

# UNDERSTANDING CNN

---

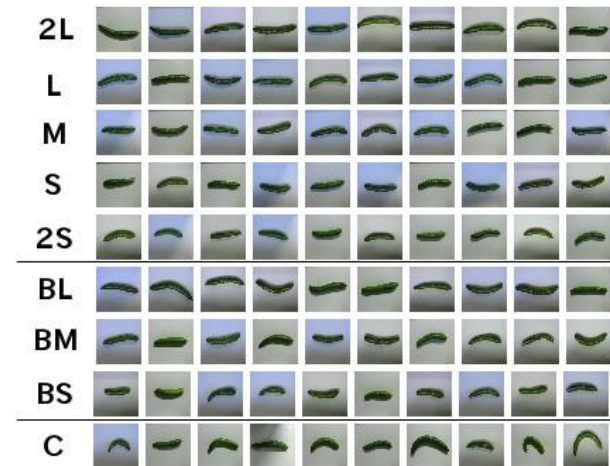
			Tasks						
			ADAS						
			Self Driving						
			Localizati on	Perception	Planning/ Control	Driver state	Vehicle Diagnosis	Smart factory	
Methods	Traditional	Non-machine Learning		GPS, SLAM		Optimal control			
		Machine-Learning based method	Supervised	MLP		Pedestrian detection (HOG+SVM)			
	CNN				Detection/ Segmentat ion/Classif ication	End-to- end Learning			
	RNN (LSTM)				Dry/wet road classificati on	End-to- end Learning			
	DNN								
	Reinforcement								
	Unsupervised								

# TENSORFLOW-POWERED CUCUMBER SORTER

---

# Cucumber sorting

- Each cucumber has different color, shape, quality and freshness.
- At Makoto's farm, they sort them into nine different classes, and his mother sorts them all herself — spending up to eight hours per day at peak harvesting times.

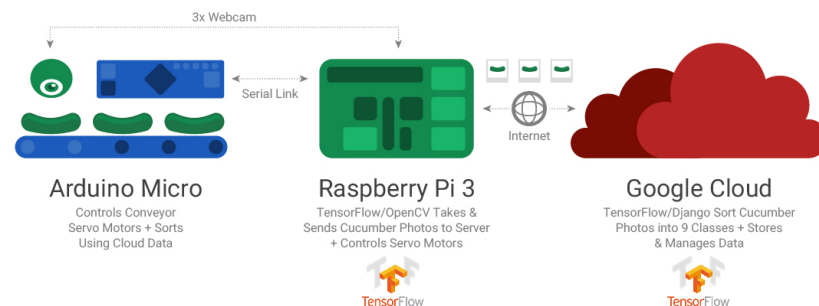


# Cucumber sorting

- You have to look at not only the size and thickness, but also the color, texture, small scratches, whether or not they are crooked and whether they have prickles. It takes months to learn the system and **you can't just hire part-time workers during the busiest period**. I myself only recently learned to sort cucumbers well,” Makoto said.
- Makoto doesn't think sorting is an essential task for cucumber farmers. "**Farmers want to focus and spend their time on growing delicious vegetables**. I'd like to automate the sorting tasks before taking the farm business over from my parents.

# Tensorflow-powered cucumber sorter

- Makoto used the sample TensorFlow code **Deep MNIST for Experts** with minor modifications to the convolution, pooling and last layers, changing the network design to adapt to the pixel format of cucumber images and the number of cucumber classes.



# Cucumber sorter by Makoto Koike



# MNIST & LENET

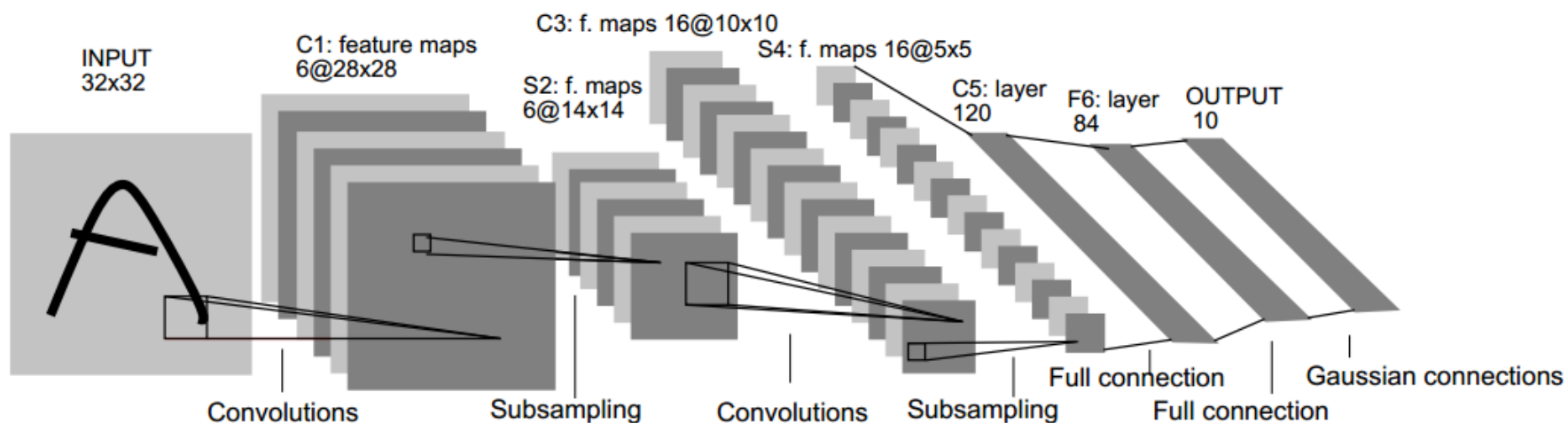
---





# LeNet

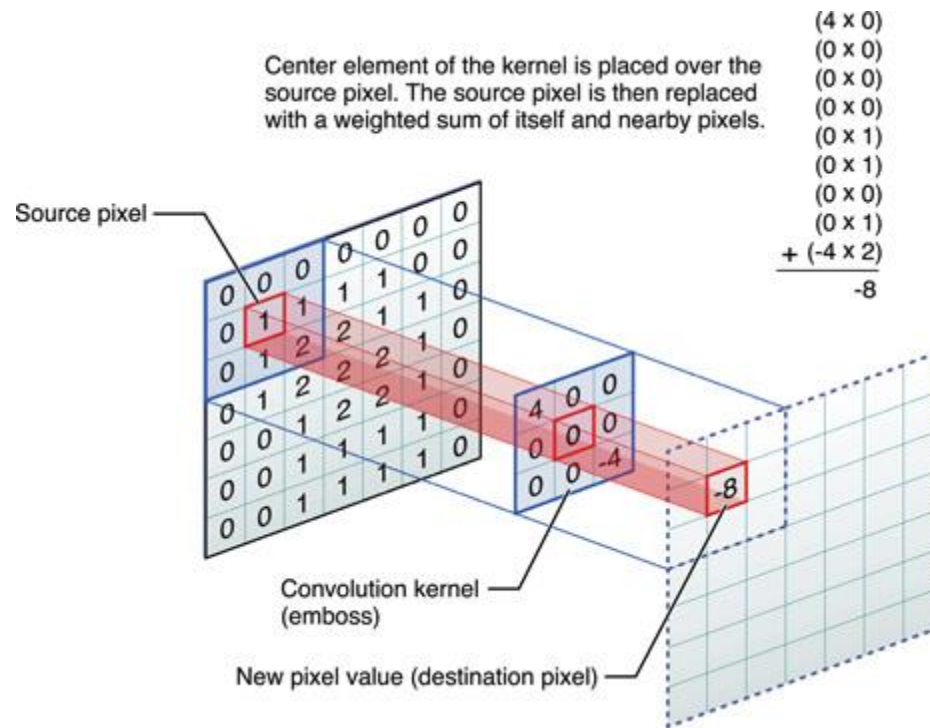
- Yann LeCun and his collaborators developed a recognizer for handwritten digits by using back-propagation in a feed-forward net



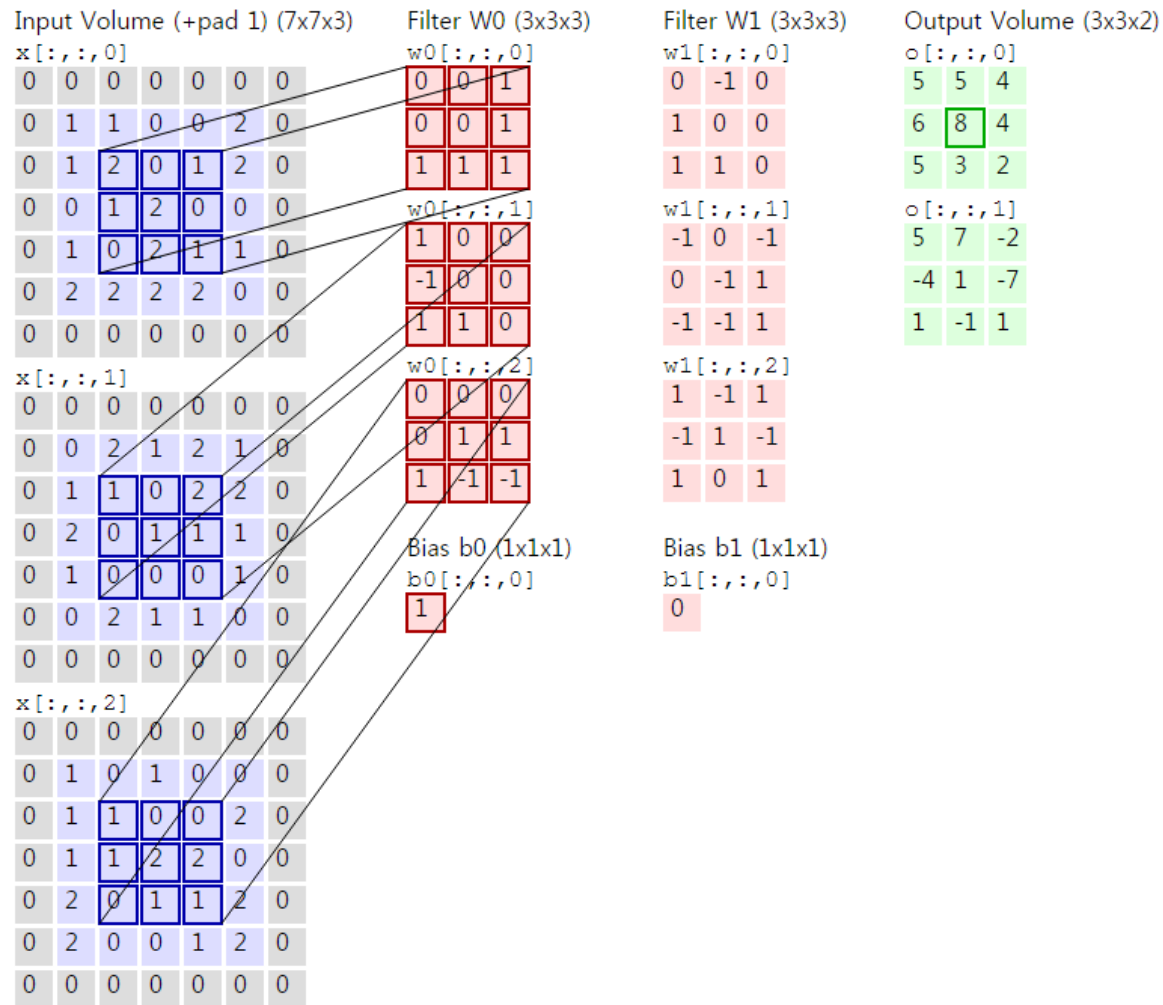
# CNN BUILDING BLOCKS

---

# Convolution

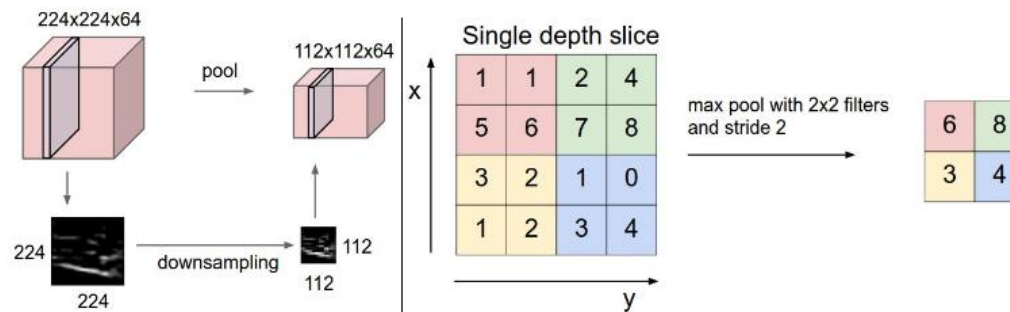


# Convolutions in CNNs



# Pooling

- Max vs Average pooling



Pooling layer downsamples the volume spatially, independently in each depth slice of the input volume. **Left:** In this example, the input volume of size  $[224 \times 224 \times 64]$  is pooled with filter size 2, stride 2 into output volume of size  $[112 \times 112 \times 64]$ . Notice that the volume depth is preserved. **Right:** The most common downsampling operation is max, giving rise to **max pooling**, here shown with a stride of 2. That is, each max is taken over 4 numbers (little  $2 \times 2$  square).

# DEEP MNIST FOR EXPERTS

---

# Deep MNIST for Experts

```
import tensorflow as tf
from tensorflow.examples.tutorials.mnist import input_data
mnist = input_data.read_data_sets('MNIST_data', one_hot=True)

def weight_variable(shape):
    # tf.truncated_normal: Outputs random values from a truncated normal distribution.
    # values whose magnitude is more than 2 standard deviations from the mean are dropped and re-picked.
    initial = tf.truncated_normal(shape, stddev=0.1)
    return tf.Variable(initial)

def bias_variable(shape):
    initial = tf.constant(0.1, shape=shape)
    return tf.Variable(initial)

# convolution & max pooling
def conv2d(x, W):
    return tf.nn.conv2d(x, W, strides=[1, 1, 1, 1], padding='SAME')

def max_pool_2x2(x):
    return tf.nn.max_pool(x, ksize=[1, 2, 2, 1], strides=[1, 2, 2, 1], padding='SAME')

x = tf.placeholder(tf.float32, [None, 784])
y_ = tf.placeholder(tf.float32, [None, 10])

# [5, 5, 1, 32]: 5x5 convolution patch, 1 input channel, 32 output channel.
W_conv1 = weight_variable([5, 5, 1, 32])
b_conv1 = bias_variable([32])

x_image = tf.reshape(x, [-1,28,28,1])

# convolution, relu, max pooling
h_conv1 = tf.nn.relu(conv2d(x_image, W_conv1) + b_conv1)
h_pool1 = max_pool_2x2(h_conv1)

# [5, 5, 32, 64]: 5x5 convolution patch, 32 input channel, 64 output channel.
W_conv2 = weight_variable([5, 5, 32, 64])
b_conv2 = bias_variable([64])

# convolution, relu, max pooling
h_conv2 = tf.nn.relu(conv2d(h_pool1, W_conv2) + b_conv2)
h_pool2 = max_pool_2x2(h_conv2)

# fc layer 1
W_fc1 = weight_variable([7 * 7 * 64, 1024])
b_fc1 = bias_variable([1024])
```



# Deep MNIST for Experts

```
h_pool2_flat = tf.reshape(h_pool2, [-1, 7*7*64])
h_fc1 = tf.nn.relu(tf.matmul(h_pool2_flat, W_fc1) + b_fc1)

keep_prob = tf.placeholder("float")
h_fc1_drop = tf.nn.dropout(h_fc1, keep_prob)

# fc layer 2
W_fc2 = weight_variable([1024, 10])
b_fc2 = bias_variable([10])

y_conv = tf.nn.softmax(tf.matmul(h_fc1_drop, W_fc2) + b_fc2)

cross_entropy = -tf.reduce_sum(y * tf.log(y_conv))
train_step = tf.train.AdamOptimizer(1e-4).minimize(cross_entropy)
correct_prediction = tf.equal(tf.argmax(y_conv, 1), tf.argmax(y_, 1))
accuracy = tf.reduce_mean(tf.cast(correct_prediction, "float"))
init = tf.initialize_all_variables()

ce_sum = tf.scalar_summary("cross entropy", cross_entropy)
acc_sum = tf.scalar_summary("accuracy", accuracy)
merged = tf.merge_summary([ce_sum, acc_sum])

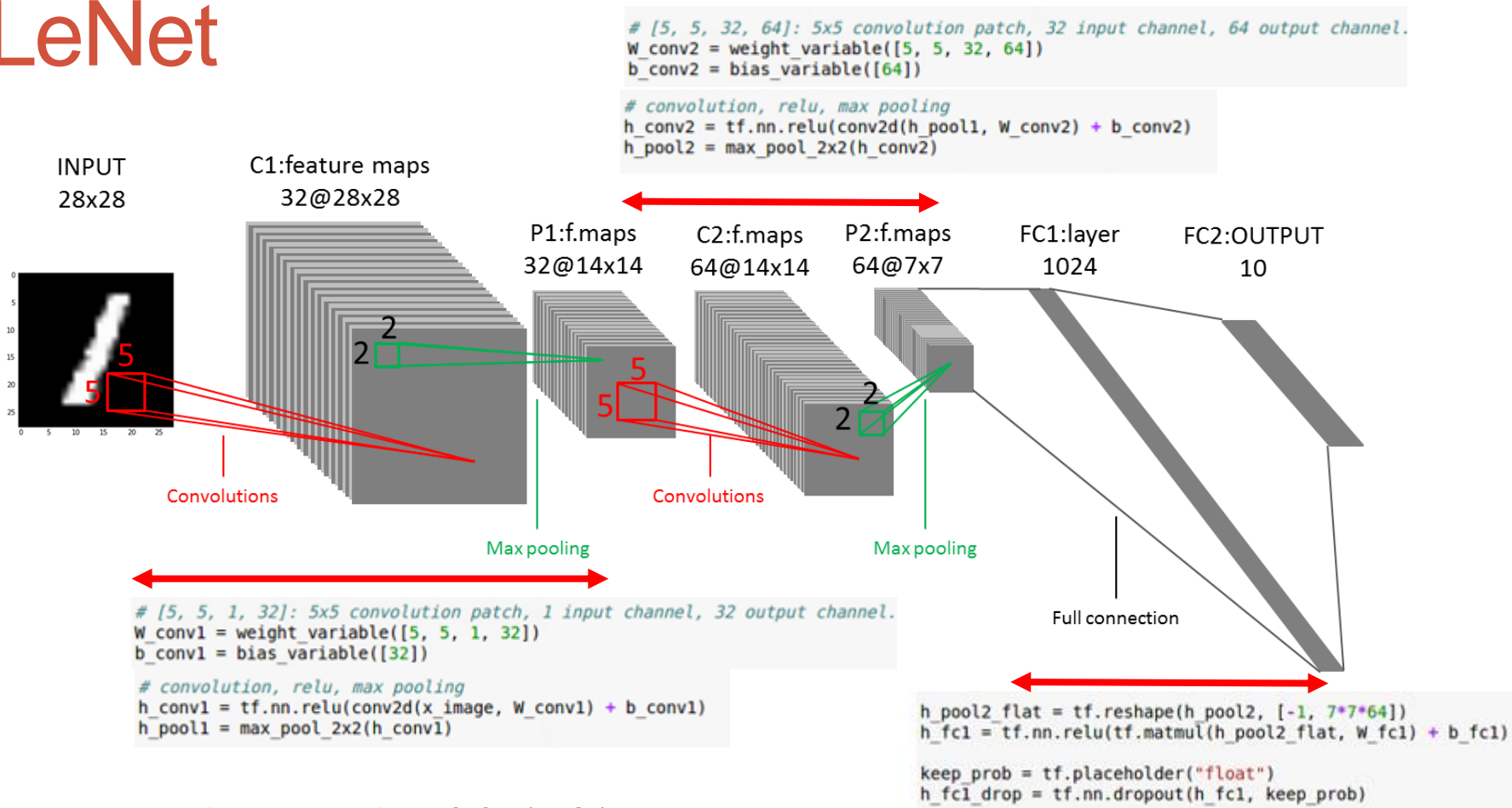
sess = tf.Session()
writer = tf.train.SummaryWriter("./sumlog", sess.graph)

sess.run(init)

with sess.as_default():
    for i in range(20000):
        batch = mnist.train.next_batch(50)
        if i%400 == 0:
            train_accuracy = accuracy.eval(feed_dict={x:batch[0], y_: batch[1], keep_prob: 1.0})
            print "step %d, training accuracy %g" % (i, train_accuracy)
            print "test accuracy %g" % accuracy.eval(feed_dict={x: mnist.test.images, y_: mnist.test.labels,
                                                                keep_prob: 1.0})
            writer.add_summary(merged.eval(feed_dict={x: mnist.test.images, y_: mnist.test.labels,
                                                                keep_prob: 1.0}), i)

        train_step.run(feed_dict={x: batch[0], y_: batch[1], keep_prob: 0.5})
```

# LeNet



#(Parameter) = 3,274,634

Layer	C1	C2	FC1	FC2
Weight	800	51,200	3,211,264	10,240
Bias	32	64	1,024	10

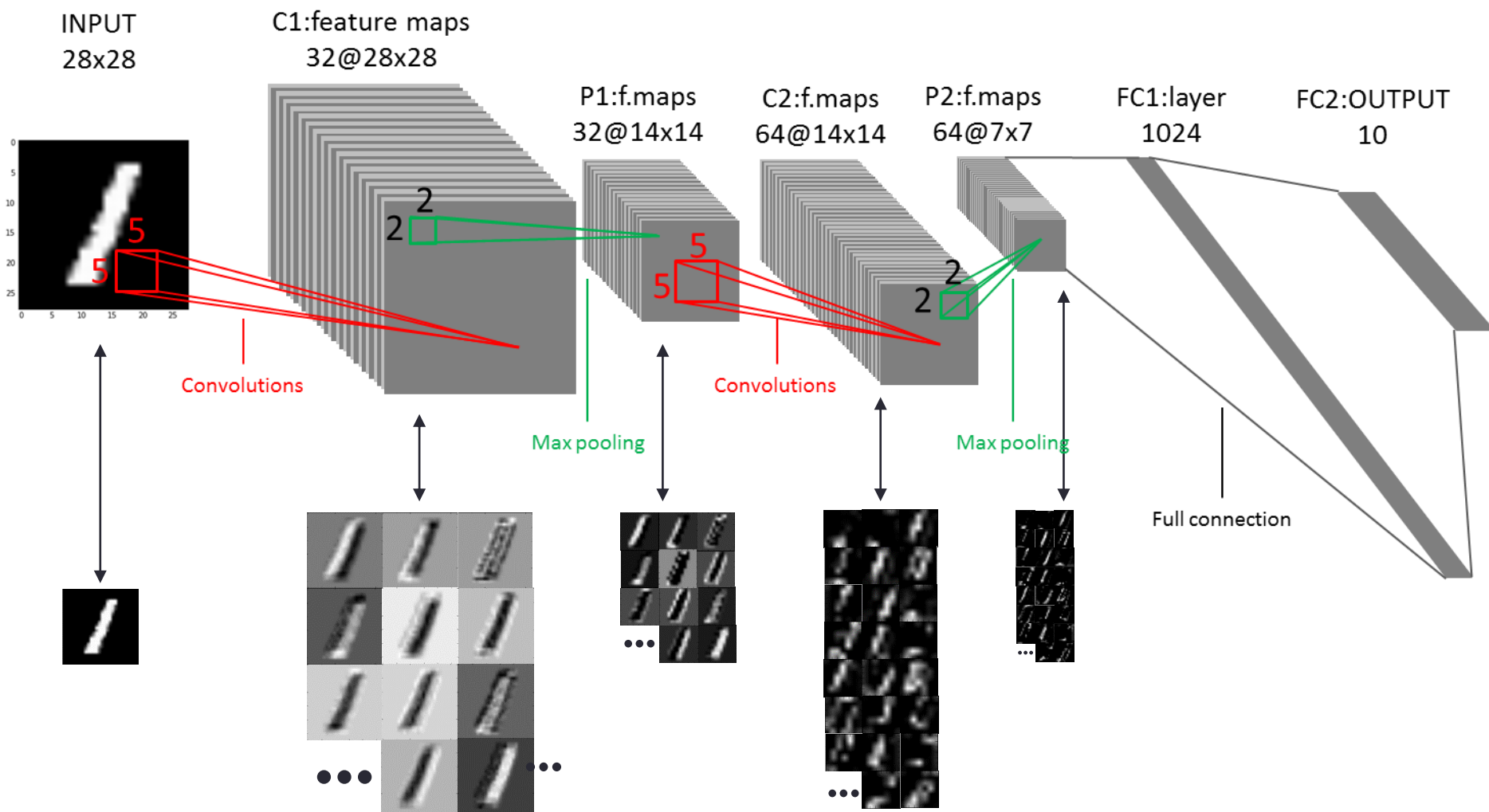
# The 82 errors by LeNet5



Notice that most of the errors are cases that people find quite easy.

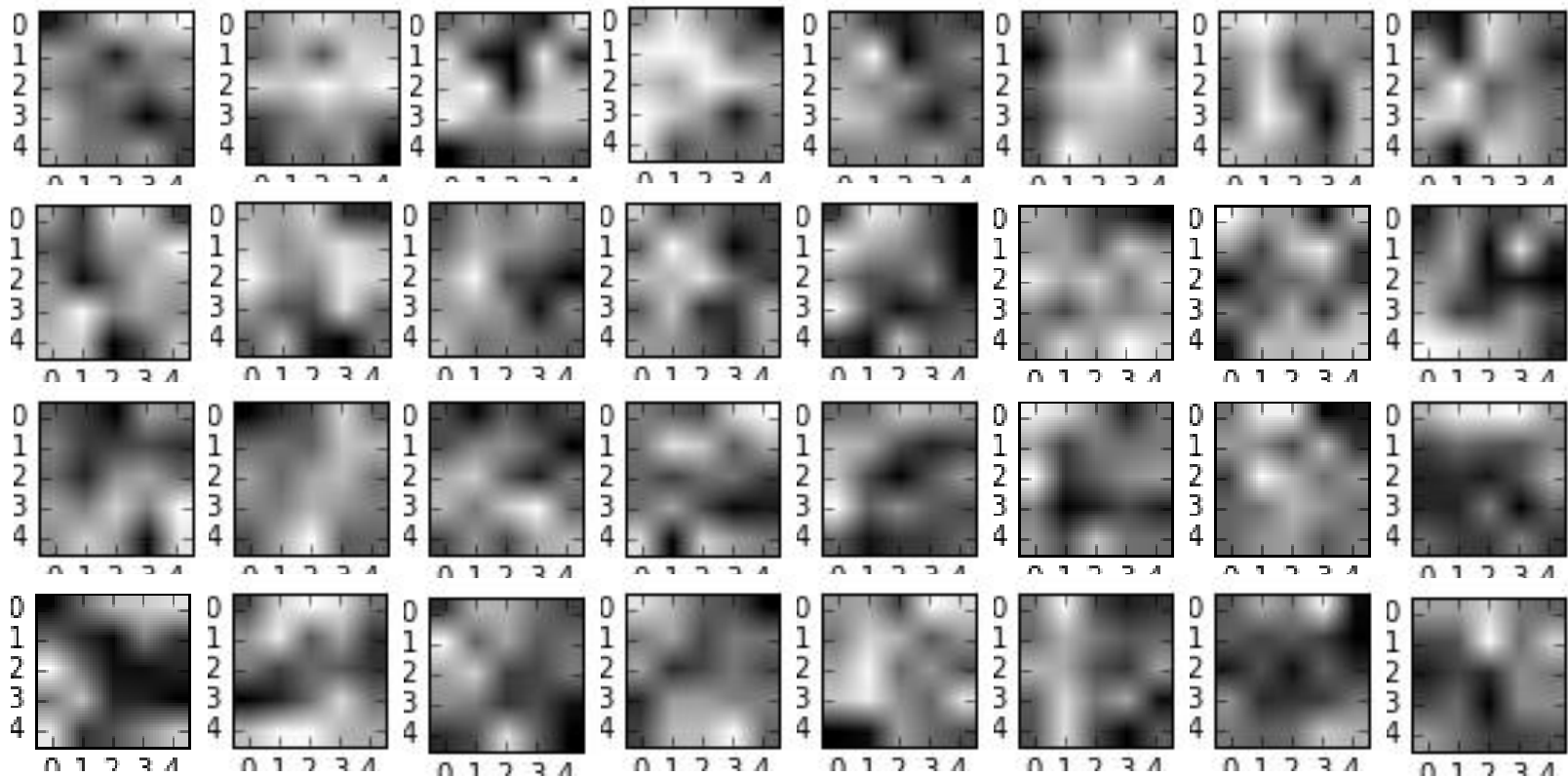
The human error rate is probably 20 to 30 errors but nobody has had the patience to measure it.

# Feature map results

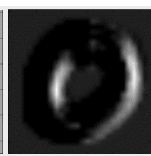
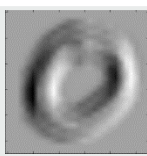
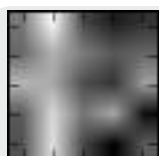


# Learned Filters

Trained 32 filters on C1 layer

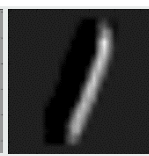
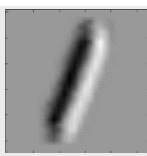


# Learned Filters



Filtered  
result

ReLU



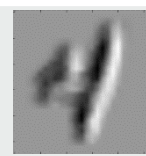
Filtered  
result

ReLU



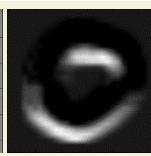
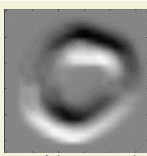
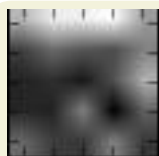
Filtered  
result

ReLU



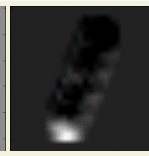
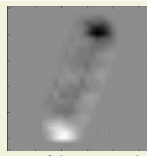
Filtered  
result

ReLU



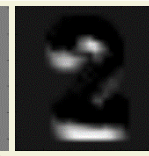
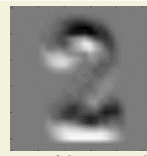
Filtered  
result

ReLU



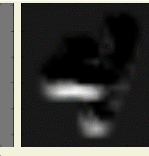
Filtered  
result

ReLU



Filtered  
result

ReLU



Filtered  
result

ReLU

# TensorFlow codes

- [LeNet tensorflow codes](#)

# IMAGE CLASSIFICATION

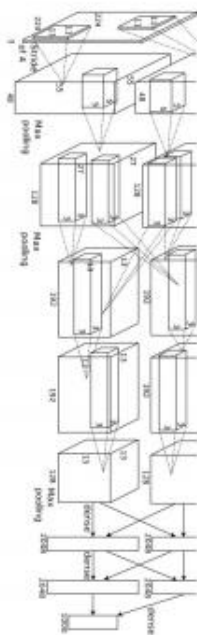
---



# Image Classification (ImageNet)

**Year 2012**

SuperVision



[Krizhevsky NIPS 2012]

**Year 2014**

GoogLeNet

VGG



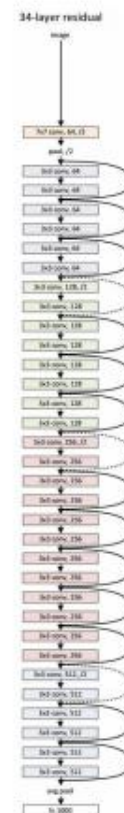
[Szegedy arxiv 2014]



[Simonyan arxiv 2014]

**Year 2015**

MSRA

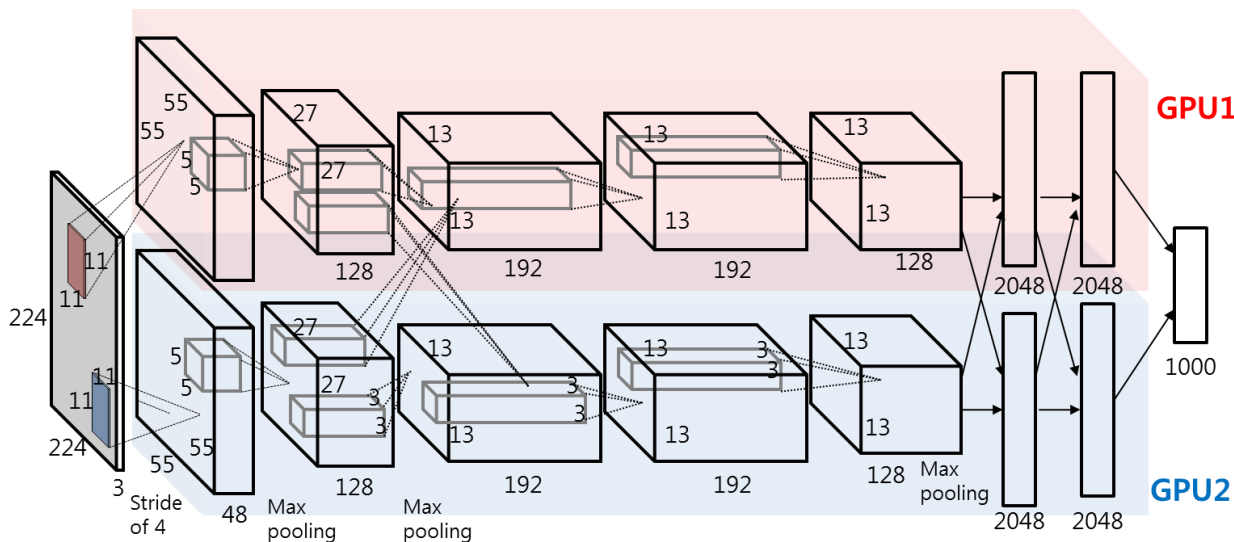


ALEXNET

---

# AlexNet

- AlexNet: won the 2012 ImageNet competition by making 40% less error than the next best competitor
  - It is composed of 5 convolutional layers
  - The input is a color RGB image
  - Computation is divided over 2 GPU architectures



# AlexNet results

- [AlexNet TensorFlow codes and some results](#)



# AlexNet Visualization

- Filters learned by the first convolutional layer. The top half corresponds to the layer on one GPU, the bottom on the other. From Krizhevsky et al. (2012)
- Each of the 96 filters is of size  $[11 \times 11 \times 3]$



# VISUALIZATION

---

# Motivation

- It is well known that Artificial Neural Networks show **remarkable performance** in image classification
- However, **we actually understand little** of why certain models work and others don't
- There have been some attempts to visualize at each layer in the neural network
  - to know “how neural networks work and what each layer has learned”

# Why is this important?

- There is a need of training networks with information we want to learn

But this program couldn't ignore what we don't care about

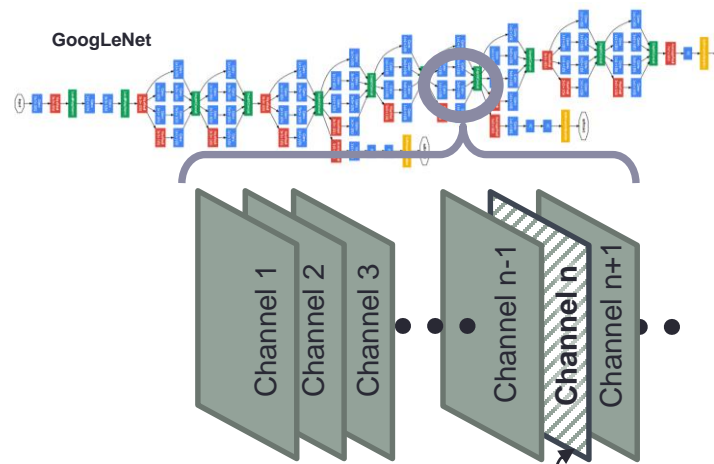




# Visualization method

- Deconvolution
  - Matthew D. Zeiler and Rob Fergus, “Visualizing and Understanding Convolutional Networks,” ECCV 2014
- Input optimization
  - Naïve visualization
  - Low/High frequency normalization
    - With image prior
    - With Laplacian (pyramid gradient) normalization

# Naïve visualization



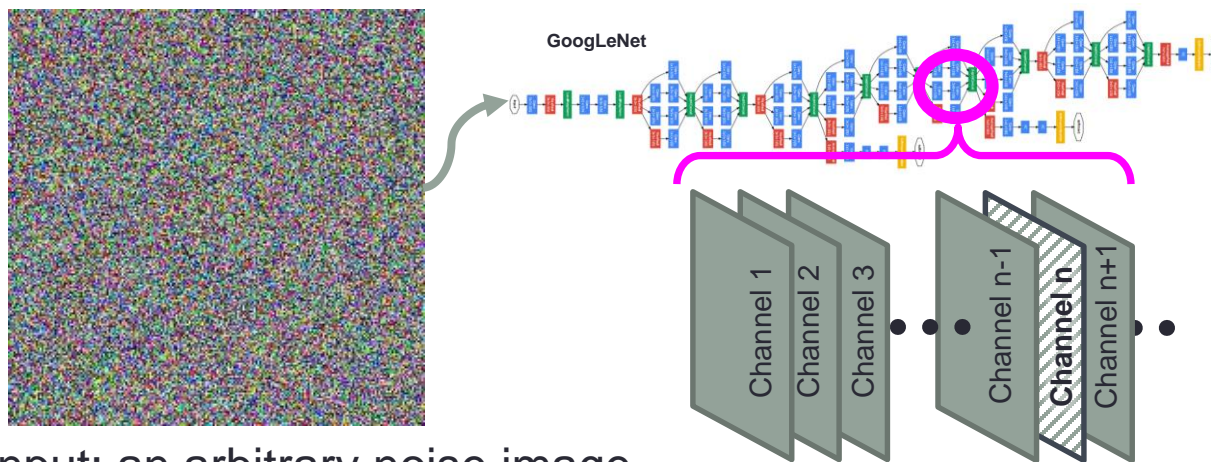
$T$ : Selected layer and channel

Objective function:  $L = \text{mean}(T)$

*GRADIENT ASCENT:*

$$img_{new} \leftarrow img_{old} + \alpha \times \left. \frac{\partial(L)}{\partial(img)} \right|_{img_{old}}$$

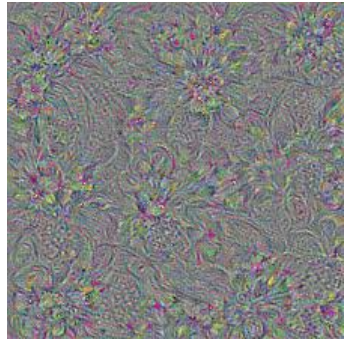
# Naïve visualization



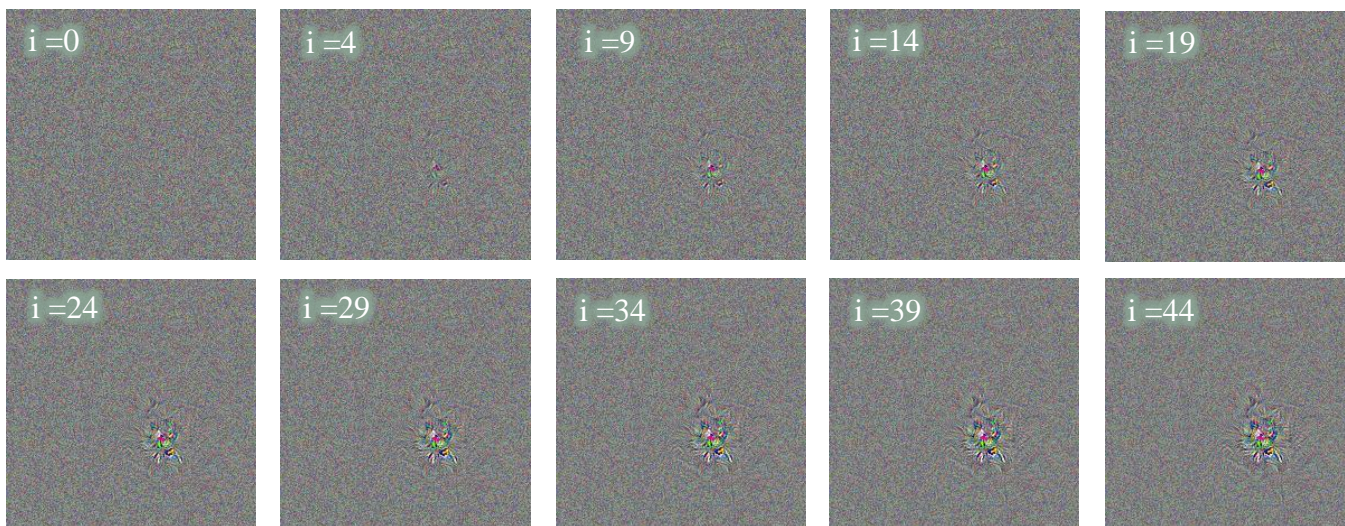
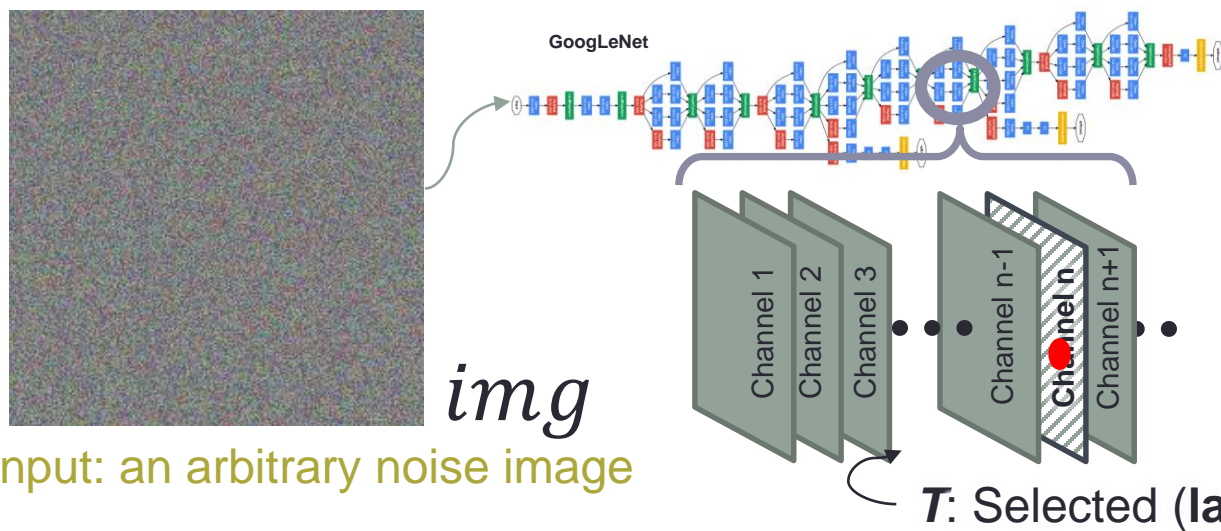
Initial input: an arbitrary noise image

Selected (layer, channel)

The block contains three square images showing the evolution of patterns. The first image is labeled  $L=34.99$  and  $i=0$ . The second image is labeled  $L=470.40$  and  $i=9$ . The third image is labeled  $L=773.42$  and  $i=19$ . Below these images is a large gray arrow pointing to the right, labeled *GRADIENT ASCENT*. The text 'Output images' is written in yellow below the third image.

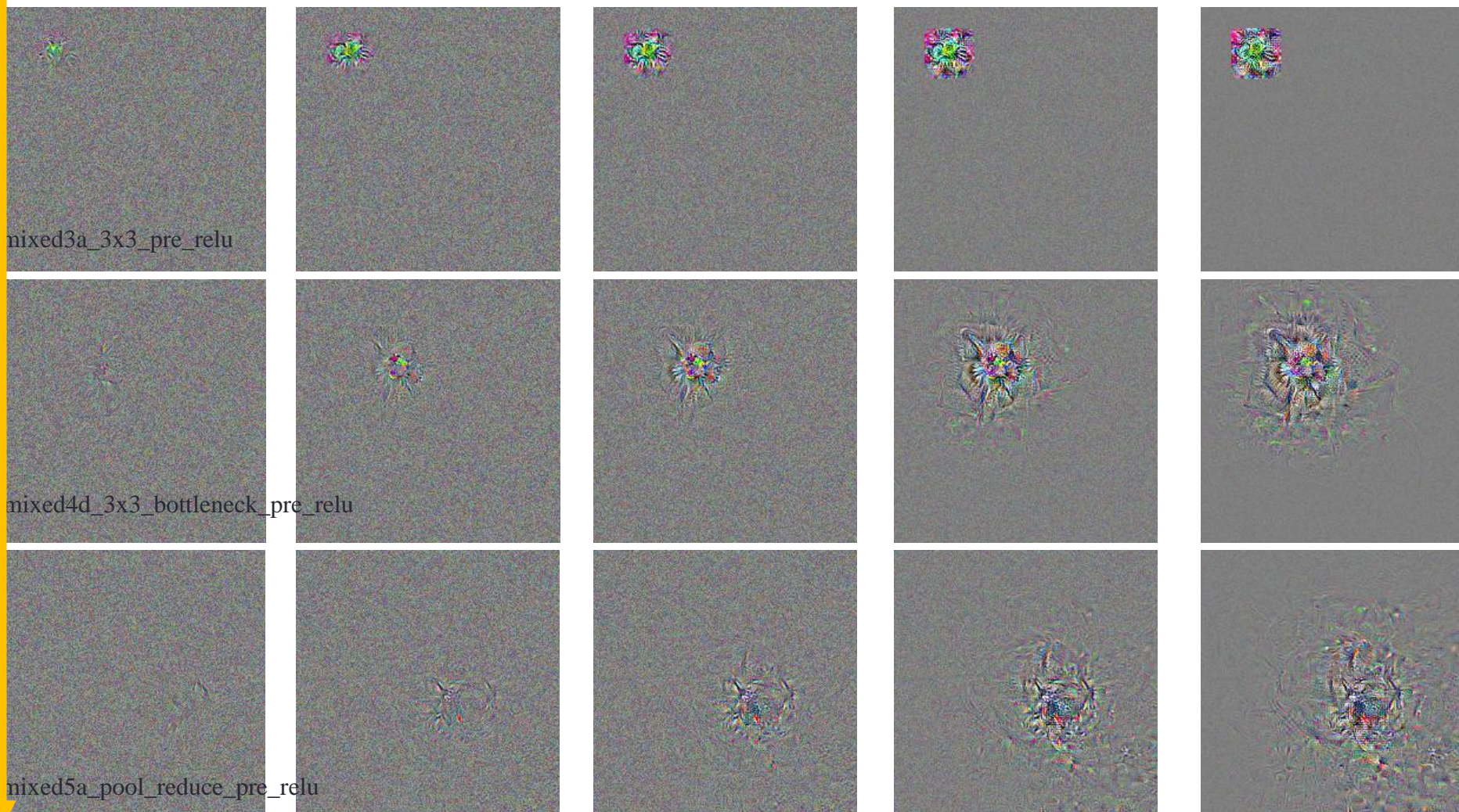


# Single neuron activation



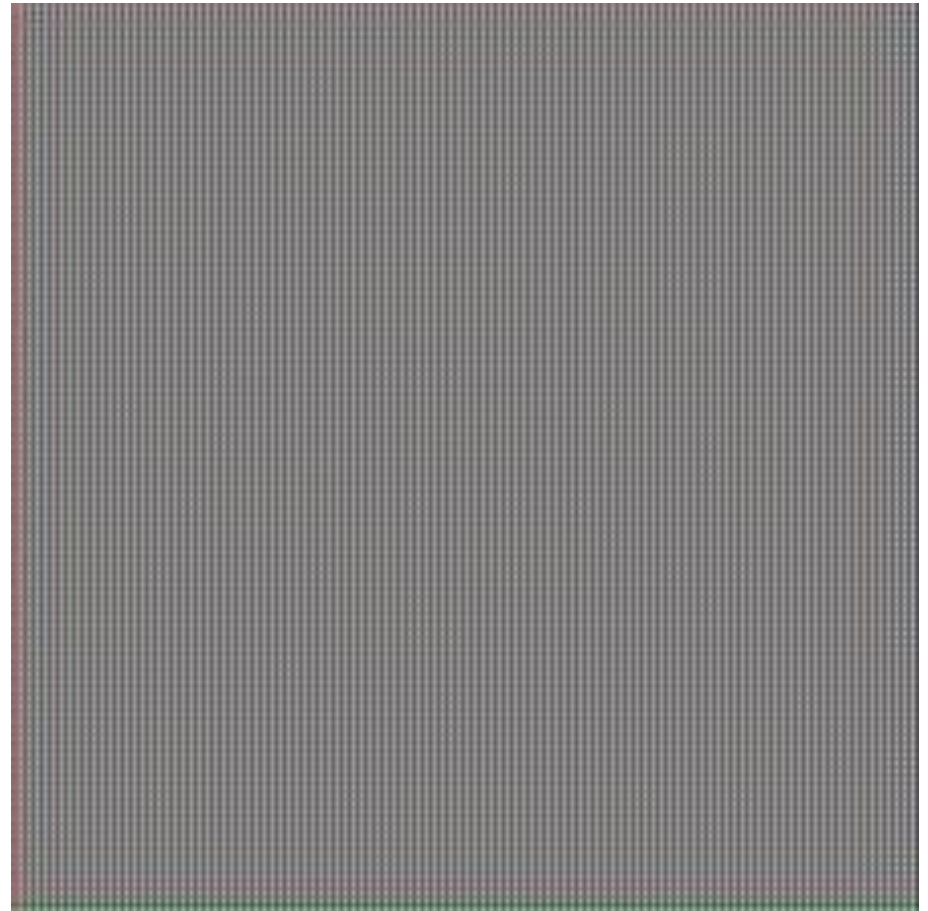
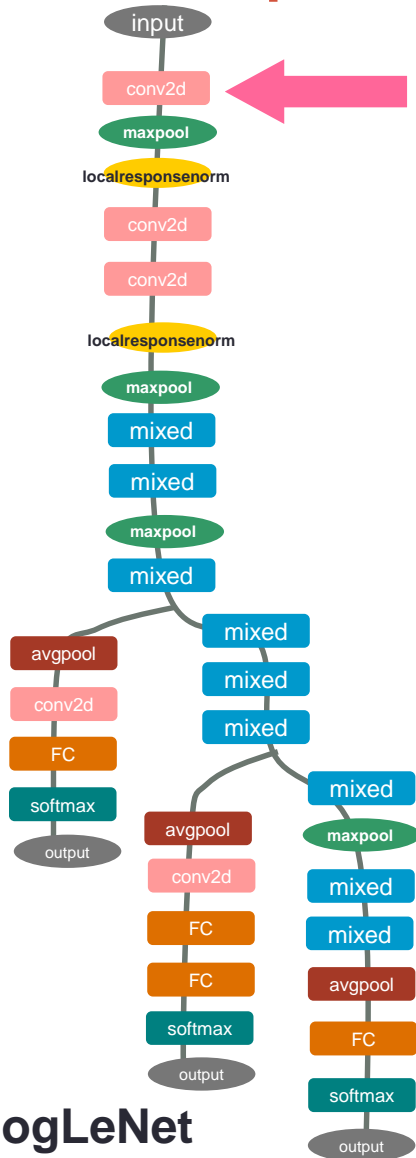
# Single neuron activation results

iteration

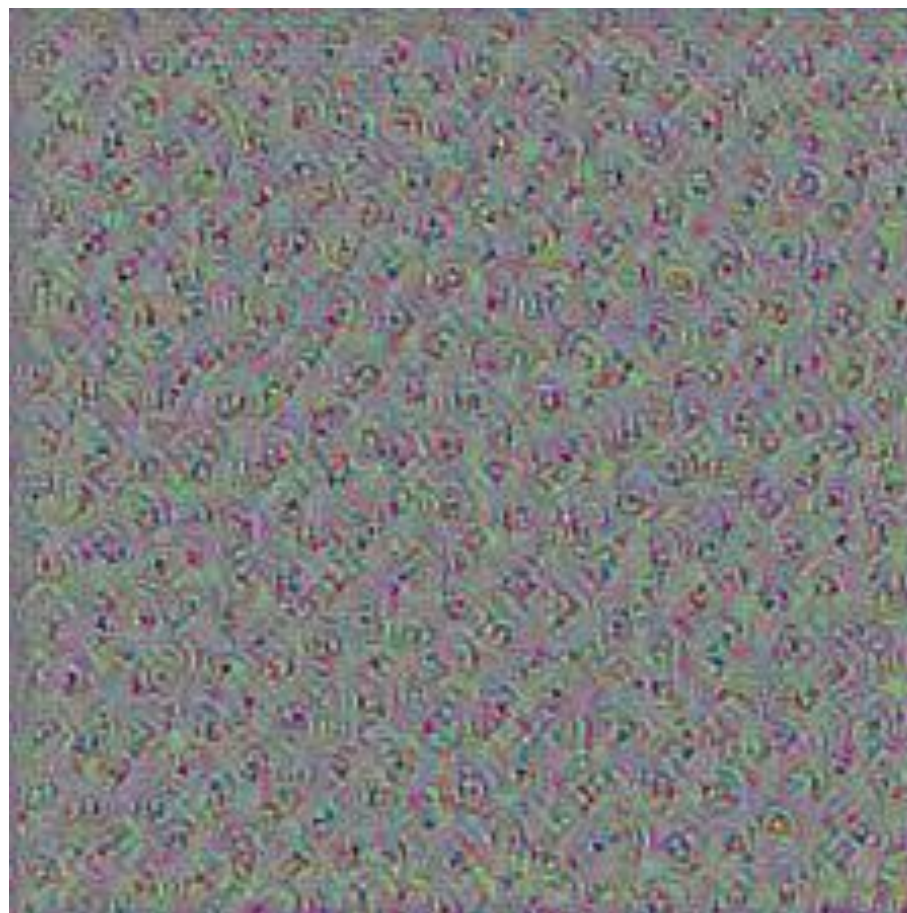
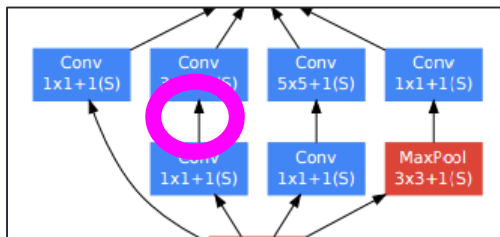
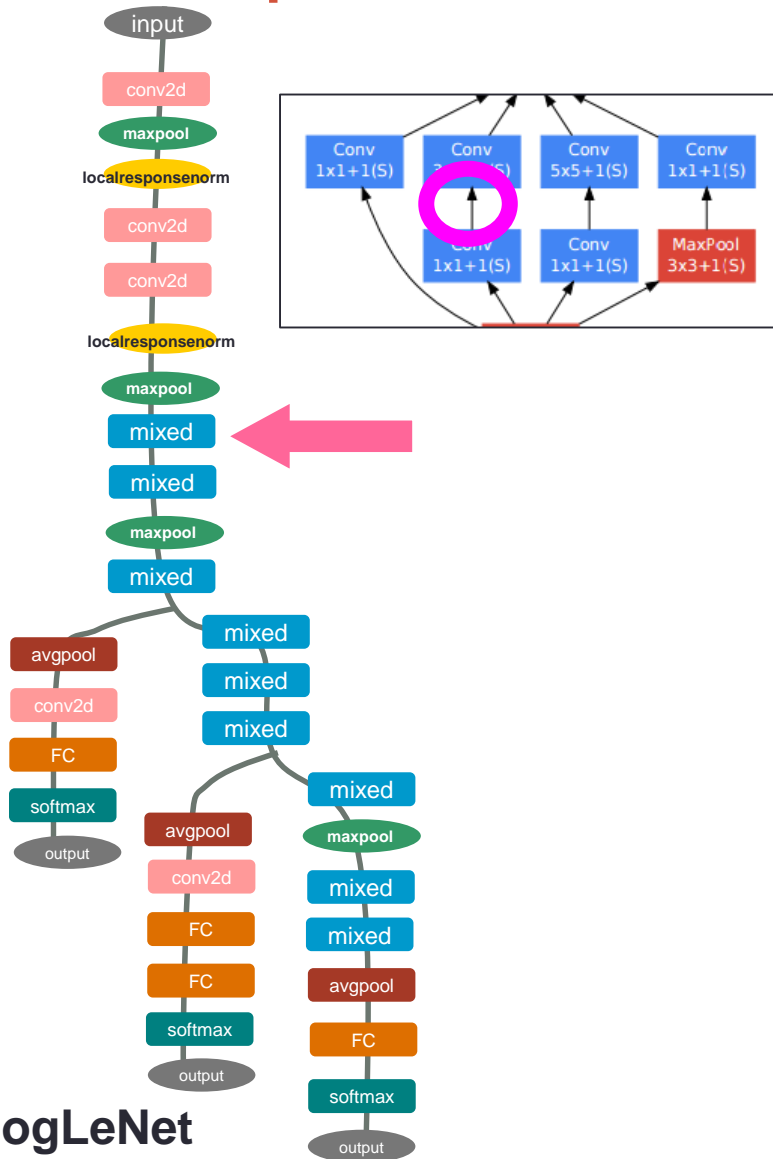


layer

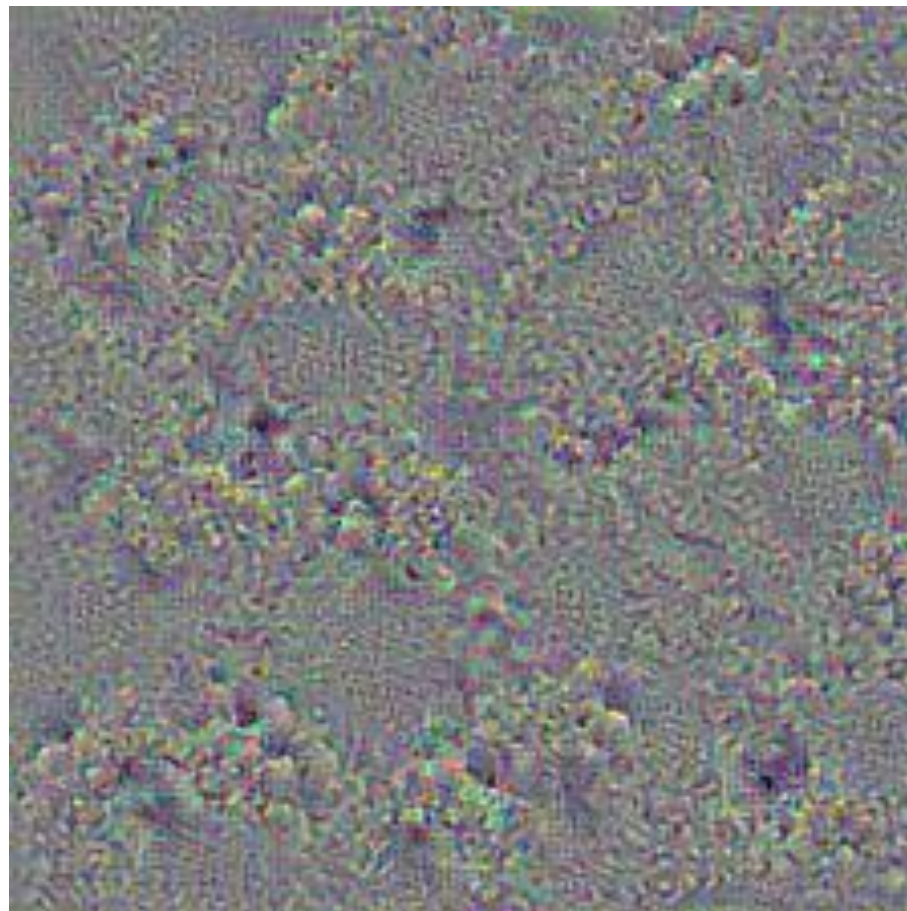
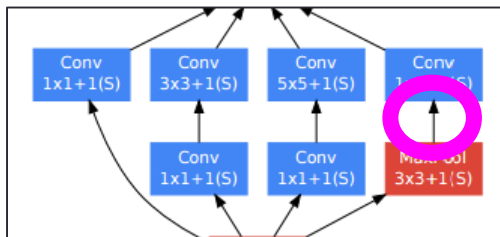
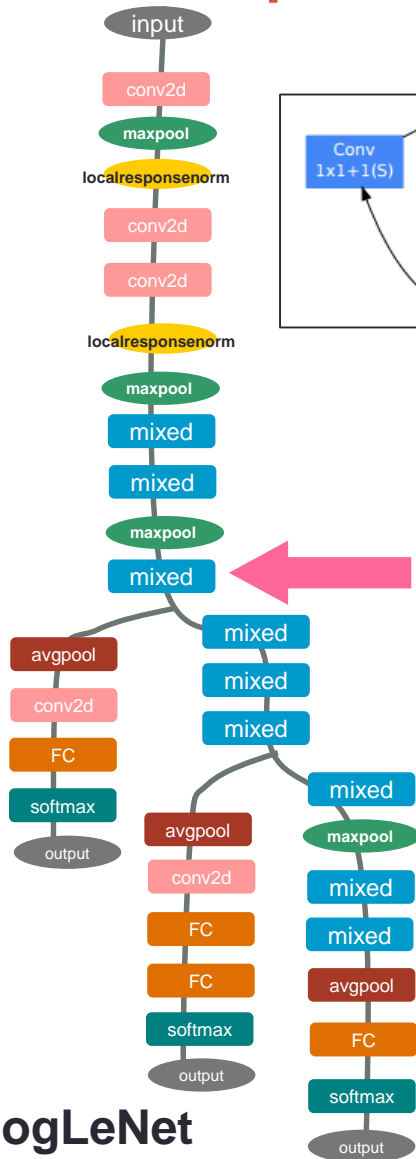
# Examples of naïve feature visualization



# Examples of naïve feature visualization



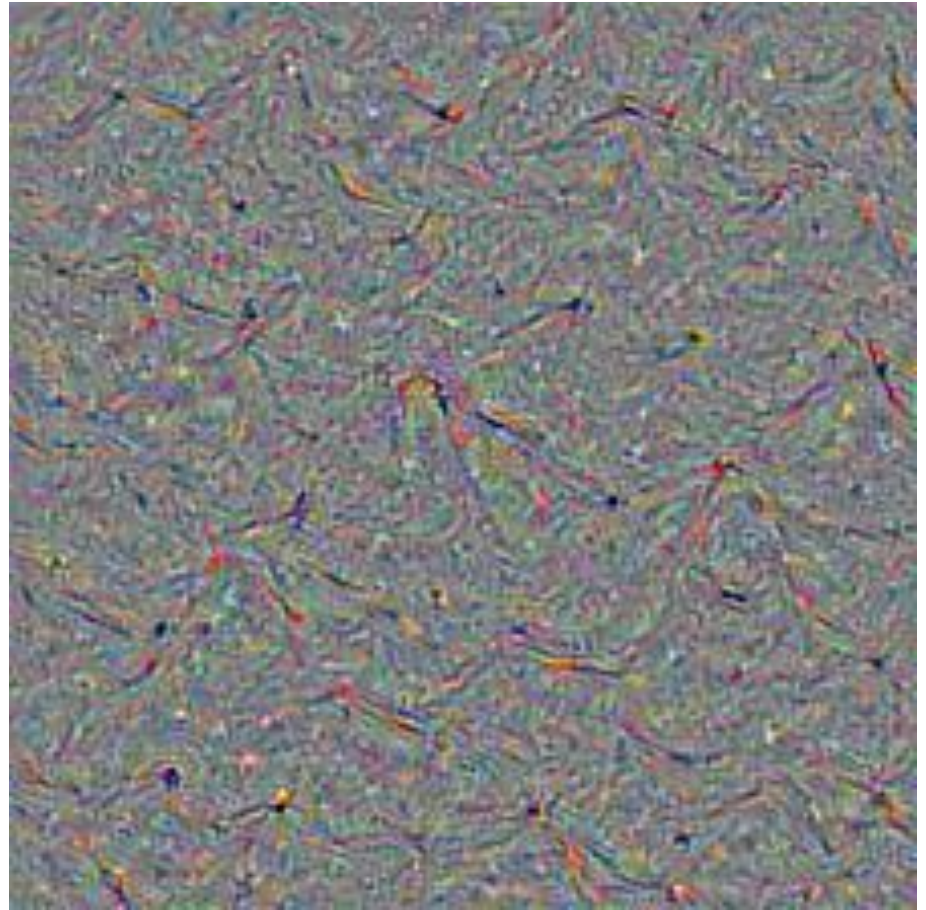
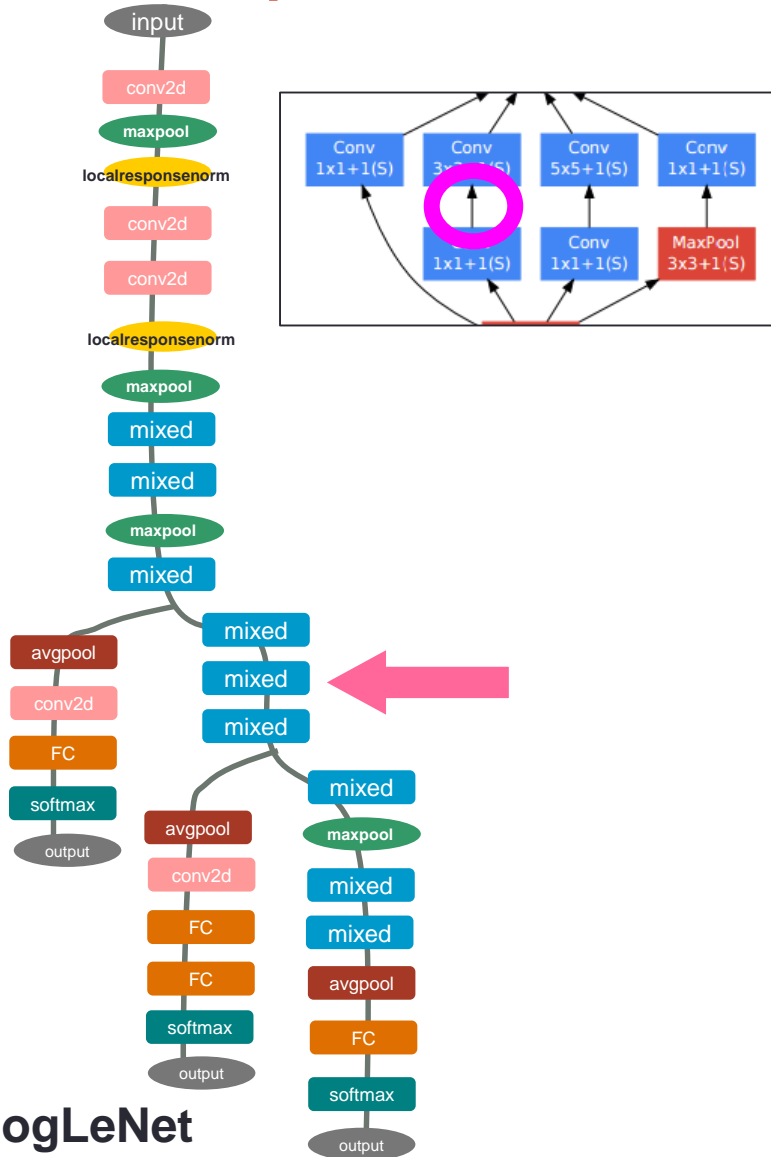
# Examples of naïve feature visualization



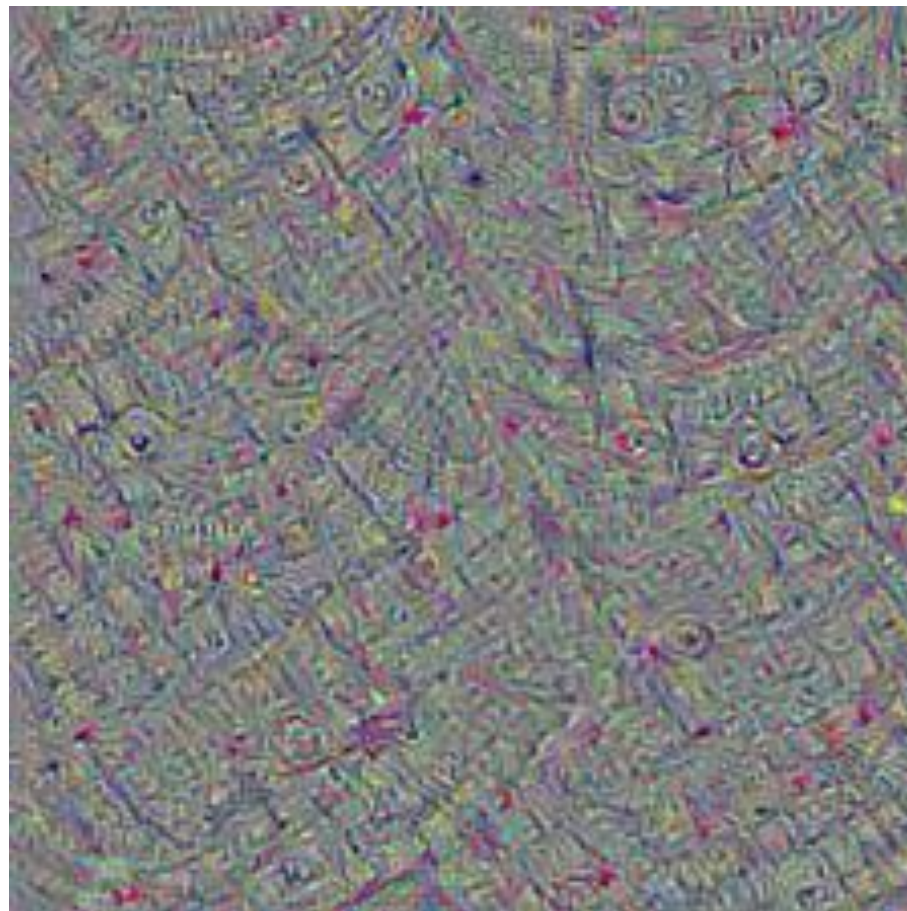
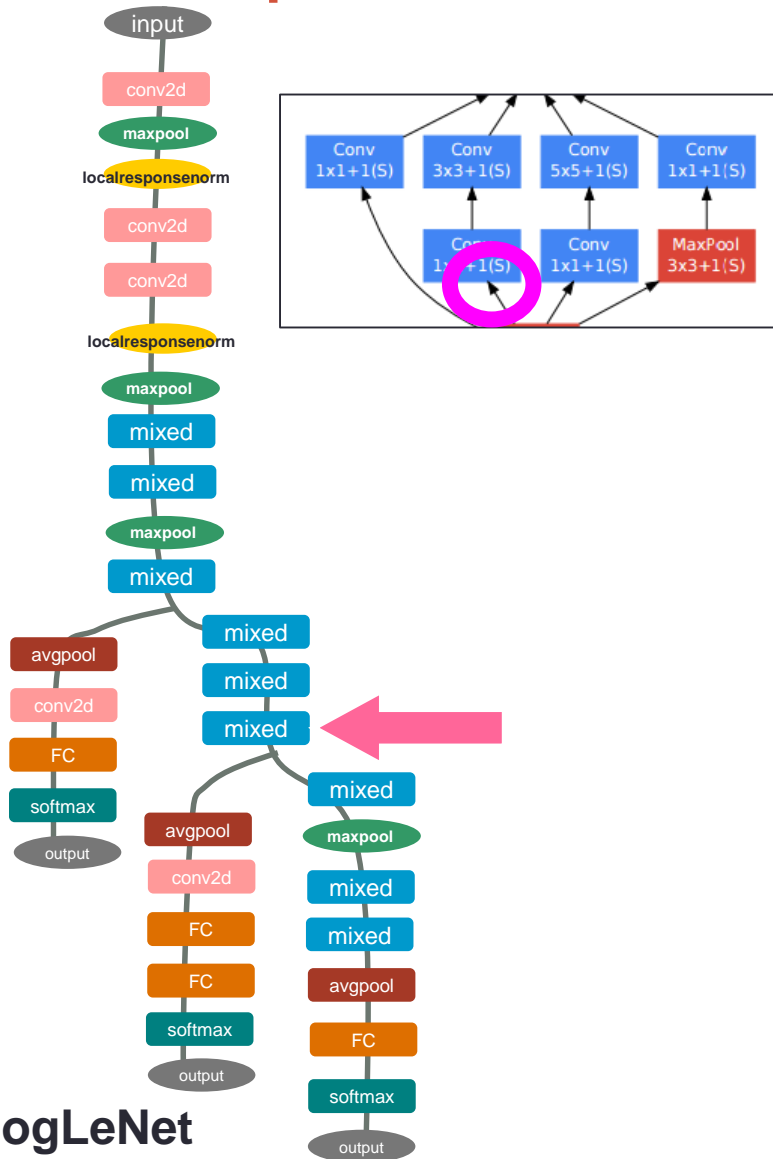




# Examples of naïve feature visualization

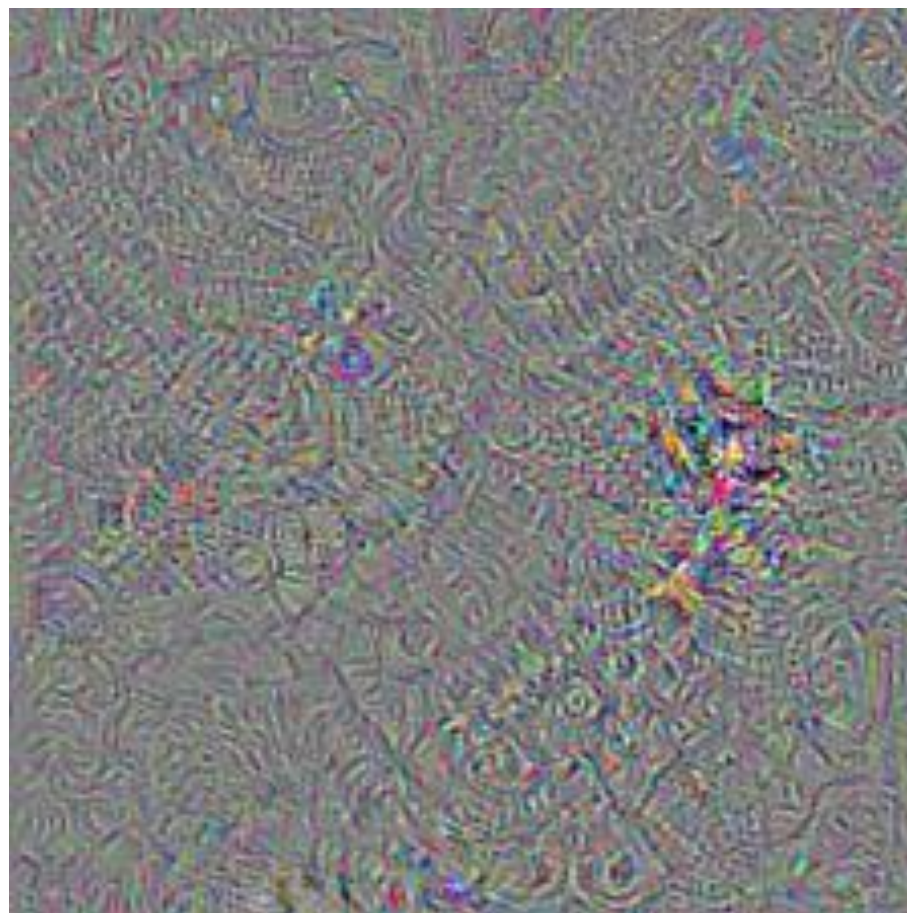
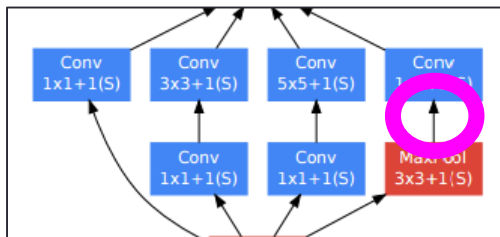
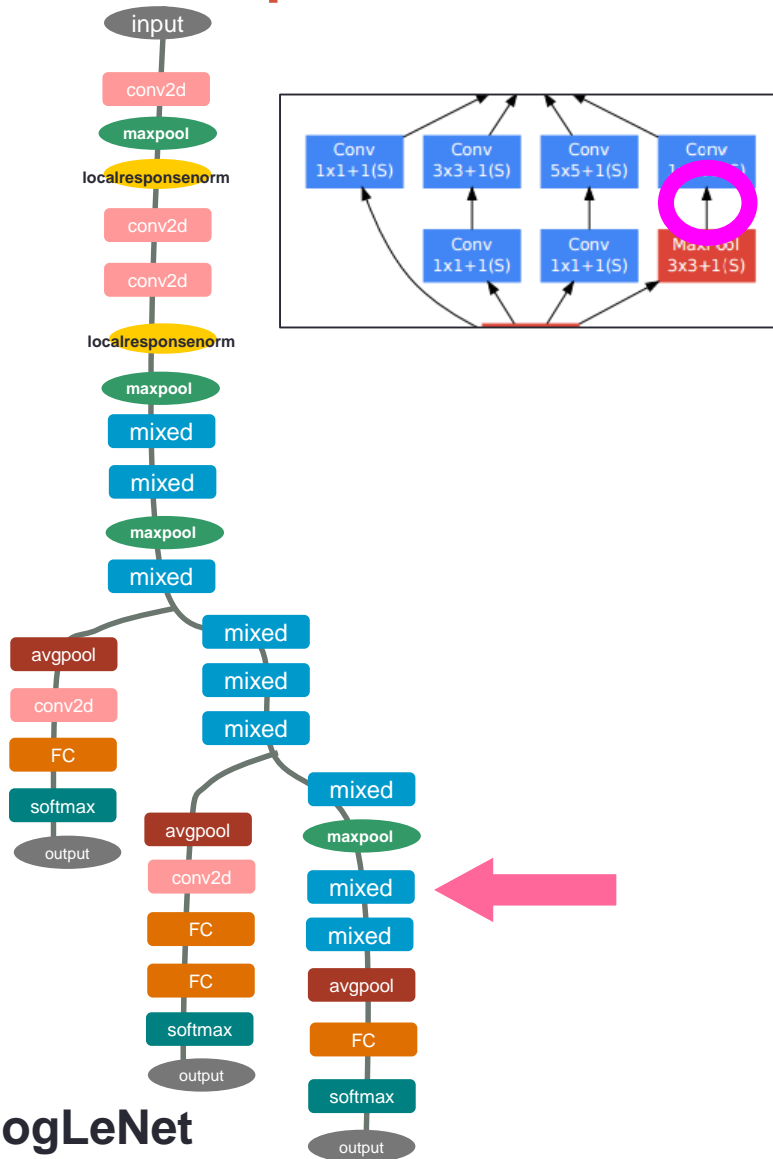


# Examples of naïve feature visualization

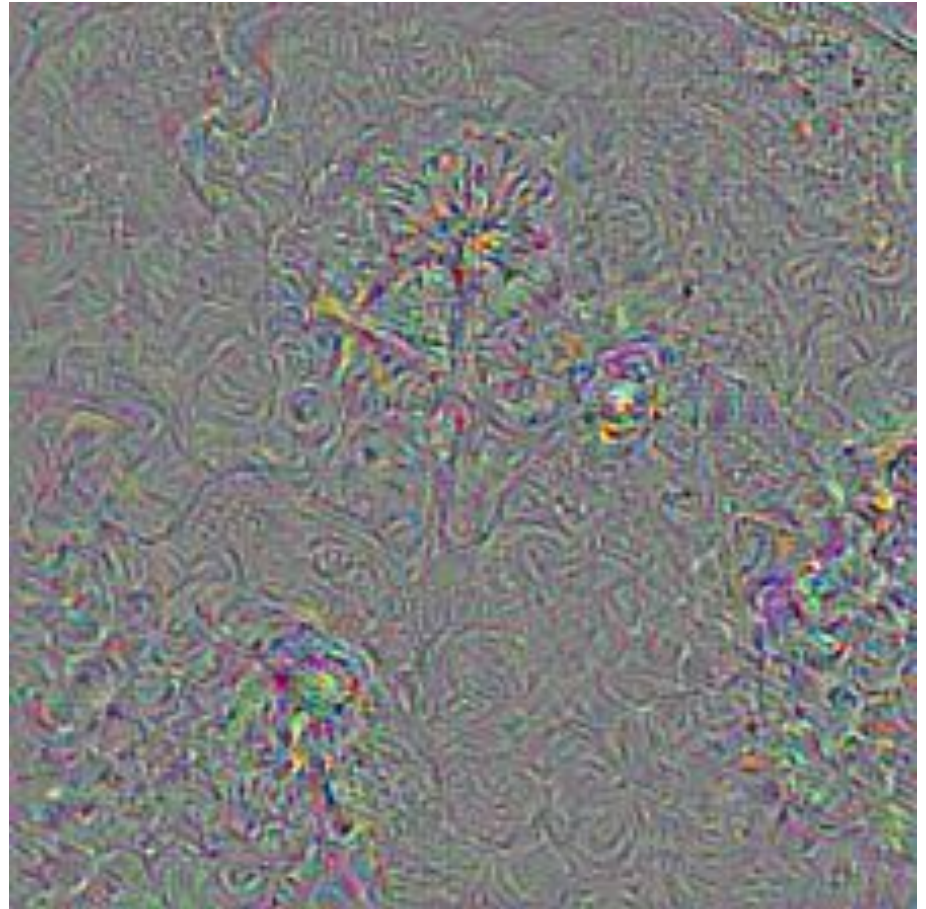
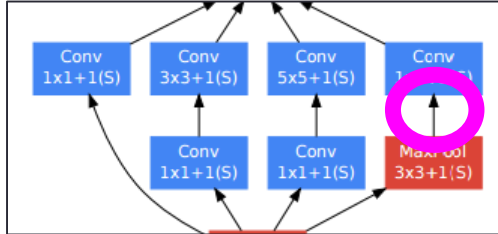
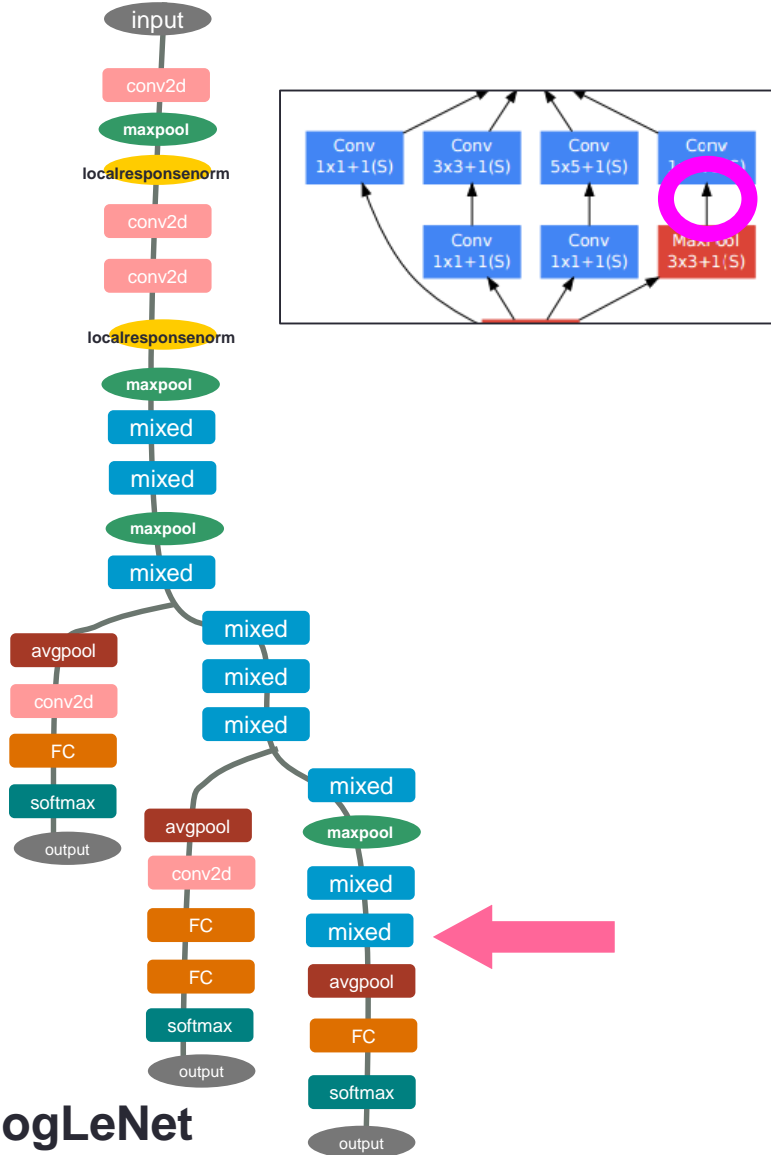




# Examples of naïve feature visualization



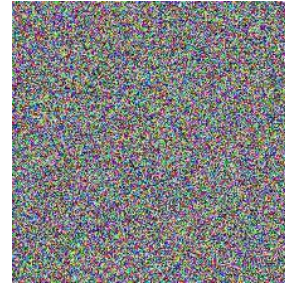
# Examples of naïve feature visualization.



# LOW/HIGH FREQUENCY NORMALIZATION

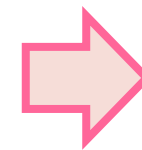
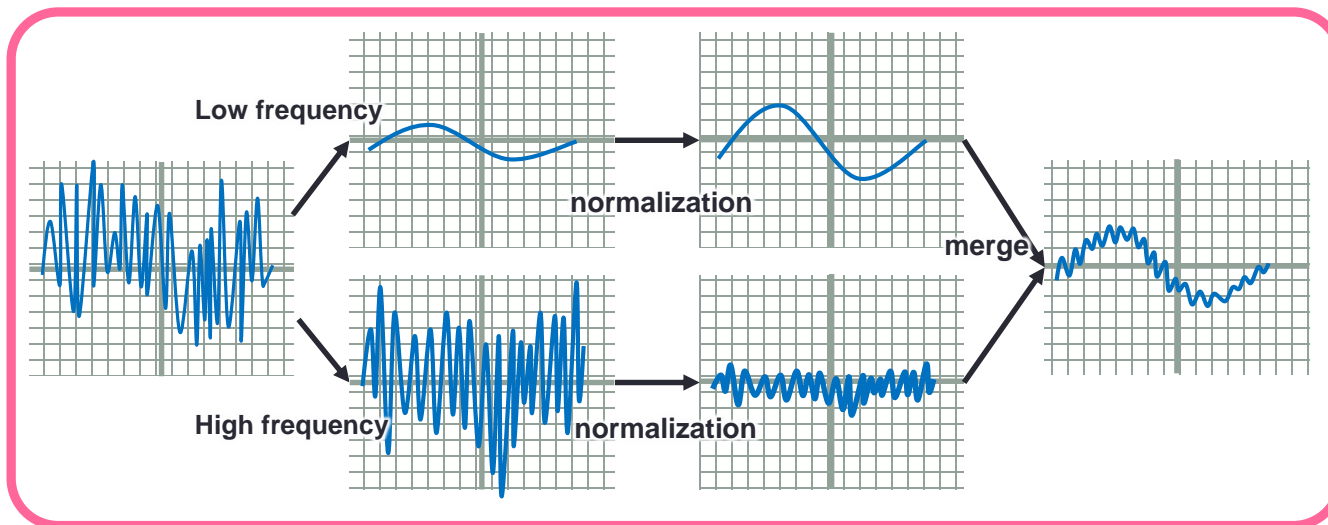
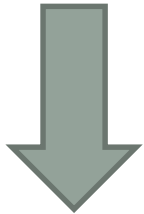
---

# Gradient normalization



Initial input:  
an arbitrary noise image

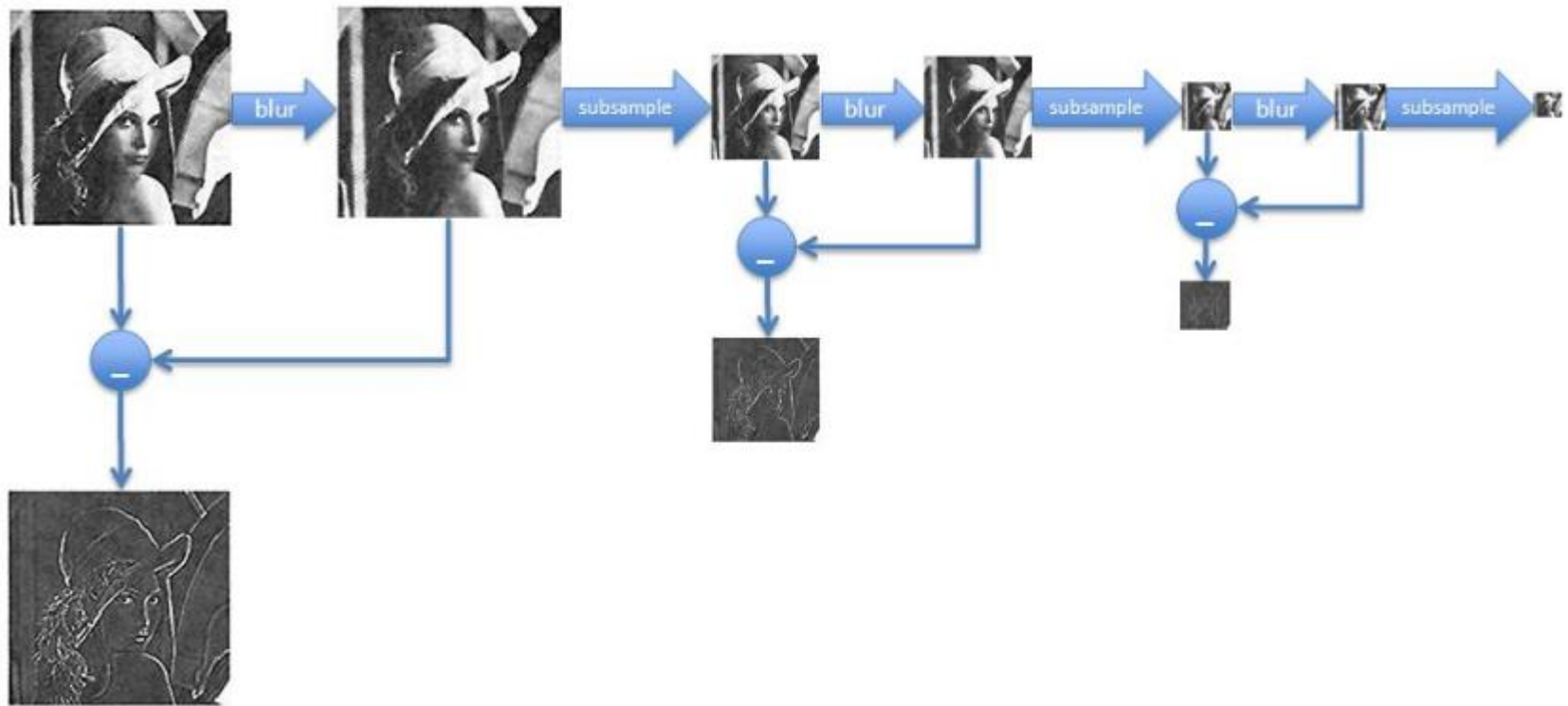
*GRADIENT  
Computation*



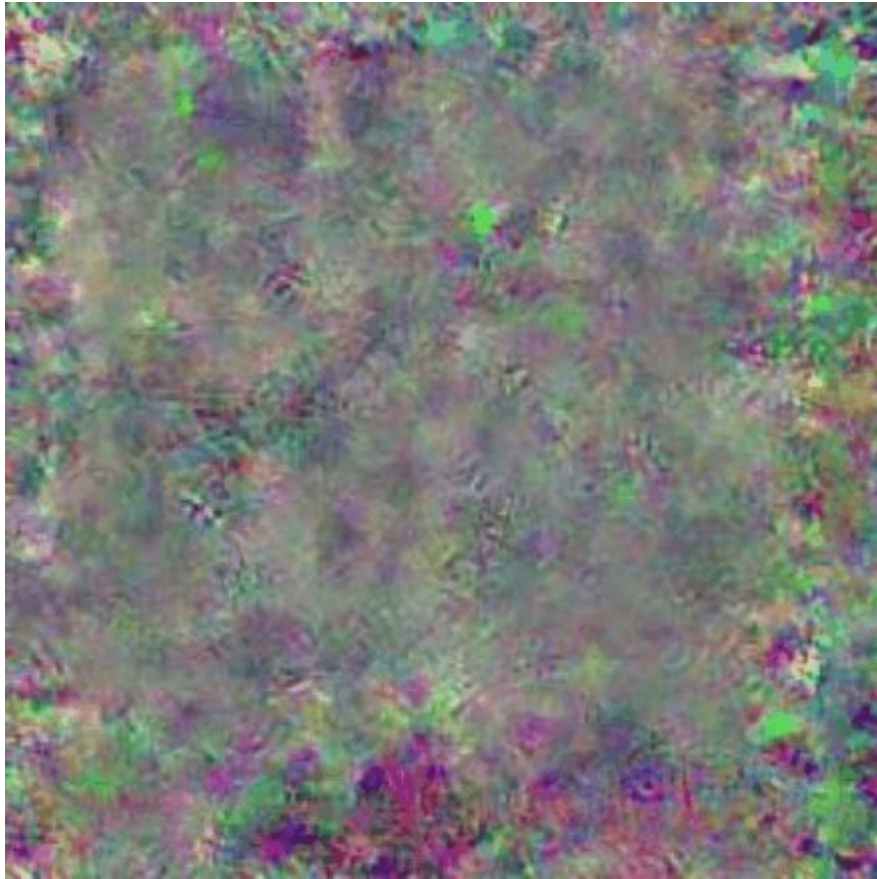
Normalized  
gradient



# Laplacian pyramid

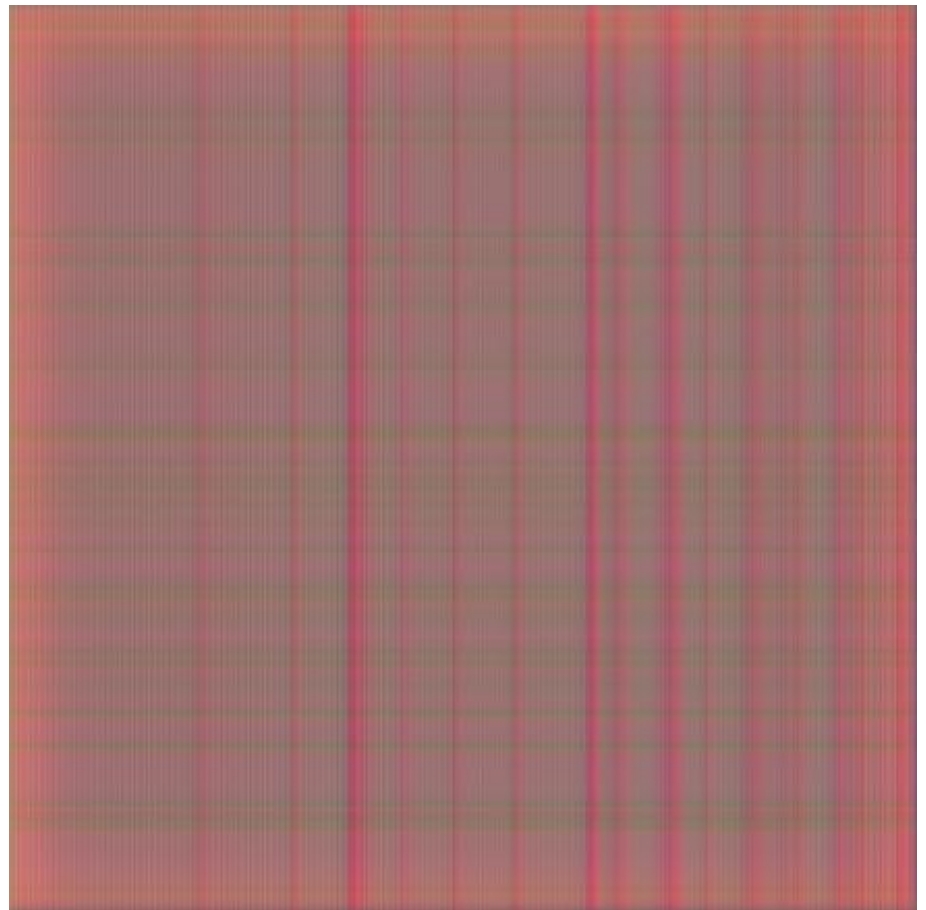
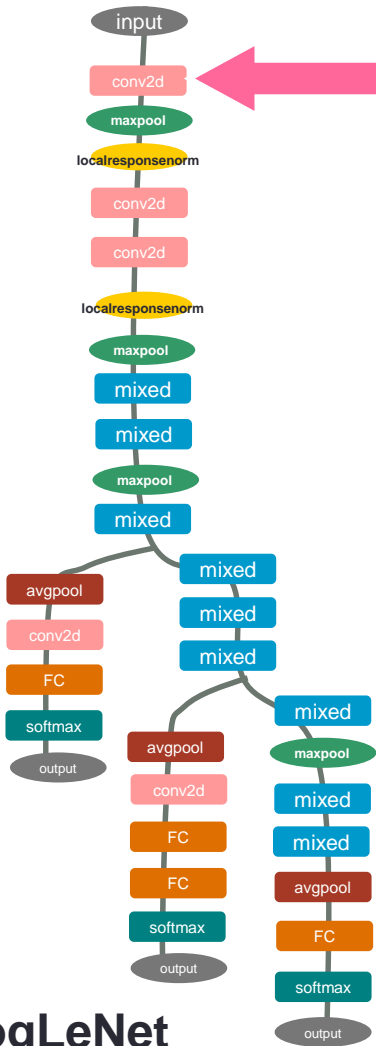


# Convergence example

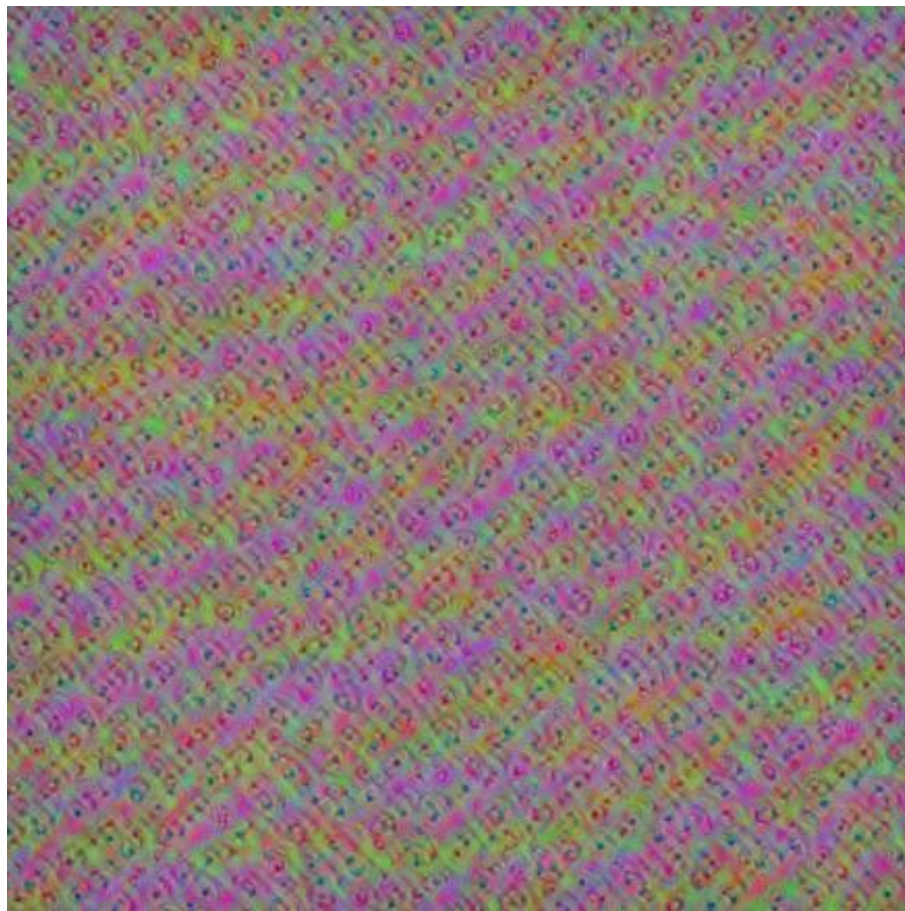
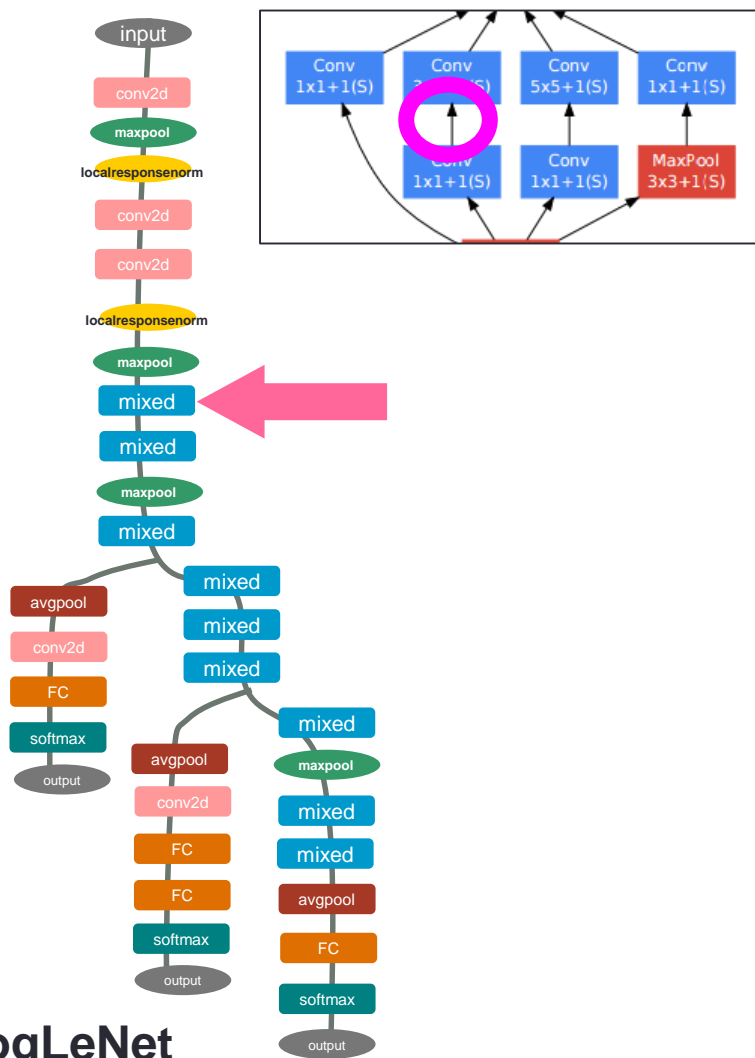


L=102.42

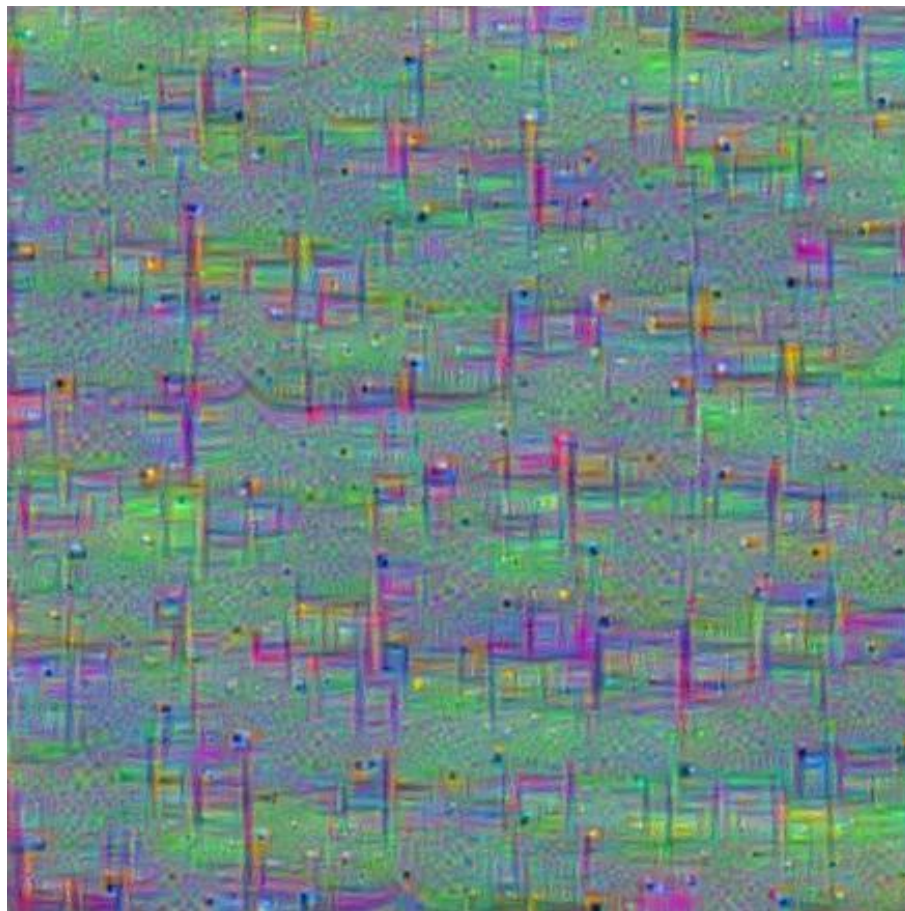
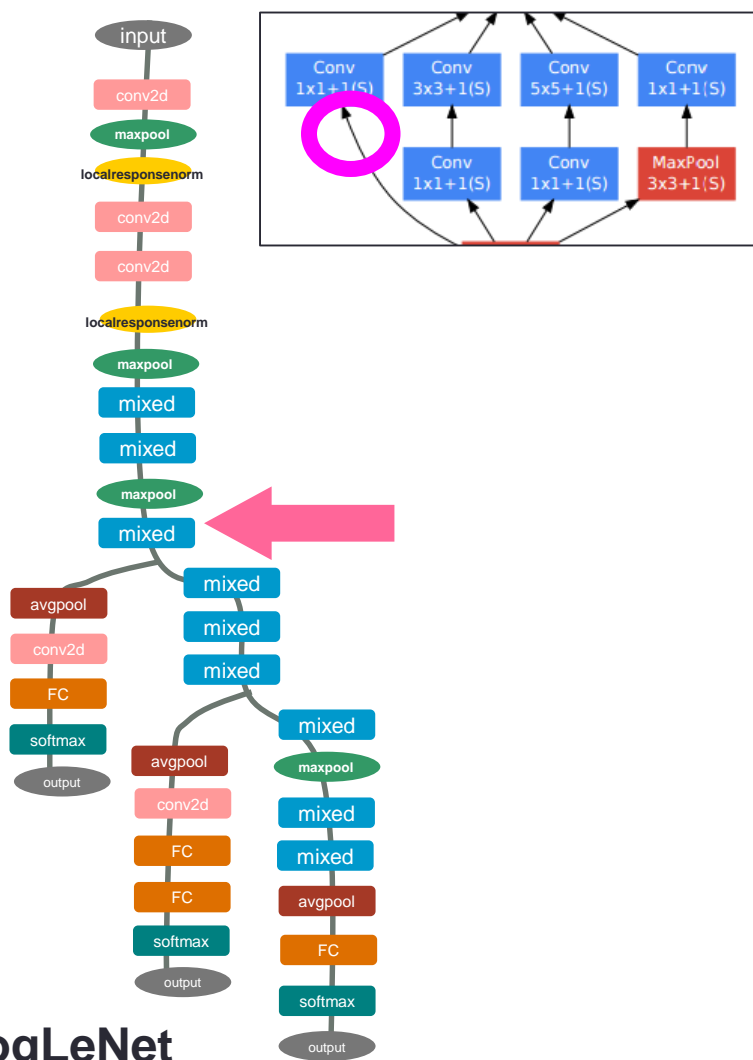
# Result of Laplacian pyramid method



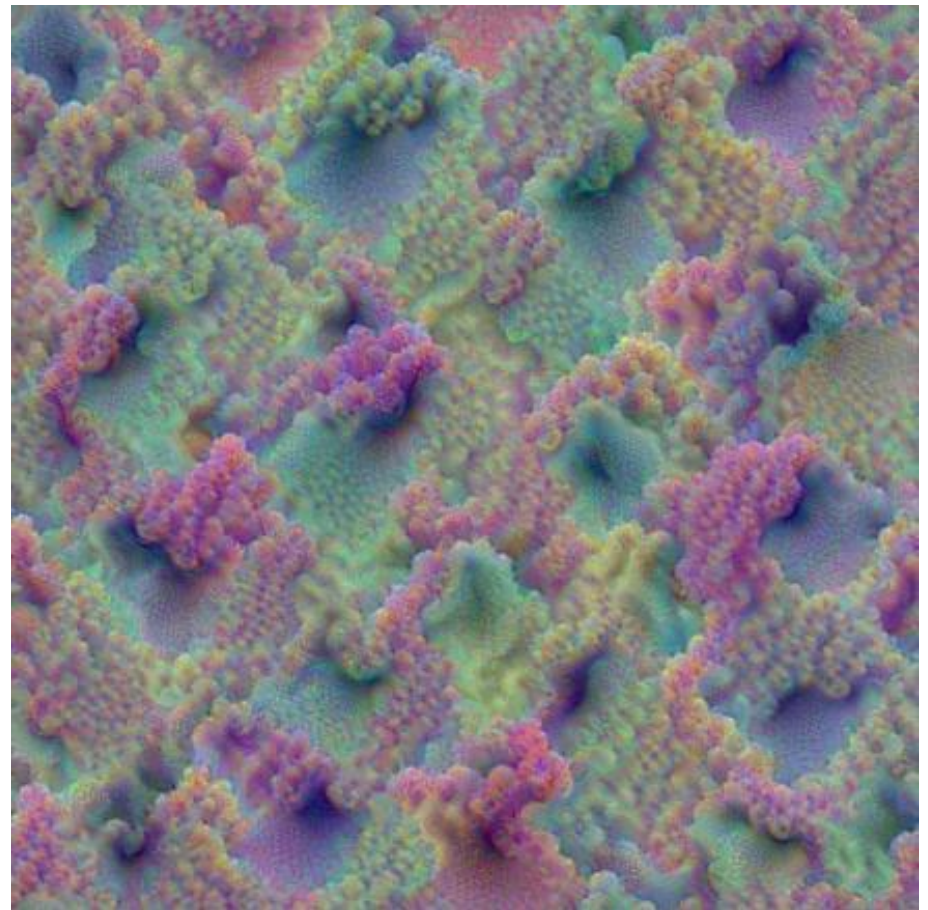
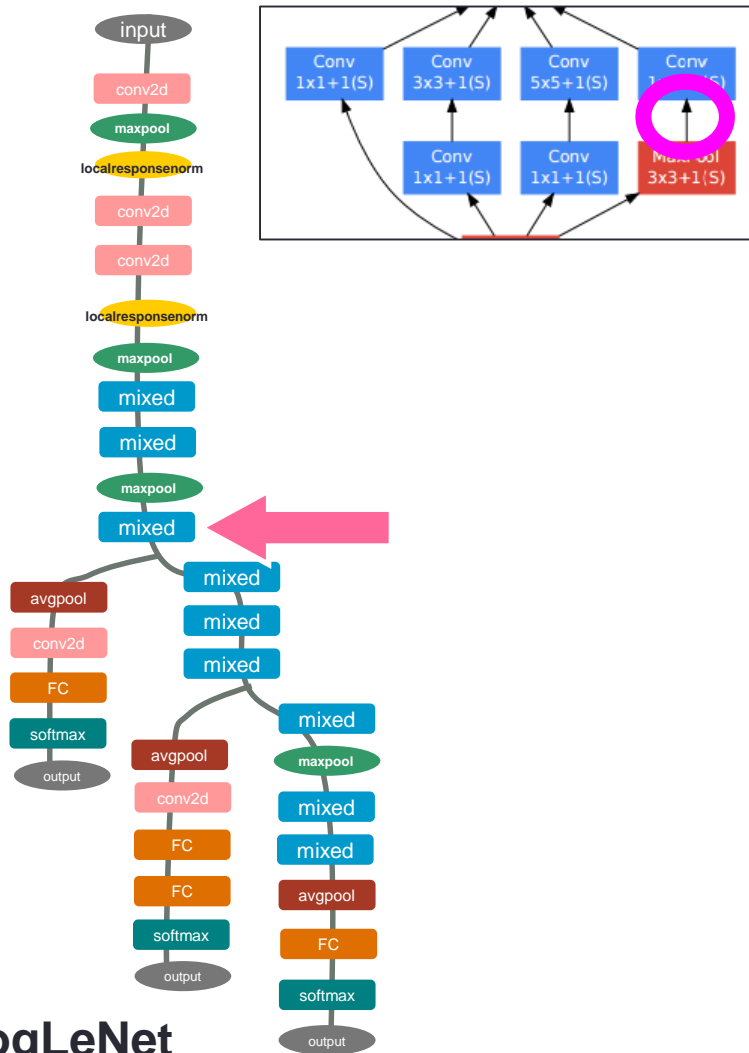
# Result of Laplacian pyramid method



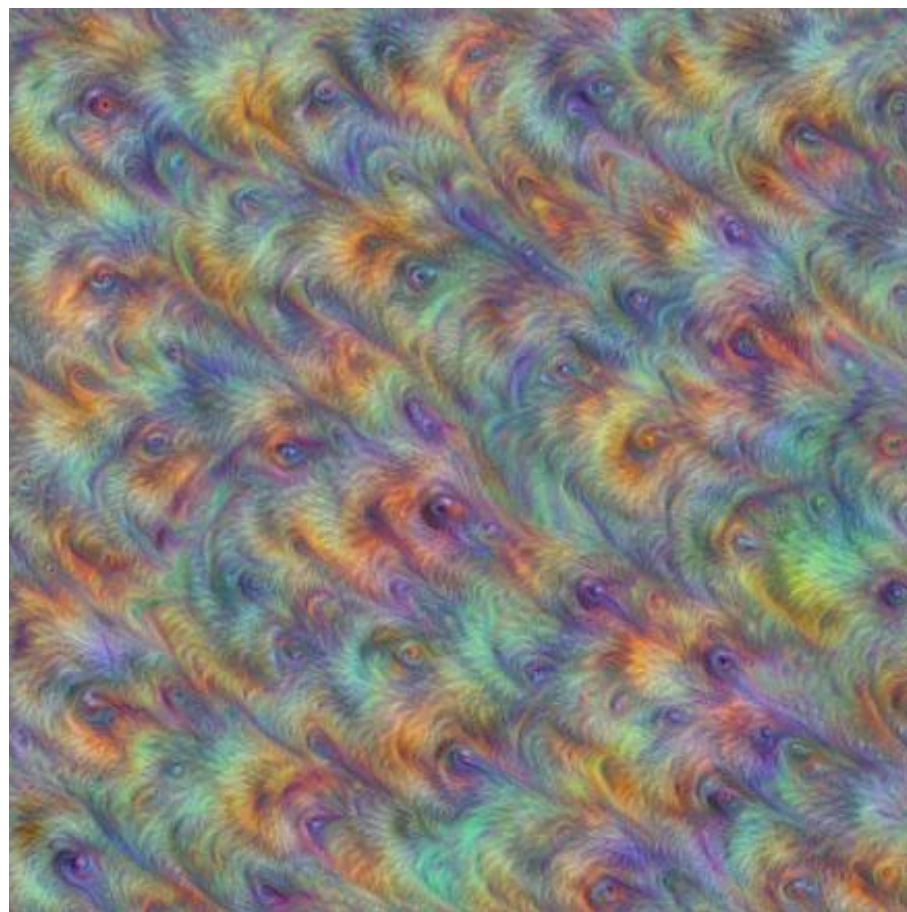
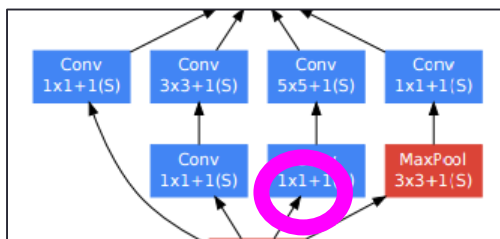
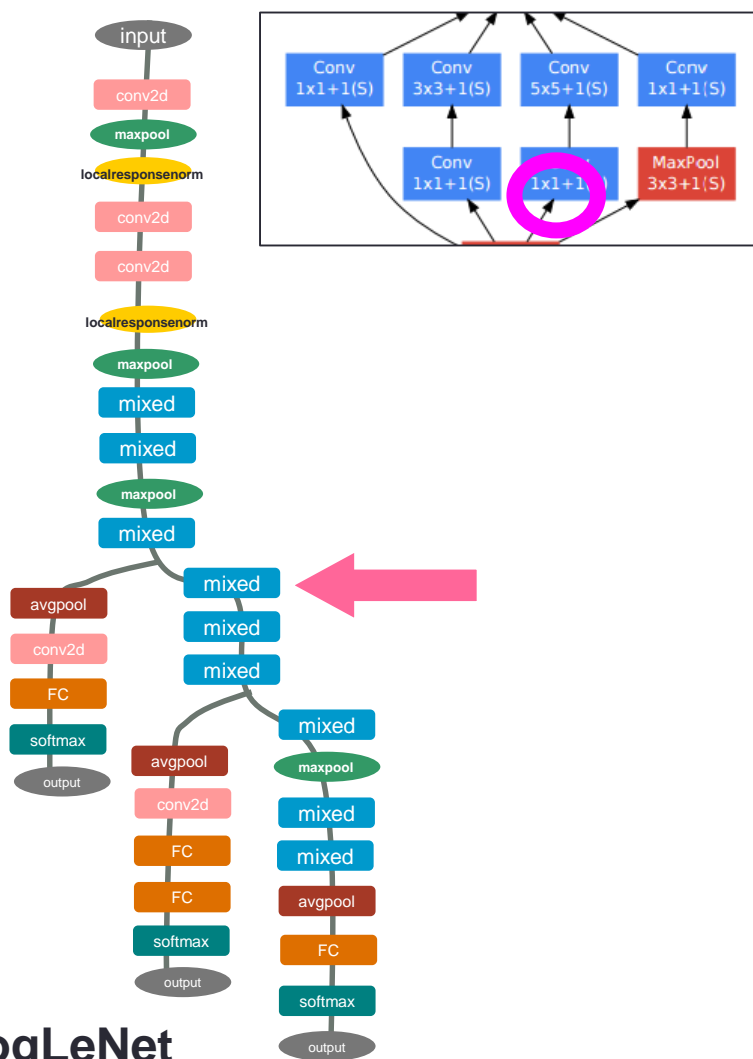
# Result of Laplacian pyramid method



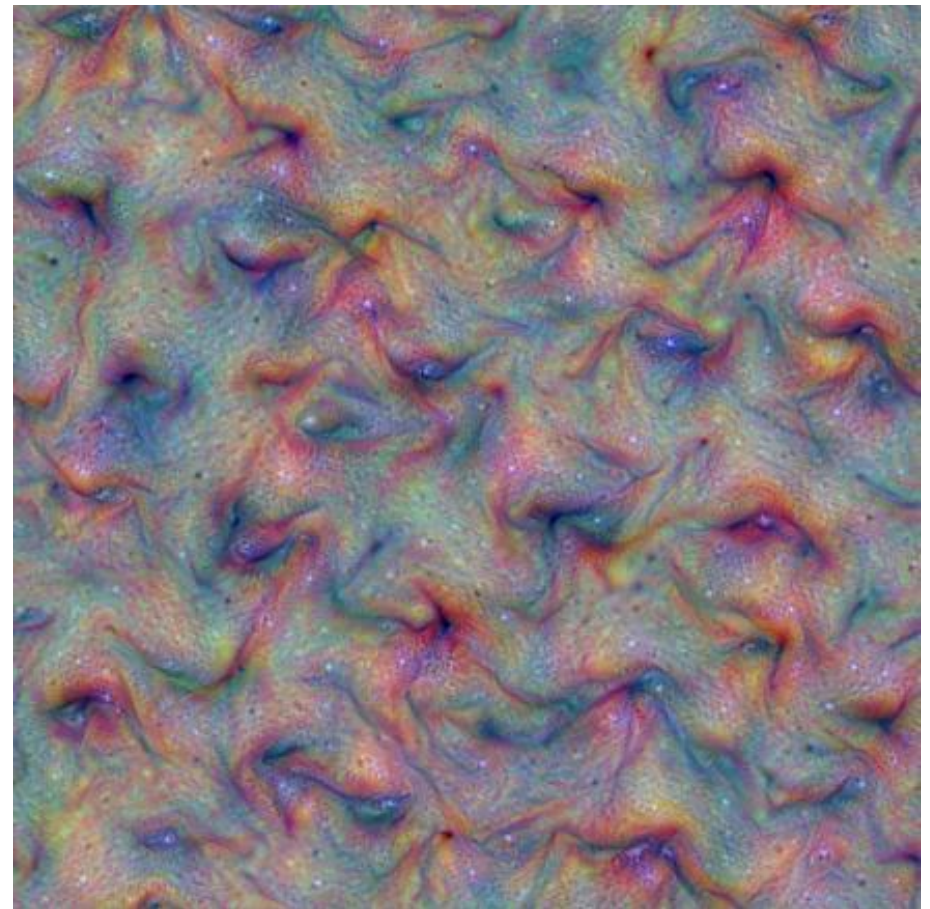
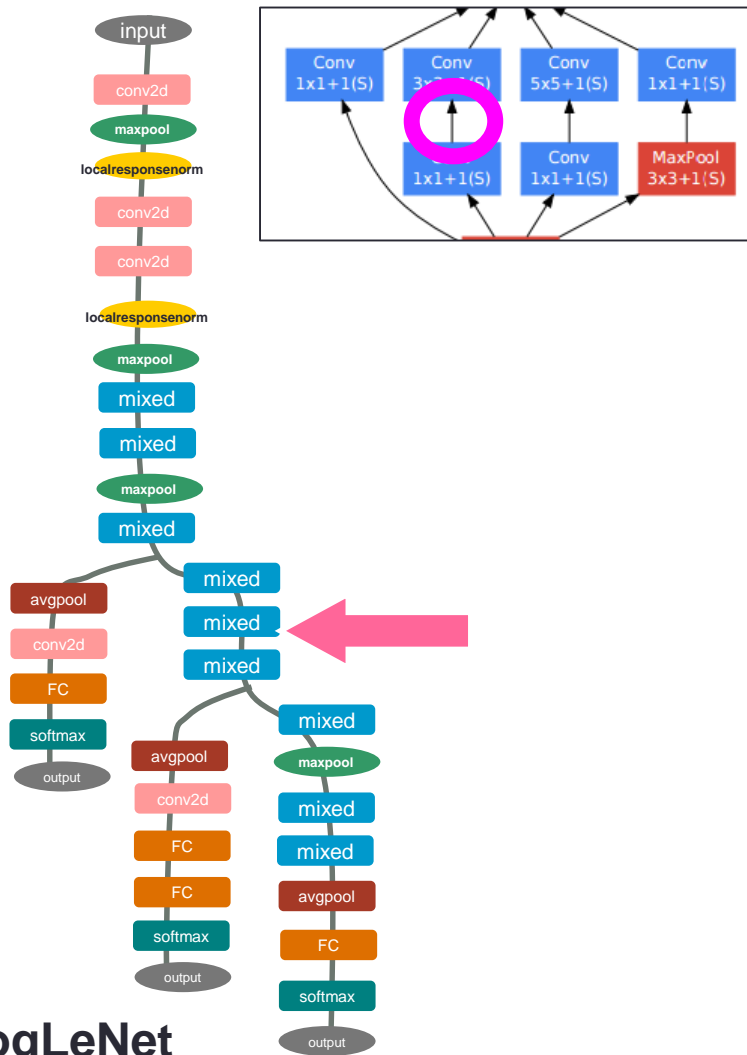
# Result of Laplacian pyramid method



# Result of Laplacian pyramid method

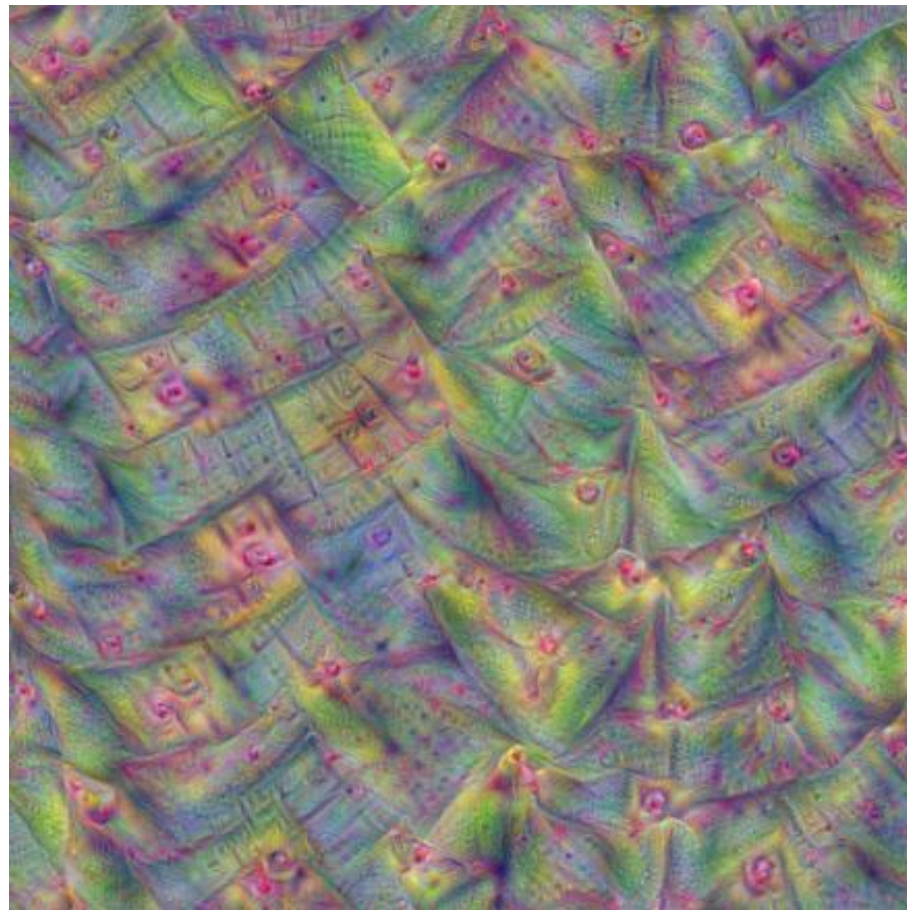
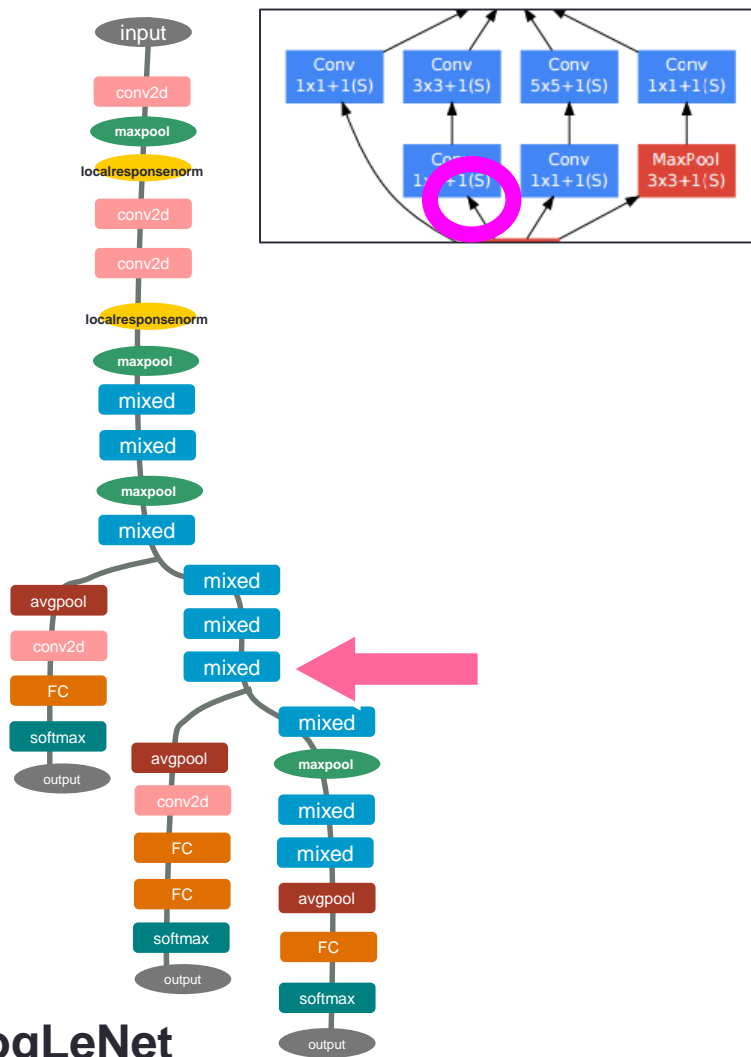


# Result of Laplacian pyramid method

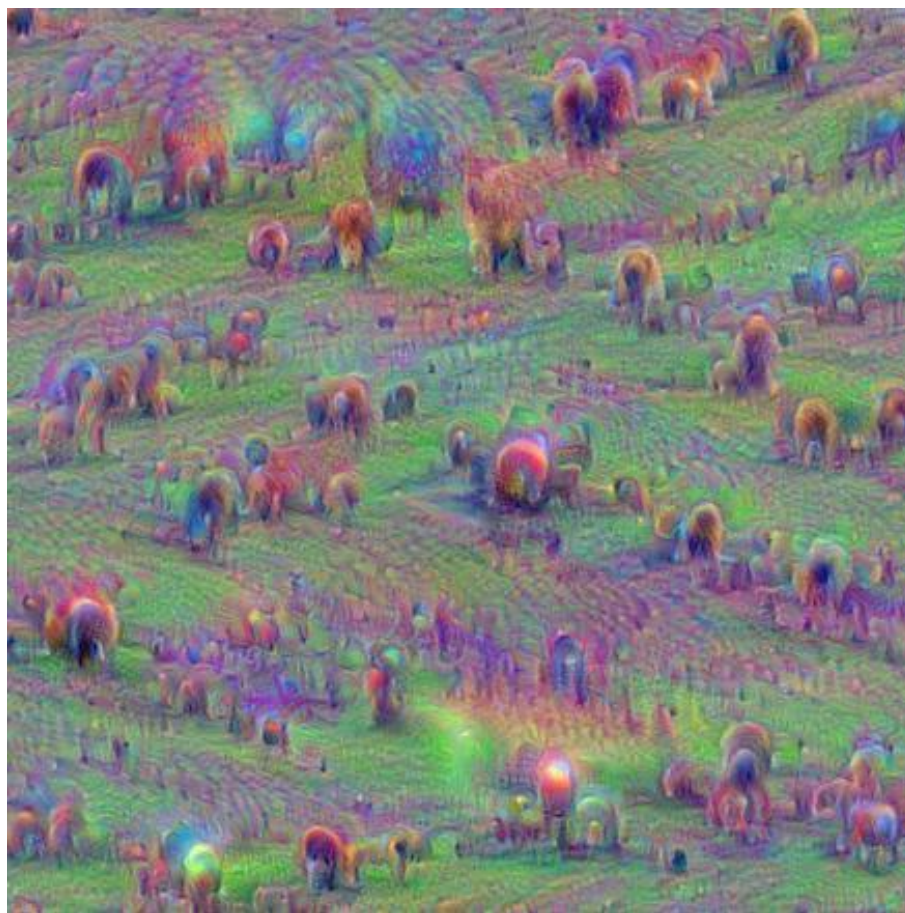
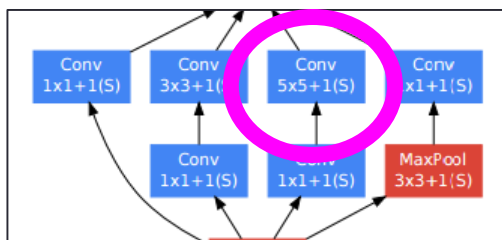
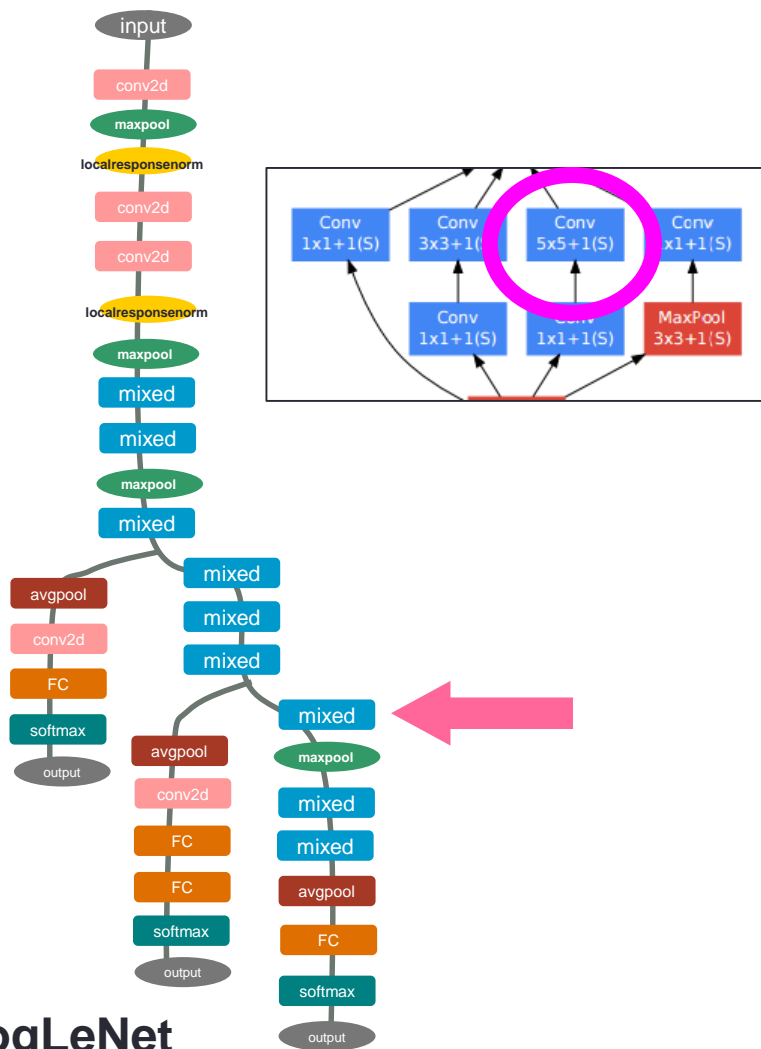




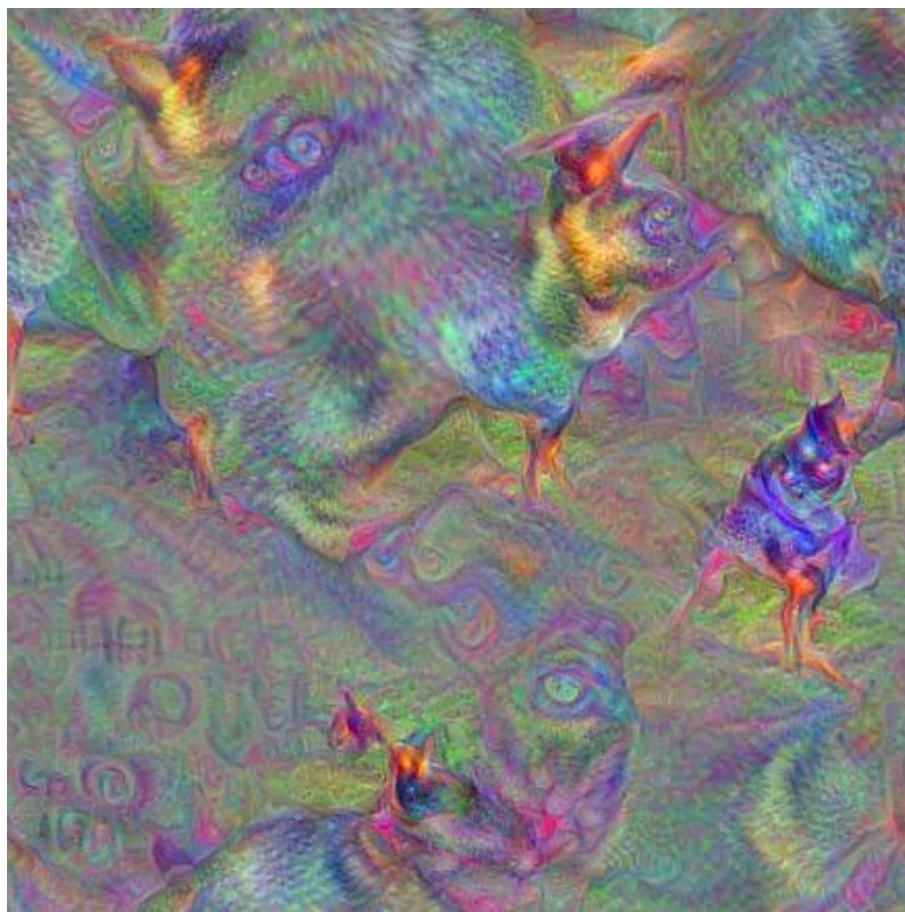
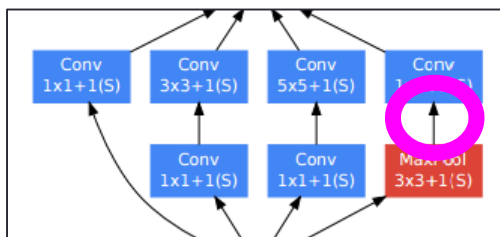
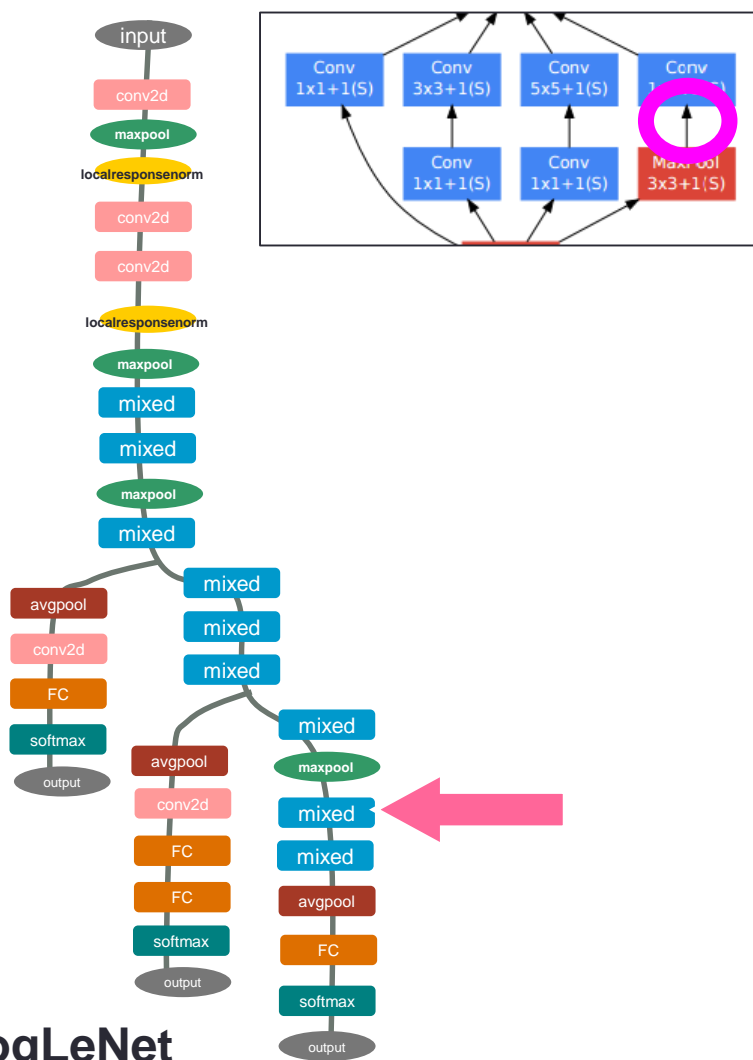
# Result of Laplacian pyramid method



# Result of Laplacian pyramid method

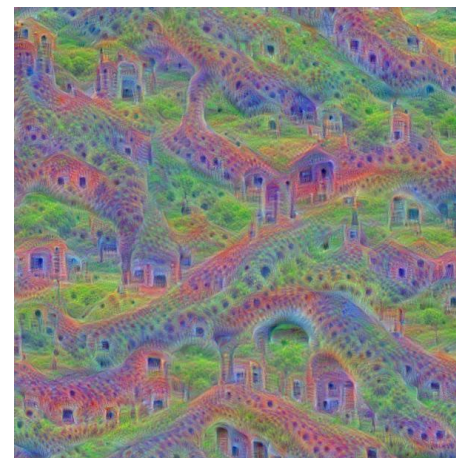
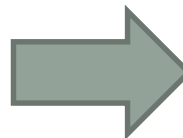
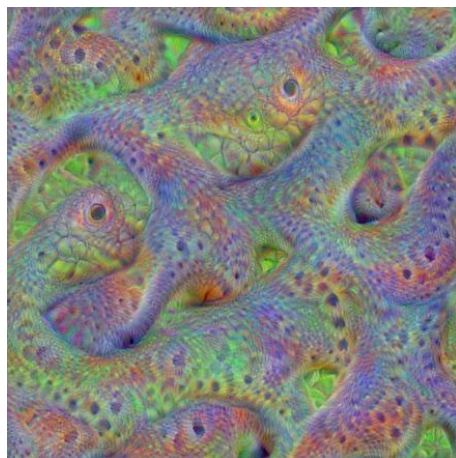
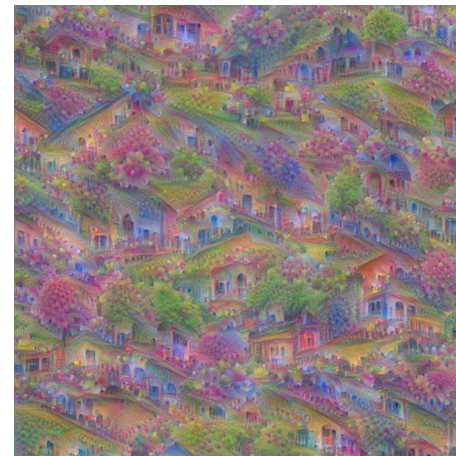
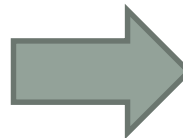


# Result of Laplacian pyramid method

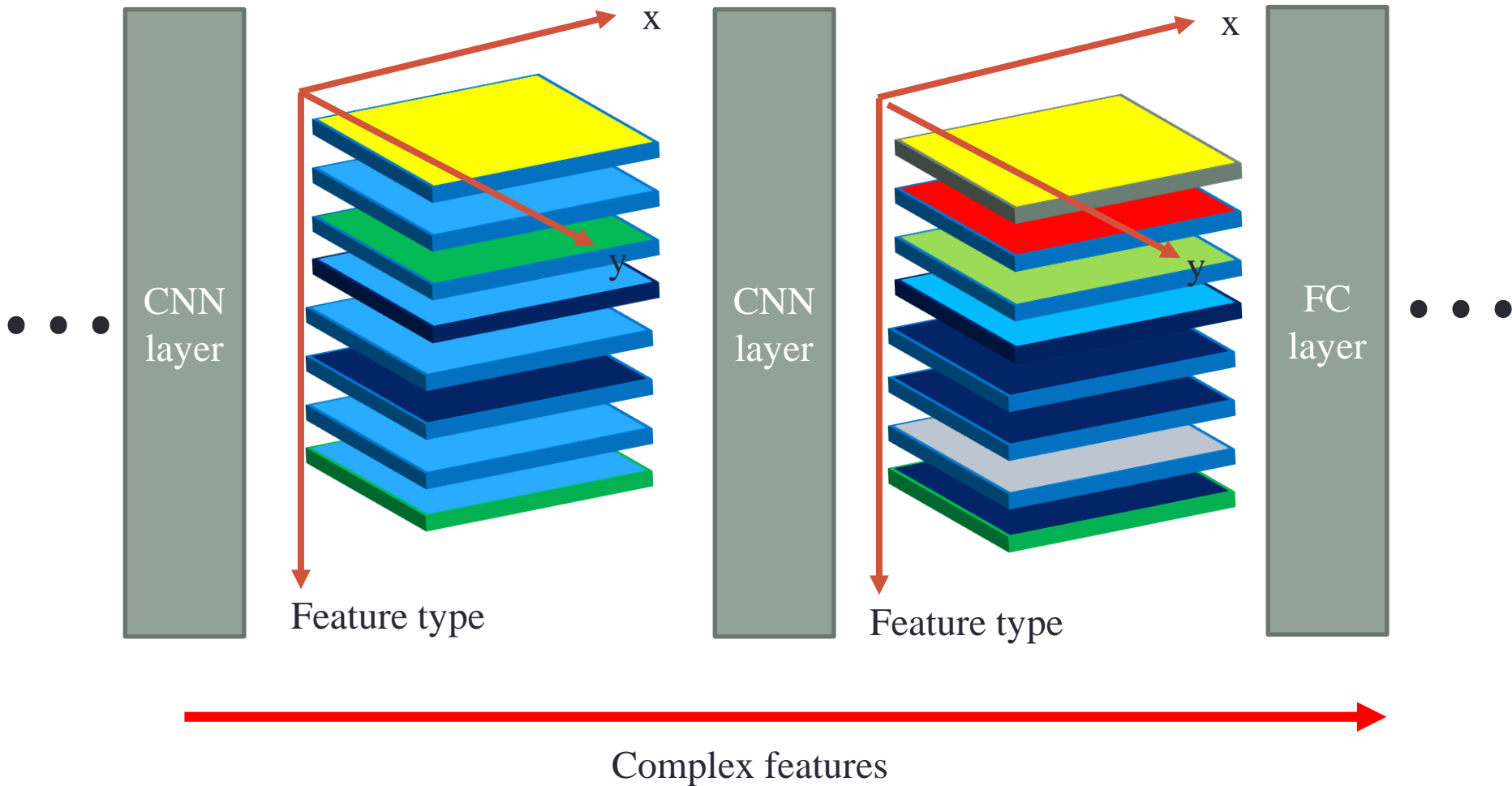




# Results with two channels



# Summary



DEEPA DREAM

---

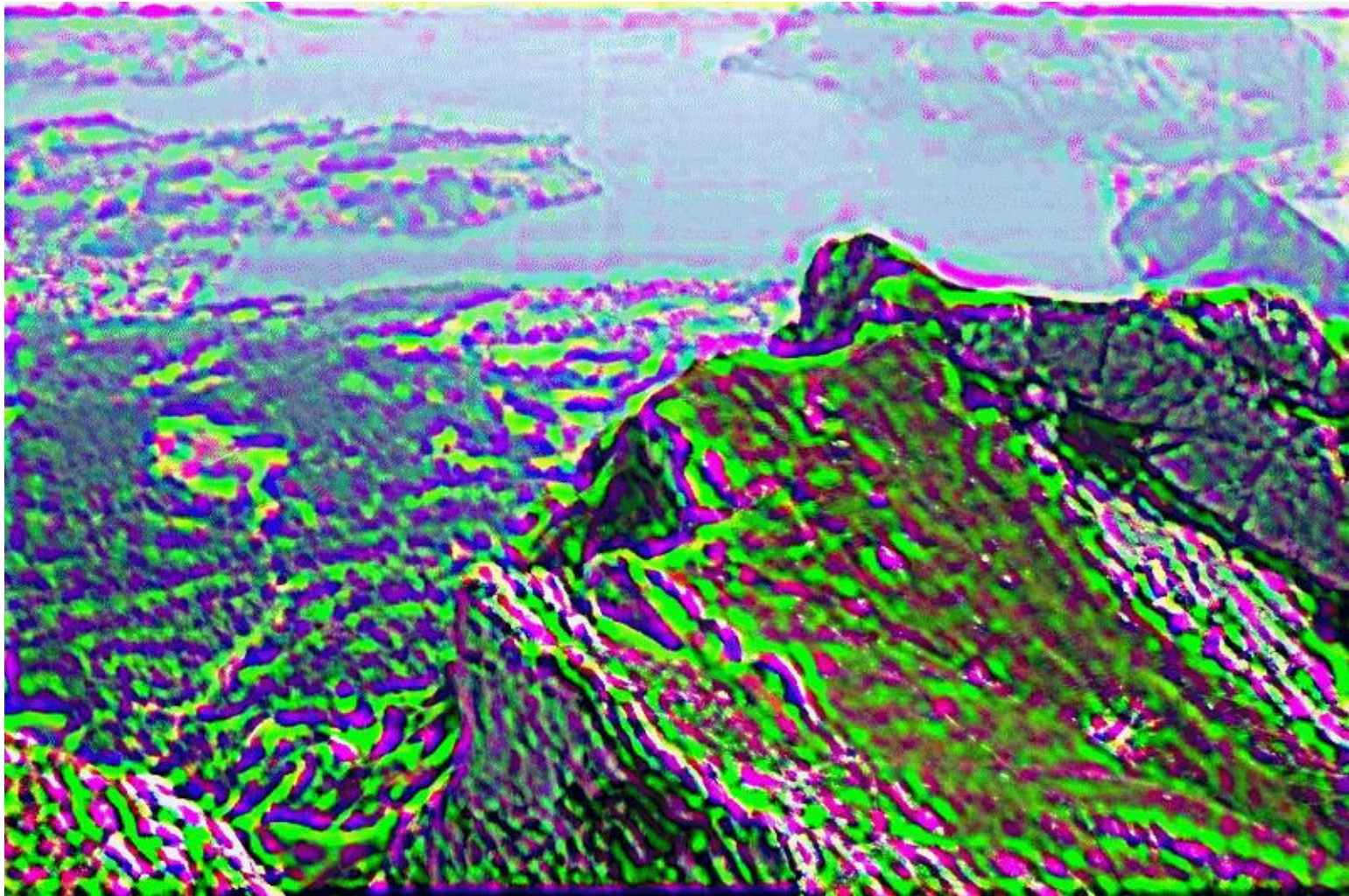
# DeepDream

- Square cost function
  - $(1 + 1)^2$  vs  $1^2$
  - $(100 + 1)^2$  vs  $100^2$





# Examples (all feature maps in a layer)



# DeepDream example



<http://www.pyimagesearch.com/2015/08/03/deep-dream-visualizing-every-layer-of-googlenet/>

# DeepDream example





conv2/3x3



inception\_3a/3x3



inception\_3b/3x3\_reduce



inception\_4b/3x3

# OTHER TOPICS IN CNN

---

# NETWORK COMPARISON

---

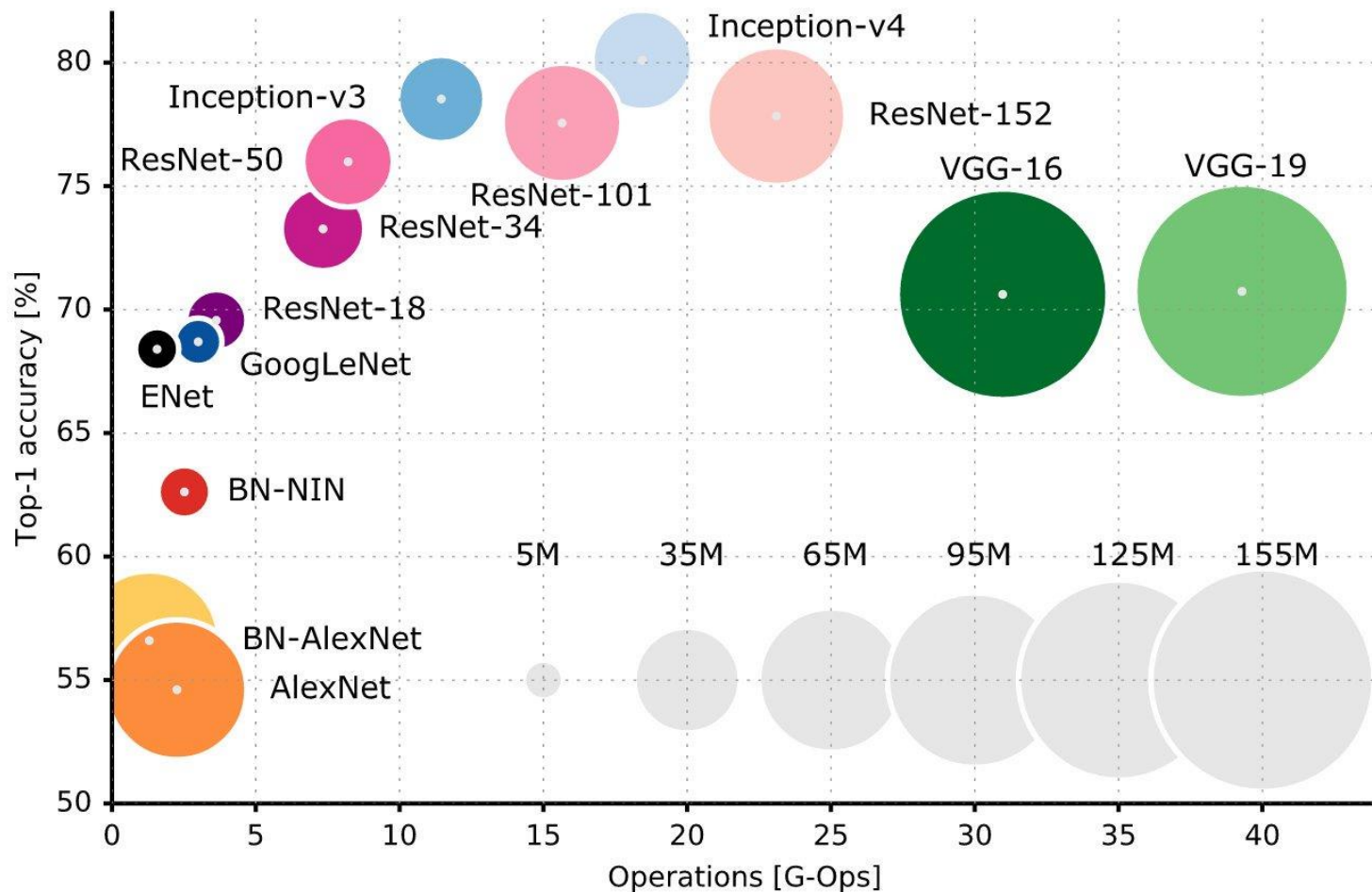
# AlexNet VS VGG-19 VS GoogLeNet

	Parameters	Operations (MACs)	*Top-1 accuracy (%)	*Top-5 accuracy %
<a href="#">AlexNet</a>	60 M	832 M	56.9	80.1
<a href="#">VGG-19</a>	144 M	19,632 M	68.5	88.5
<a href="#">GoogLeNet</a>	6.8 M	1,502 M	68.7	89.0

\*Evaluated with ImageNet2012

\*<https://github.com/BVLC/caffe/wiki/Models-accuracy-on-ImageNet-2012-val>

# Comparison



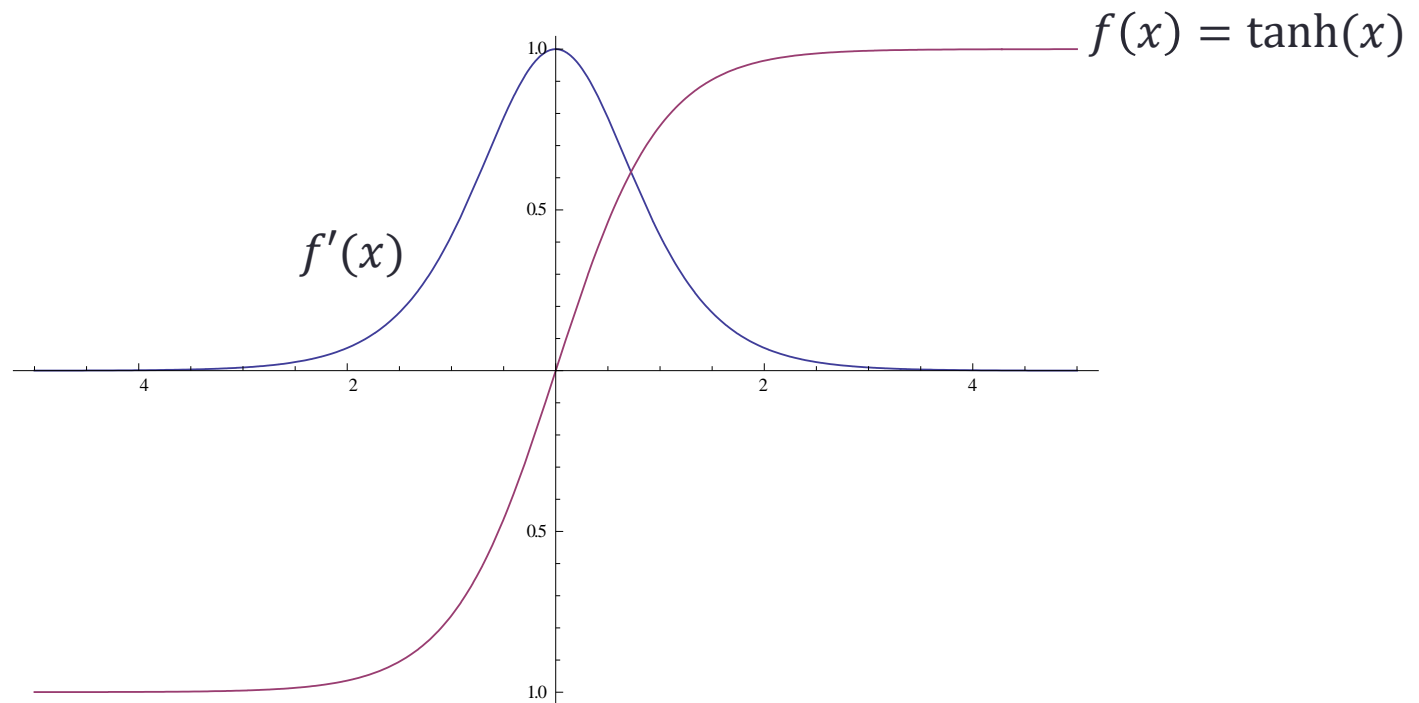


# ISSUES IN TRAINING DNN

---

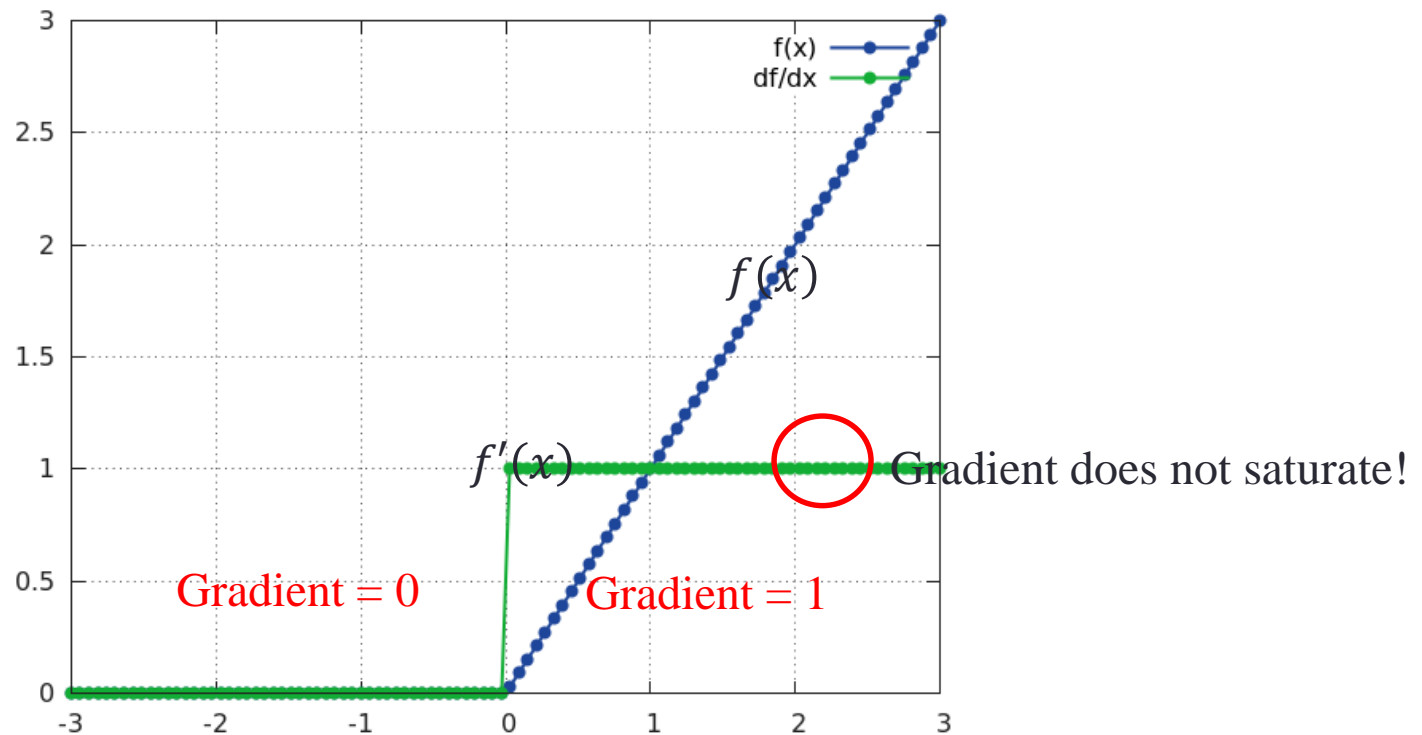
# Vanishing Gradient

- Saturating nonlinearities (e.g., tanh) can not be used for deep networks as they tend to get stuck in the saturation region as the network grows deeper



# Solution of Gradient Vanishing Problem

- ReLU, LeakyReLU



# Internal covariate shift

- Each layer in a neural net has a simple goal, to model the input from the layer below it, so each layer tries to adapt to its input but for hidden layers, things get a bit complicated.
- The input's statistical distribution changes after a few iterations, so if the input statistical distribution keeps changing, called **internal covariate shift**, the hidden layers will keep trying to adapt to that new distribution hence slowing down convergence. It is like a goal that keeps changing for hidden layers.

# Batch Normalization

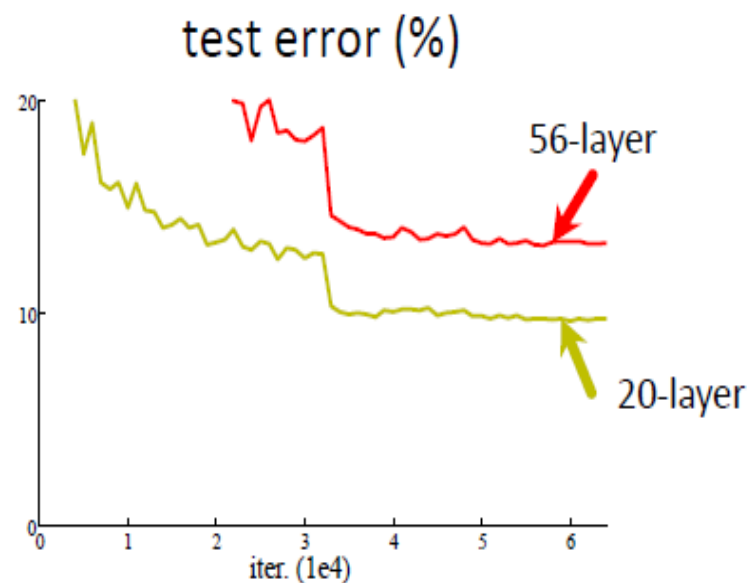
- Batch Normalization
  - [Example Code](#)

# RESNET

---

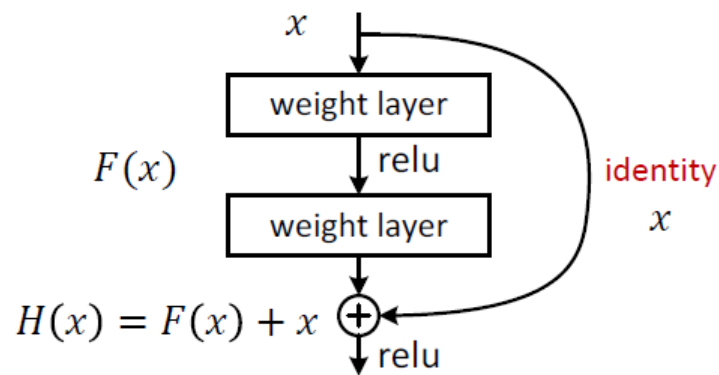
# Degradation Problem

- Overly deep plain nets have higher training error



# ResNet

- Very Deep Network + Batch Normalization + Skip connection
- Skip connection (Residual Learning)
  - Identity mapping shortcut connections
  - If identity were optimal, easy to set weight as 0
  - If optimal mapping is closer to identity, easier to find small fluctuations
  - Add neither extra parameter nor computational complexity





# ResNet Architecture

- **Netscope**

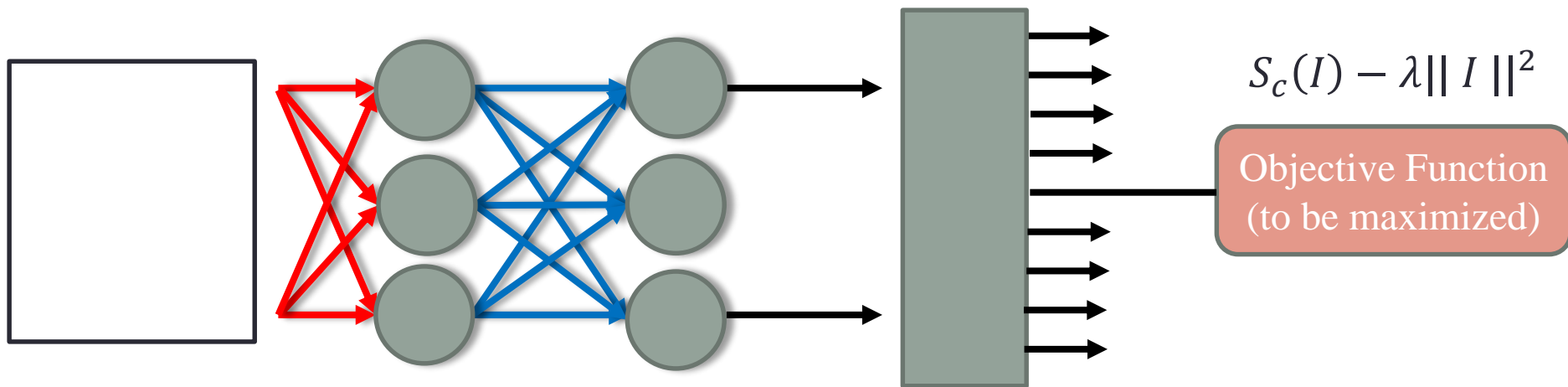
- <http://ethereon.github.io/netscope/#/gist/d38f3e60>



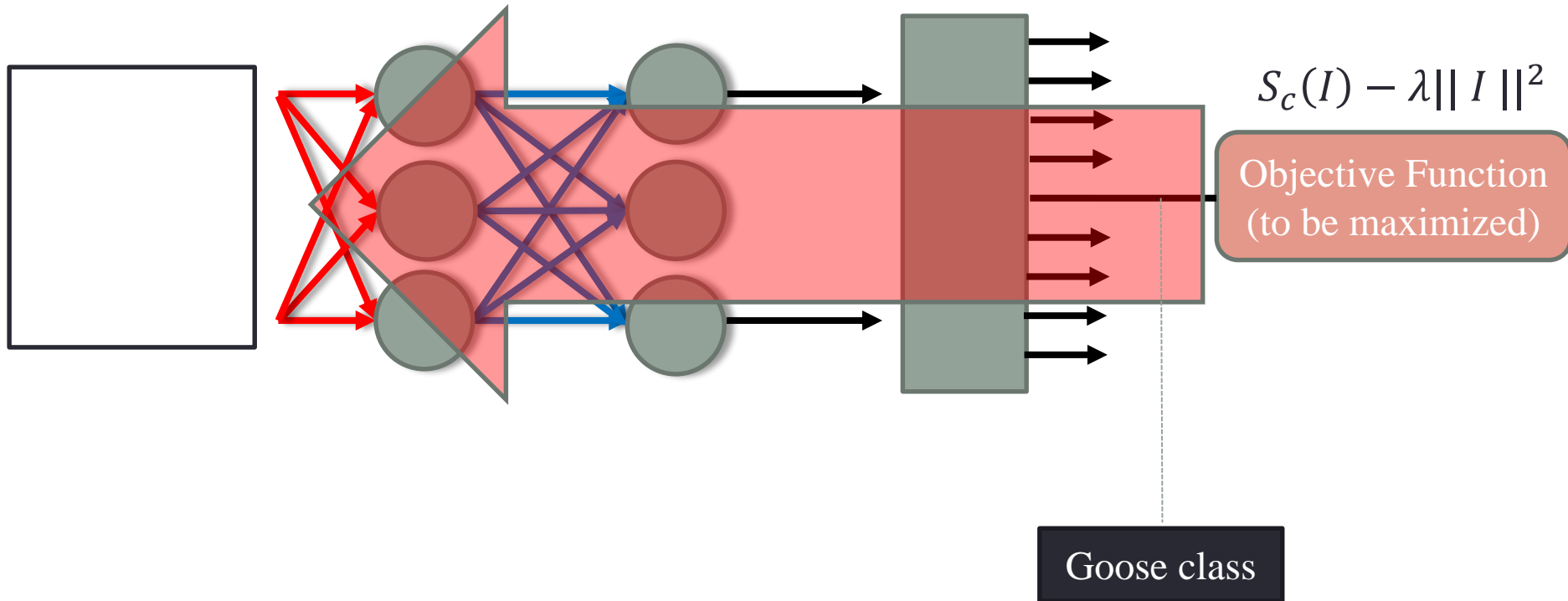
# DEEP INSIDE CONVOLUTION NETWORKS

---

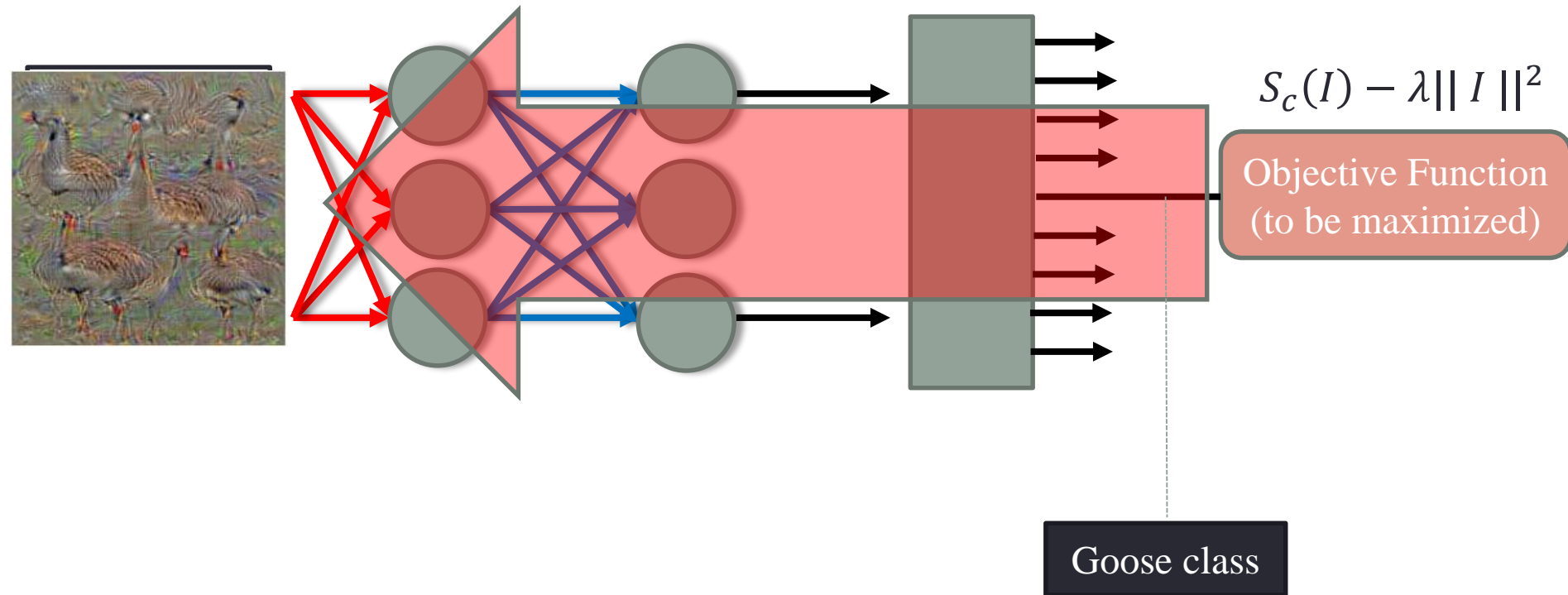
# Inputs maximizing class score



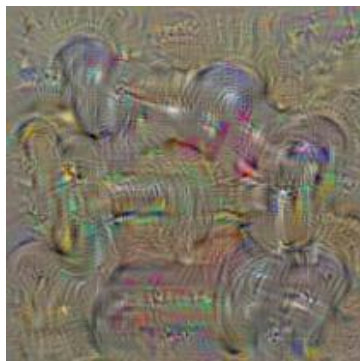
# Inputs maximizing class score



# Maximizing class score



# Inputs maximizing class score



dumbbell



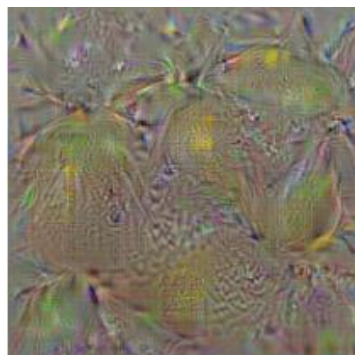
cup



dalmatian



bell pepper

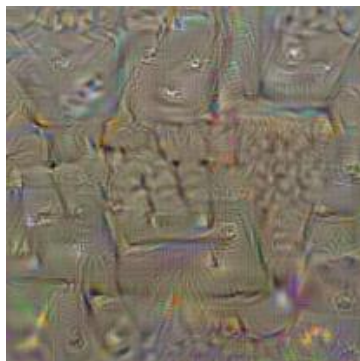


lemon

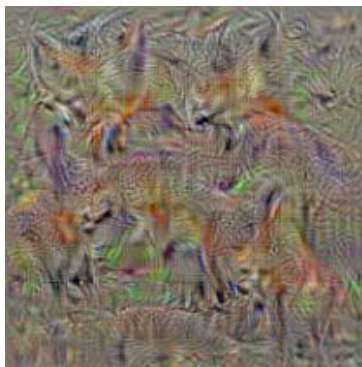


husky

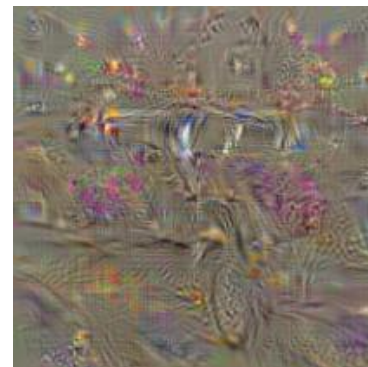
# Inputs maximizing class score



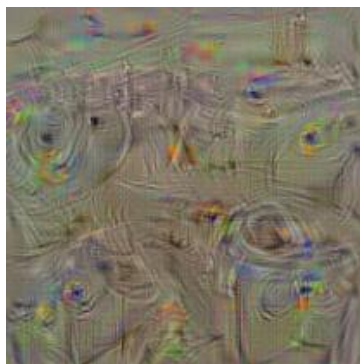
computer keyboard



kit fox



limousine



Washing machine



goose



ostrich

# SALIENCY

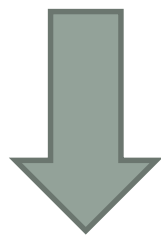
---



# Saliency visualization

- Linear score model for class  $c$ :

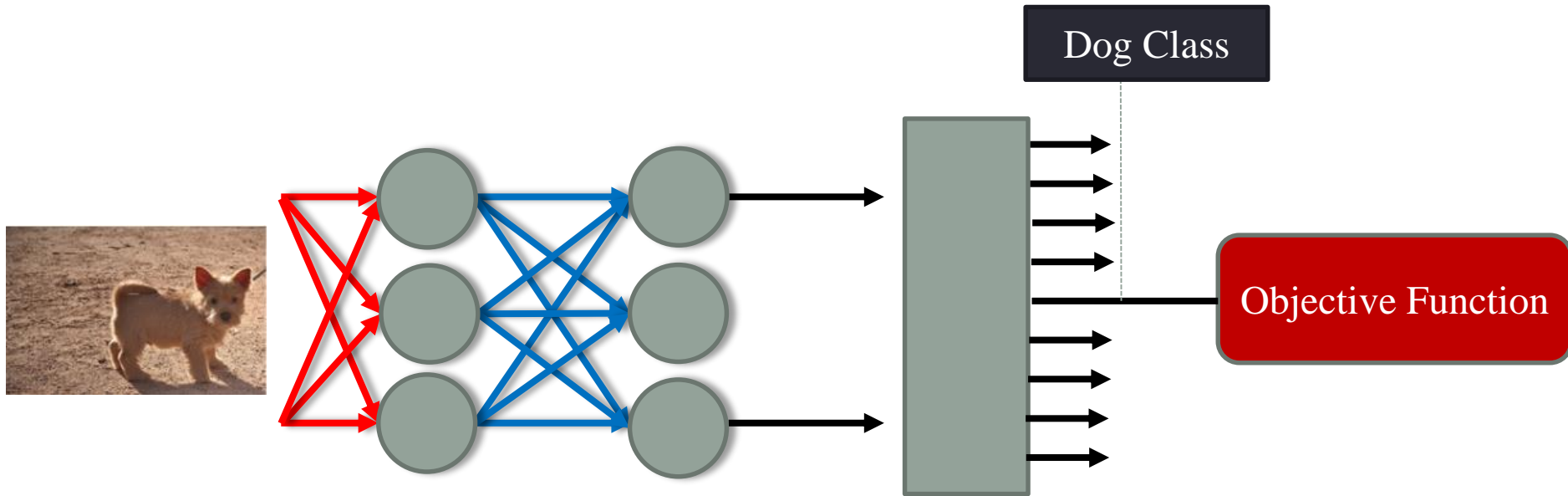
$$S_c(I) \approx wI + b$$



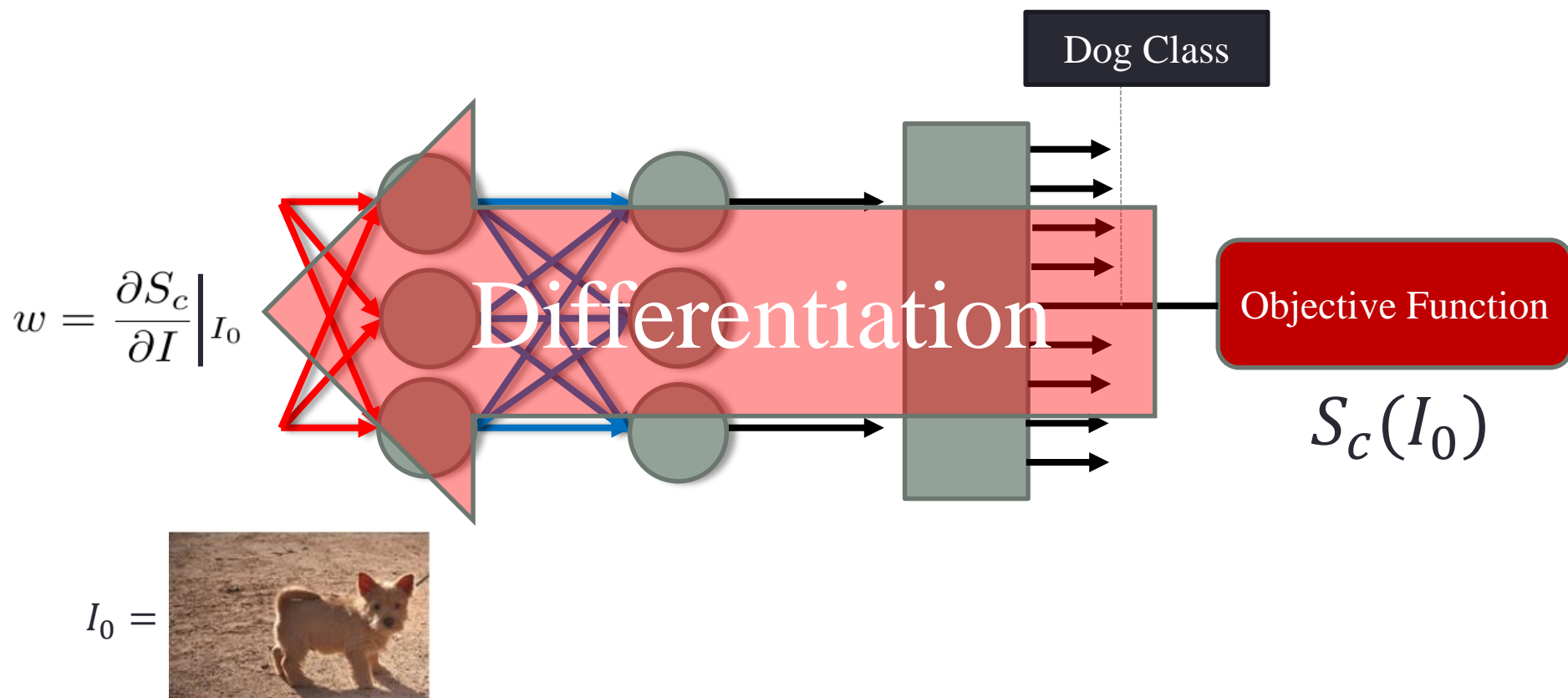
$$w = \left. \frac{\partial S_c}{\partial I} \right|_{I_0}$$

$w$  : importance of corresponding pixels of  $I$  for class  $c$

# Saliency visualization



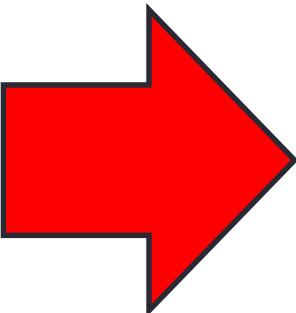
# Saliency visualization

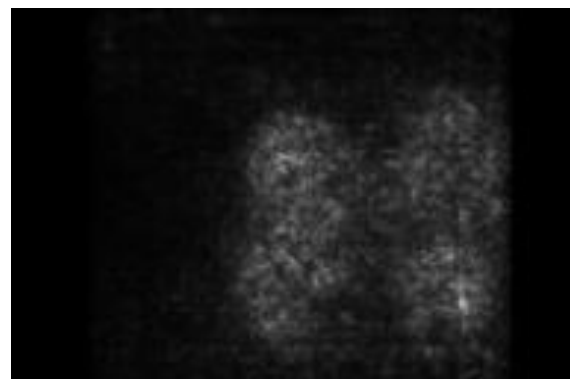


# Saliency visualization

$I_0 =$



$$w = \left. \frac{\partial S_c}{\partial I} \right|_{I_0}$$




saliency map

yacht



dog



monkey



washing machine



cow

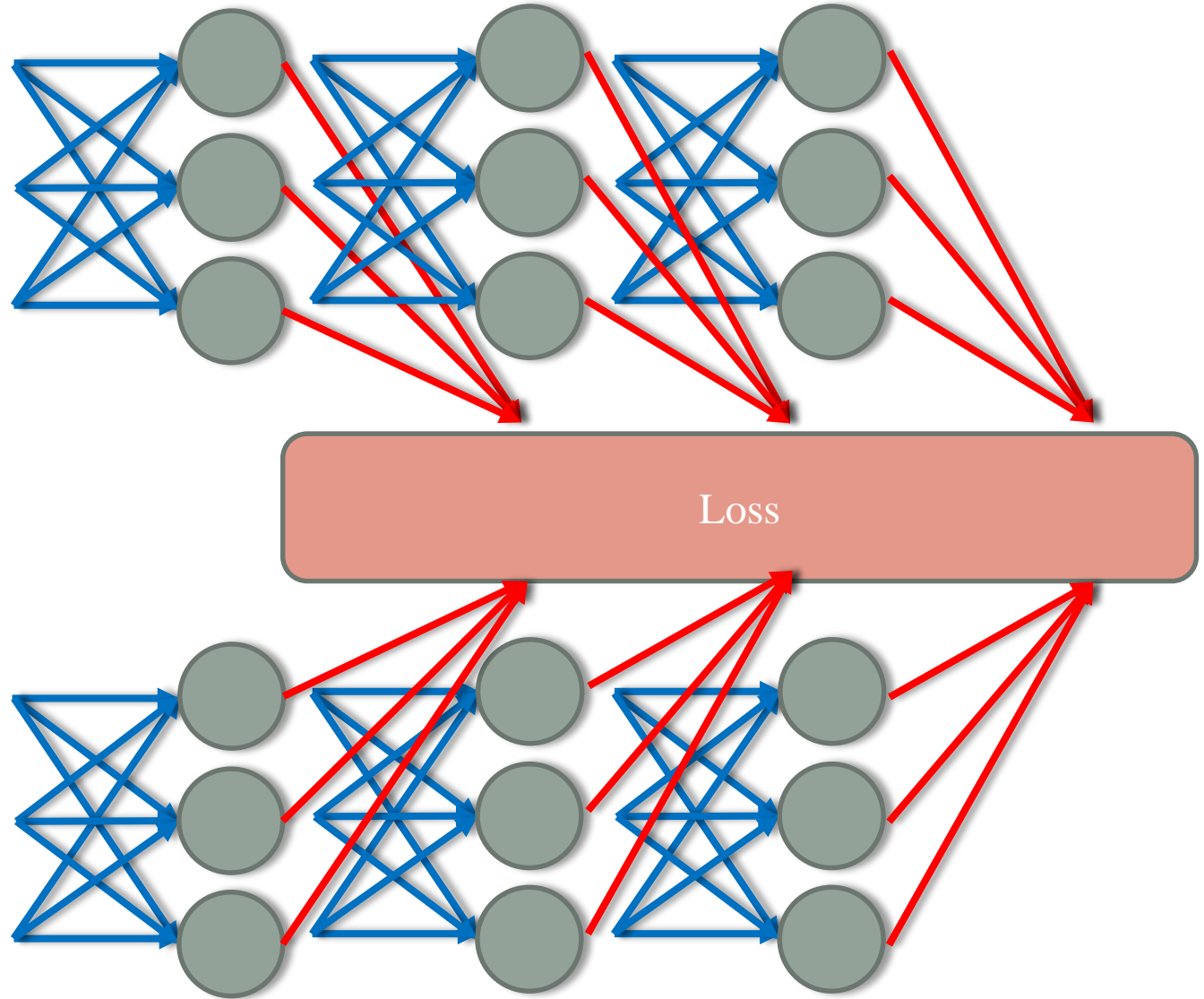
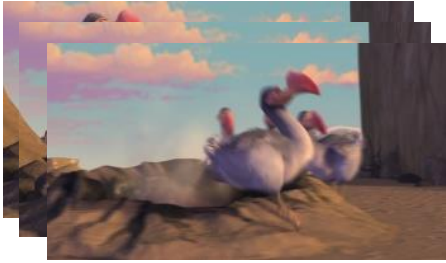


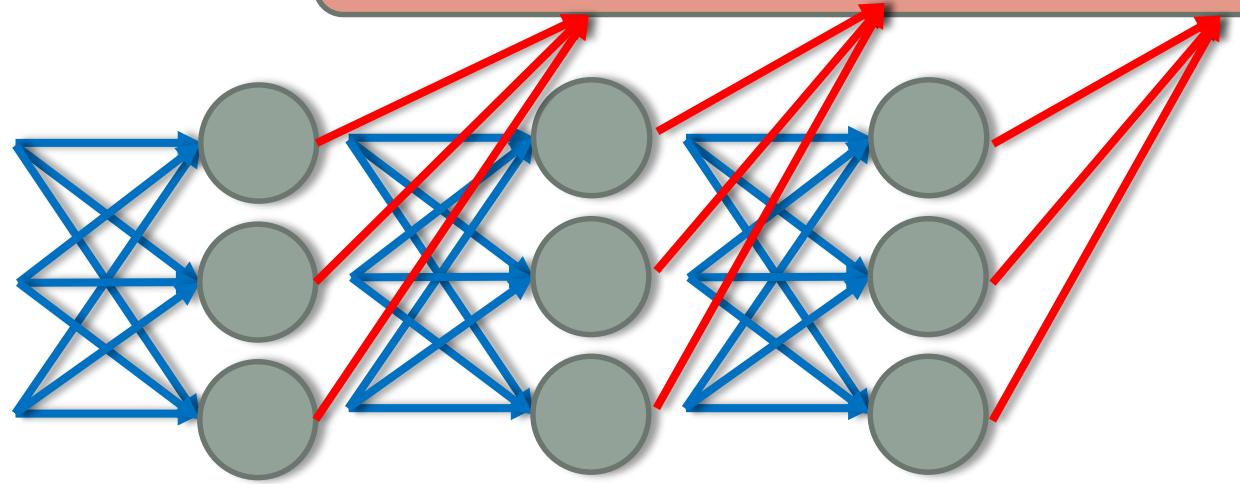
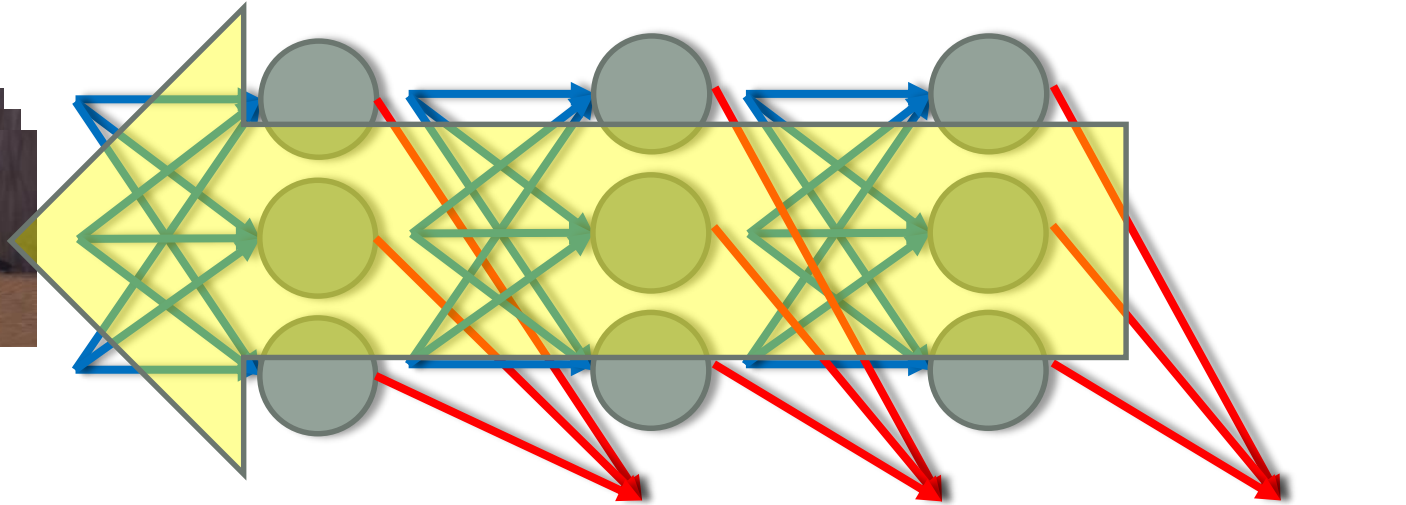
building



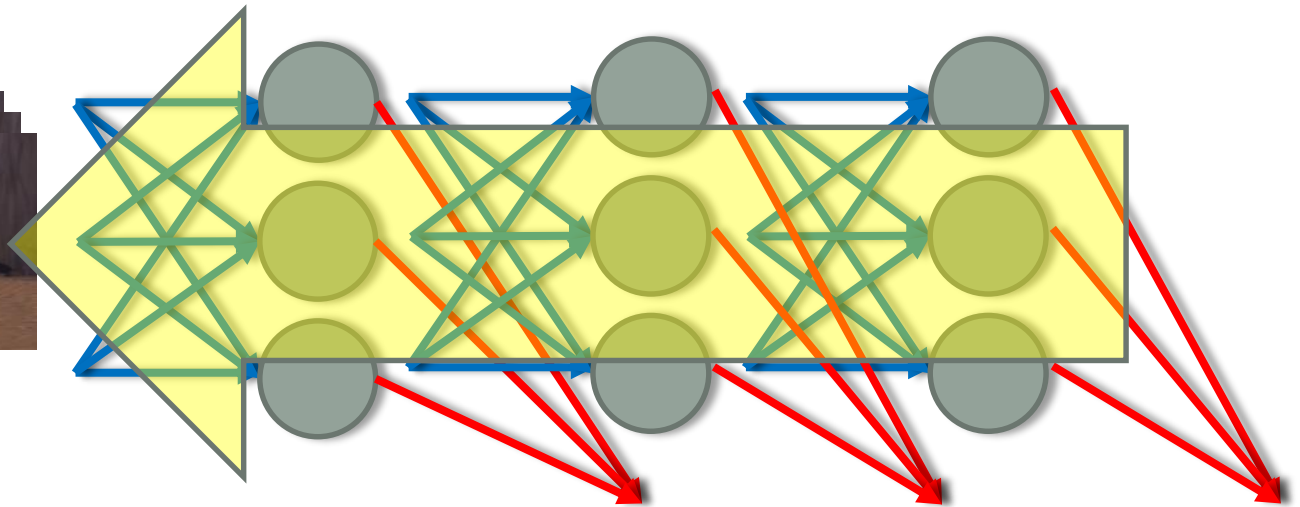
# A NEURAL ALGORITHM OF ARTISTIC STYLE

---

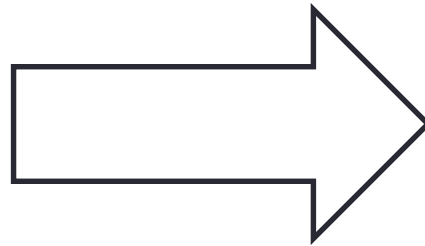




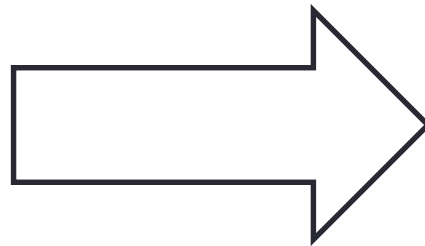
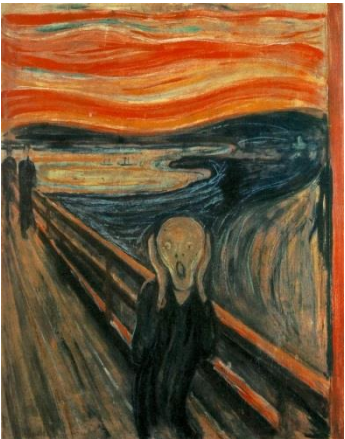




# Artistic style

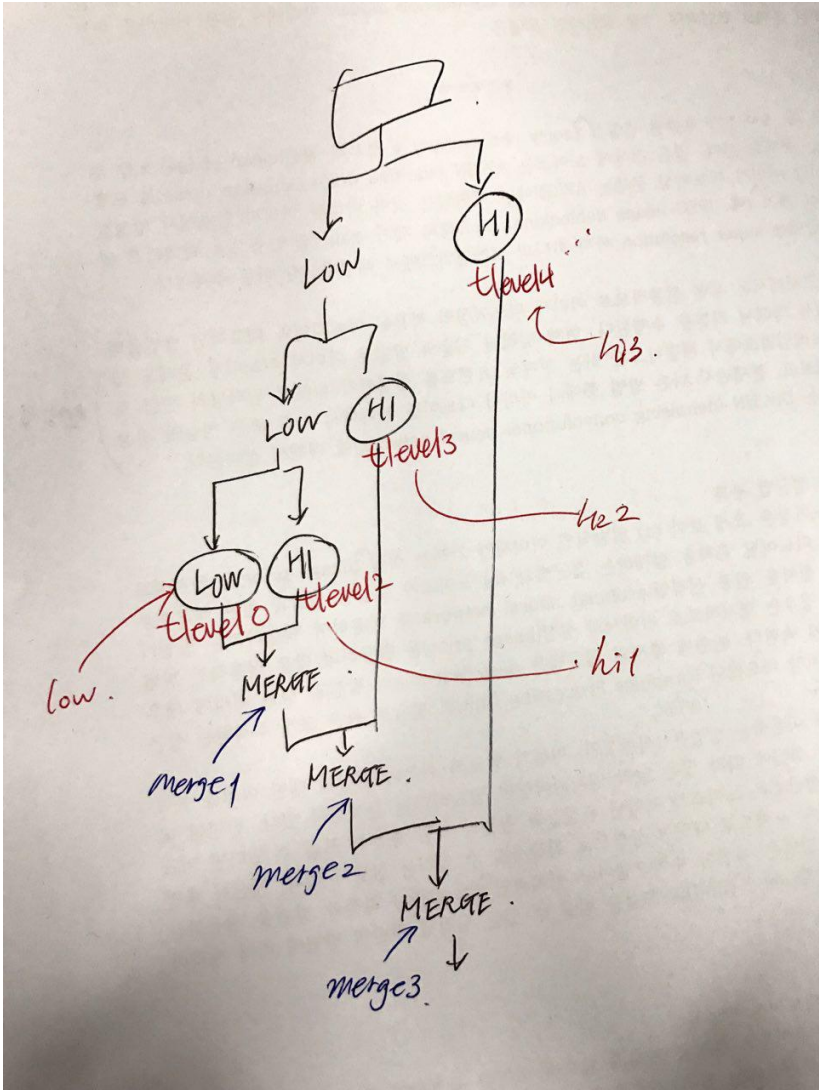


# Artistic style

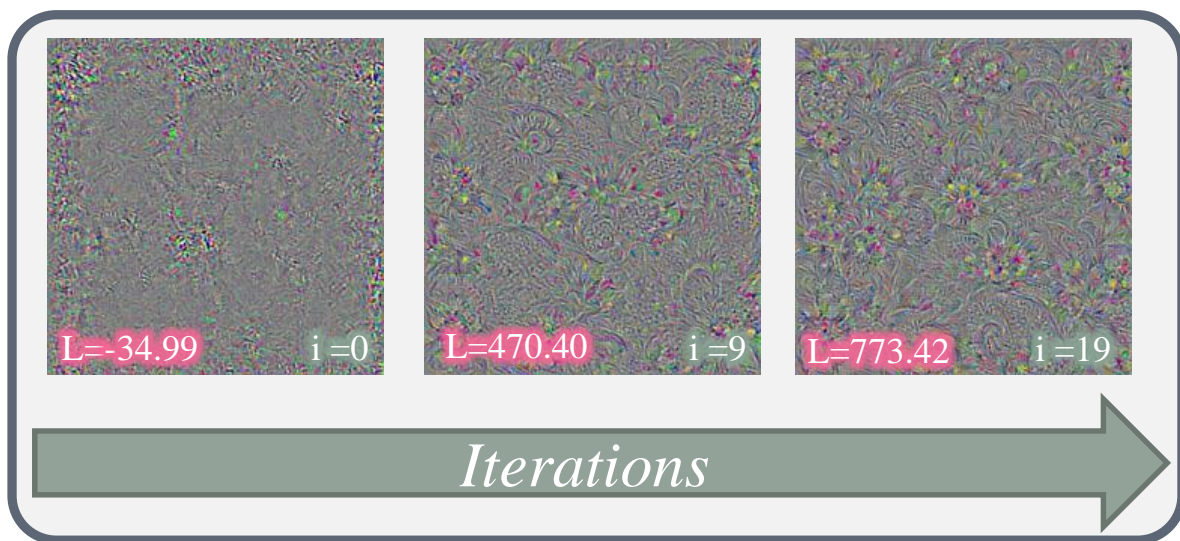


# BACKUPS

---



# Objective function values



L=102.42

# CAFFE 코드

---

# Example

- [Neural Network](#)
- [LeNet example](#)
- [Non-image example](#)



# Netscope

- <http://ethereon.github.io/netscope/quickstart.html>