SK hynix i-TAP
반도체 Data Scientist를 위한
ML/DL 심화 커리큘럼

3강 Network Visualization

Ernest K. Ryu (류경석)

2020.11.20

3강. Network Visualization

- Batch Normalization
- Residual Network
- Network visualization

Batch Normalization

- In many classical methods involving data, the first step is to normalize data to have zero mean and unit variance.
  - Step 1. Compute $\hat{\mu} = \frac{1}{N}\sum_{i=1}^{N} X_i$, $\widehat{\sigma^2} = \frac{1}{N}\sum_{i=1}^{N}(X_i - \hat{\mu})^2$

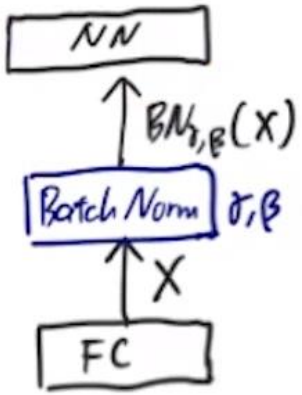
$$\hat{X}_i = \frac{X_i - \hat{\mu}}{\sqrt{\widehat{\sigma^2} + \varepsilon}}$$

  - Step 2. Run method with data $\hat{X}_1, \ldots, \hat{X}_N$

- Batch normalization n (BN) (sort of) enforces this normalization layer-by-layer. (Ioffe & Szegedy 2015)
- BN has become an indispensable tool for training very deep neural networks.
- Theoretical justification for BN is weak.

BN for Fully Connected Layer

nn.BatchNorm1d



Input: X (batch size) $\times$ (# entries)

Output: $BN_{\gamma,\beta}(X)$ Shape($BN_{\gamma,\beta}(X)$) = Shape(X)

$BN_{\gamma,\beta}$ acts independently over entries

$$\hat{\mu}[:] = \frac{1}{B}\sum_{b=1}^{B} X[b,:]$$

$$\hat{\sigma}[:] = \frac{1}{B}\sum_{b=1}^{B} (X[b,:] - \hat{\mu}[:])^2 + \epsilon$$

learned standard dev. and mean parameters

$\circledast$ elementwise mult

$$BN_{\gamma,\beta}(X) = \gamma \circledast \frac{X - \hat{\mu}}{\hat{\sigma}} + \beta$$

$$BN_{\gamma,\beta}(X)[b,:] = \gamma[:] \circledast \frac{X[b,:] - \hat{\mu}[:]}{\hat{\sigma}[:]} + \beta[:] \quad b=1,\dots,B$$

BN normalizes X and controls the mean and variance through learned parameters γ and β.

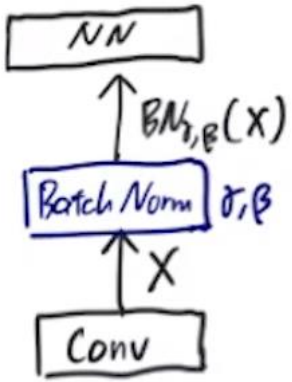
BN for Convolutional Layer

nn.BatchNorm2d

Assuming translation invariance, each batch, horizontal pixel, and vertical pixel is like an IID sample

Input: X (batch size) × (# channels) × (# horizontal pixels) × (# vertical pixels)

Output: $BN_{\gamma,\beta}(X)$ Shape $(BN_{\gamma,\beta}(X)) = $ Shape (X)



$BN_{\gamma,\beta}$ acts independently over channels

$$\hat{\mu}[:] = \frac{1}{BPQ}\sum_{b=1}^{B}\sum_{i=1}^{P}\sum_{j=1}^{Q} X[b,:,i,j]$$

$$\hat{\sigma}^2[:] = \frac{1}{BPQ}\sum_{b=1}^{B}\sum_{i=1}^{P}\sum_{j=1}^{Q} (X[b,:,i,j] - \hat{\mu}[:])^2 + \epsilon$$

$$BN_{\gamma,\beta}(X) = \gamma \circledast \frac{X-\hat{\mu}}{\hat{\sigma}} + \beta$$

learned standard dev. and mean parameters

$\circledast$ elementwise mult

$$BN_{\gamma,\beta}(X)[b,:,i,j] = \gamma[:] \circledast \frac{X[b,:,i,j] - \hat{\mu}[:]}{\hat{\sigma}[:]} + \beta[:]$$

$b = 1,\dots,B$
$i = 1,\dots,P$
$j = 1,\dots,Q$

Batch Norm During Prediction

- The $\hat{\mu}$ and $\hat{\sigma}$ are estimated from batches during training.
- During testing, where we evaluate the NN without changing it, $\hat{\mu}$ and $\hat{\sigma}$ are fixed.
- 2 strategies for computing final values of $\hat{\mu}$ and $\hat{\sigma}$:
  1) After training, fix weights and evaluate NN on full training set to compute $\hat{\mu}$ and $\hat{\sigma}$ layer-by-layer. (Computation of $\hat{\mu}$ and $\hat{\sigma}$ for initial layers must be done first.)
  2) During training, compute running average of $\hat{\mu}$ and $\hat{\sigma}$. This is the default behavior of PyTorch.
- In PyTorch, use model.train() and model.eval() to switch BN behavior between training and testing.

Discussion of Batch Normalization

- With BN, the choice of batch size becomes more important.
- For some reason, BN seems to subsume the regularization of Dropout. Since BN has been popularized, Dropout is used less often. (Not understood very well.)
- BN is indispensable in practice, but has very little theoretical justification.

Discussion of Batch Normalization

- In the original paper, Ioffe and Szegedy offered the mitigation of “internal covariate shift” but (Santukar et al. 2018) showed that BN worsens the internal covariate shift and yet improves the training process.

- Remember that BN has trainable parameters

Batch Norm has Trainable Parameters

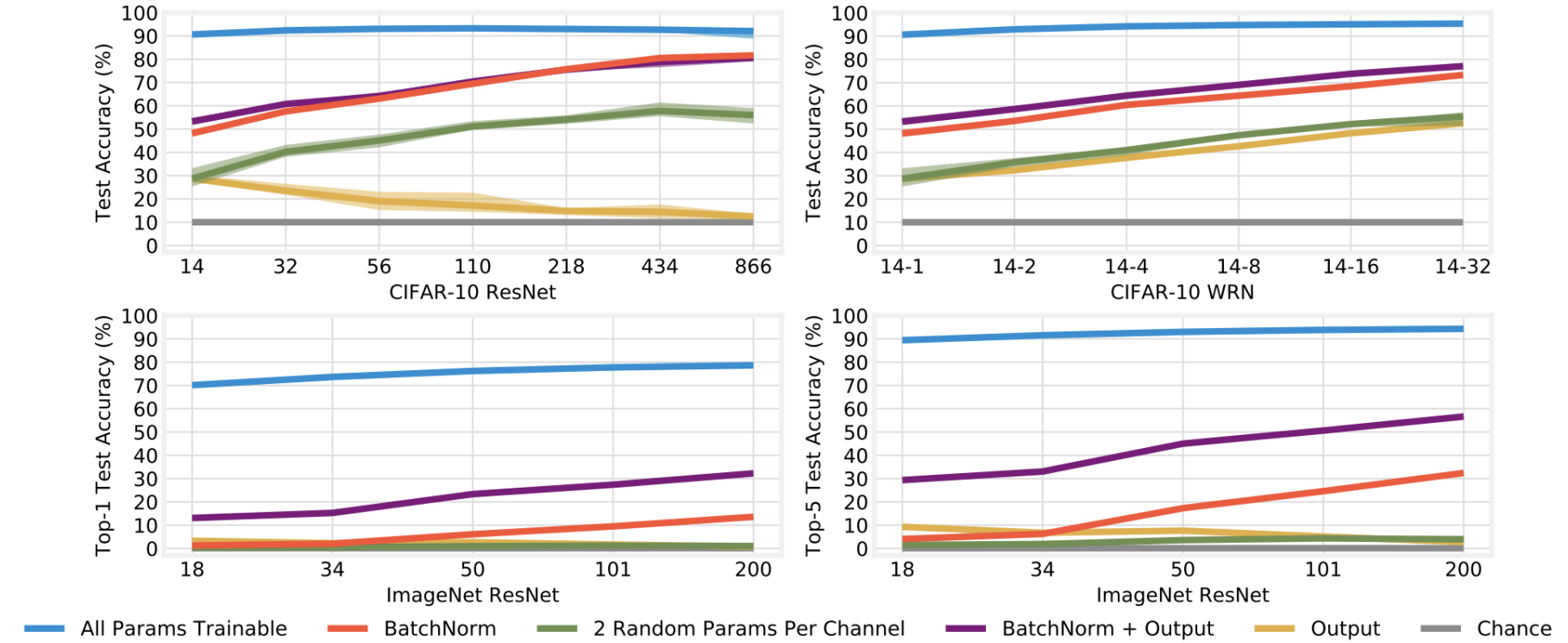
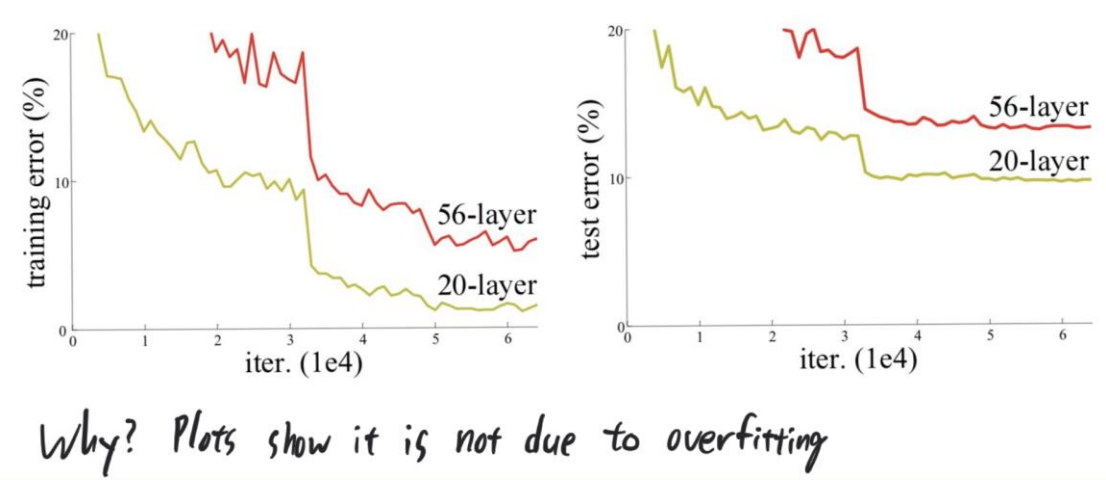


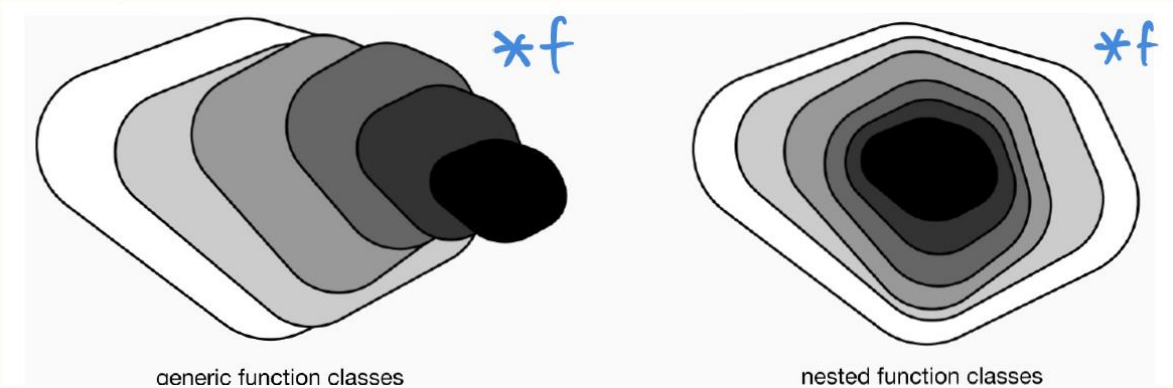
Figure 2: Accuracy of ResNets for CIFAR-10 (top left, deep; top right, wide) and ImageNet (bottom left, top-1 accuracy; bottom right, top-5 accuracy) with different sets of parameters trainable.

Frankle, Schwab, Morcos, Training BatchNorm and Only BatchNorm: On the Expressive Power of Random Features in CNNs, NeurIPS SEDL Workshop, 2019.

Residual Network (ResNet)

- Winner of 2015 ImageNet Challenge (Kaiming He et al. 2015)
- Observation: Excluding the issue of computation cost, more layers it not always better



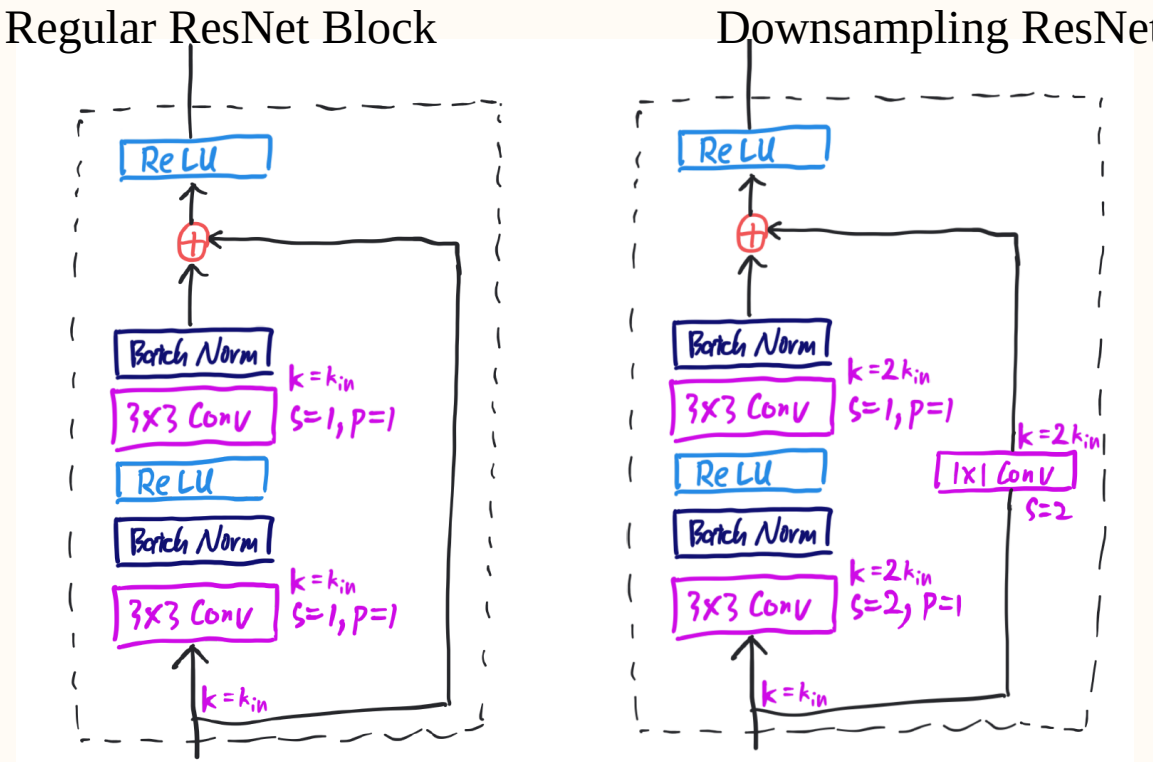


- Hypothesis 1: Deeper networks are harder to train.
  Is there a way to train a shallow network and embed it in a deeper network?
- Hypothesis 2: The deeper networks may be worst approximations of the true unknown function. Find an architecture representing a strictly increasing function class as a function of depth.

He, Zhang, Ren, Sun, Deep Residual Learning for Image Recognition, CVPR, 2016.

ResNet Blocks

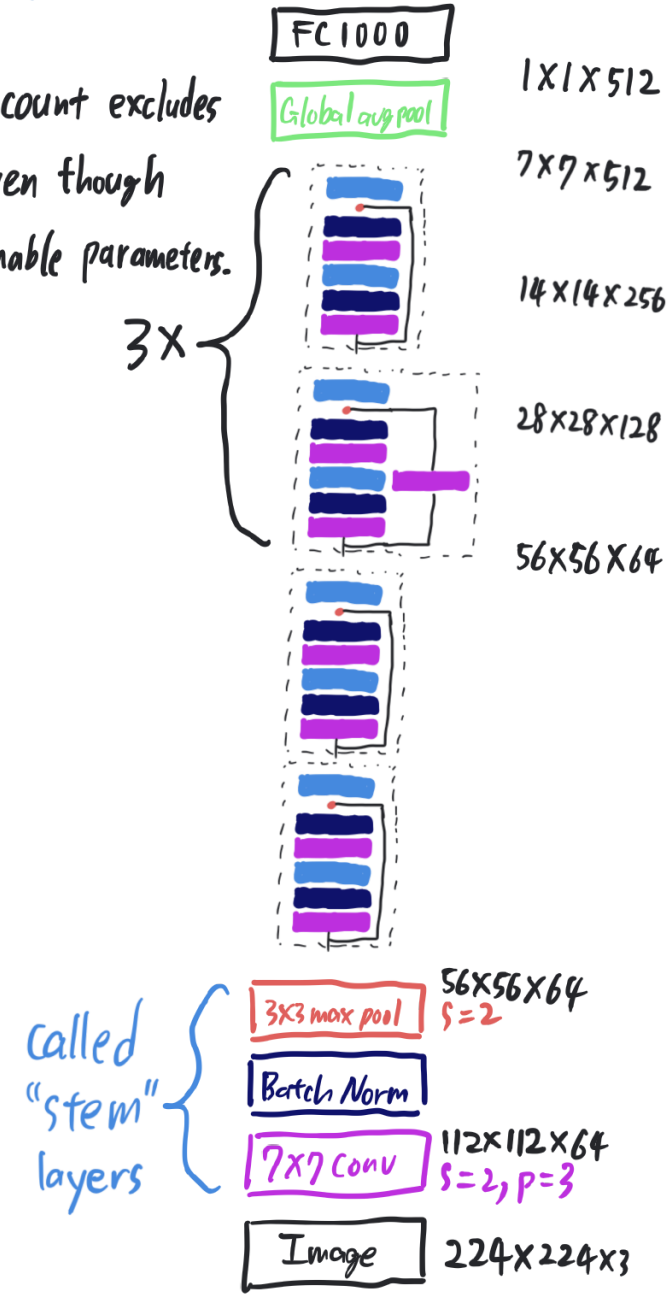
- Use a residual connection so that [all weights=0] correspond to [block=identity]



- Residual connection requires sptial dimension and number of channels to be preserved
- Downsampling version uses 1×1 convolution in residual to modify the number of channels and spatial resolutions.

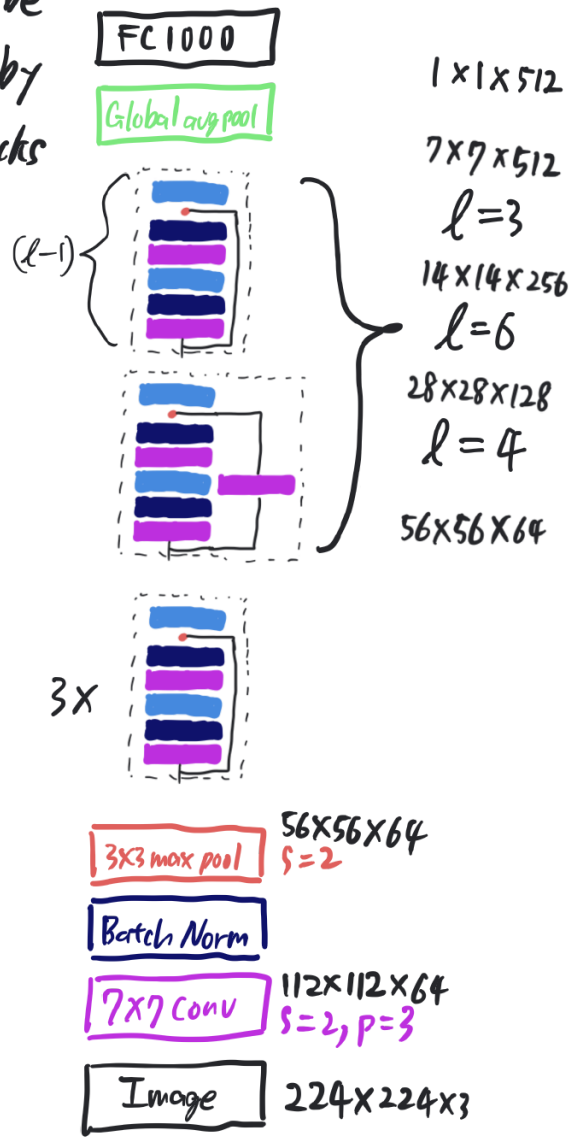
ResNet 18

- The layer count excludes batch norm even though BN has trainable parameters.



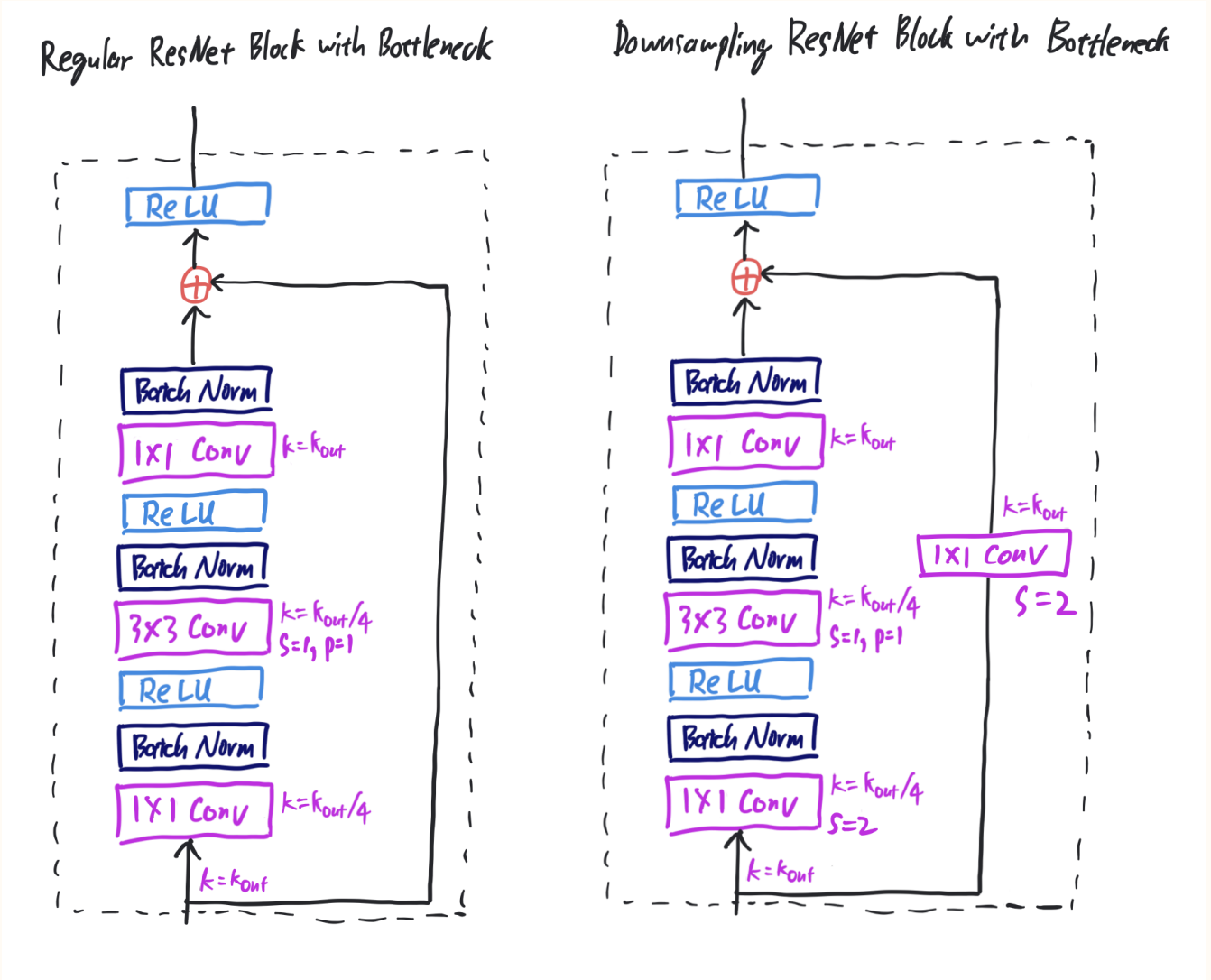
ResNet 34

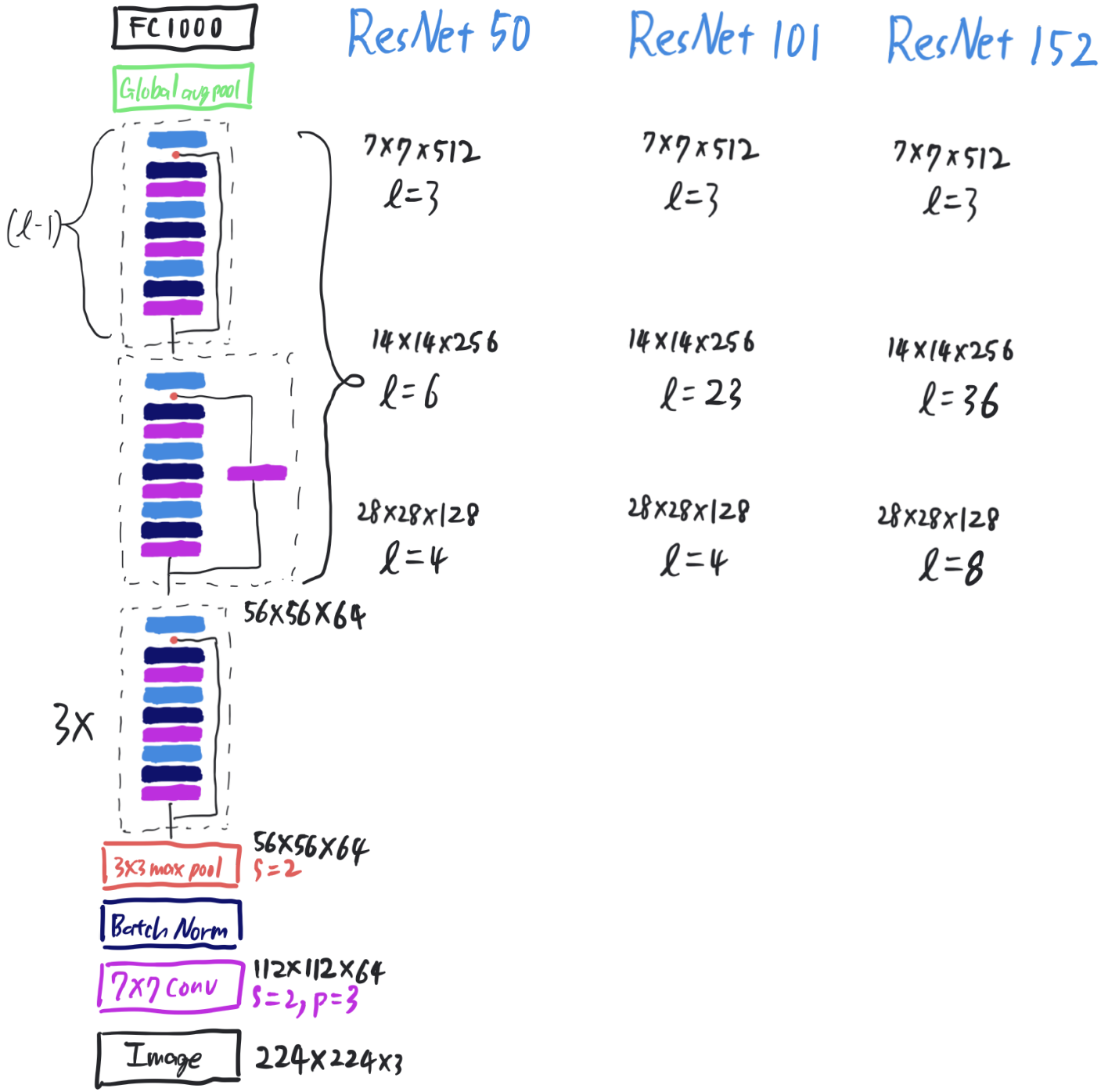
- Shallower ResNet can easily be embedded in deeper network by setting weights of additional blocks to zero.



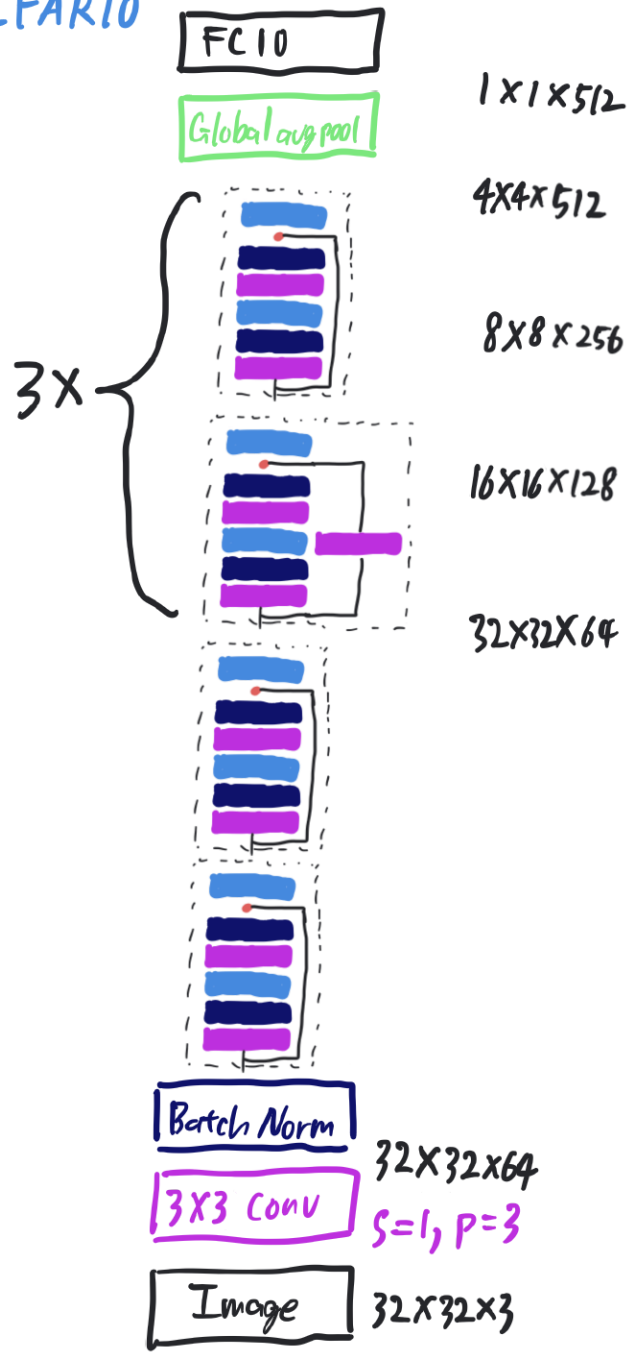
ResNet Blocks for Deeper ResNets

Use 1×1 "bottleneck" convolutions as in GoogLeNet

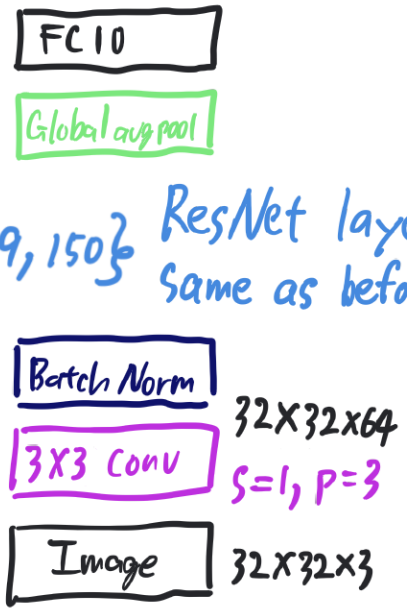



FC1000
Global avg pool
(l-1)
7x7x512
l=3
14x14x256
l=6
28x28x128
l=4
56x56x64
3x
56x56x64
3x3 max pool
S=2
Batch Norm
112x112x64
7x7 conv
S=2, p=3
Image
224x224x3
ResNet 50
ResNet 101
ResNet 152
7x7x512
l=3
14x14x256
l=23
28x28x128
l=4
7x7x512
l=3
14x14x256
l=36
28x28x128
l=8

ResNet 18 for CIFAR10



ResNet {34, 50, 101, 152} for CIFAR10



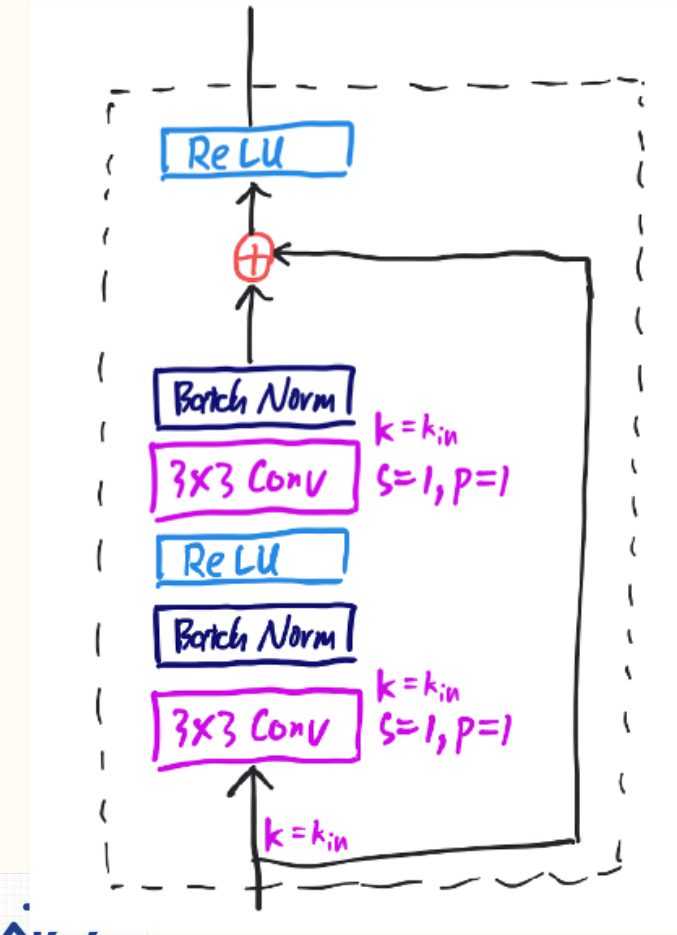
{32, 48, 99, 150} ResNet layers same as before

ResNet Variants: ResNet V1.5

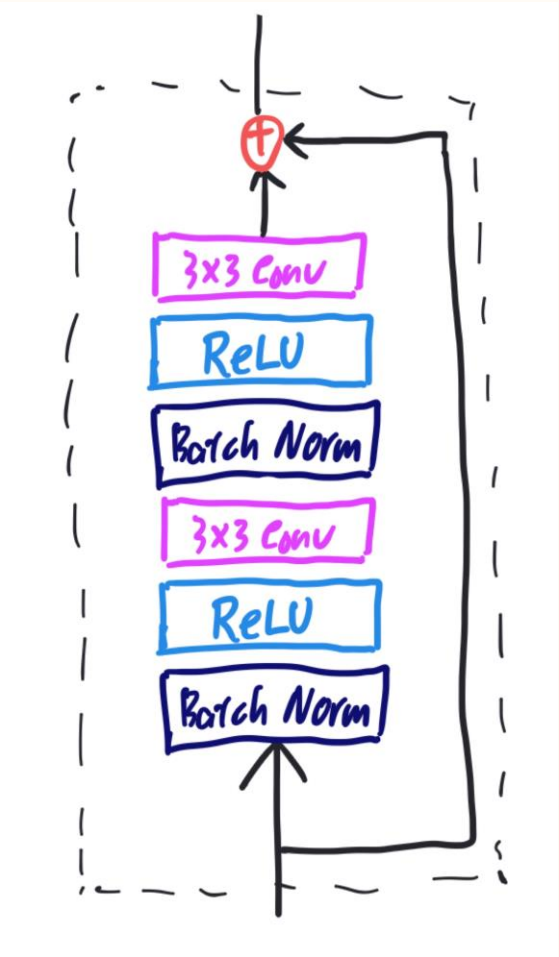
In the bottleneck blocks which requires downsampling, v1 has stride = 2 in the first 1x1 convolution, whereas v1.5 has stride = 2 in the 3x3 convolution.

ResNet Variants: conv-BN-ReLU vs. BN-ReLU-conv

Original Residual block
With conv-BN-ReLU



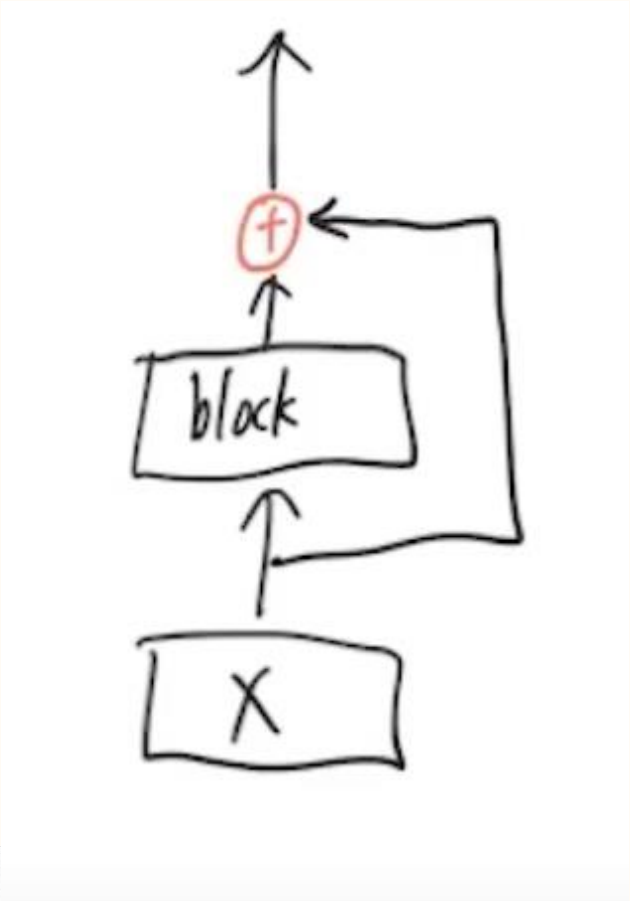
Modifies Residual block
With BN-ReLU-conv



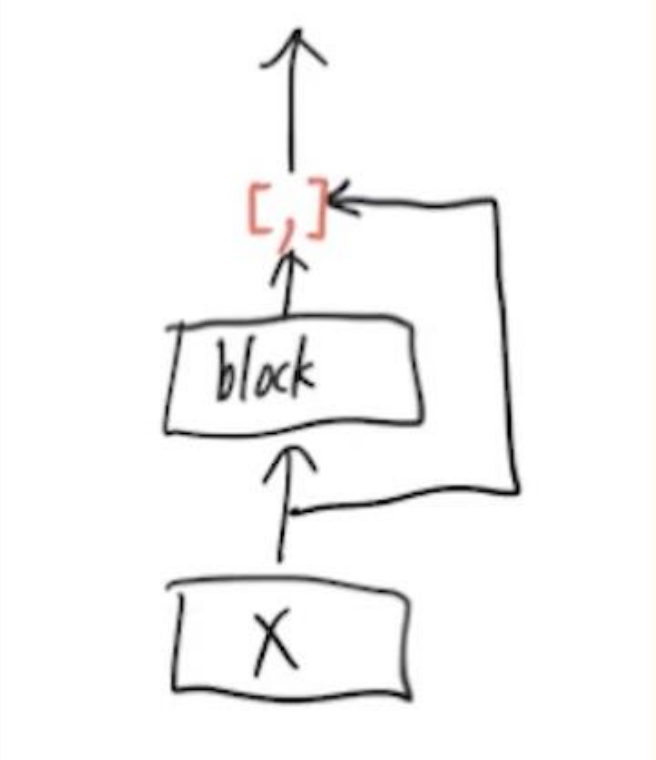
He, Zhang, Ren, Sun, Identity Mappings in Deep Residual Networks, ECCV, 2016.

DenseNet (2017 CVPR, Best Paper Award)

ResNet blocks have the form



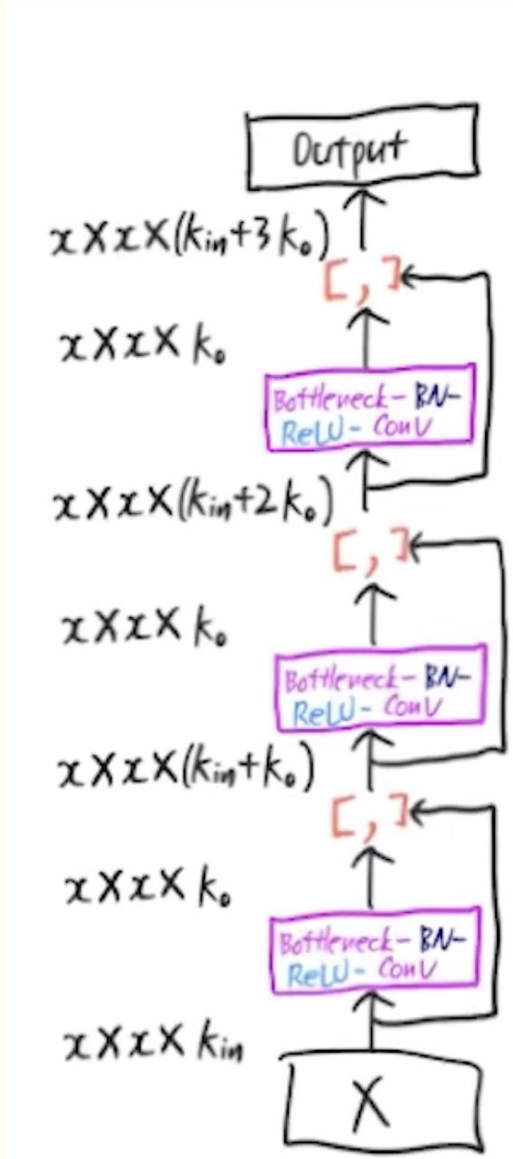
Instead of addition, what if we use concatenation?



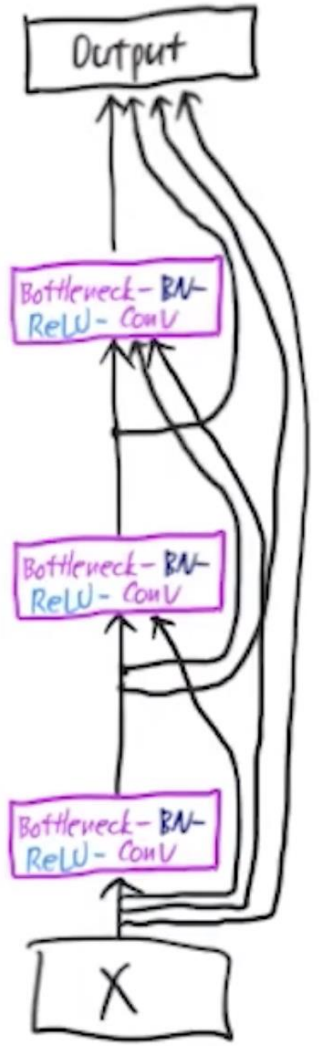
The subsequent layer can decide to add the concatenated channels and thereby replicate ⊕ or it can do other things.

Dense Block

- DenseNet's building block continues the concatenation.
- Use bottleneck layer = (BN-ReLU-1×1 conv, $k = 4k_0$)
- $k_0 = 32$ is called the growth rate
- Use BN-ReLU-conv instead of conv-BN-ReLU

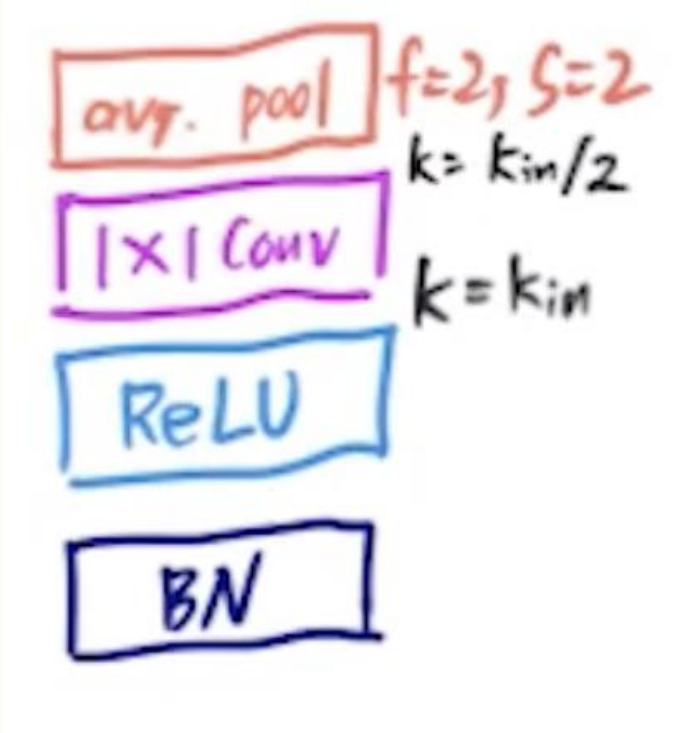


People also illustrate the block with more arrows. The name comes from the denseness of the arrows

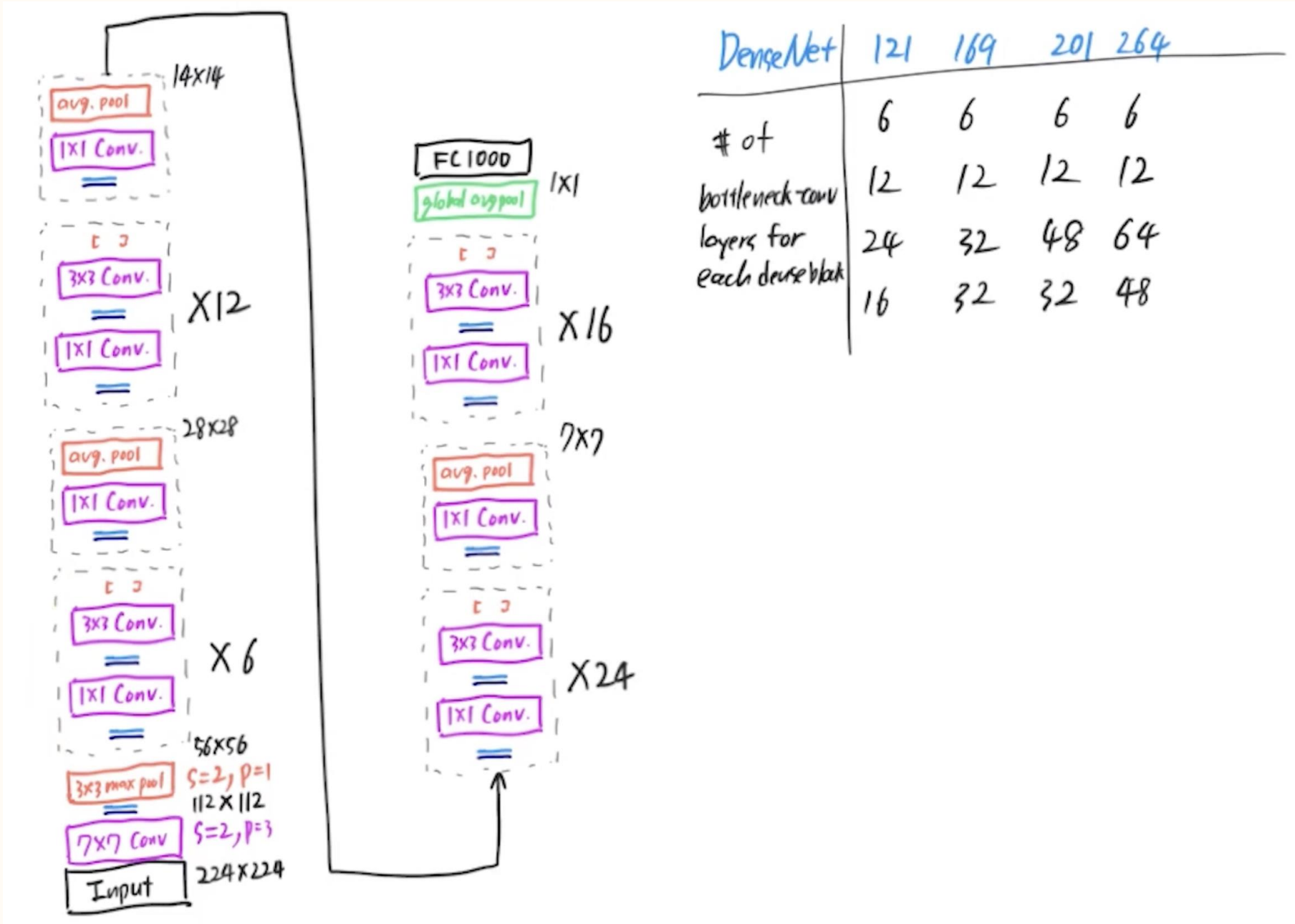


Transition Layer

- Each layer in a dense block increases the number of channels.
- The transition layers reduce the number of channels with a 1×1 convolution and reduces the spatial dimension with a pooling layer.



DenseNet 121



DenseNet	121	169	201	264
# of bottleneck conv layers for each dense block	6	6	6	6
	12	12	12	12
	24	32	48	64
	16	32	32	48

Section: Saliency Map and Attribution

Identify which part of an image affects the class score

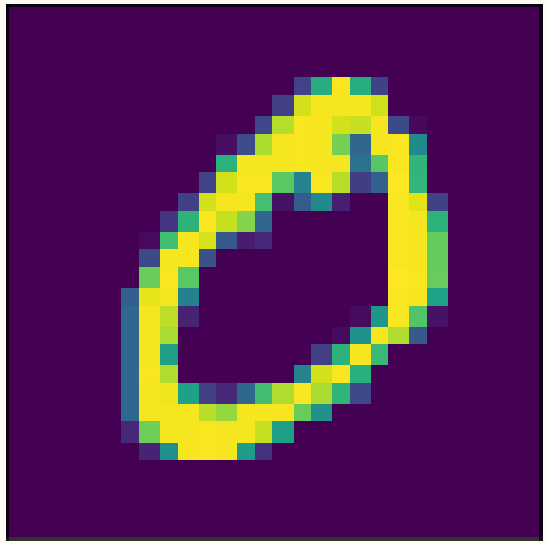
Saliency Map via Occlusion

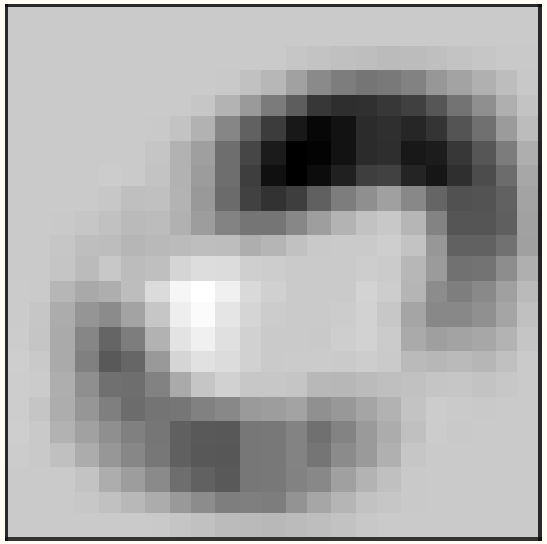
Key idea: Given an image, occlude (cover up) parts of the image and see how it affects the class score. The regions which affect the class score the most should be most likely to be salient (important).

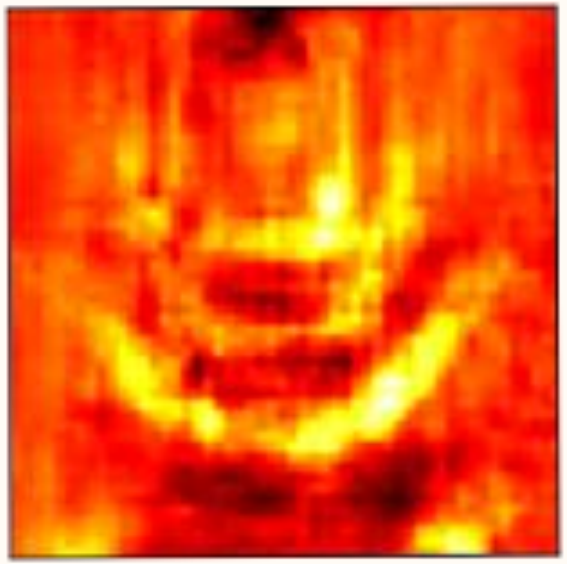
Pseudo-code

1. Load pre-trained network
2. Load data
3. Load a single image (call “(input,label)”)
4. Set k
   for i,j=1,…,28-k
   Occluded_input = input
   Occluded_input[i:i+k,j:j+k] = mean(input)
   saliency_map[i,j]=class_scores[label]
5. Plot saliency_map

Saliency Map via Occlusion Results







Saliency via Backprop

Key idea: Given an image and a network, fix the network and backprop on the class score with respect to the image.

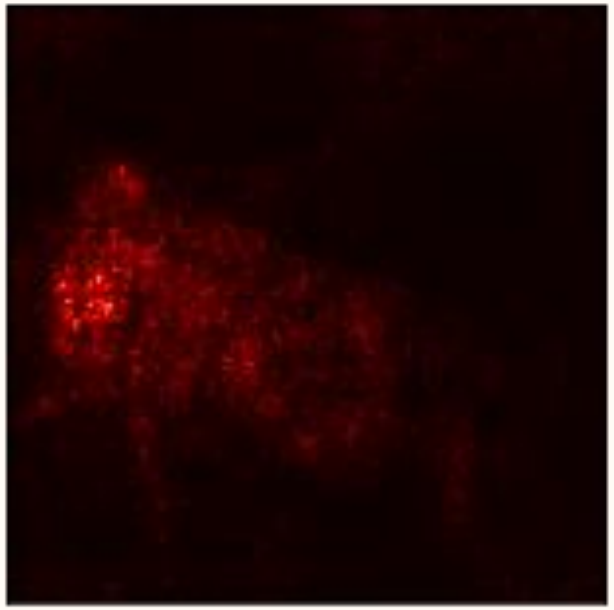
$$\frac{\partial score}{\partial image}$$

The pixels with large derivatives should be most likely to be salient (important).

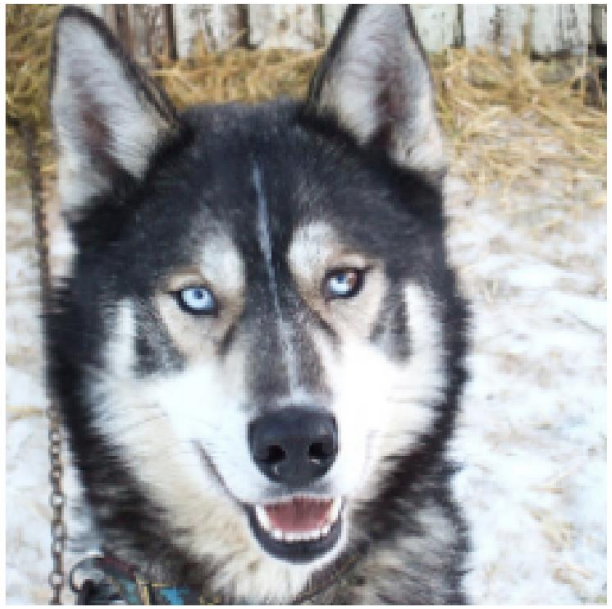
Pseudo-code

1. Load pre-trained network
2. Load data
3. Load a single image (call “(input,label)”)
4. for param in model.parameters():
       param.requires_grad = False
5. input.requires_grad = True
6. Evaluate network(input)
7. class_scores[label].backprop()
8. grad = input.grad()
9. saliency_map = abs(grad)
10. Plot saliency_map

Saliency via Backprop Results



Application of Saliency Maps: Uncovering Biases



(a) Husky classified as wolf

(b) Explanation

Sometimes, the saliency map is informative of why a network makes a certain decision

Ribeiro, Singh, Guestrin, "Why Should I Trust You?": Explaining the Predictions of Any Classifier, KDD, 2016

Feature Visualization: Model Inversion

Key idea: reconstruct image with only (a subset) of features.

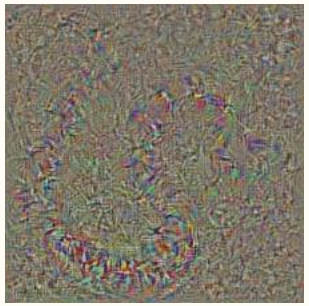
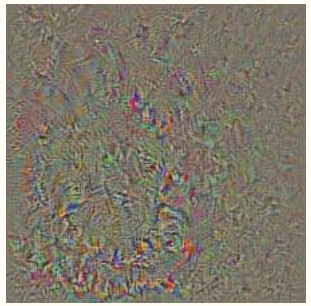
Pseudo-code

1. Load pre-trained VGG19
2. Load ImageNet data
3. Load a single ImageNet image (call “input”)
4. for param in model.parameters():
   param.requires_grad = False
5. output = randomNoise(input.size())
6. output.requires_grad = True
7. Loss = ||model(input).features() – model(output).features()||^2
8. Loss = Loss + lambda*(\beta-Tvregularizer(output))
9. Optimize (using BFGS) output image

Model Inversion Results





More Advanced Model Inversion

Key Idea:
Train network to perform inversion

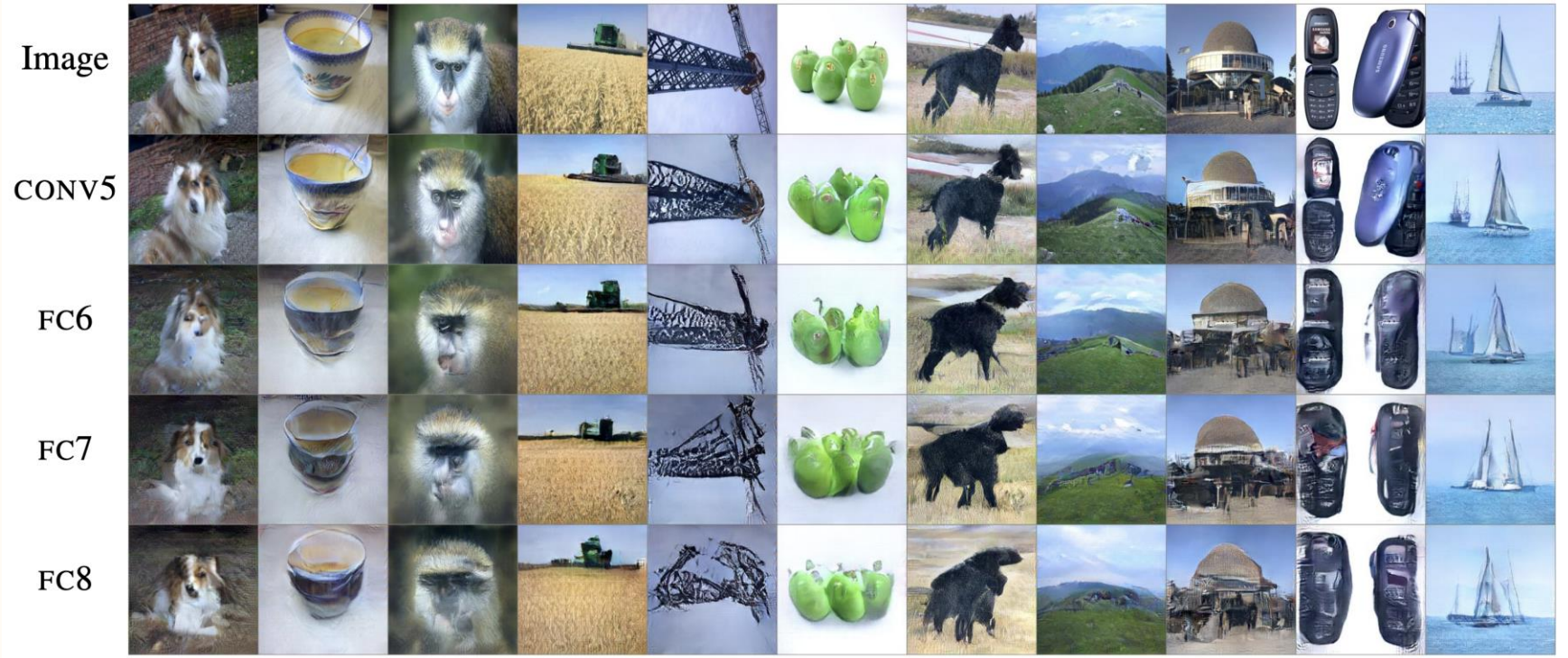


Figure 3: Representative reconstructions from higher layers of AlexNet. General characteristics of images are preserved very well. In some cases (simple objects, landscapes) reconstructions are nearly perfect even from FC8. In the leftmost column the network generates dog images from FC7 and FC8.

Section: Visualizing (Intermediate) Features

- When we wish to visualize intermediate features, we do the following.

Dataset Examples

Key Idea: Go through dataset and find images that maximally activate a certain neuron or collection of neurons.

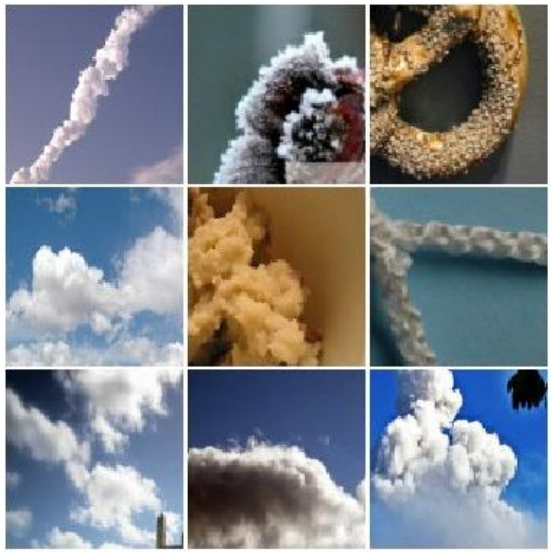
Pseudo-code

1. Load pre-trained network
2. Load data
3. Select a neuron
4. For index,image in enumerate(dataset):
   network.forward(image) #forward evaluation
   activation[index] = network[neuron]
5. Find top k (k=20 appropriate?) activation values and plot corresponding images

Dataset Examples Results

Baseball—or stripes?
mixed4a, Unit 6

Animal faces—or snouts?
mixed4a, Unit 240



Clouds—or fluffiness?
mixed4a, Unit 453

Buildings—or sky?
mixed4a, Unit 492

Improving Visualization with guided backprop

With $\sigma(x) = \mathrm{Re}LU(x)$, backprop computes

$$\frac{d}{dx}\sigma(x) = (x \geq 0)$$

$$\frac{d}{dx} f(\sigma(g(x)) = f'(\sigma(g(x))(g(x) \geq 0)g'(x)$$

In guided backprop, we instead have

$$f'(\sigma(g(x))(f'(x) \geq 0)(g(x) \geq 0)g'(x)$$

Improving Visualization with guided backprop

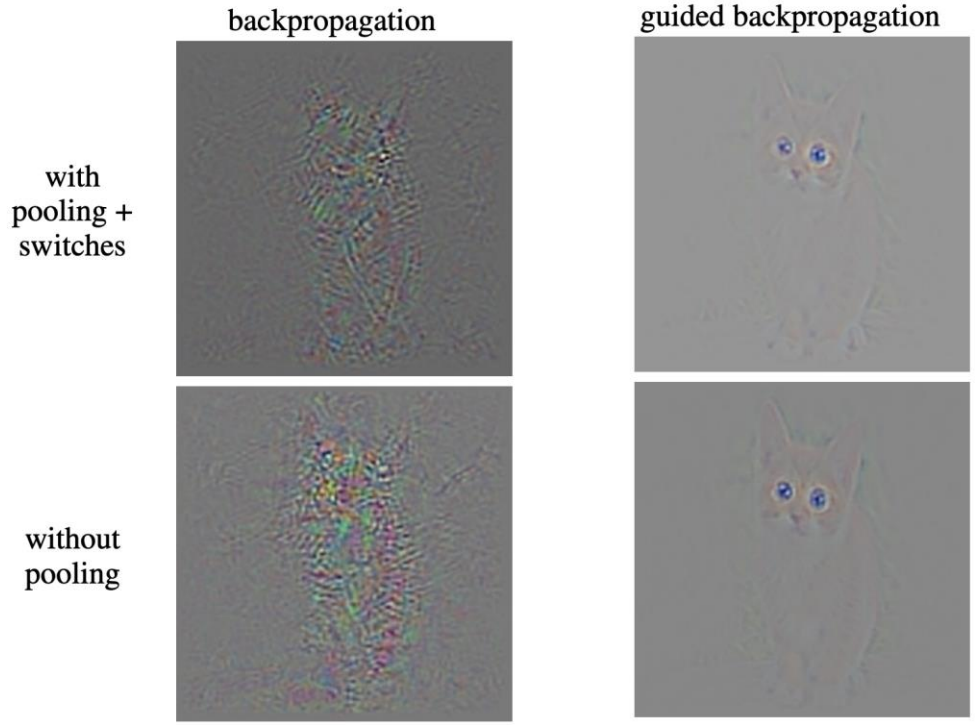


Figure 5: Visualization of descriptive image regions with different methods from the single largest activation in the pre-softmax layer global_pool of the network trained on ImageNet.

Improving Visualization with guided backprop

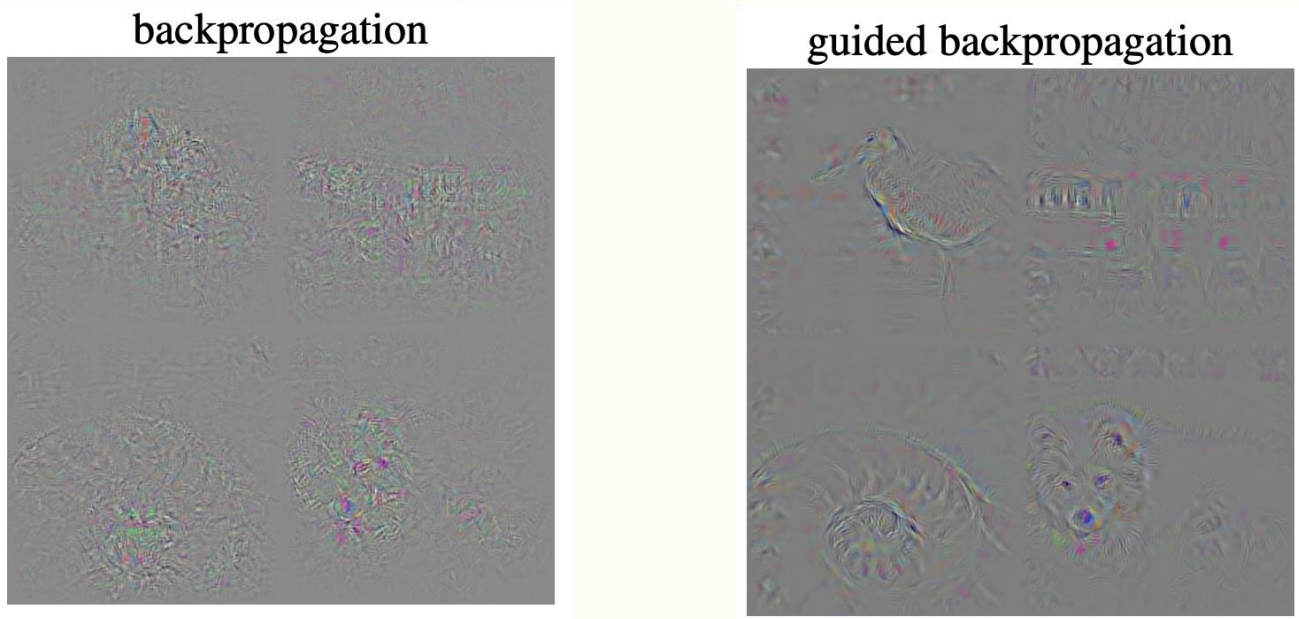


Figure 6: Visualization of descriptive image regions with different methods from the single largest activation in the last layer fc8 of the Caffenet reference network (Jia et al., 2014) trained on ImageNet. Reconstructions for 4 different images are shown.

Visualizing Intermediate Filters with (guided) backprop

Given a set of intermediate nodes $n_1, \ldots, n_k$ compute

$$\frac{d}{d\text{image}}(n_1 + \cdots + n_k)^2$$

with (guided) backprop to visualize what kind of input image excites these neurons.

Visualizing Intermediate Filters with (guided) backprop

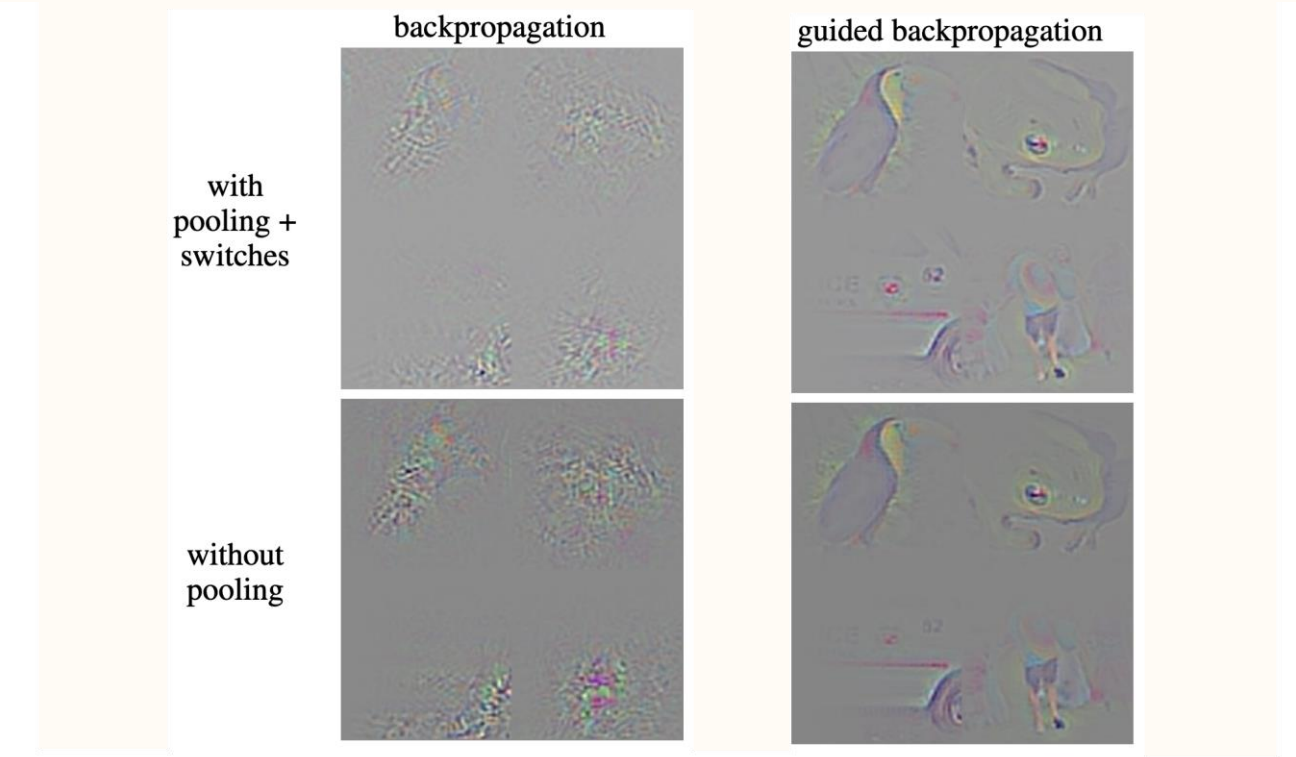
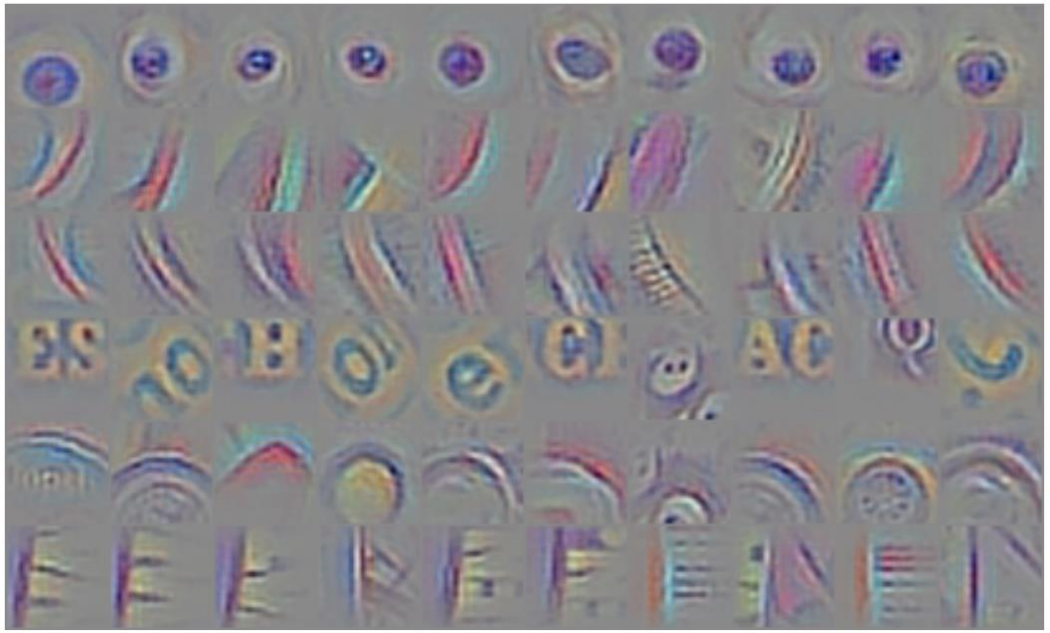


Figure 4: Visualization of descriptive image regions with different methods from the single largest activation in the last convolutional layer conv12 of the network trained on ImageNet. Reconstructions for 4 different images are shown.

Visualizing Intermediate Filters with guided backprop

guided backpropagation



corresponding image crops

Each row corresponds to a single filter