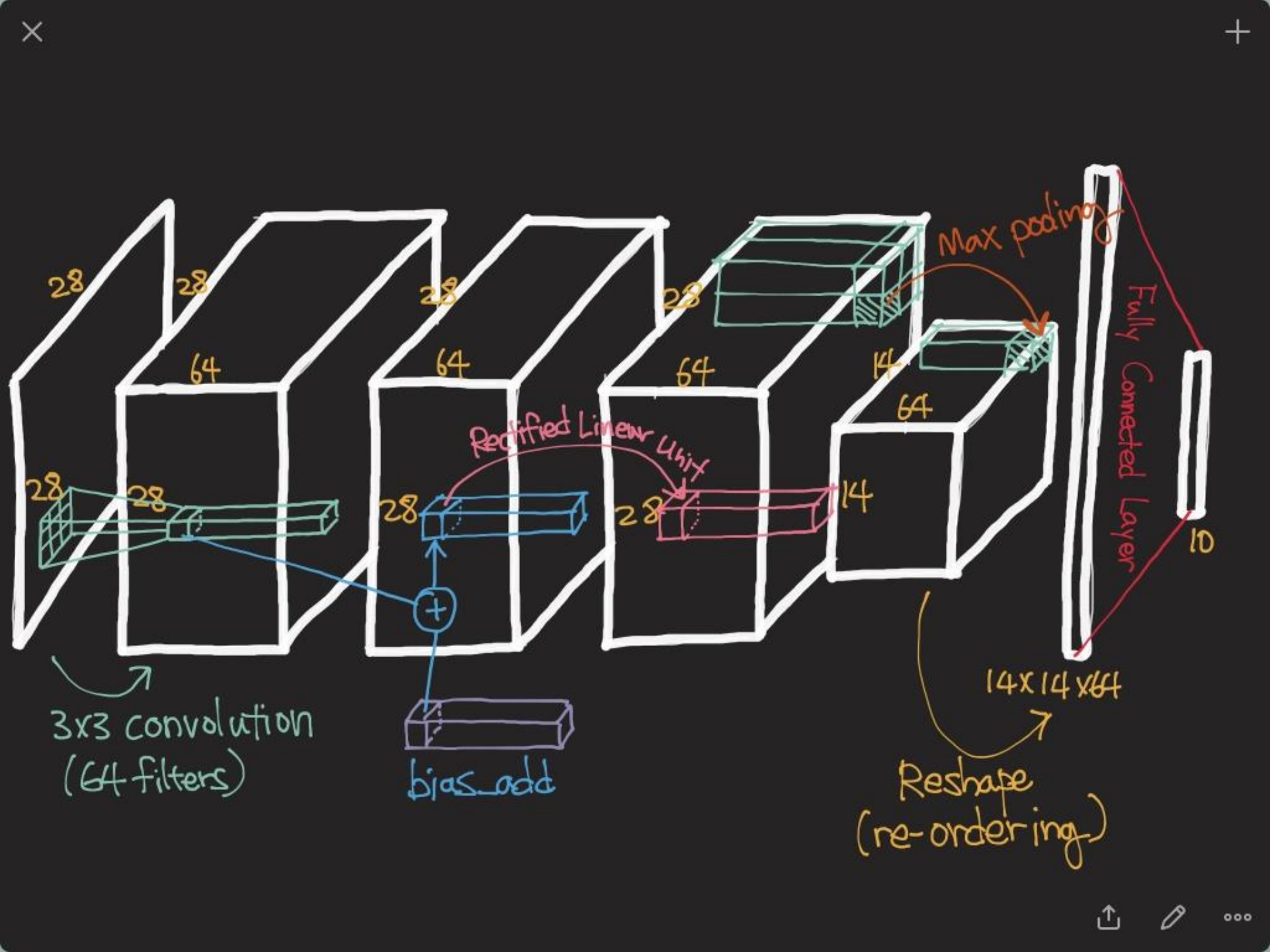

28
28
28
28
64
64
64
14
64
28
28
28
28
14
10
14x14x64
3x3 convolution
(64 filters)

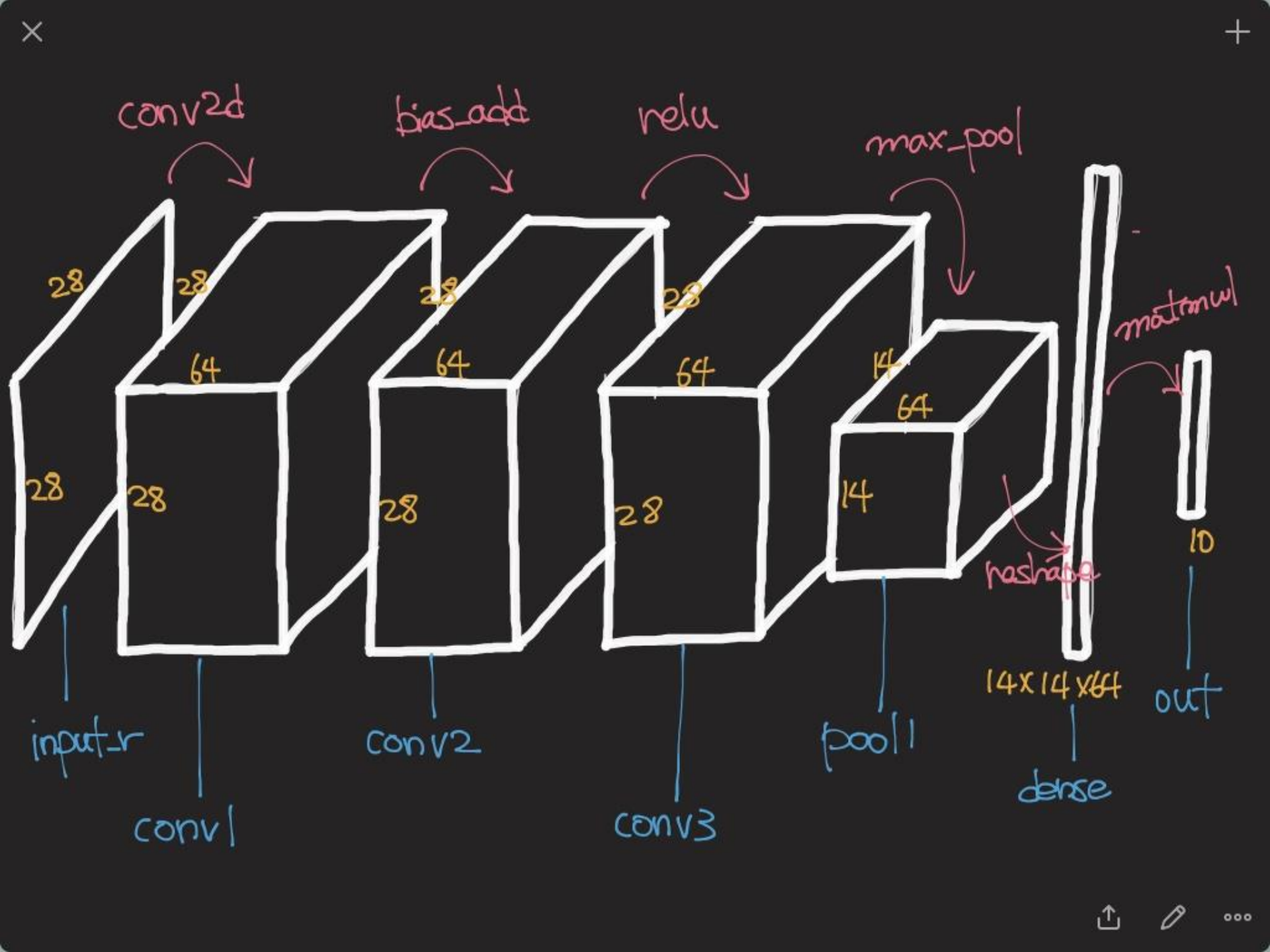

28
28
28
28
64
64
64
14
64
28
28
28
28
14
10
14x14x64
3x3 convolution
(64 filters)
bias_add

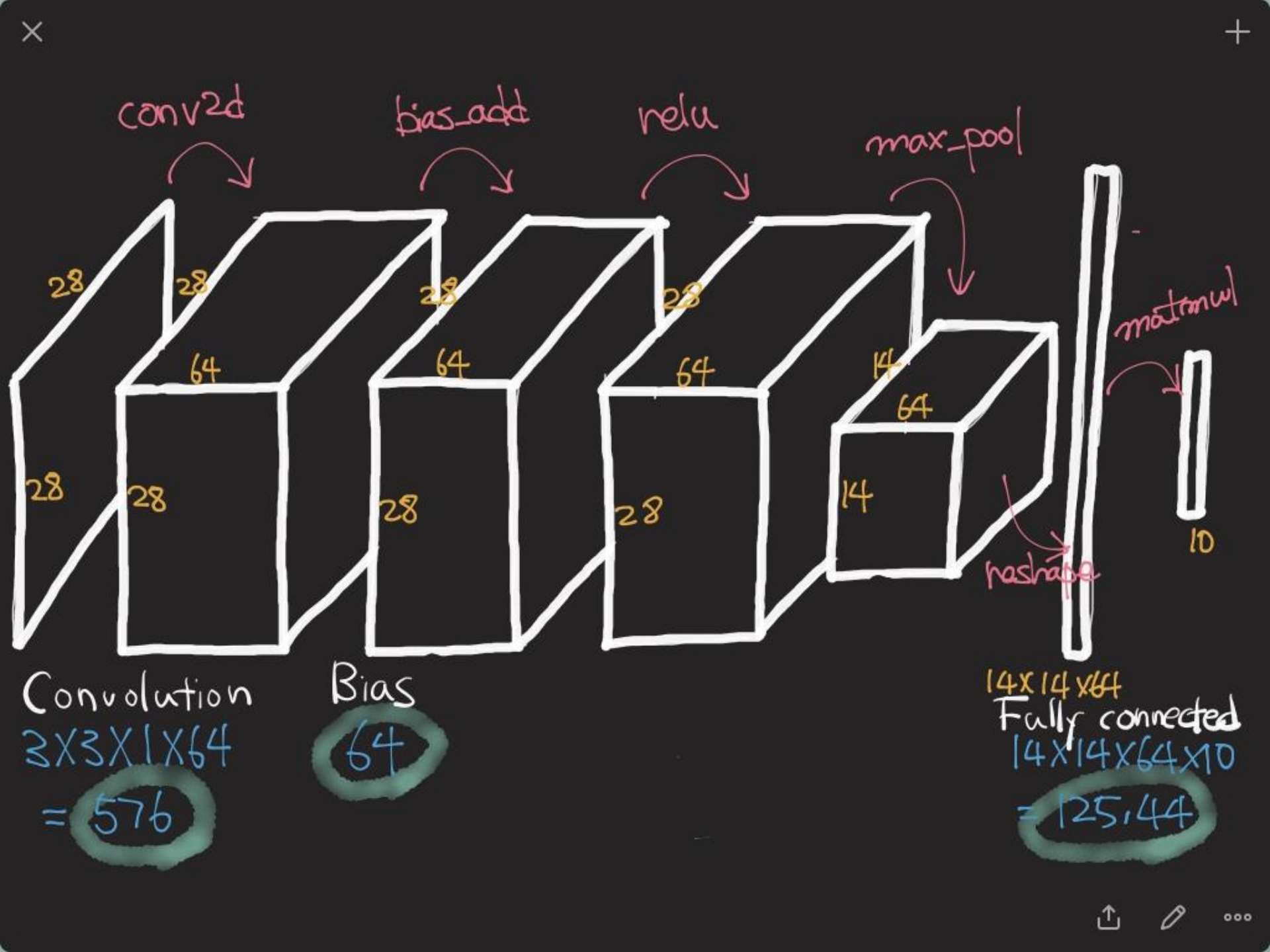

28
28
28
28
64
64
64
14
64
28
28
28
28
14
Rectified Linear Unit
10
14x14x64
3x3 convolution
(64 filters)
bias_add


28
28
28
28
64
64
64
14
64
28
28
28
28
14
Max pooling
Rectified Linear Unit
10
14x14x64
3x3 convolution
(64 filters)
bias_add


28
28
64
28
28
64
28
28
64
14
14
64
Max pooling
Rectified Linear Unit
28
28
28
28
14
10
3x3 convolution
(64 filters)
bias_add
14x14x64
Reshape
(re-ordering)


28
28
64
28
28
64
28
28
64
14
14
64
Max pooling
Rectified Linear Unit
Fully Connected Layer
10
14x14x64
3x3 convolution
(64 filters)
bias_add
Reshape
(re-ordering)


conv2d
bias_add
relu
max_pool
matmul
28
28
64
28
28
64
28
28
64
14
14
64
28
28
28
14
10
reshape
input_r
conv1
conv2
conv3
pool1
14x14x64
dense
out


conv2d
bias_add
relu
max_pool
matmul
28
28
64
28
28
28
28
64
28
28
64
14
14
64
14
reshape
10
Convolution
3X3X1X64
= 576
Bias
64
14X14X64
Fully connected
14X14X64X10
= 125,44