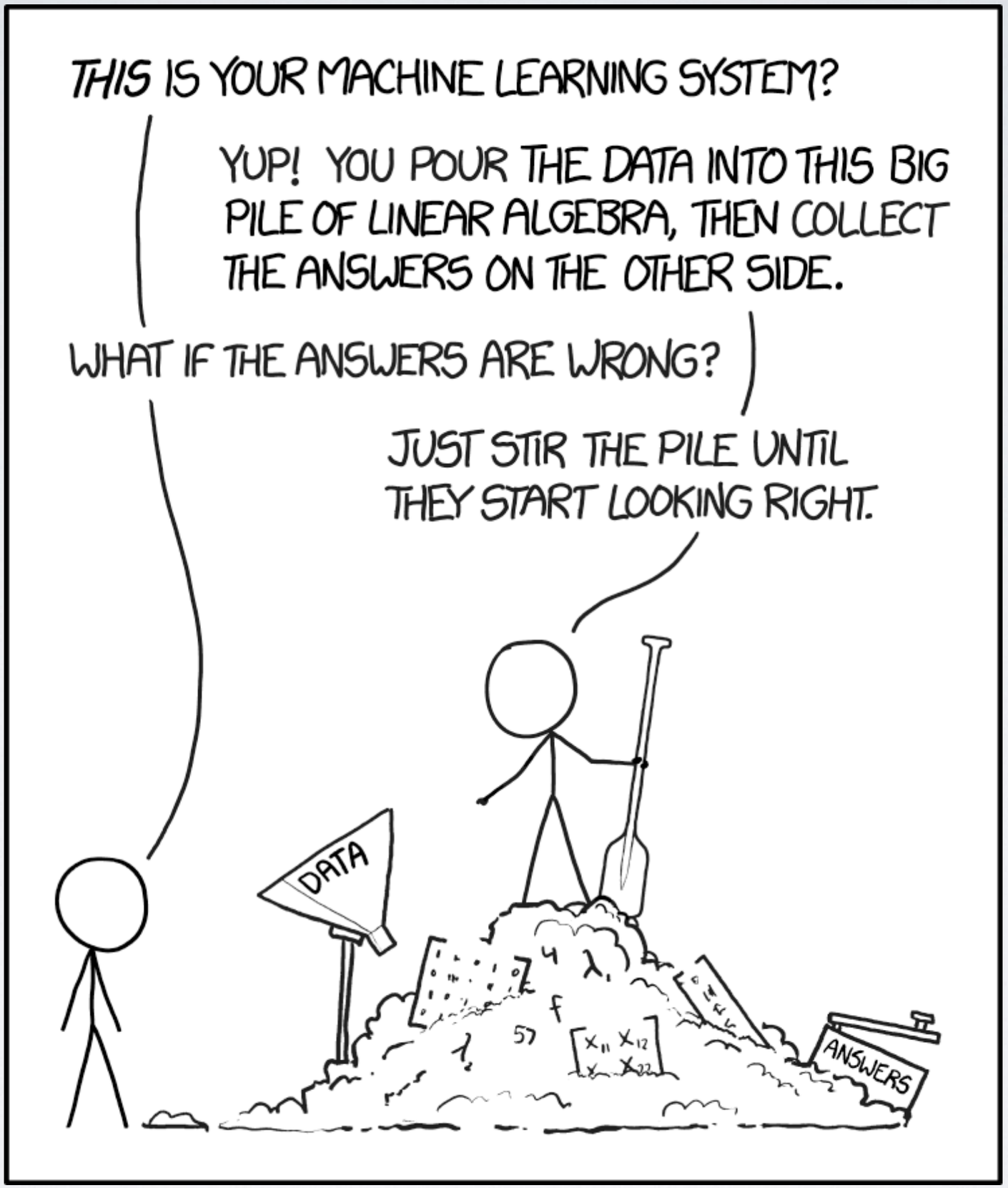
# CONVOLUTIONAL NETWORKS

Hint:  Don't do this yourself!
Always steal from other people!


THIS IS YOUR MACHINE LEARNING SYSTEM?
YUP! YOU POUR THE DATA INTO THIS BIG PILE OF LINEAR ALGEBRA, THEN COLLECT THE ANSWERS ON THE OTHER SIDE.
WHAT IF THE ANSWERS ARE WRONG?
JUST STIR THE PILE UNTIL THEY START LOOKING RIGHT.
DATA
ANSWERS

# CONVOLUTIONAL NET

$x$

$w_1$

$w_i$

$y$

**One's hot label**

| .05 | .1 | .9 | .04 |
|---|---|---|---|
| ball | car | hat | |

**grumpy cat**

$$f(x; w) \rightarrow y$$

image weights

# CONVOLUTIONAL NET

$x$

$w_1$

$w_i$

$y$

**One's hot label**

| .05 | .1 | .9 | .04 |
|---|---|---|---|
| ball | car | hat | |

**grumpy cat**

## Three ingredients

- convolutions
- non-linearities
- pooling

# CONVOLUTION

**Input image** $x$

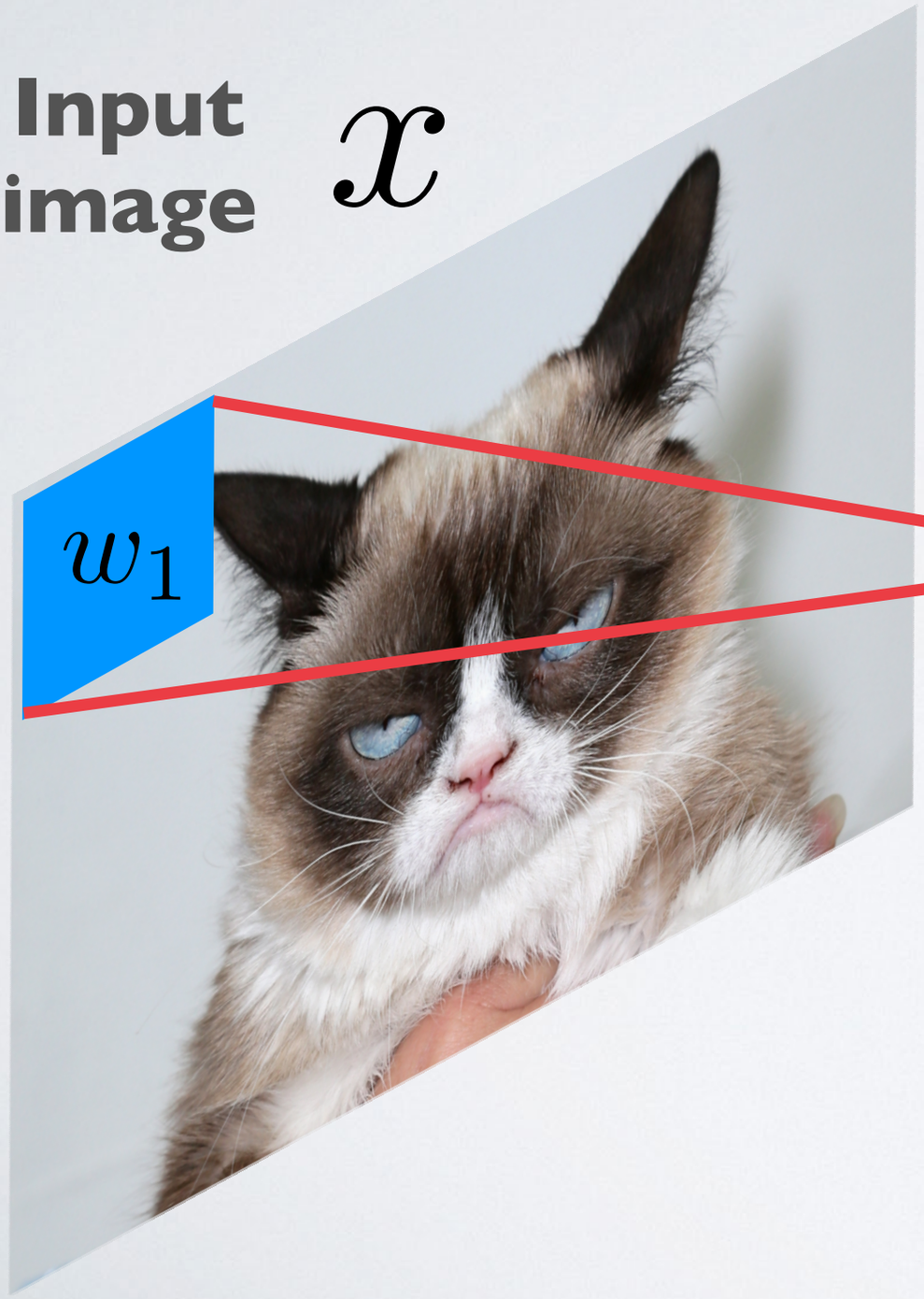
**Convolutional filter**

$w_1$

| | | |
|---|---|---|
| 5 | 3 | 8 |
| 2 | 1 | 7 |
| 1 | 9 | 4 |

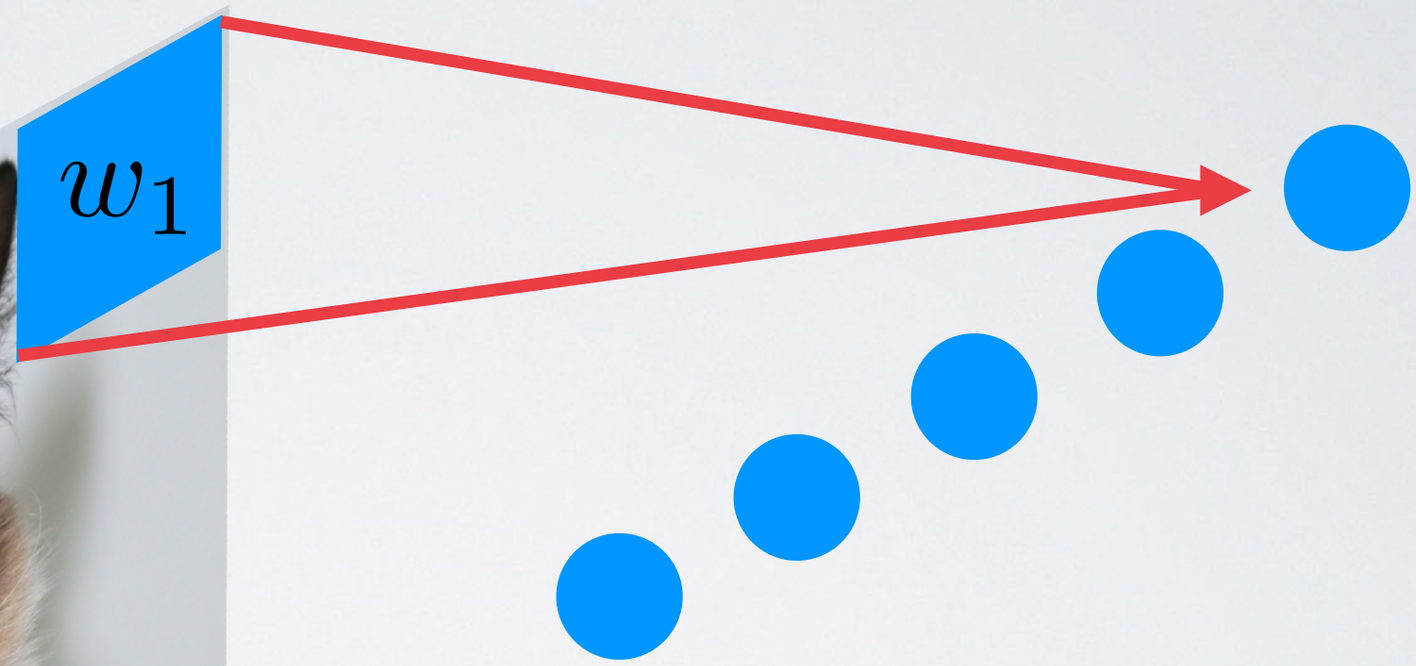
# CONVOLUTION

**Input image** $x$



# CONVOLUTION

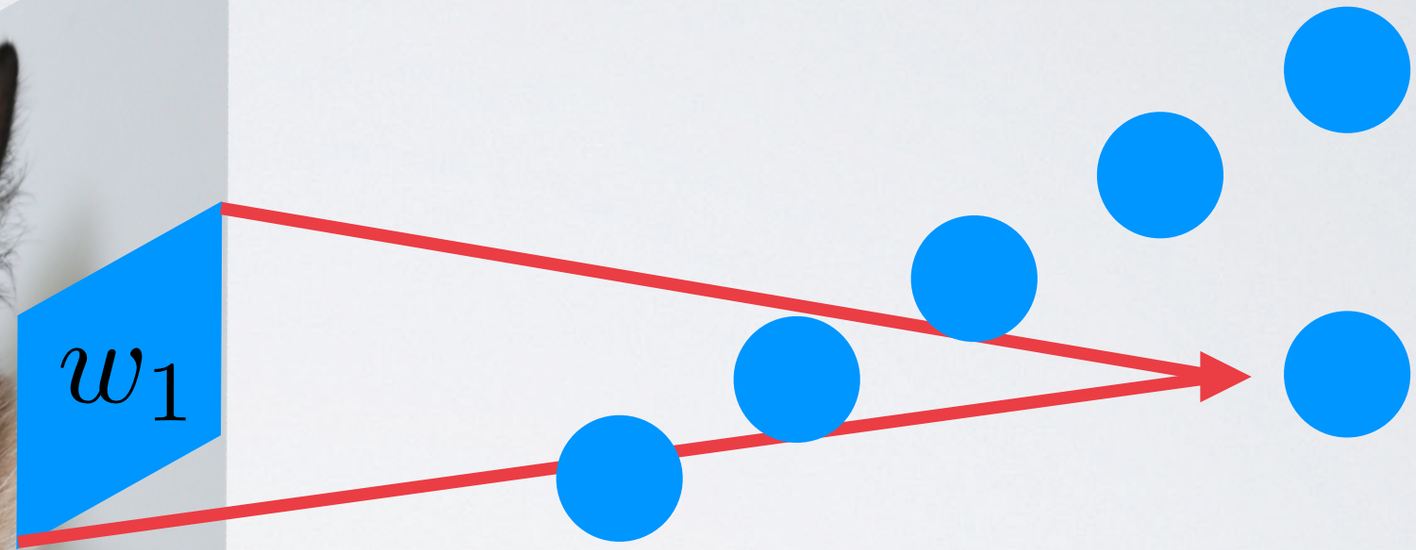
**Input image** $x$



# CONVOLUTION

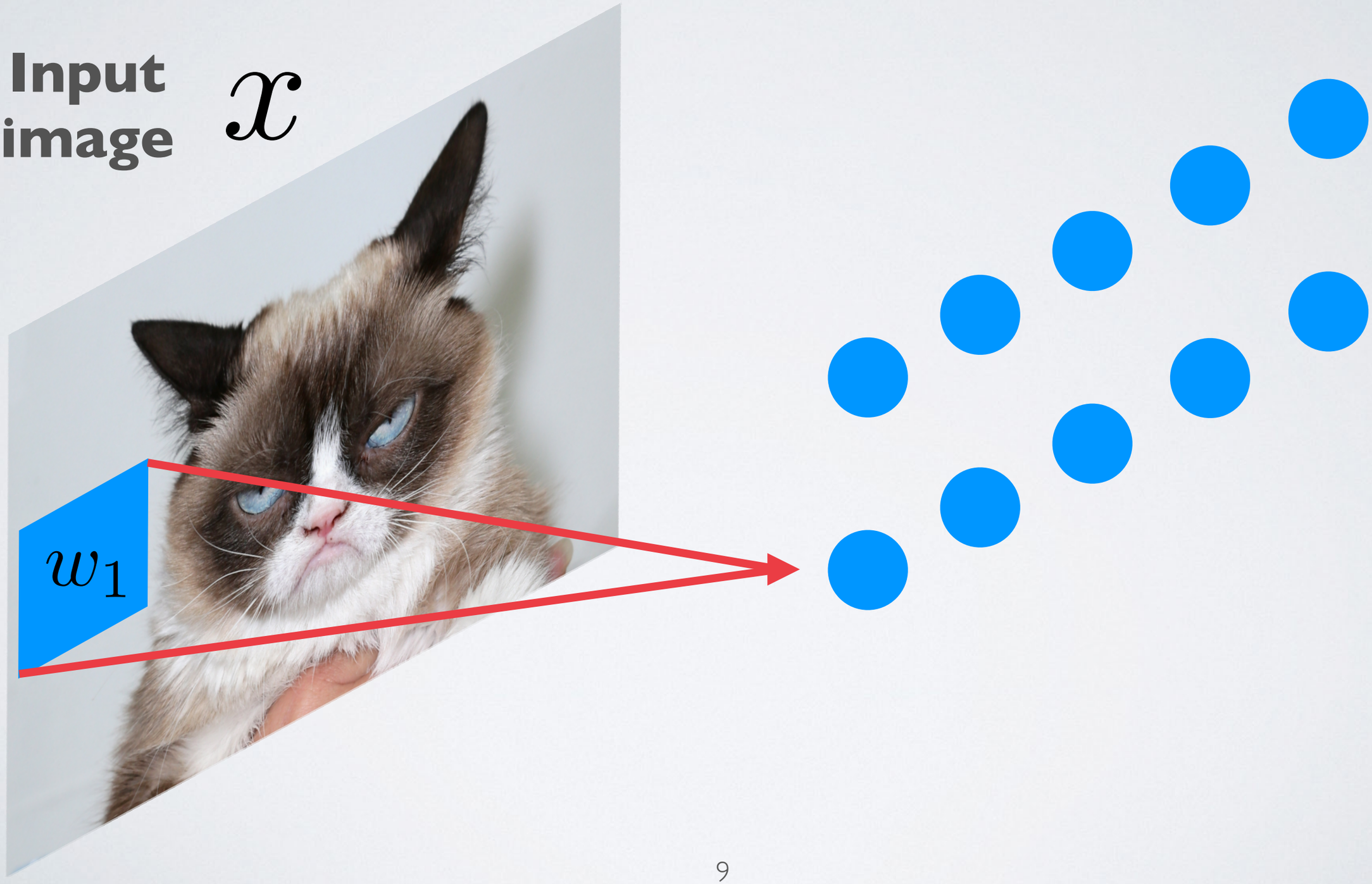
**Input image** $x$

$w_1$



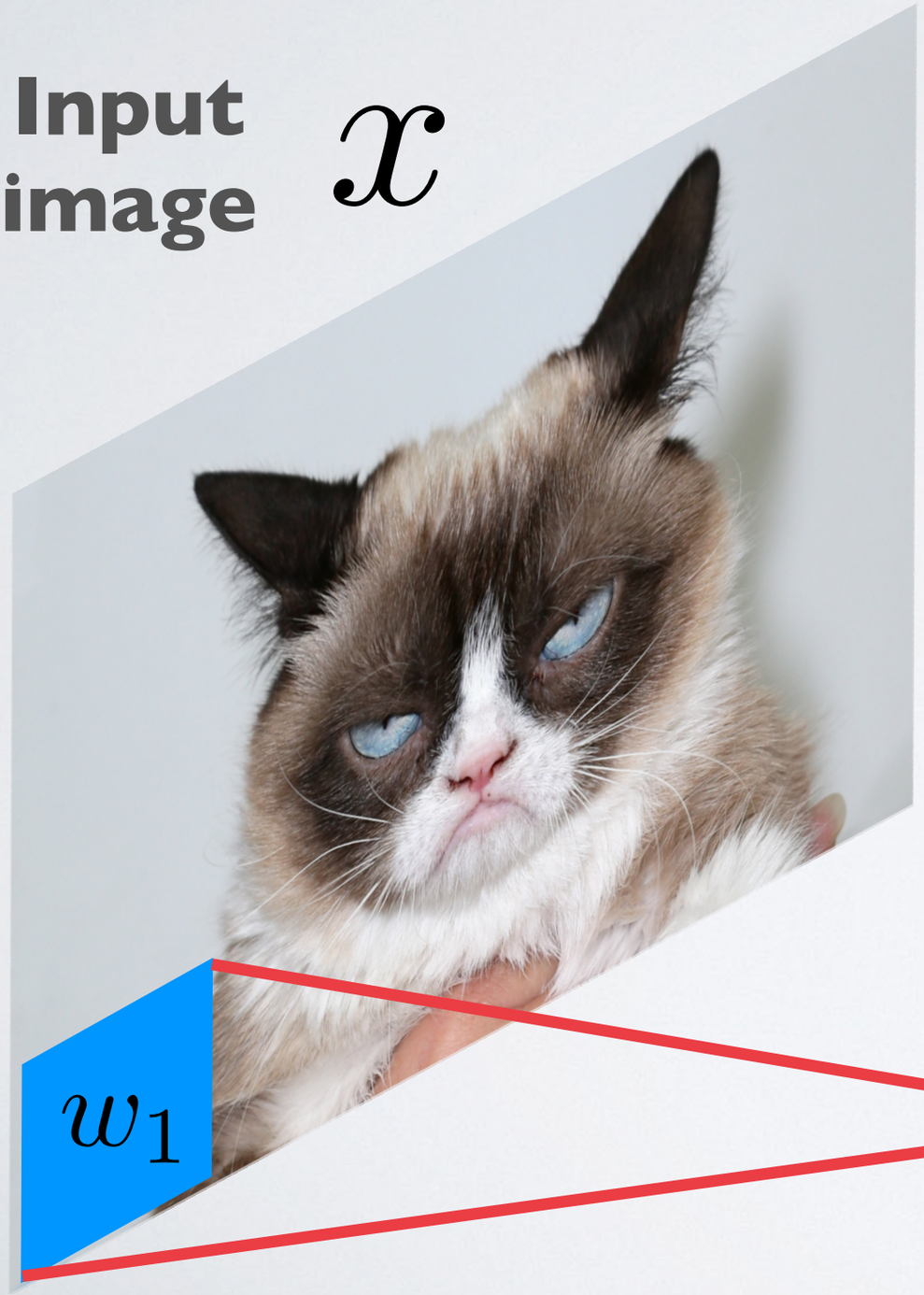
# CONVOLUTION

**Input image** $x$



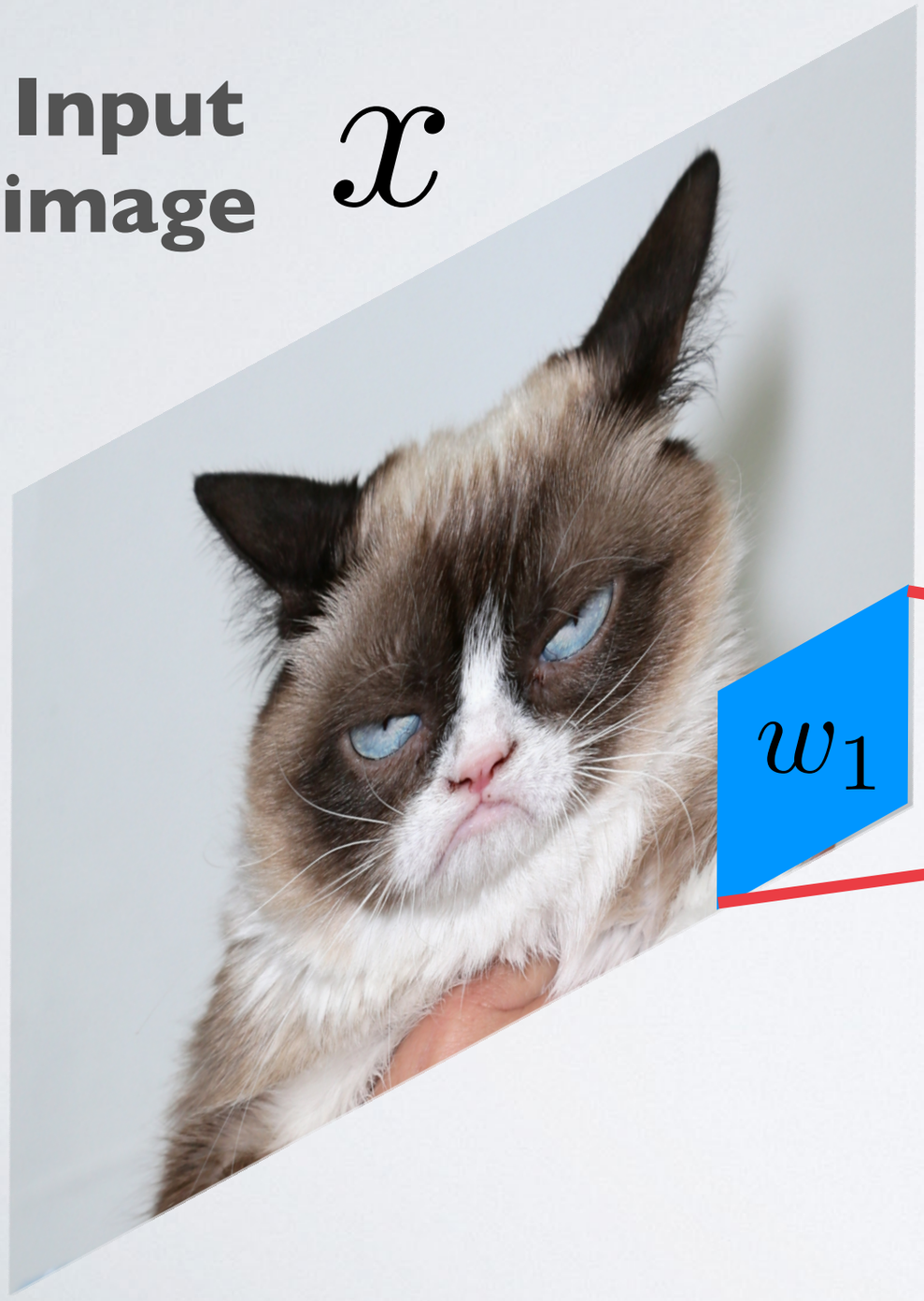
# CONVOLUTION

**Input image** $x$

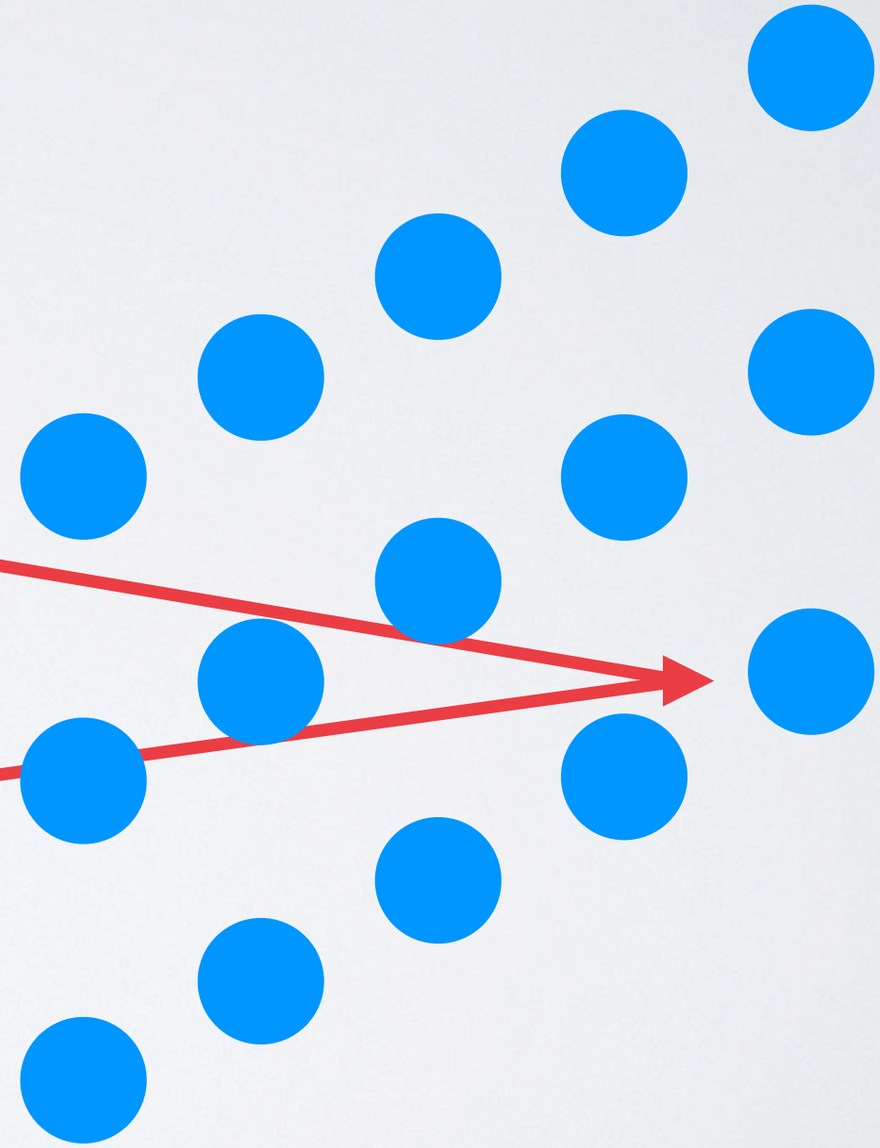


# CONVOLUTION

**Input image** $x$

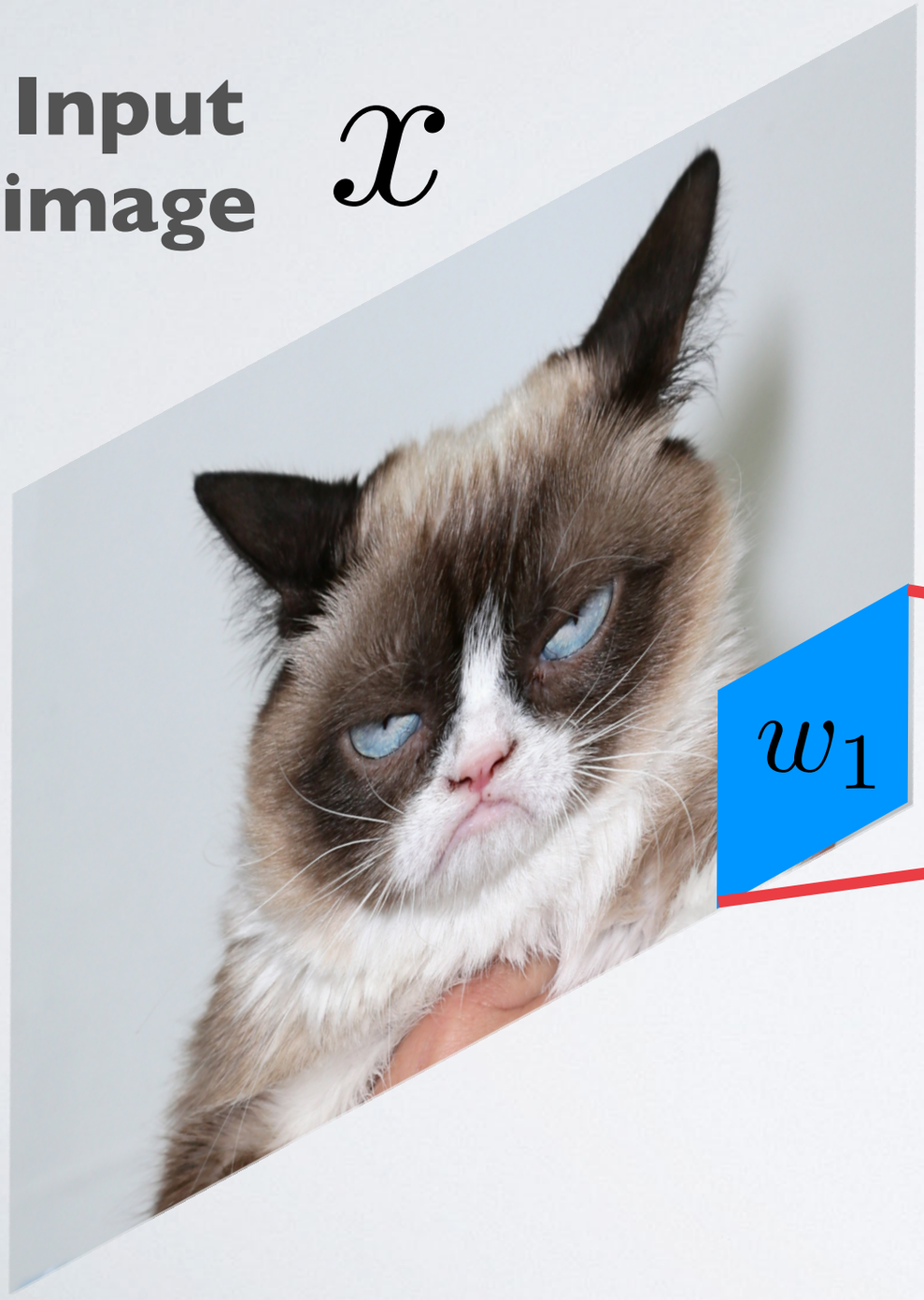


$w_1$



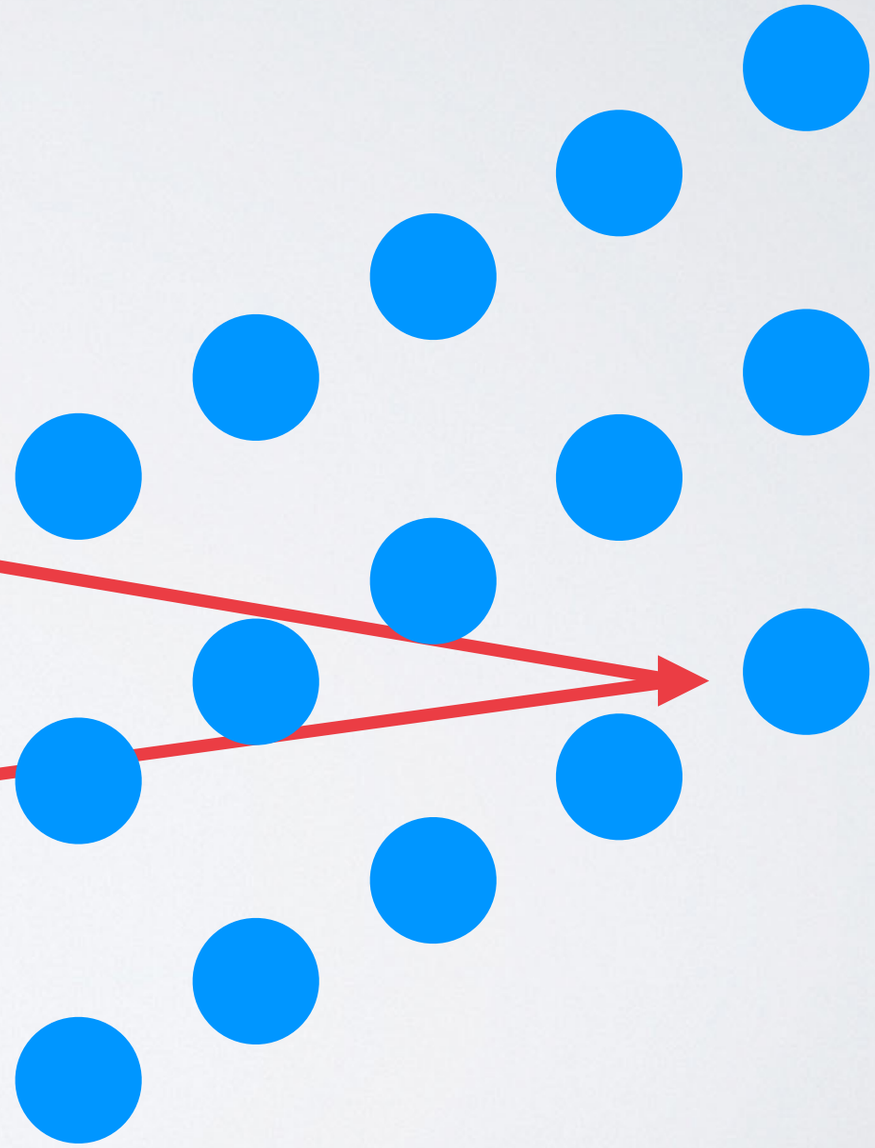
# CONVOLUTION

**Input image** $x$



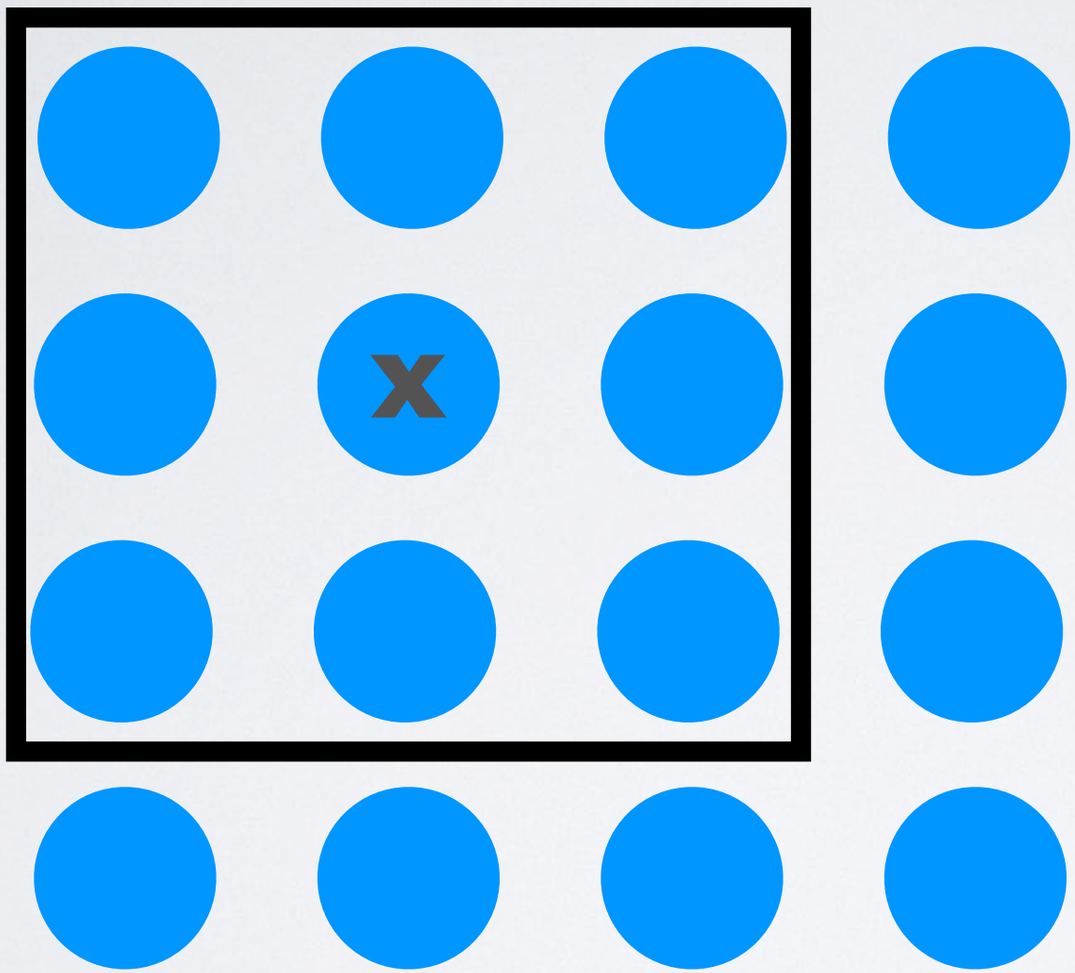
$w_1$

**“Feature map”**



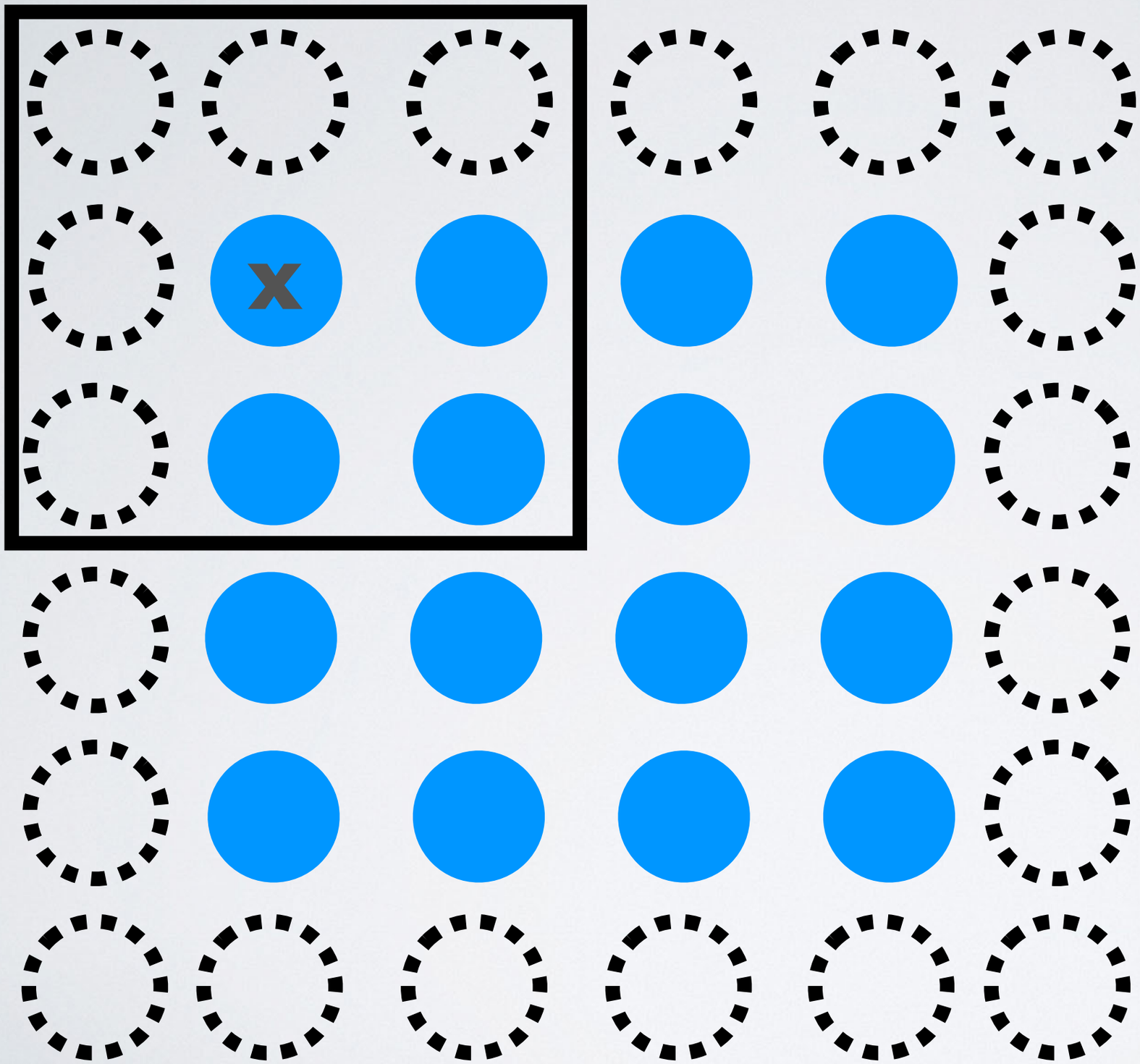
# VARIANTS OF CONV

# PADDING




**3x3 conv produces feature maps that are 2 units smaller**

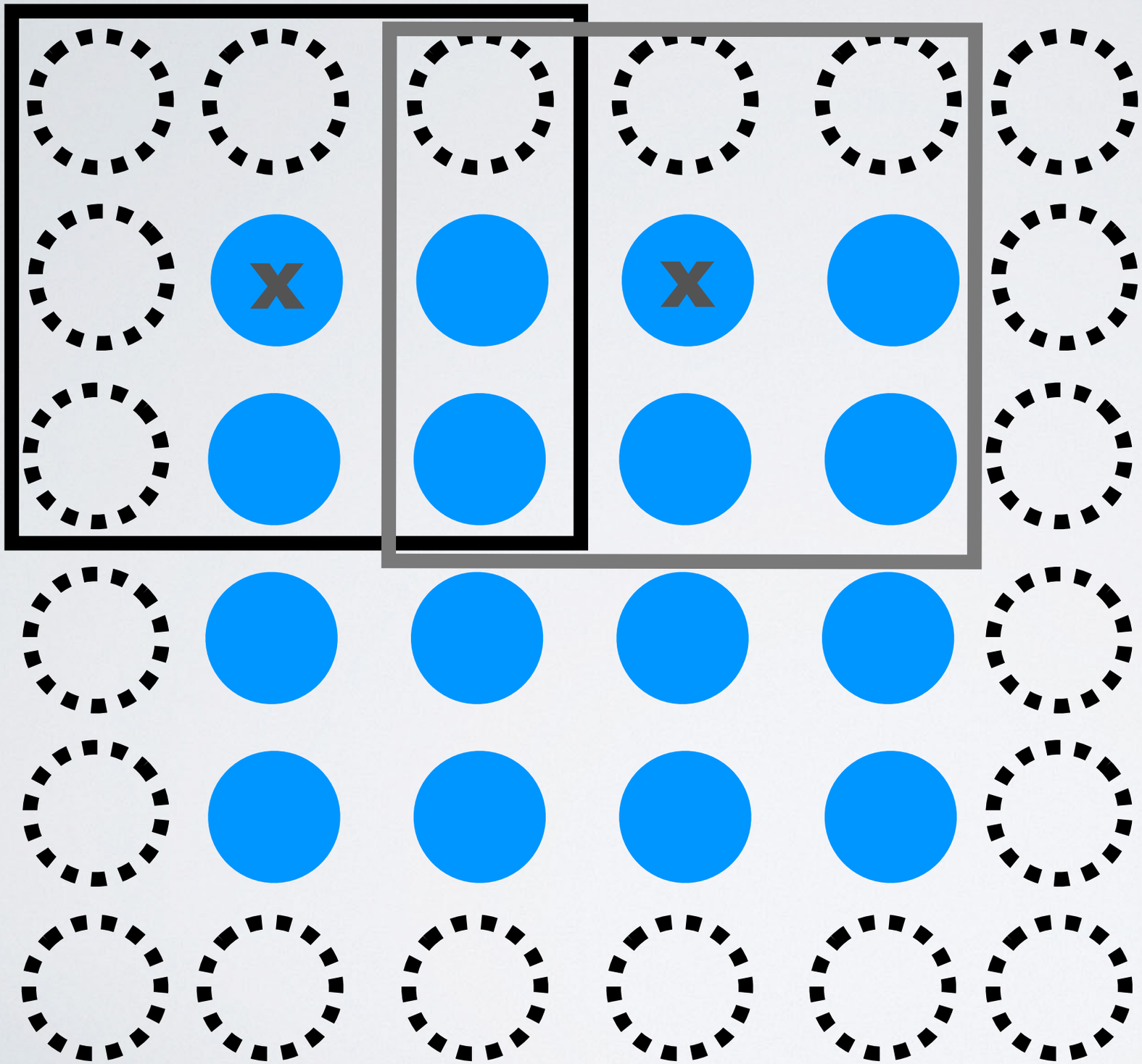
# PADDING




**padding=1**

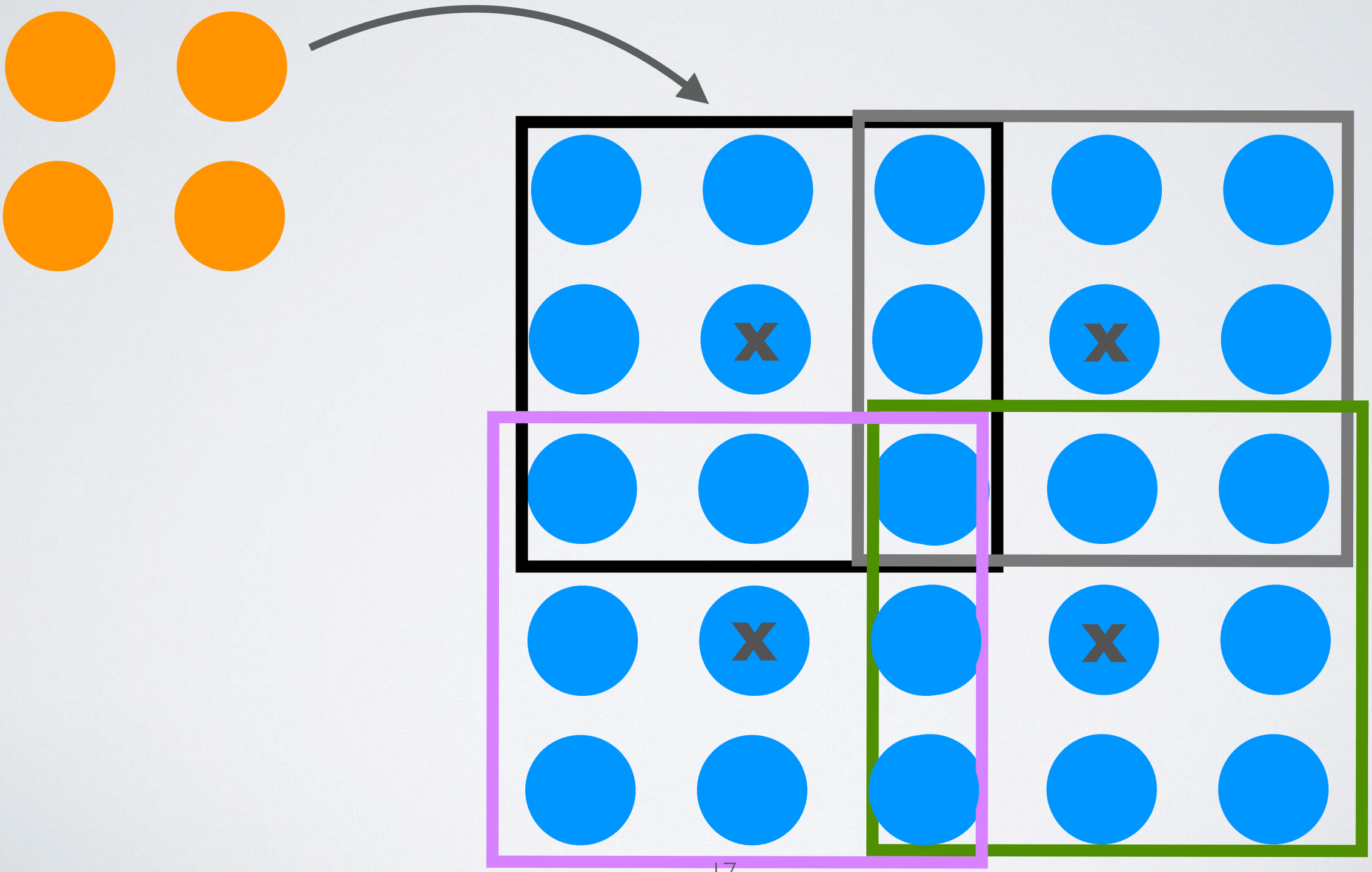
Output is same size as input

# STRIDE




**stride=2 cuts resolution by half**

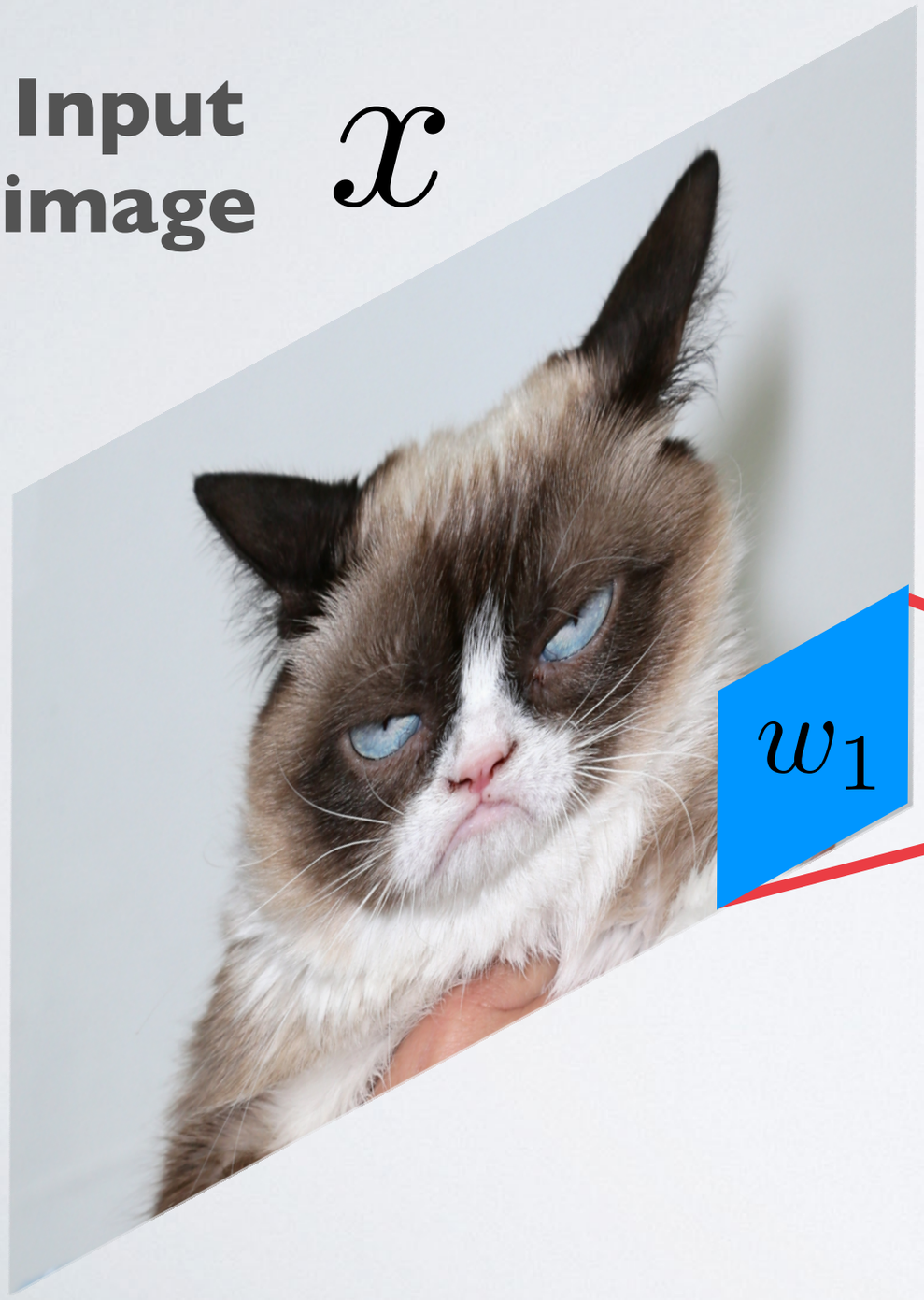
# TRANSPOSE / DECONVOLUTION



# RELU

# RELU NONLINEARITY

**Input image** $x$



$w_1$

**“Feature map”**

**remove negatives**

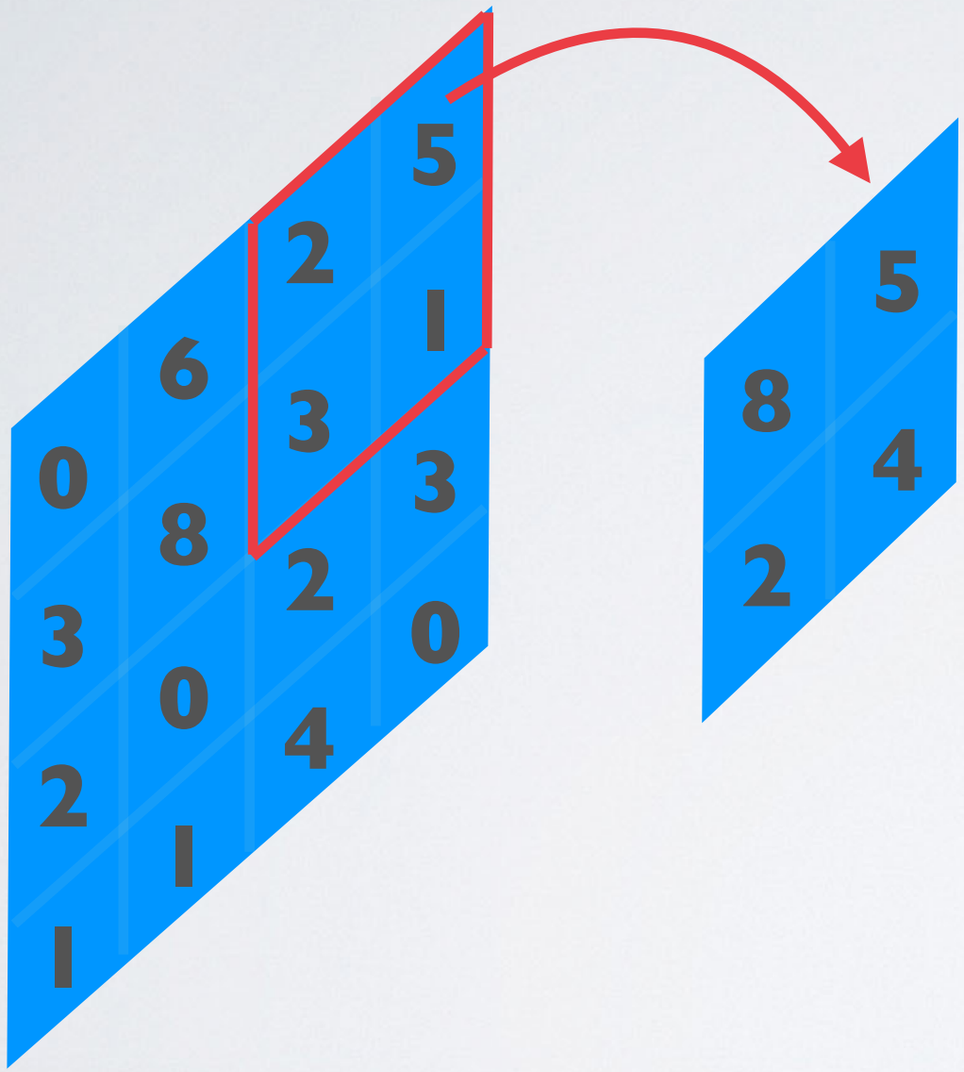
| | | |
|---|---|---|
| 5 | -3 | -2 |
| 1 | 3 | 7 |
| -9 | 2 | -8 |

| | | |
|---|---|---|
| 5 | 0 | 0 |
| 1 | 3 | 7 |
| 0 | 2 | 0 |

# POOLING

# POOLING

moving to lower resolutions







Max pooling

Average pooling

Strided convolution
(aka downsampling)

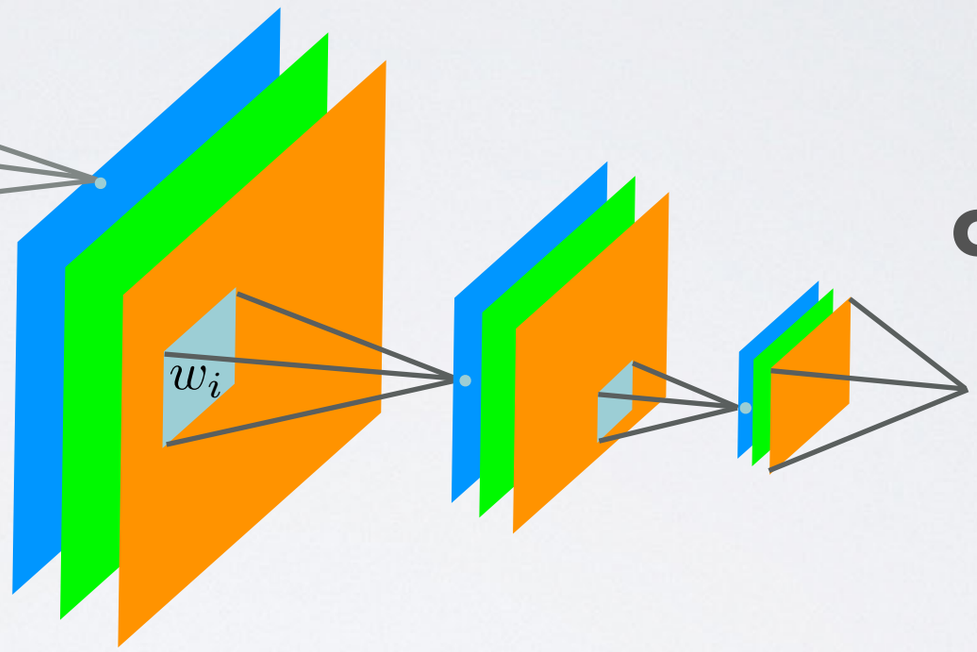
# THAT'S ALL YOU NEED!

(sort of)

$x$

$w_1$

$w_i$



$y$

**One's hot label**

| .05 | .1 | .9 | .04 |
|---|---|---|---|
| ball | car | hat | |

**grumpy cat**

$$f(x; w) \rightarrow y$$

image weights

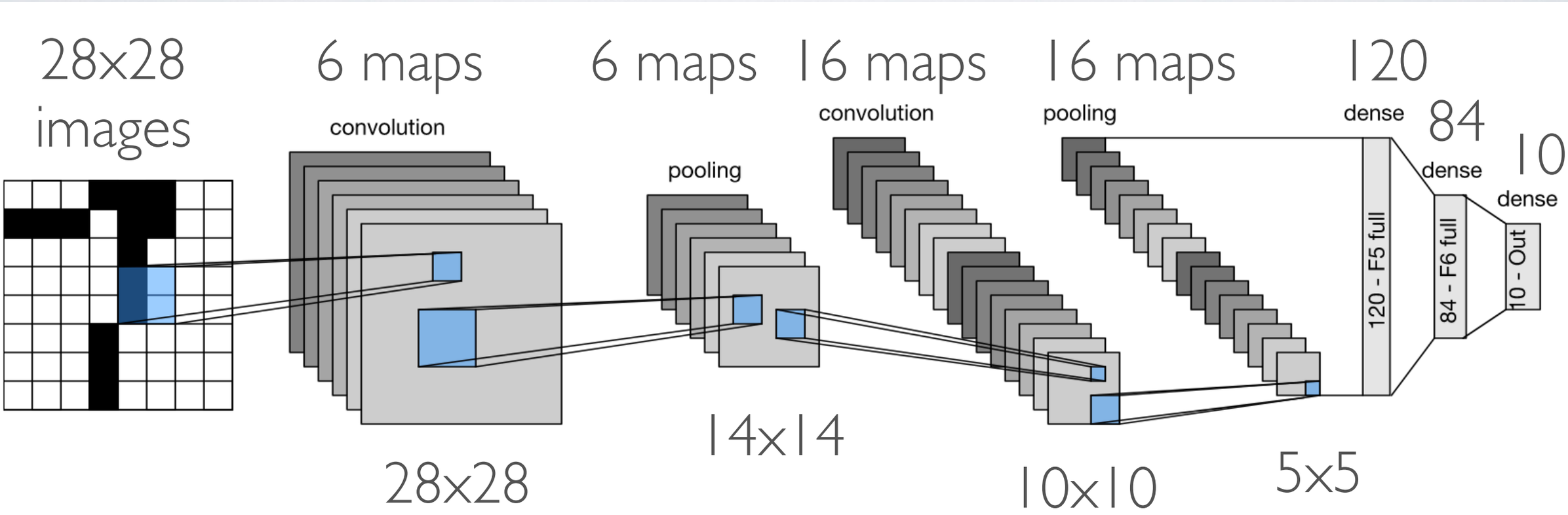
# MNIST

Modified National Institute of Standards and Technology

# LENET

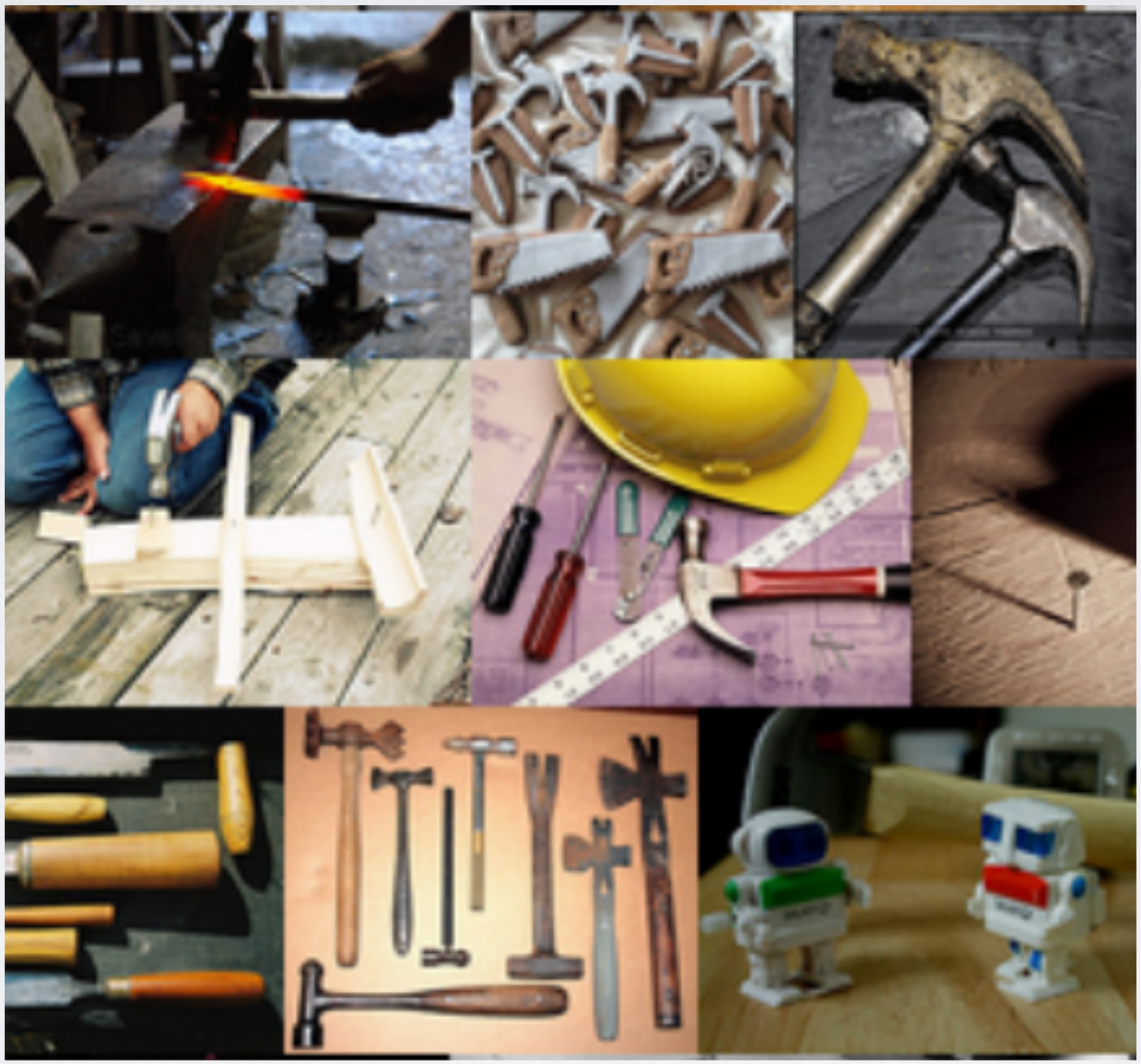
## LeCun et al, 1998

The first successful convnet



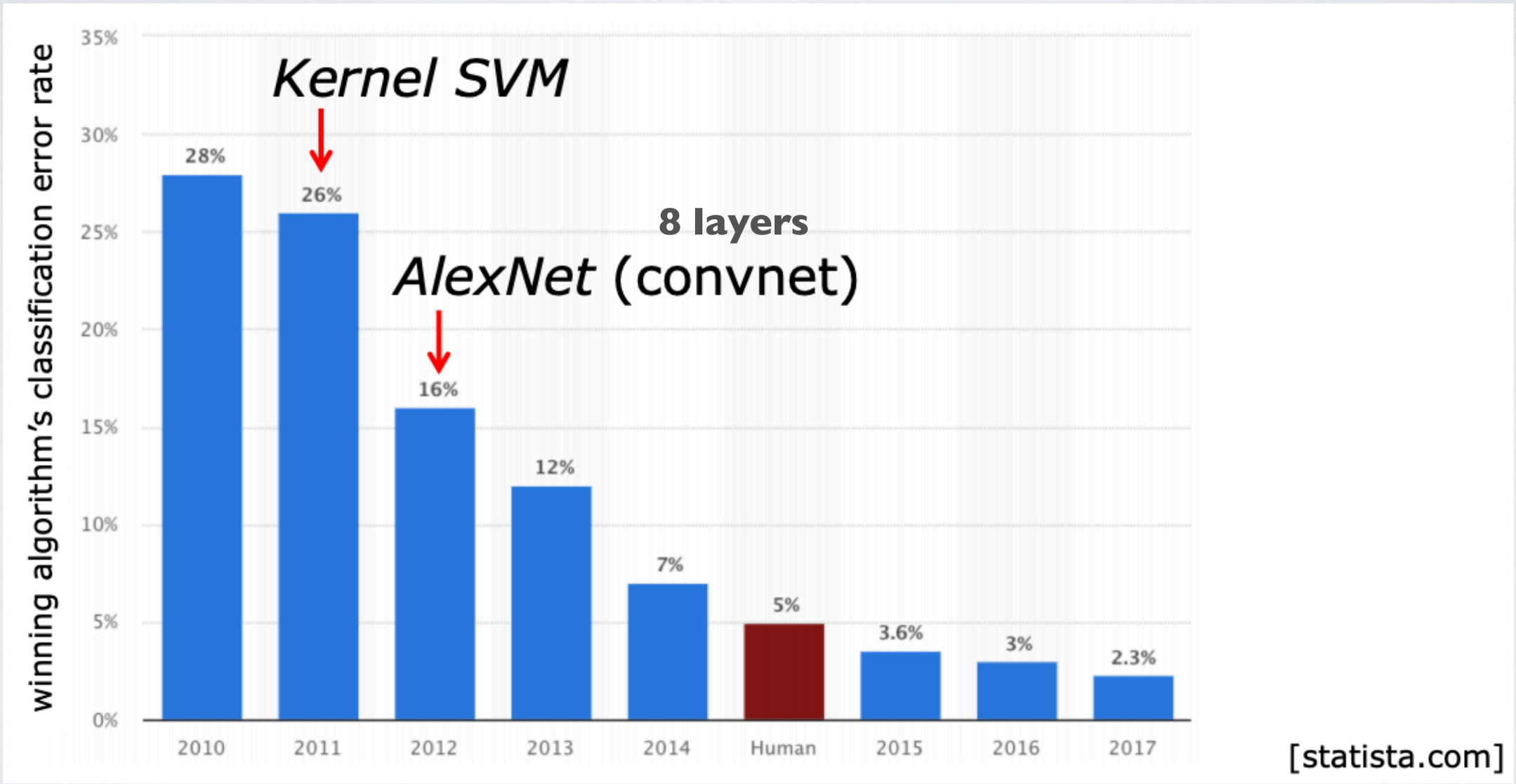
# IMAGENET DATASET

2010



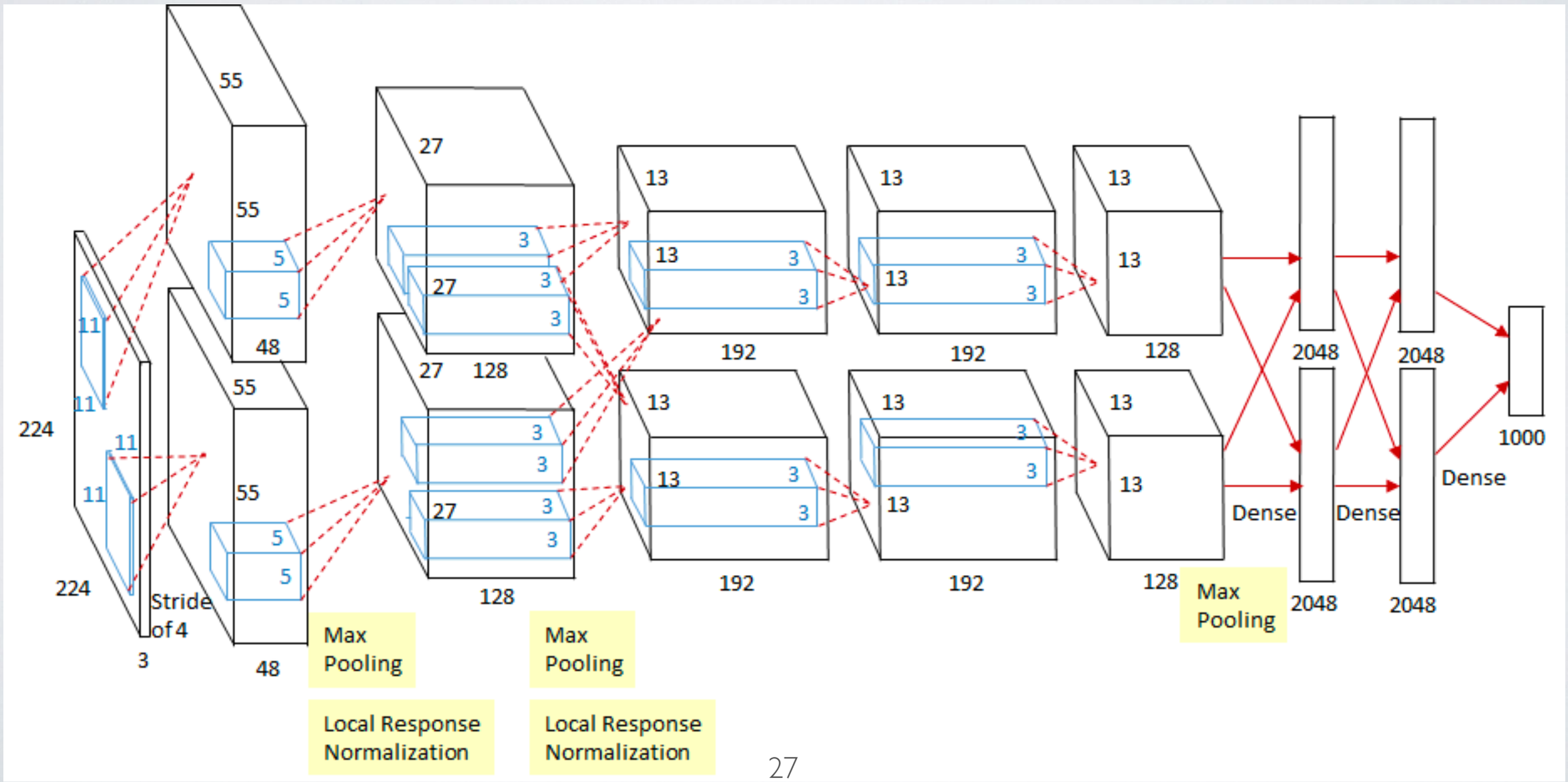
1K classes 1M Images Usually 224x224 resolution

# IMAGENET HISTORY

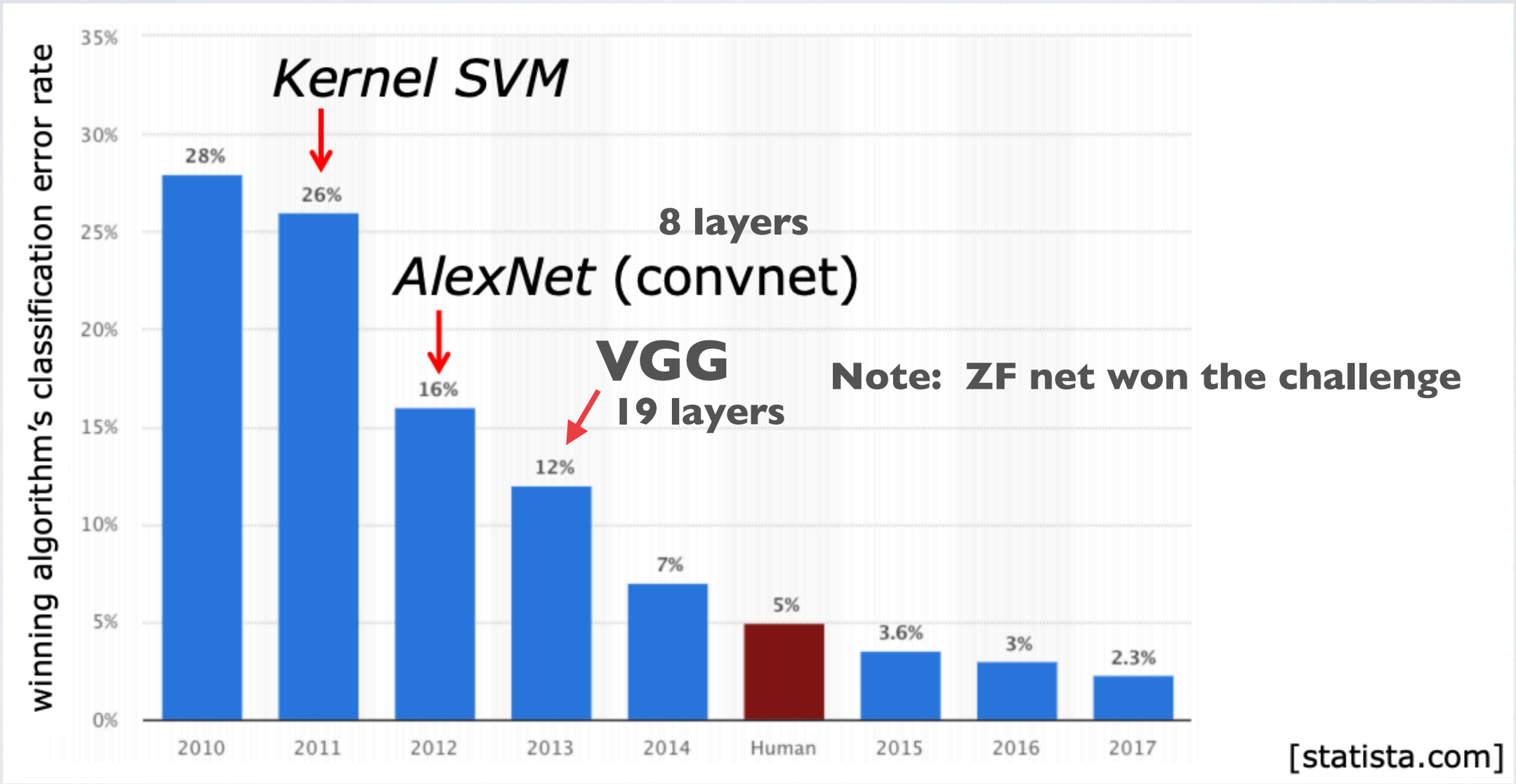


# ALEX NET

## Named for Alex Krizhevsky, 2012

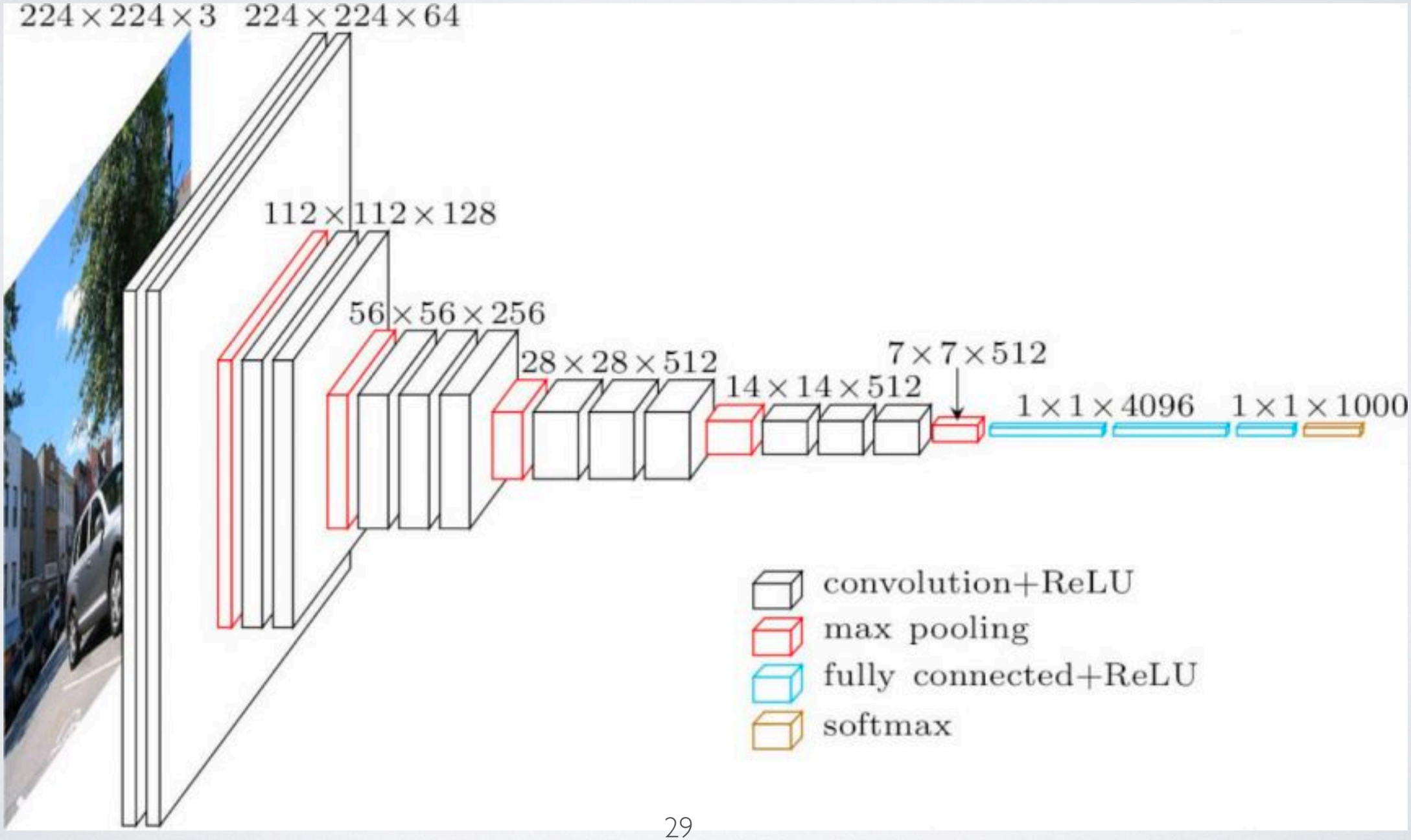


# IMAGENET HISTORY

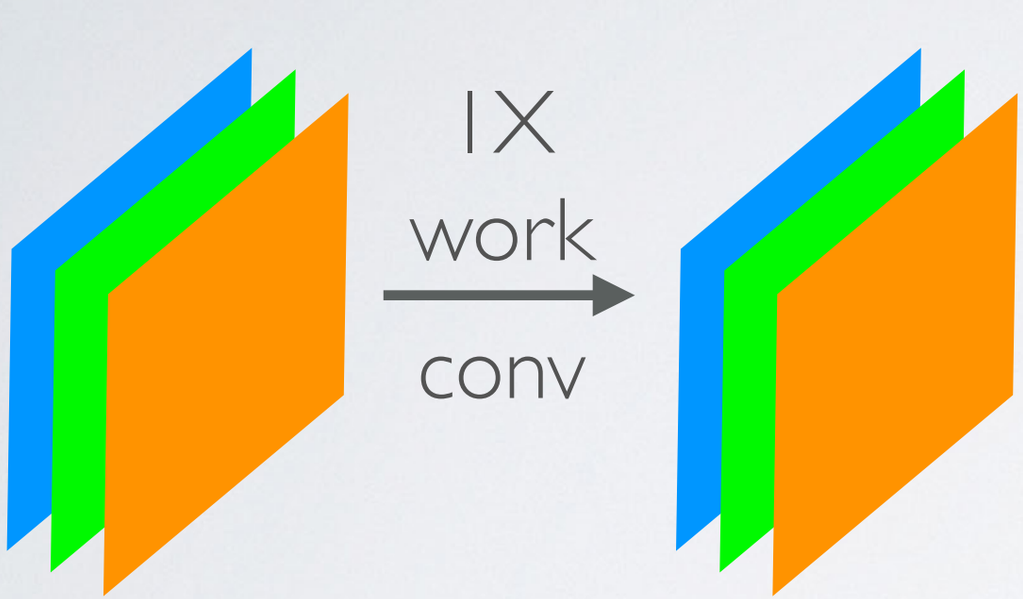


# VGG

## Visual graphics group, Oxford

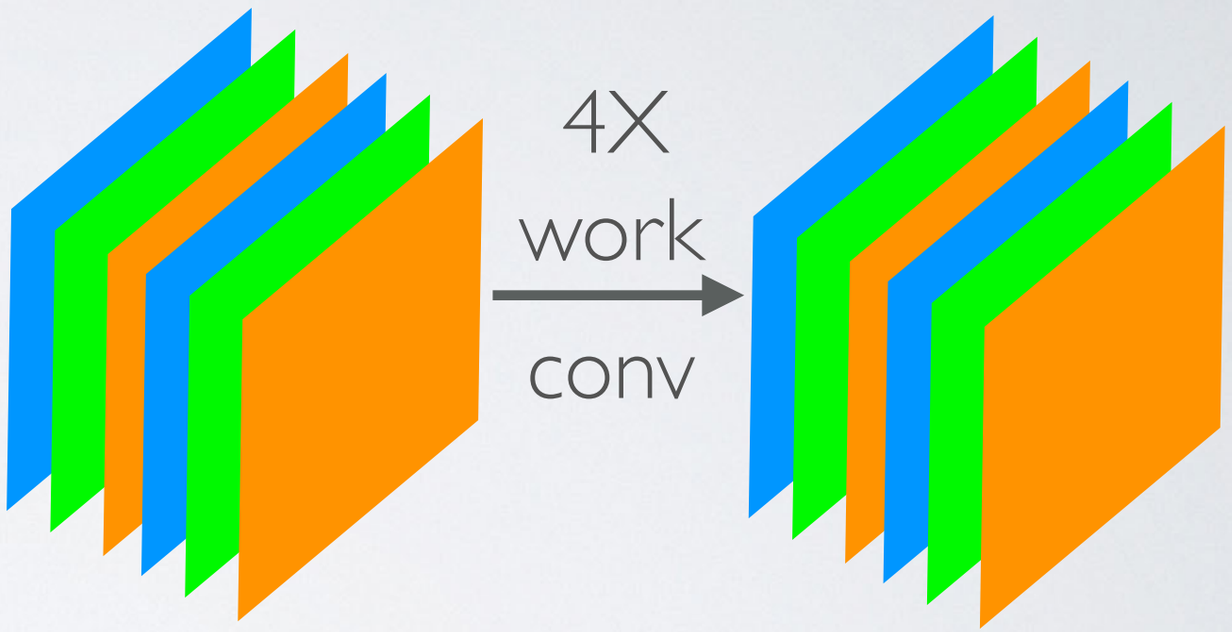


# 2D CONV NETS



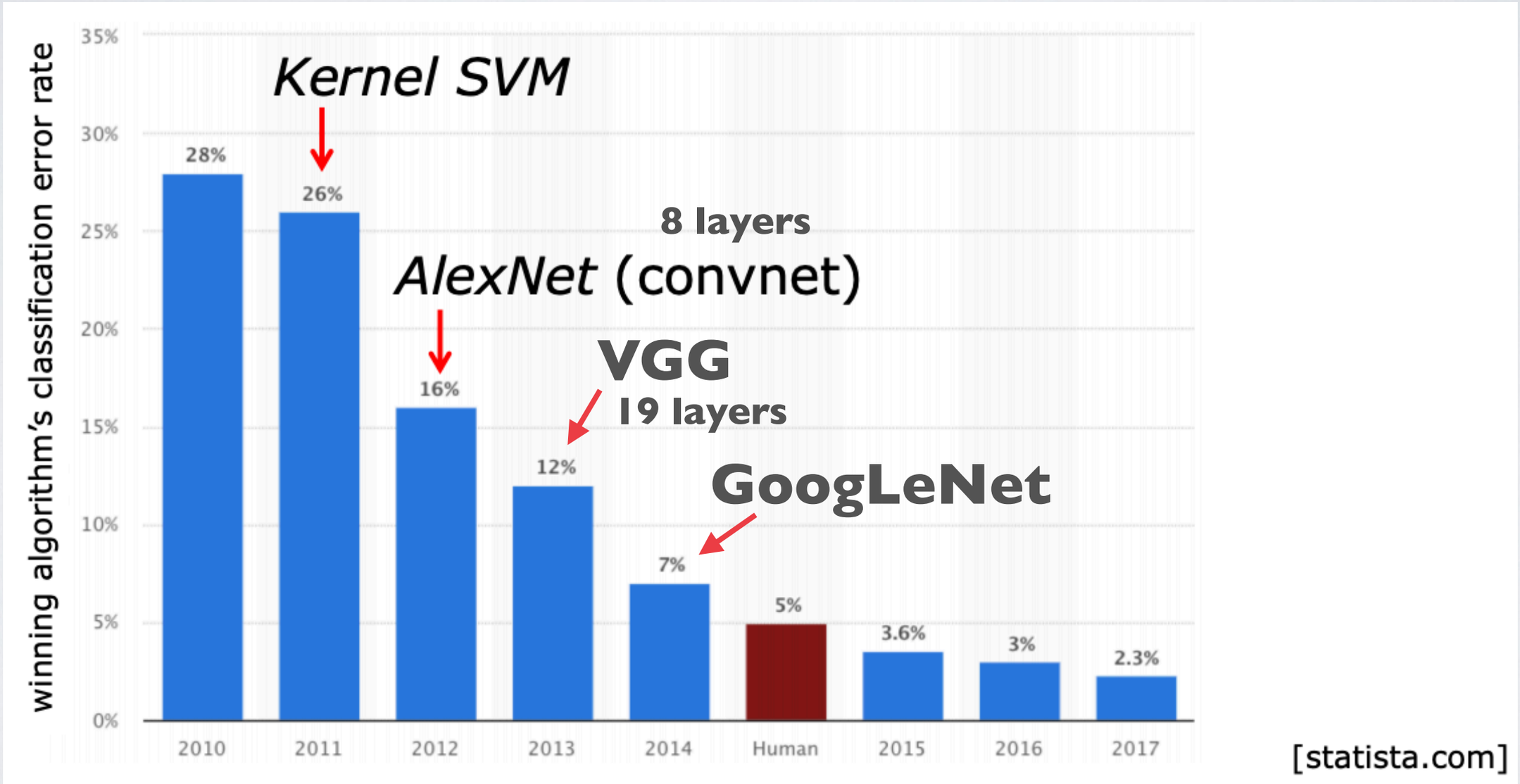

Half the resolution


Twice as many maps



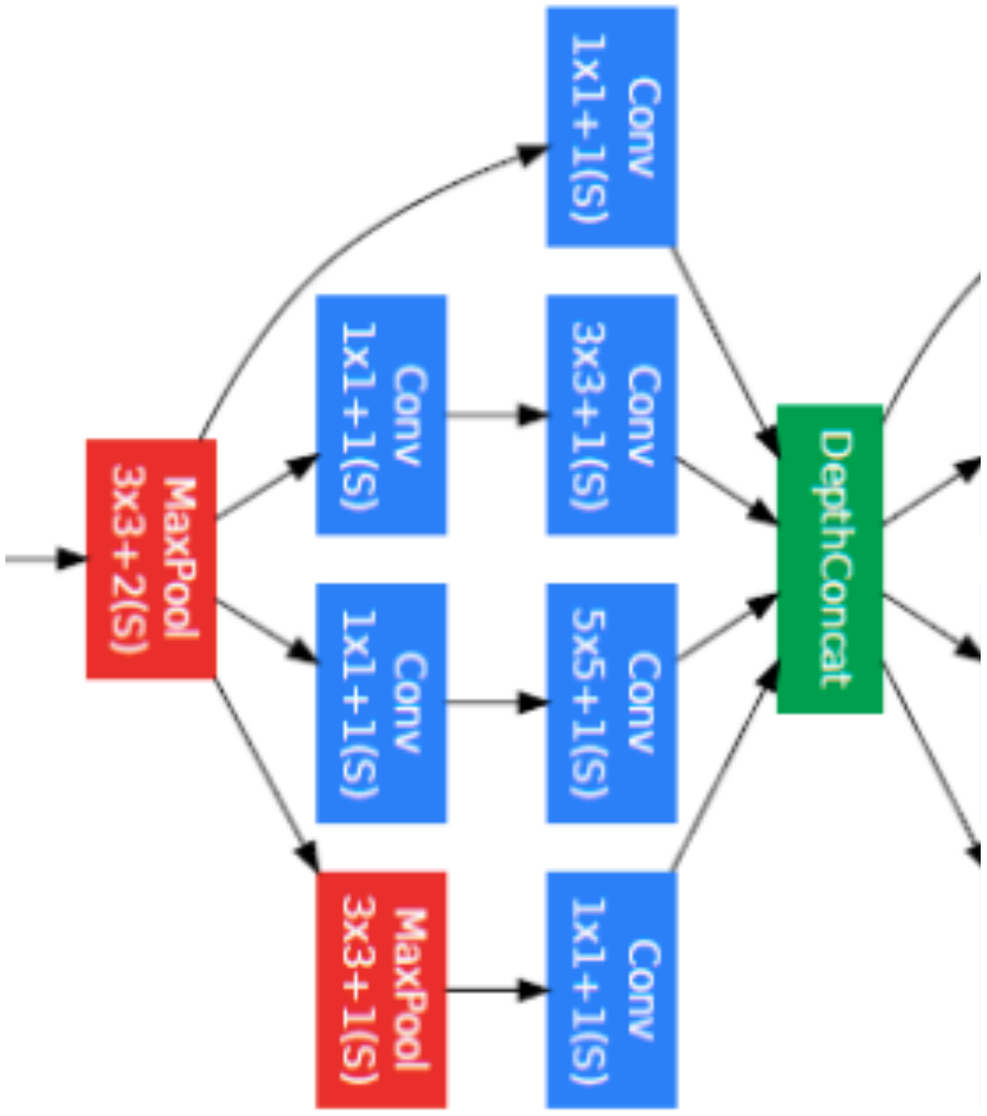

(2X feature maps) + (1/2 resolution)
=
Same number of operations

# IMAGENET HISTORY

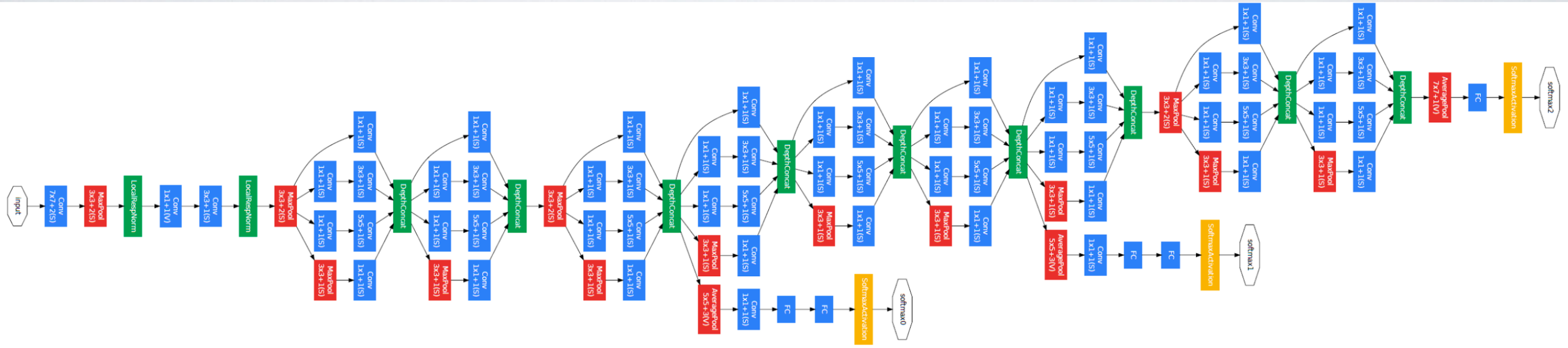



[statista.com]

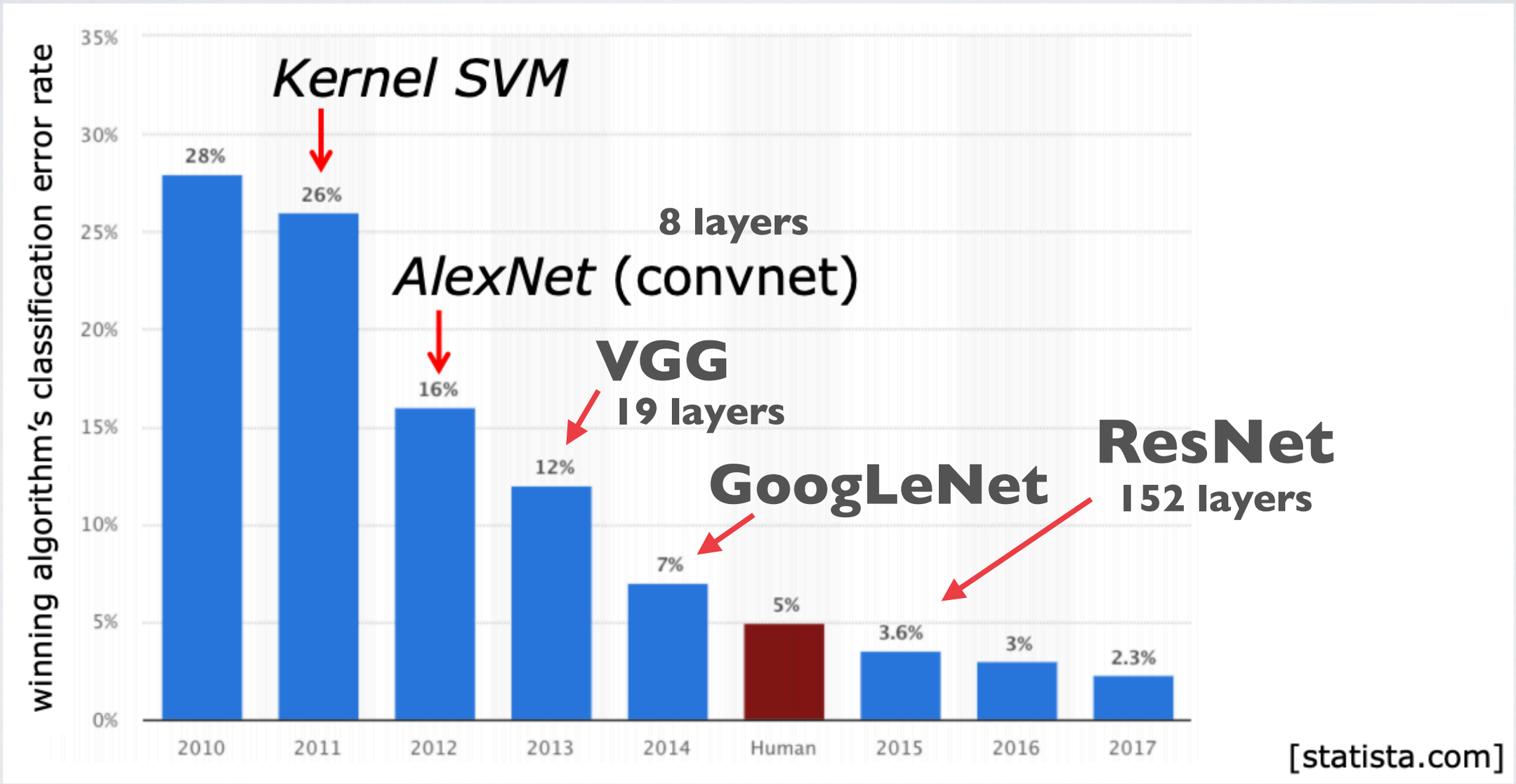
# GOOGLENET


Conv
1x1+1(S)
Conv
1x1+1(S)
Conv
3x3+1(S)
MaxPool
3x3+2(S)
Conv
1x1+1(S)
Conv
5x5+1(S)
DepthConcat
MaxPool
3x3+1(S)
Conv
1x1+1(S)

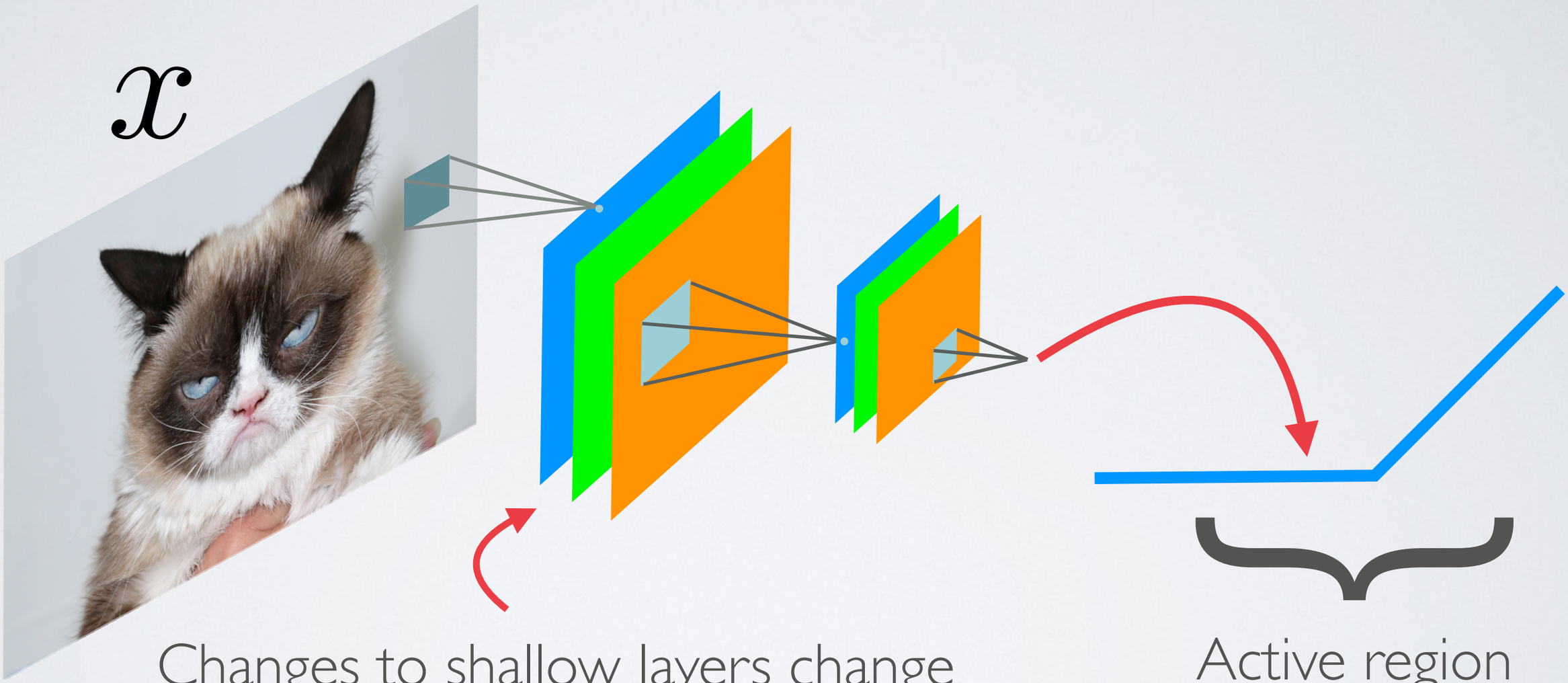
# GOOGLENET


input
Conv 7x7+2(S)
MaxPool 3x3+2(S)
LocalRespNorm
Conv 1x1+1(V)
Conv 3x3+1(S)
LocalRespNorm
MaxPool 3x3+2(S)
Conv 1x1+1(S)
Conv 3x3+1(S)
Conv 5x5+1(S)
MaxPool 3x3+1(S)
DepthConcat
MaxPool 3x3+2(S)
Conv 1x1+1(S)
Conv 3x3+1(S)
Conv 5x5+1(S)
MaxPool 3x3+1(S)
DepthConcat
AveragePool 5x5+3(V)
Conv 1x1+1(S)
FC
SoftmaxActivation
softmax0
MaxPool 3x3+2(S)
Conv 1x1+1(S)
Conv 3x3+1(S)
Conv 5x5+1(S)
MaxPool 3x3+1(S)
DepthConcat
AveragePool 5x5+3(V)
Conv 1x1+1(S)
FC
SoftmaxActivation
softmax1
MaxPool 3x3+2(S)
Conv 1x1+1(S)
Conv 3x3+1(S)
Conv 5x5+1(S)
MaxPool 3x3+1(S)
DepthConcat
AveragePool 7x7+1(V)
FC
SoftmaxActivation
softmax2

# IMAGENET HISTORY



# BATCH NORMALIZATION




Changes to shallow layers change mean of deeper activations

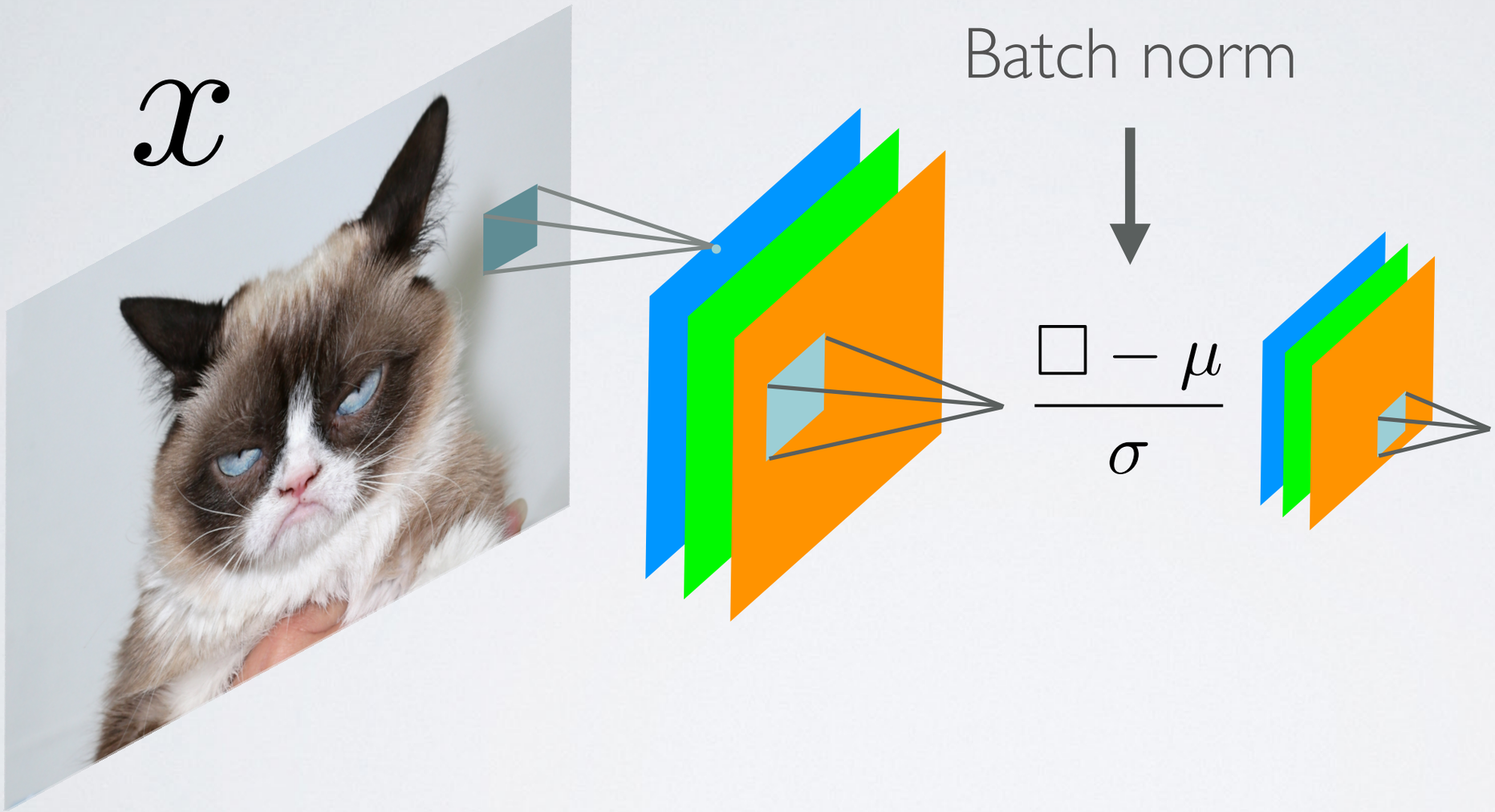
"Internal covariant shift"

Active region



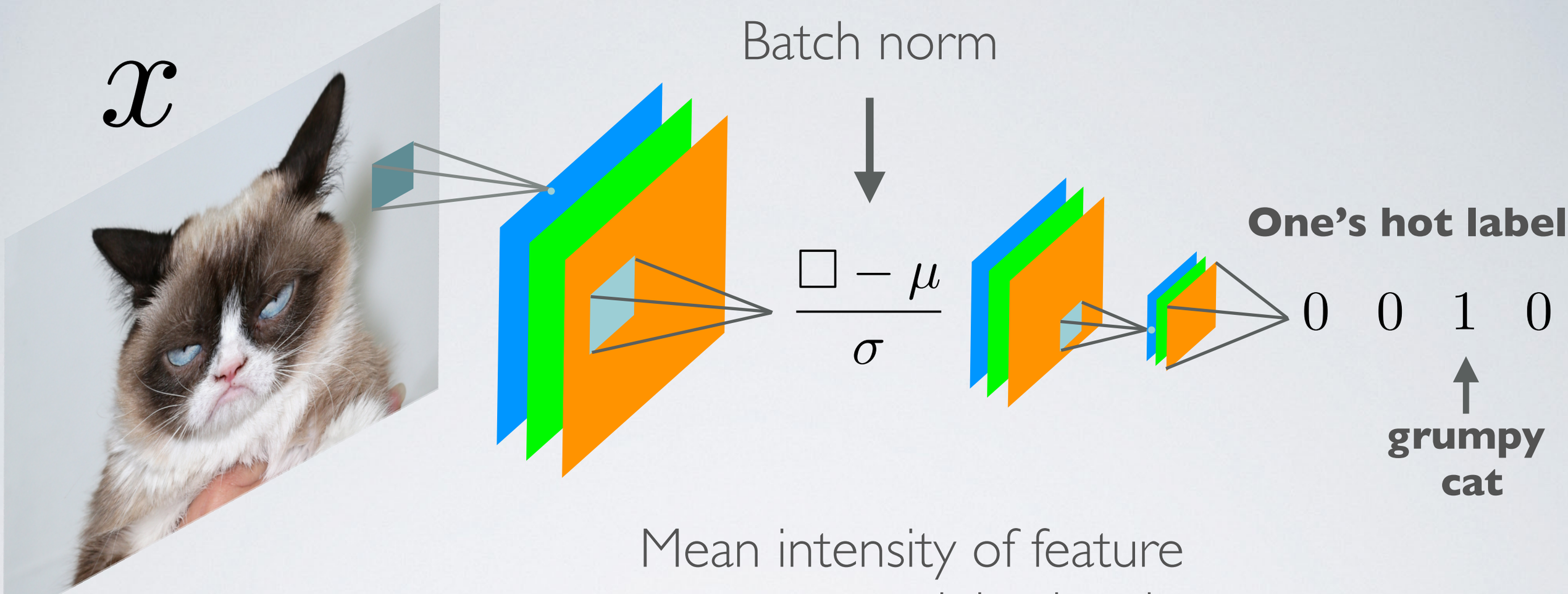
Ioffe & Szegedy, "Batch Normalization…", 2015

# BATCH NORMALIZATION



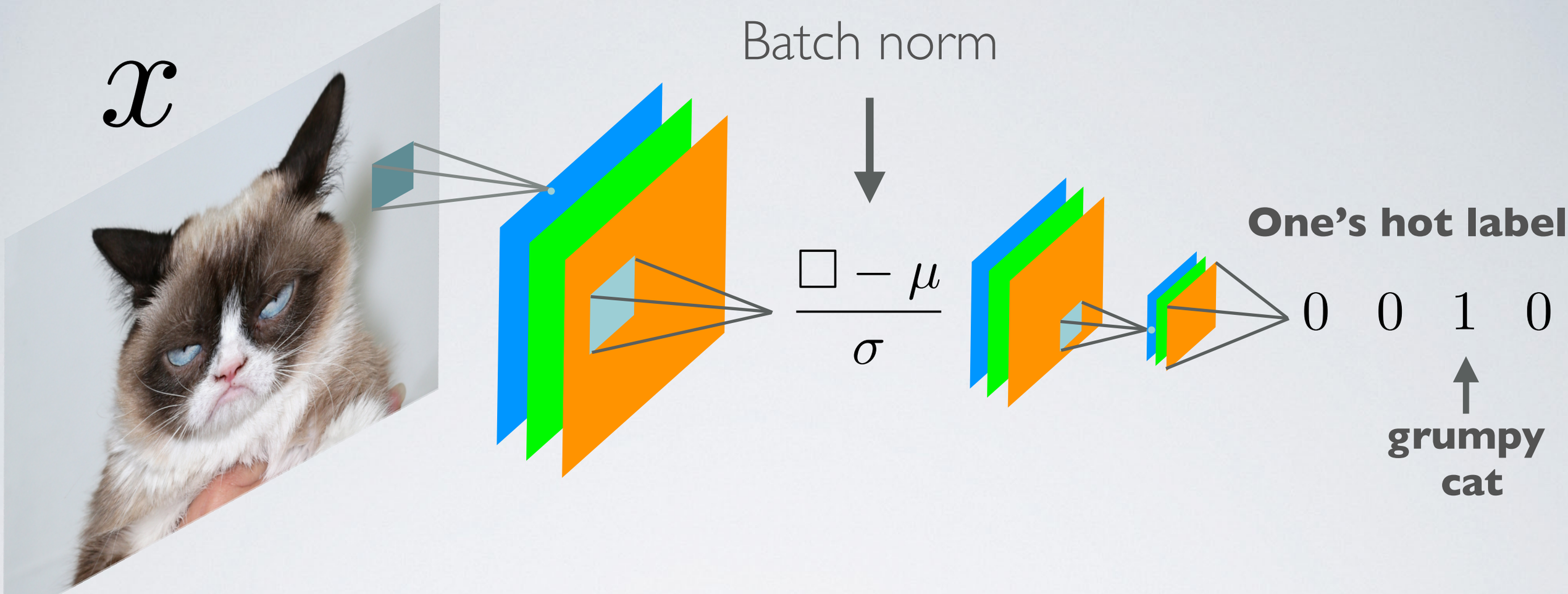



Ioffe & Szegedy, "Batch Normalization…", 2015






Ioffe & Szegedy, "Batch Normalization…", 2015



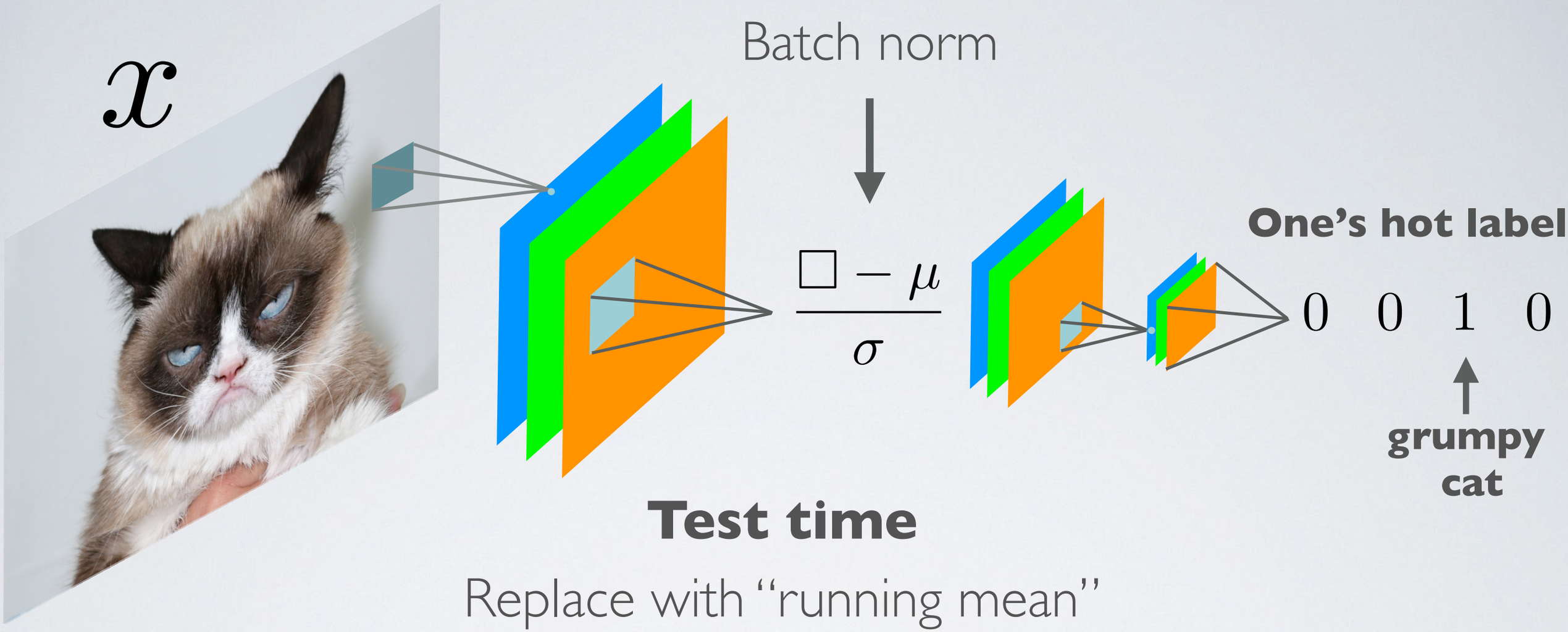

Trainable parameter

$$\gamma \frac{z - \mu}{\sigma} + \eta$$

Trainable parameter



Ioffe & Szegedy, "Batch Normalization…", 2015




## Test time

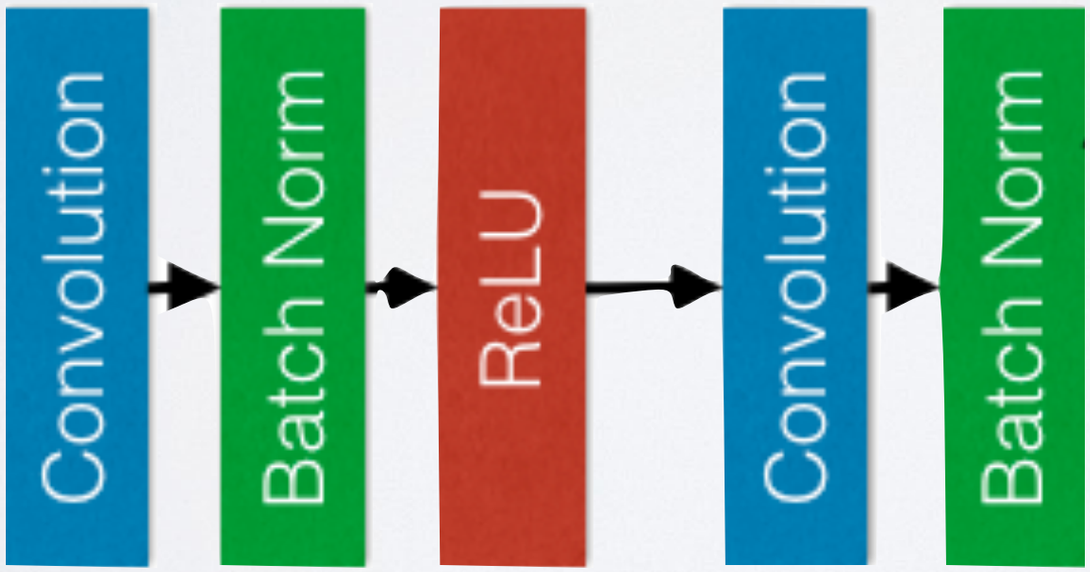
Replace with "running mean"
and "running variance"

$$\gamma \frac{z - \mu}{\sigma} + \eta$$



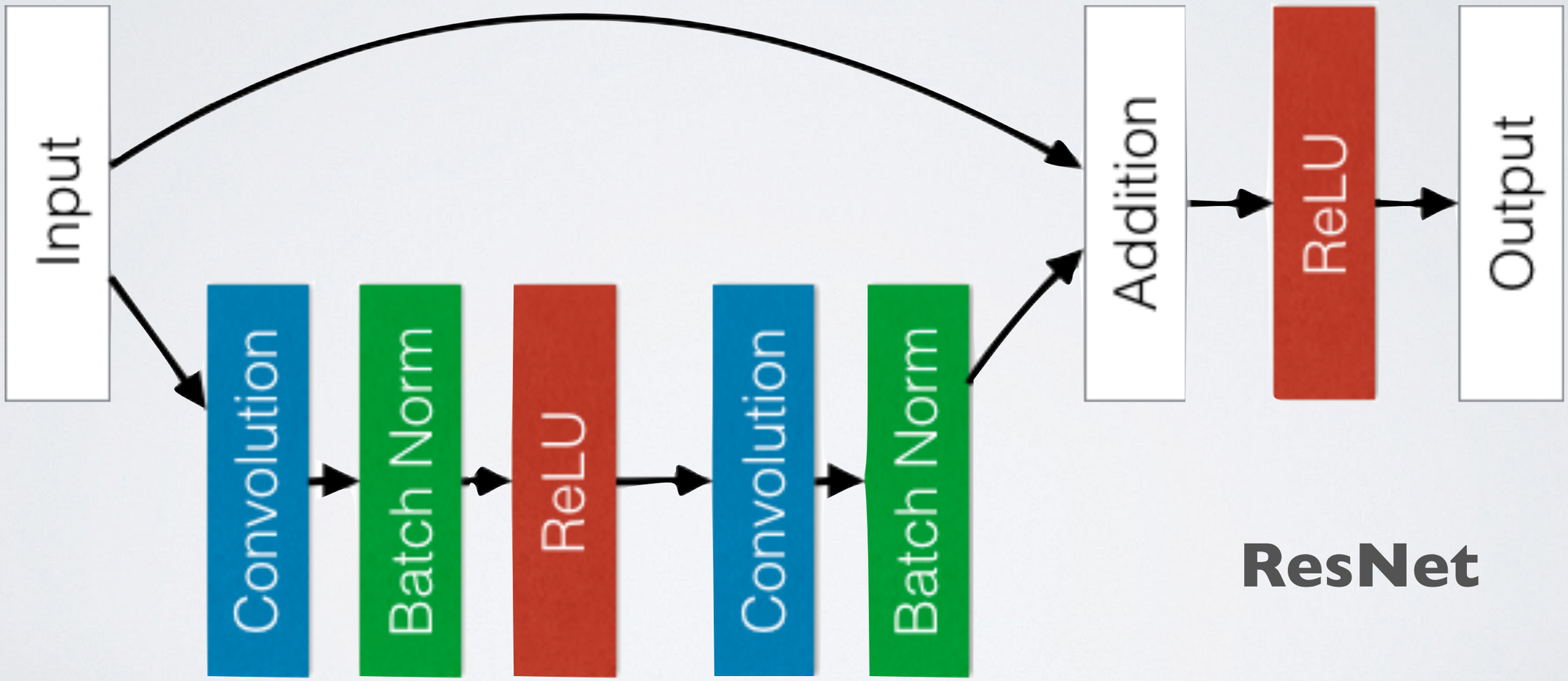
Ioffe & Szegedy, "Batch Normalization…", 2015

# RESIDUAL CONNECTIONS



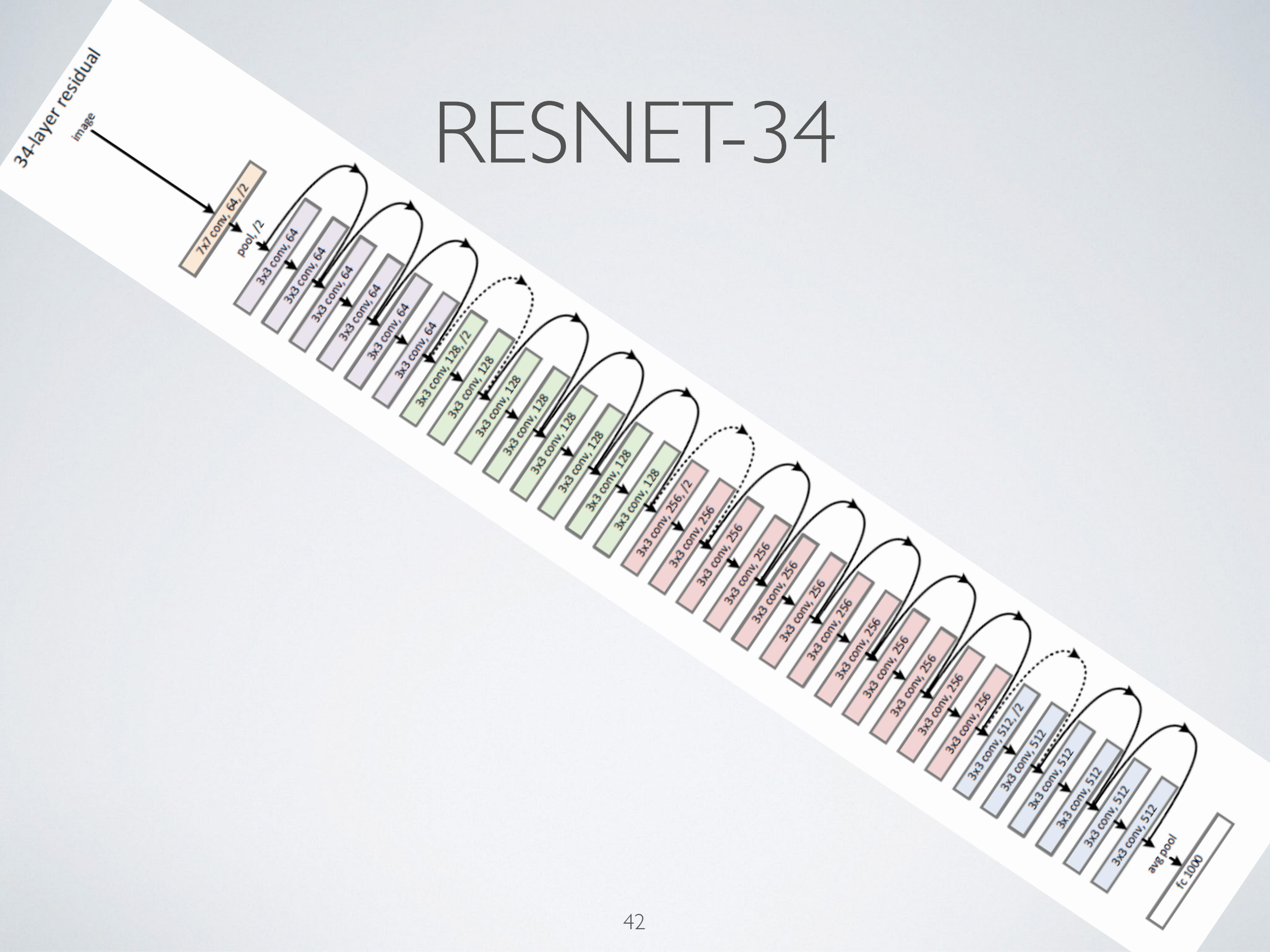

**“VGG”-like Net**

# RESIDUAL CONNECTIONS




**ResNet**

# RESNET-34

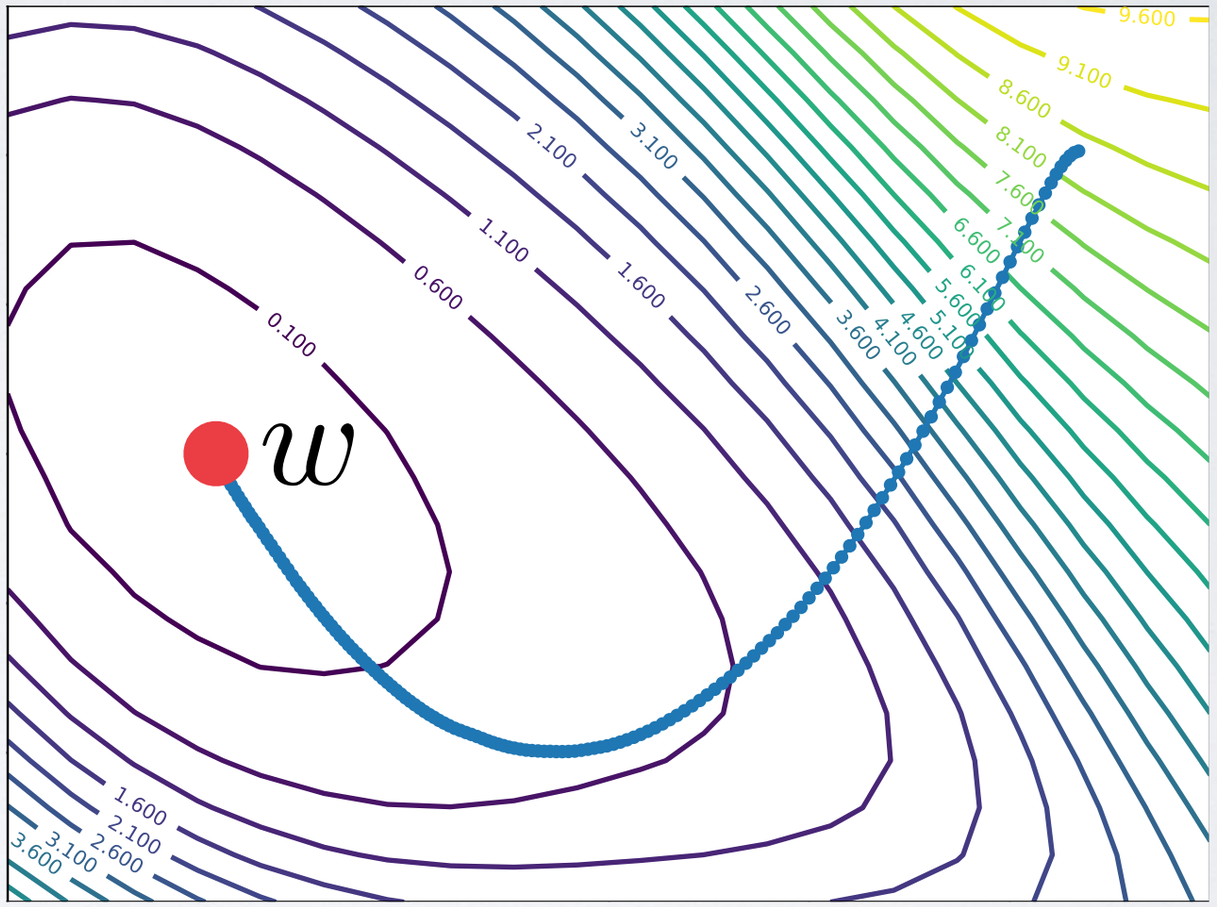

34-layer residual
image
7x7 conv, 64, /2
pool, /2
3x3 conv, 64
3x3 conv, 128, /2
3x3 conv, 128
3x3 conv, 256, /2
3x3 conv, 256
3x3 conv, 512, /2
3x3 conv, 512
avg pool
fc 1000

# WHY ARE SKIP CONNECTIONS GOOD?

# VISUALIZING LOSS FUNCTIONS: FILTER NORMALIZATION

**Step 1:**

**Find minimizer**



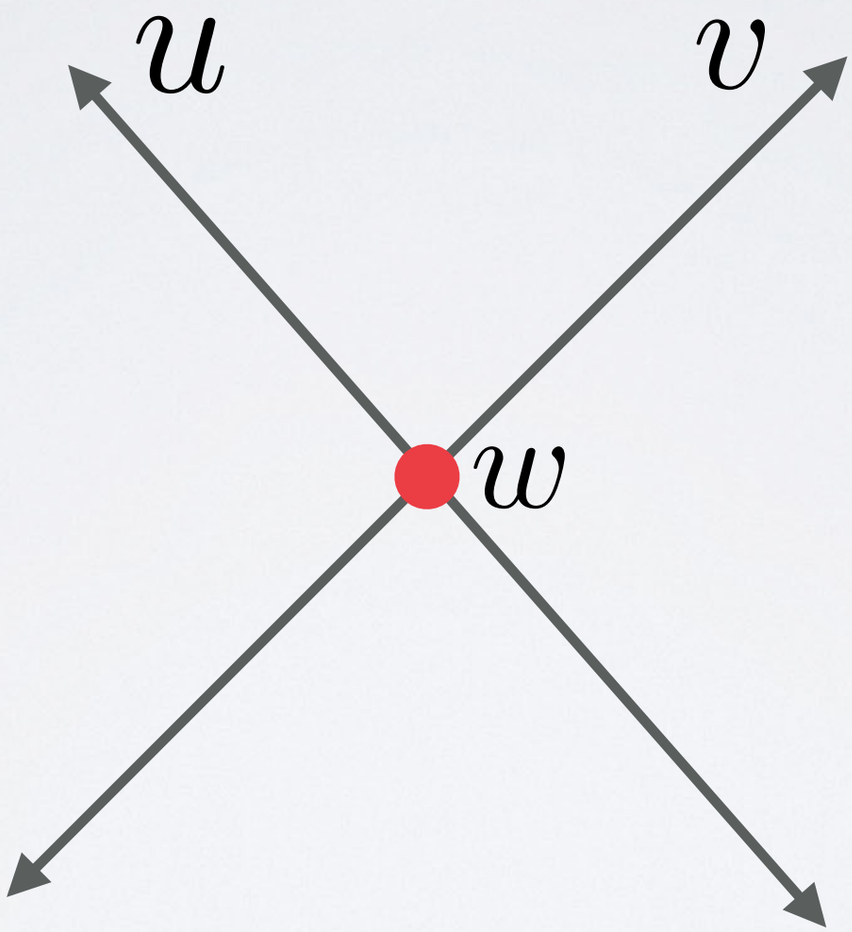

**30 million dimensions**

# VISUALIZING LOSS FUNCTIONS: FILTER NORMALIZATION

**Step 2:**

**Random directions**

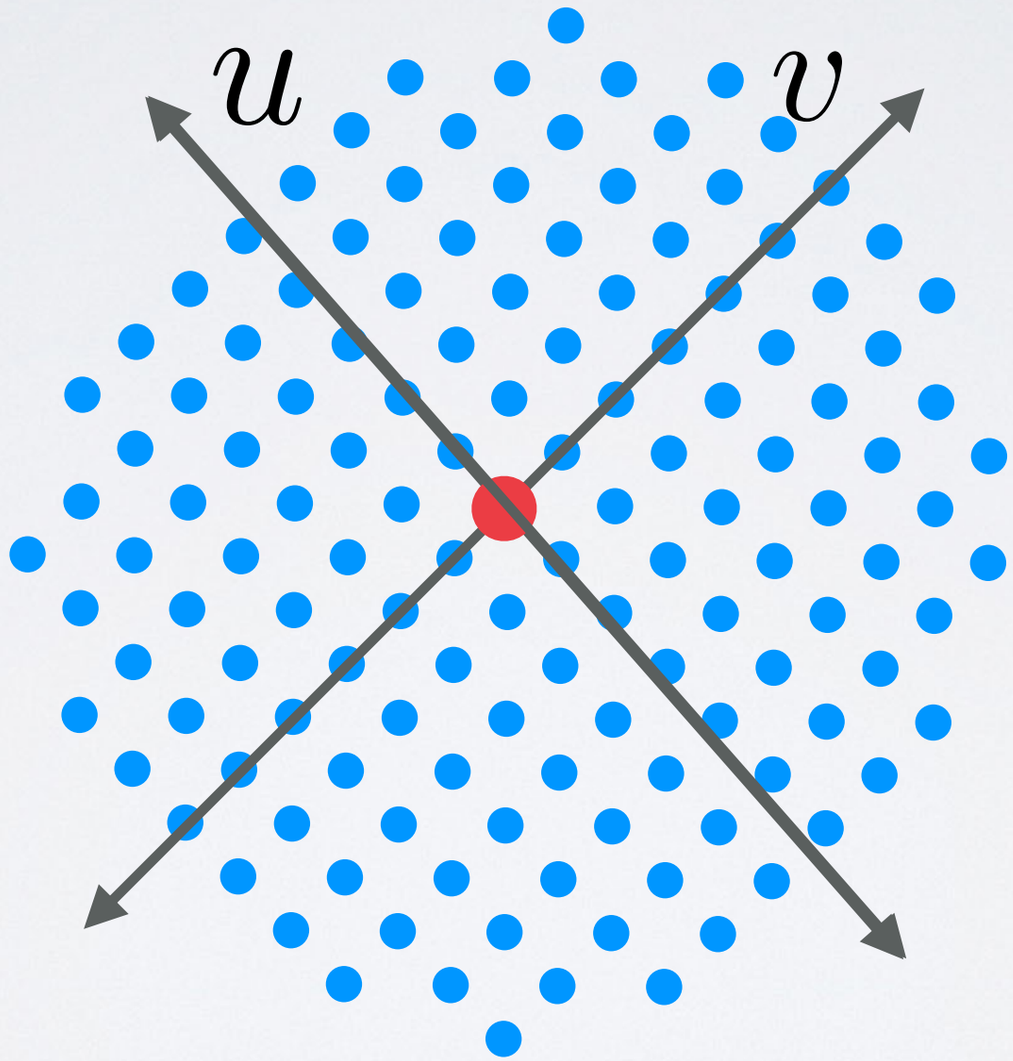
$u, v$



# VISUALIZING LOSS FUNCTIONS: FILTER NORMALIZATION

**Step 3**

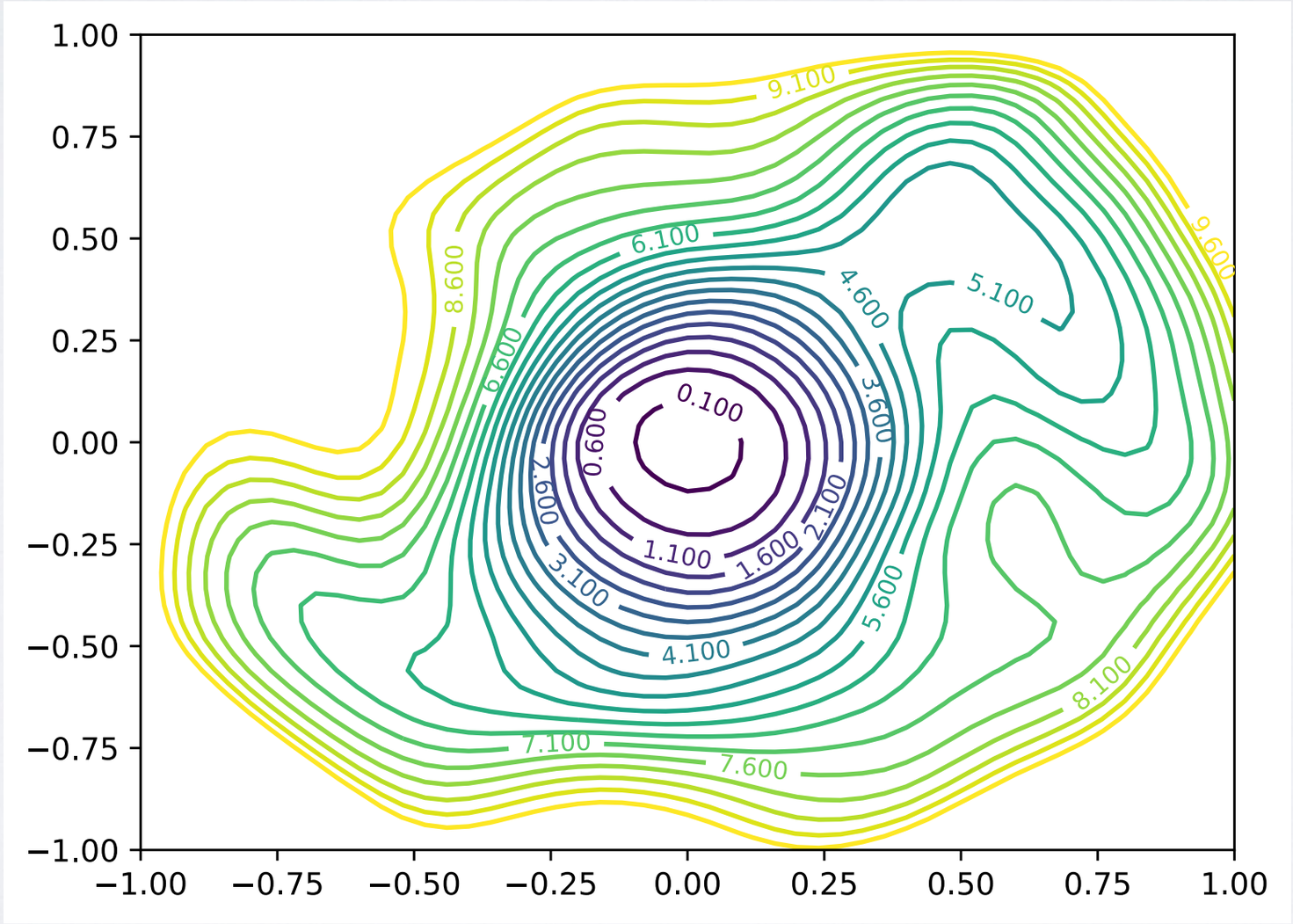
Plot




**Li, Xu, Taylor, G. "Visualizing the loss landscape of neural nets."**

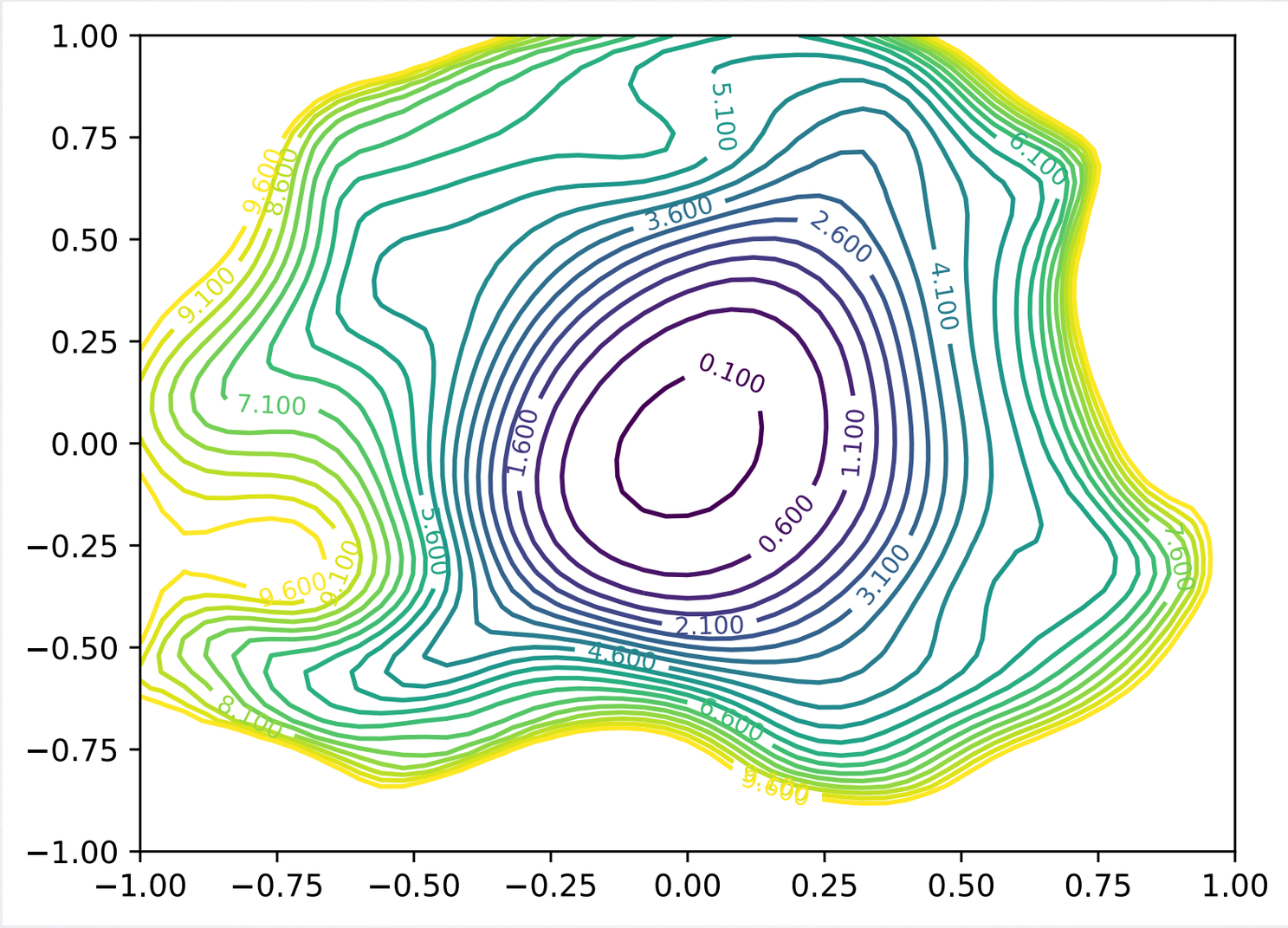
# CHAOTIC TRANSITIONS

## VGG-9



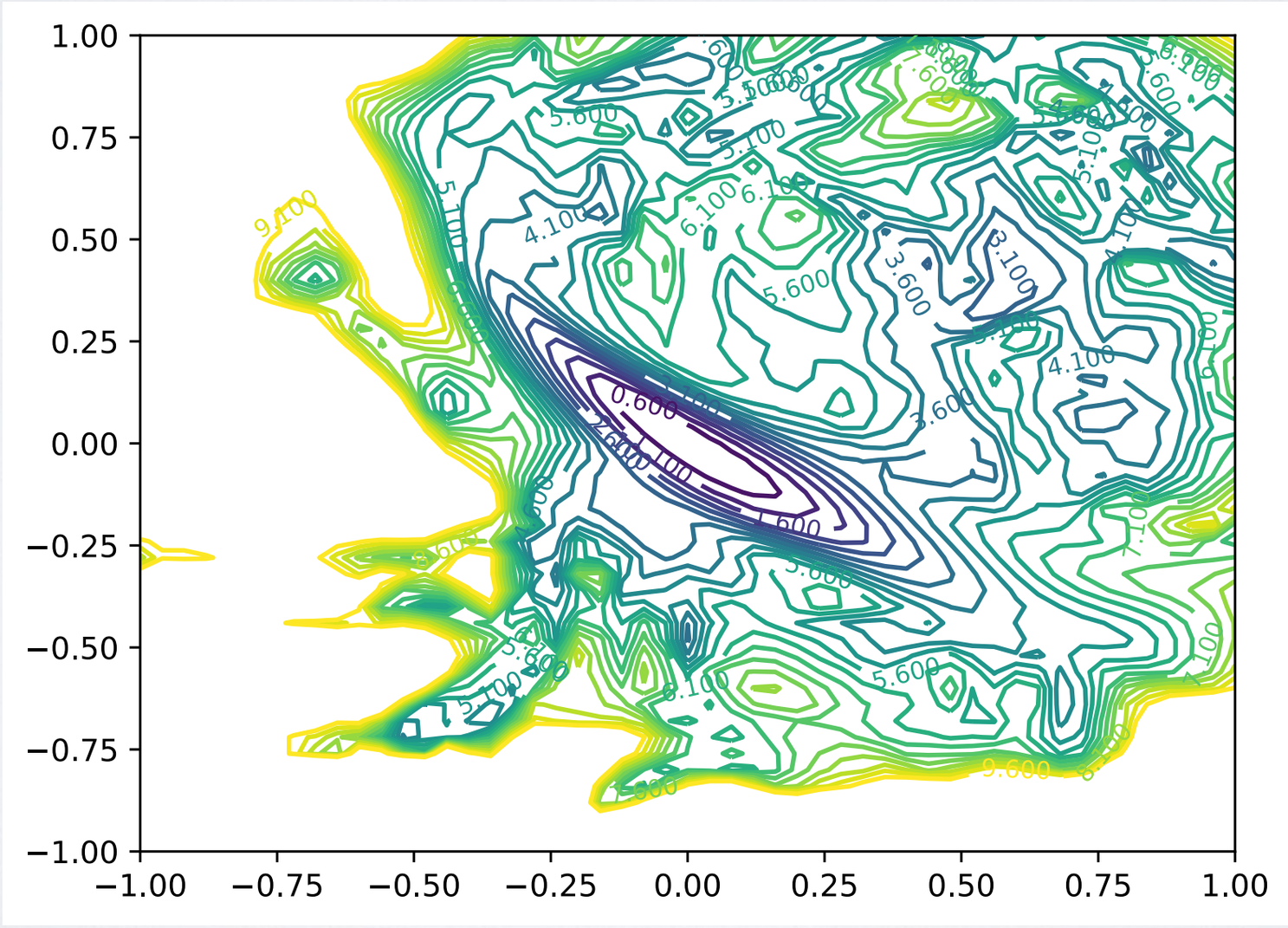
# CHAOTIC TRANSITIONS

## VGG-20



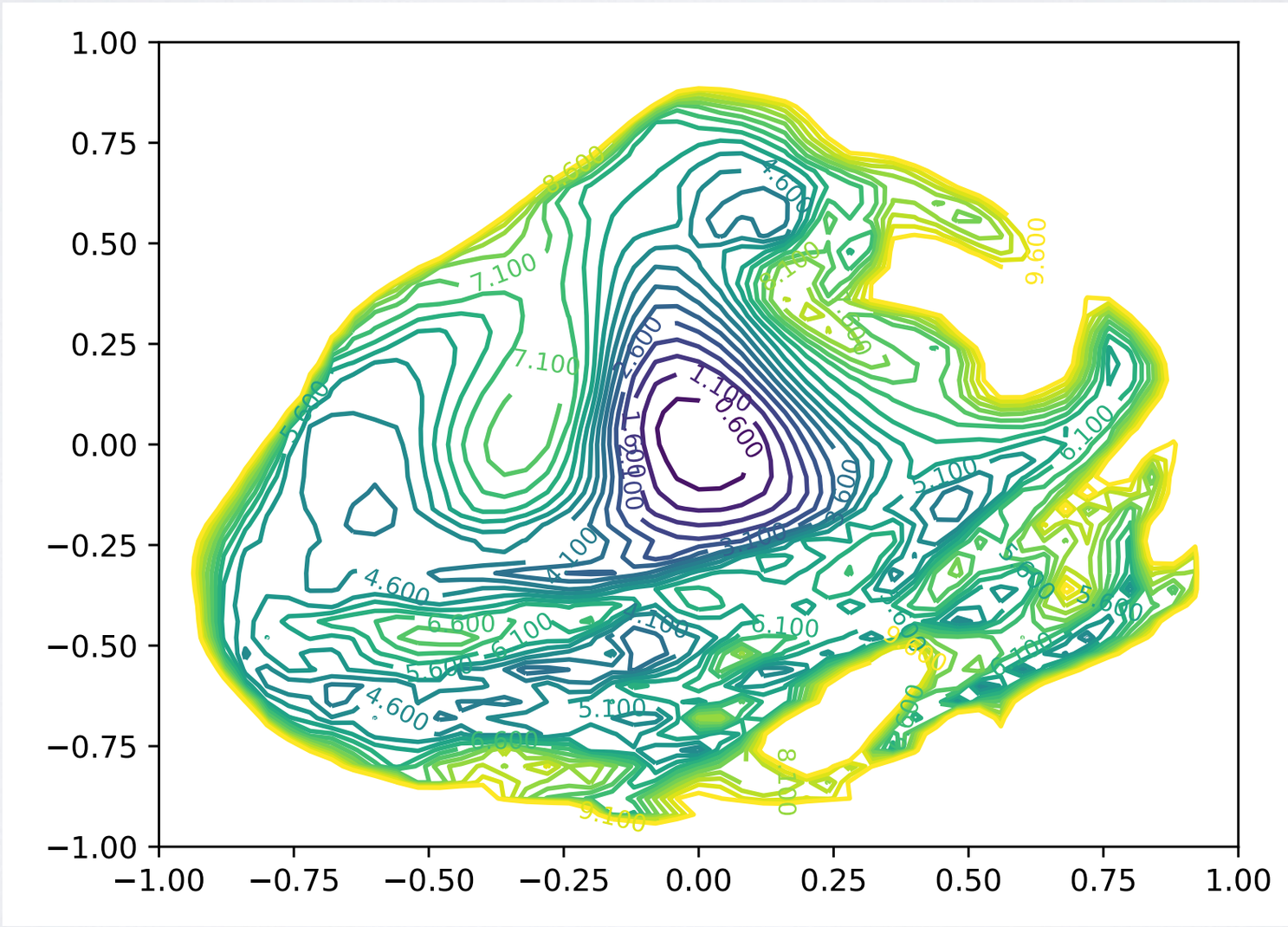
# CHAOTIC TRANSITIONS

## VGG-56



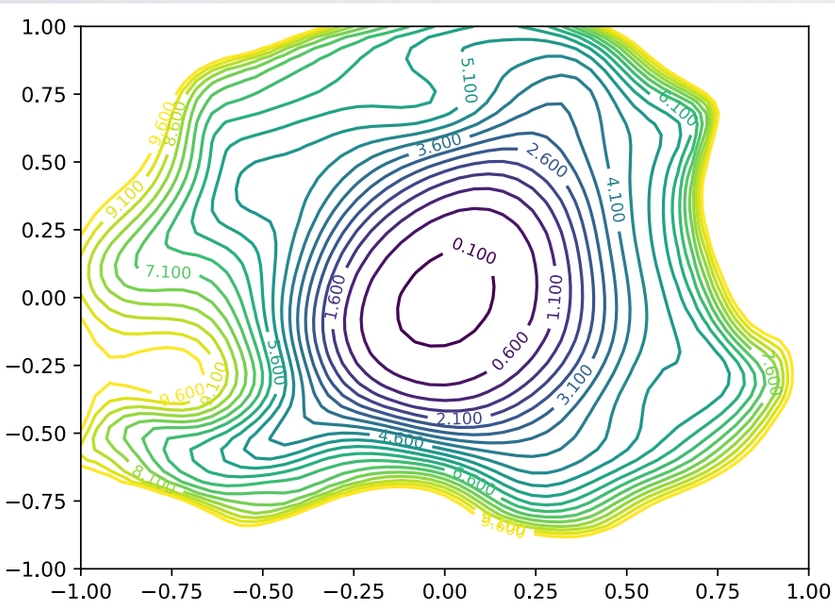
# CHAOTIC TRANSITIONS

## VGG-110

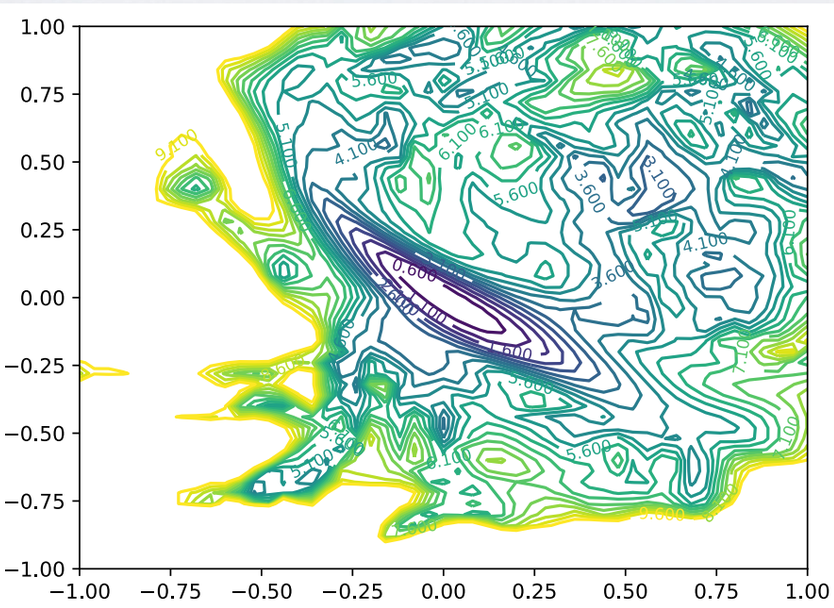


# CHAOTIC TRANSITIONS

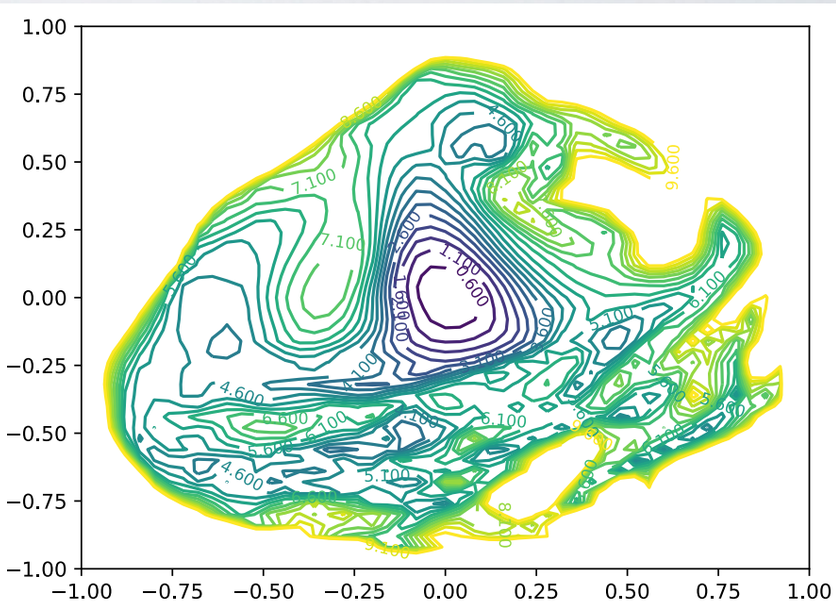
**VGG-20**




**VGG-56**



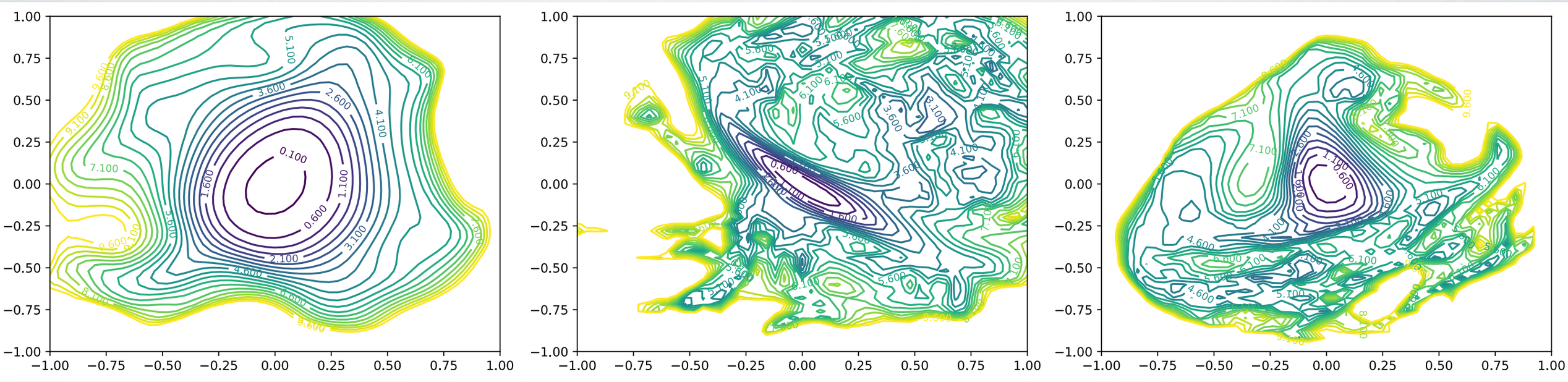

**VGG-110**




**Convexity** → **Chaos**

# CHAOTIC TRANSITIONS

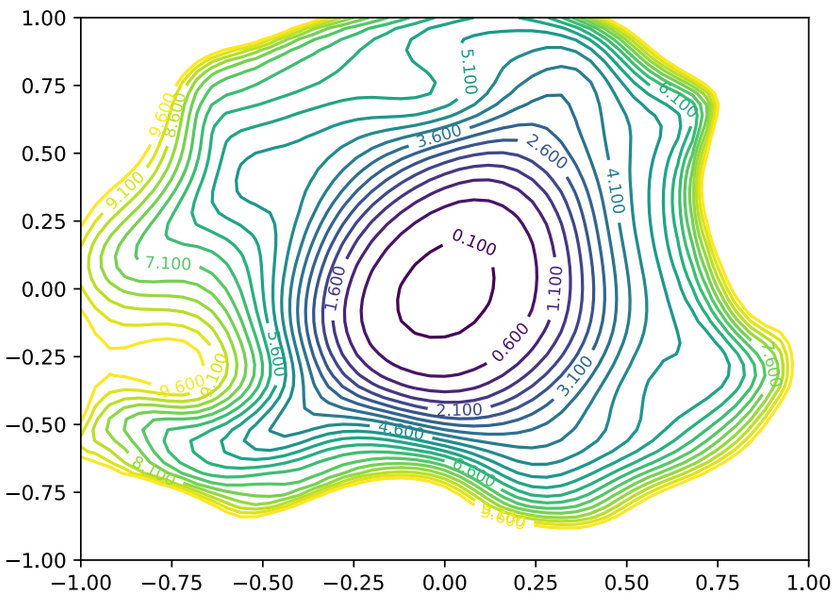
**VGG-20** **VGG-56** **VGG-110**

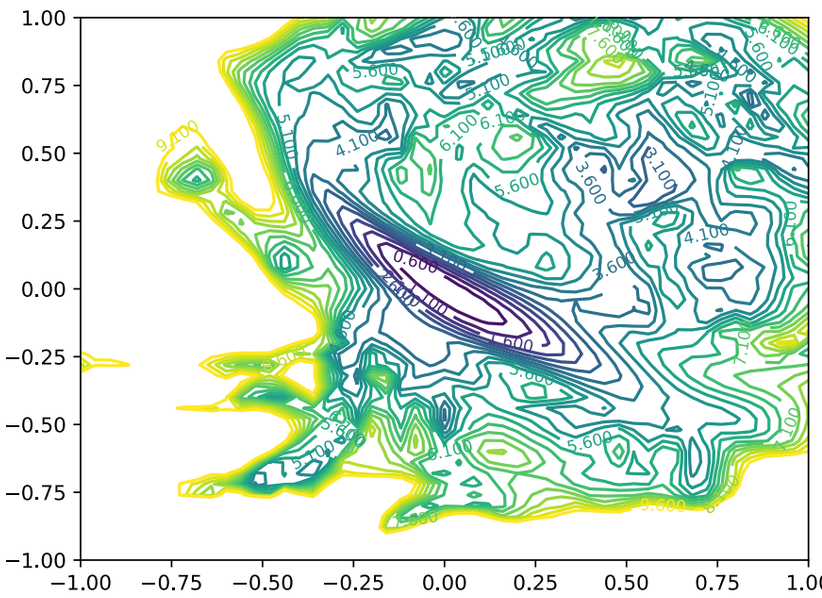


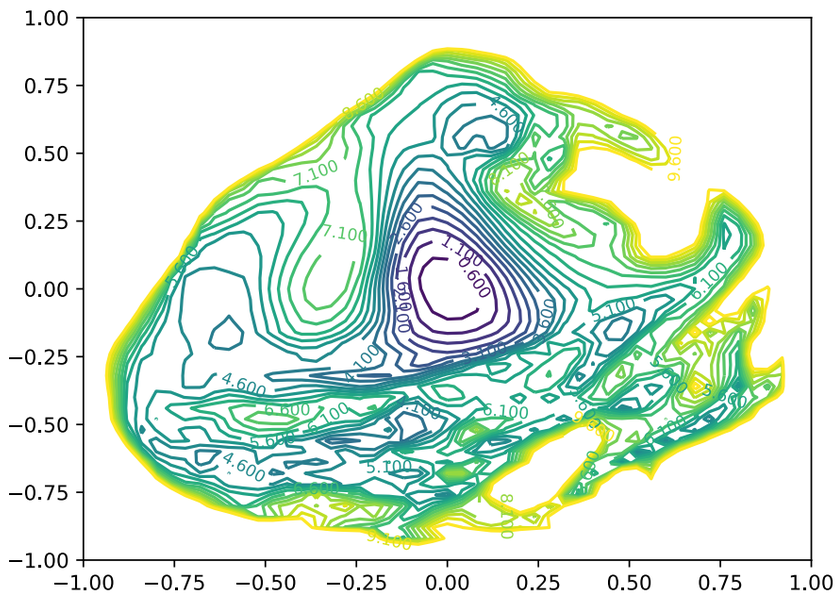
**Convexity** **Chaos**

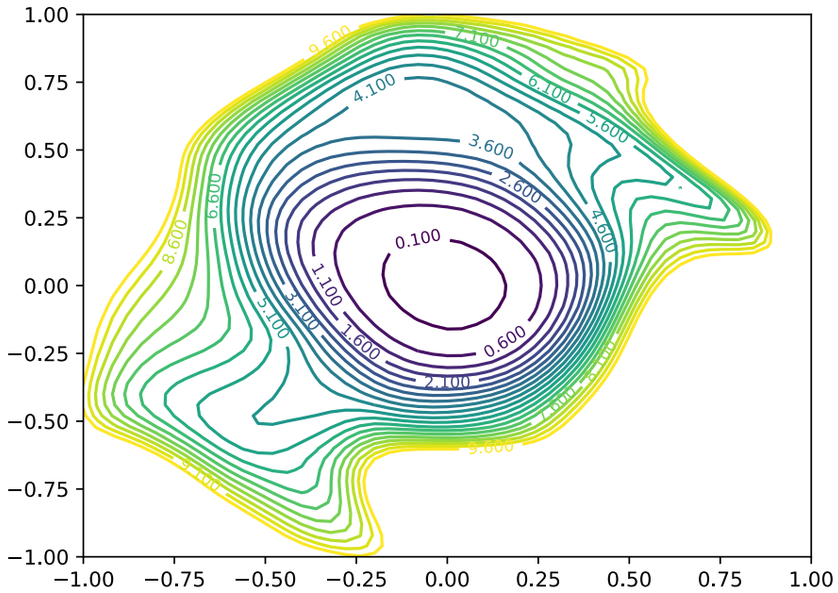
# CHAOTIC TRANSITIONS

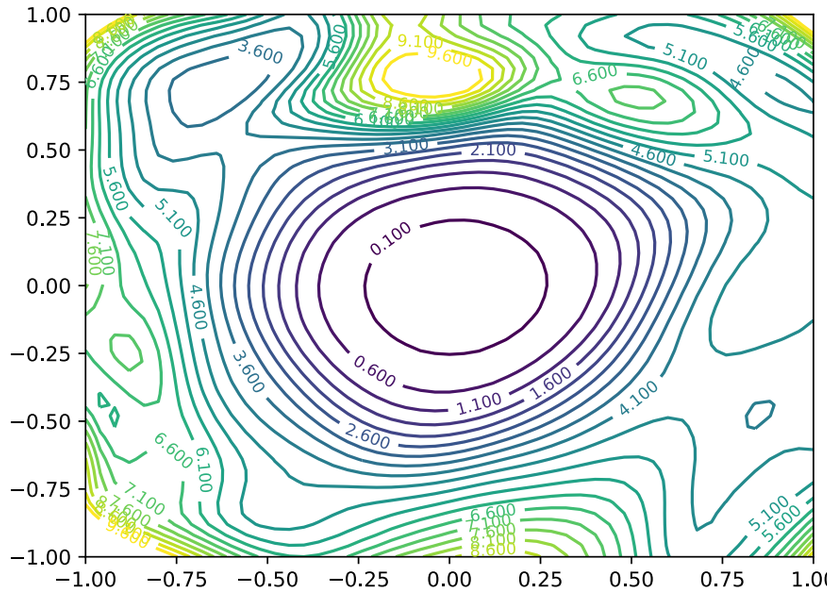
**VGG-20** **VGG-56** **VGG-110**

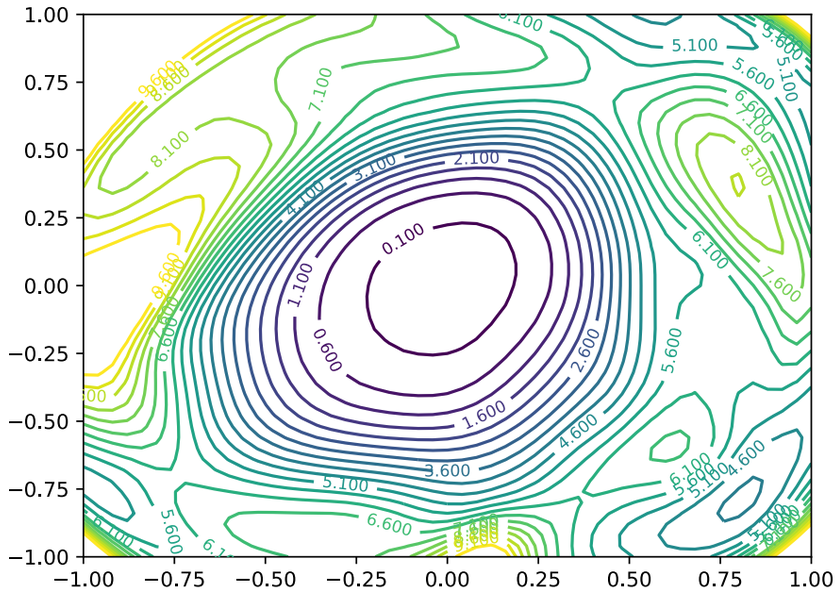






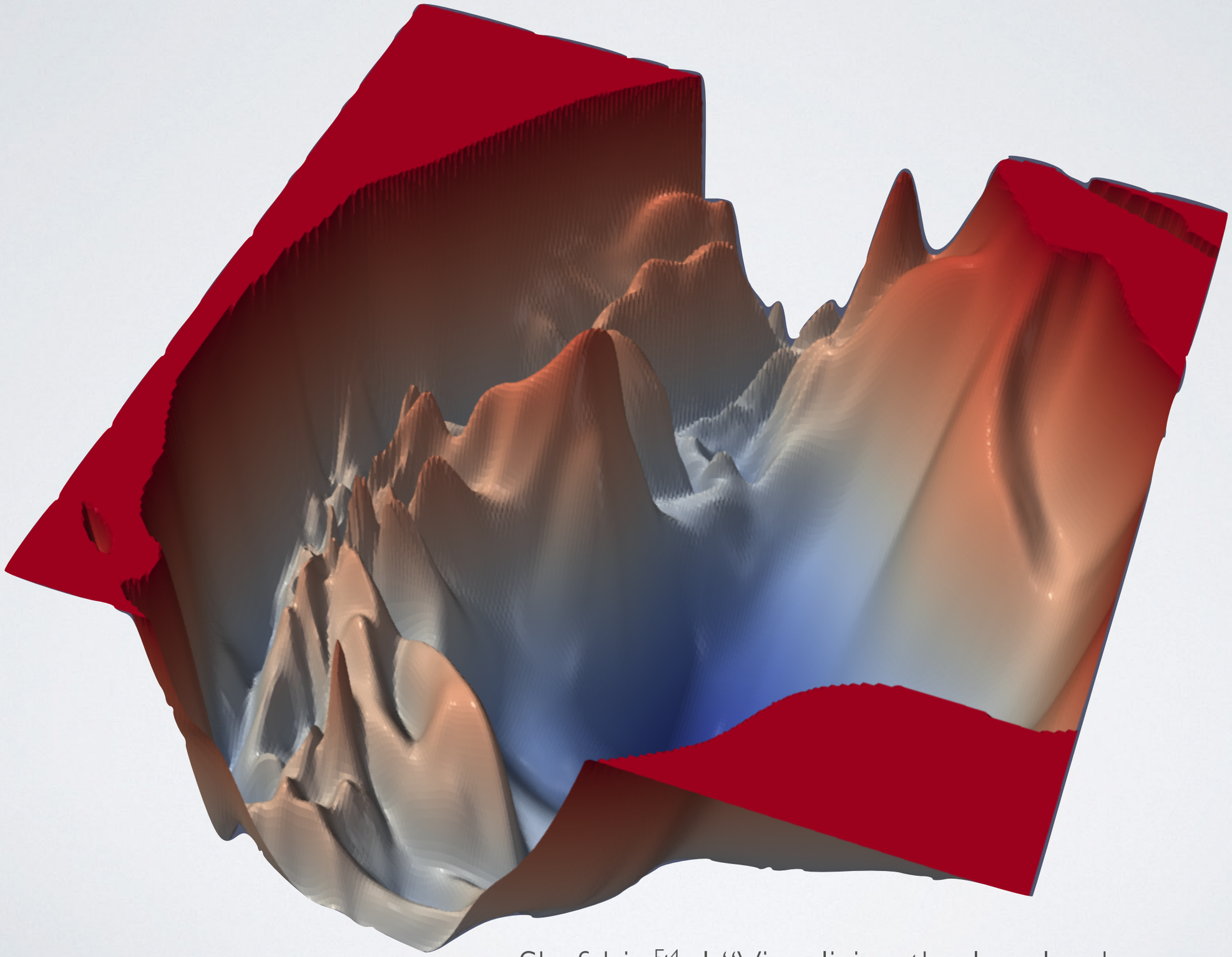






**ResNet-20** **ResNet-56** **ResNet-110**

# VGG-110

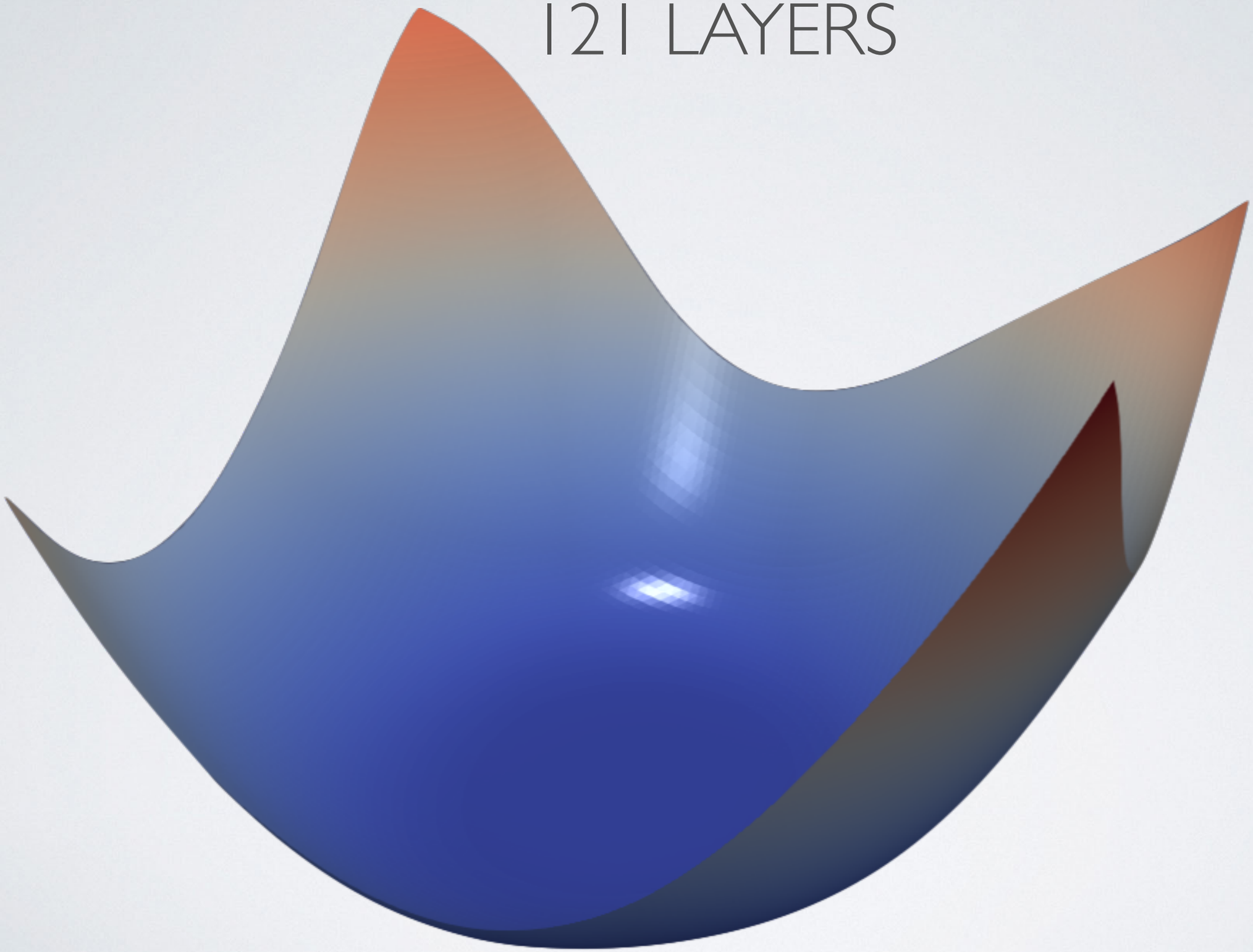




Shafahi et al. "Visualizing the loss landscape of neural nets"

# DENSENET

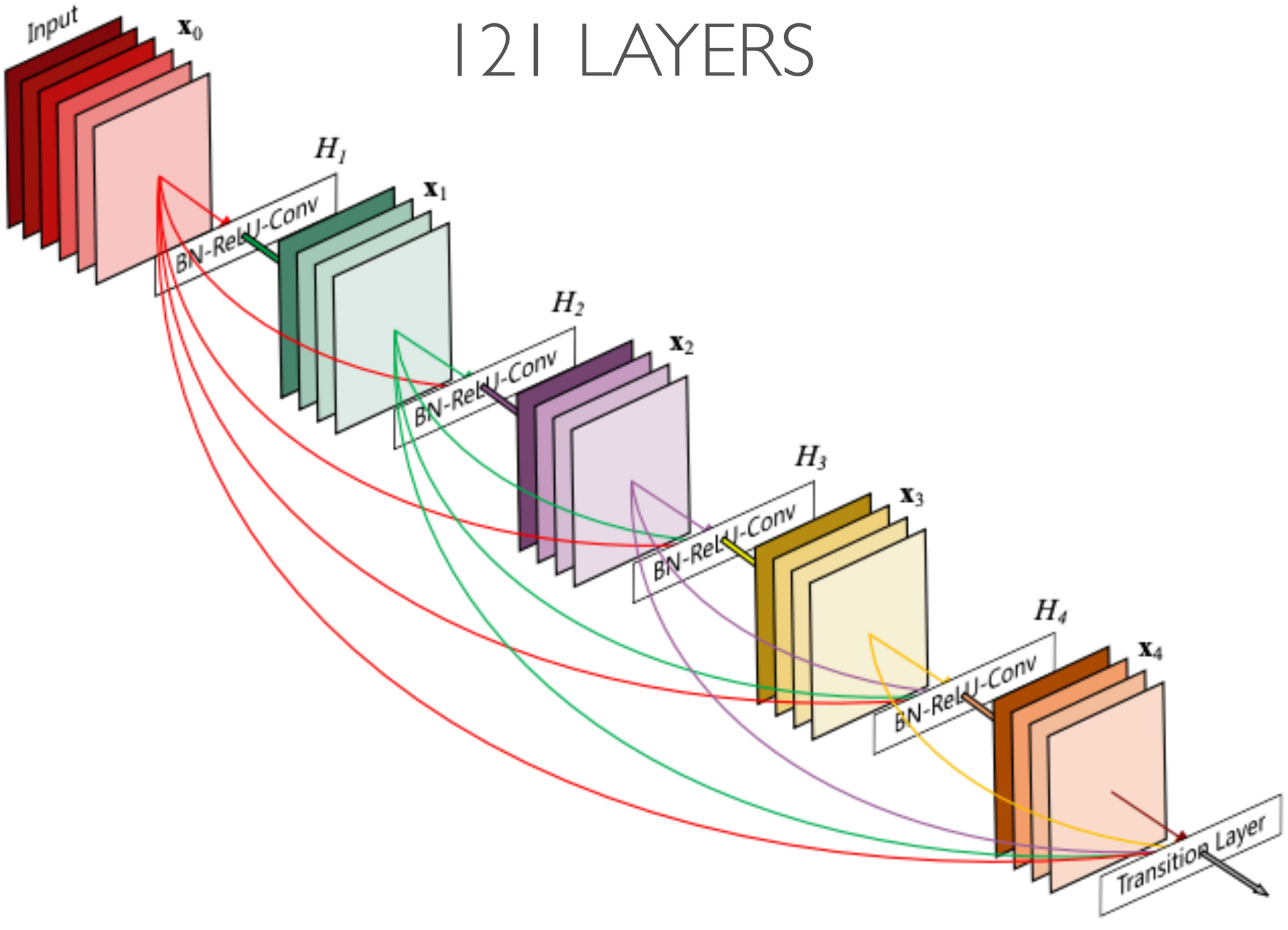
## 121 LAYERS



Shafahi et al. "Visualizing the loss landscape of neural nets"

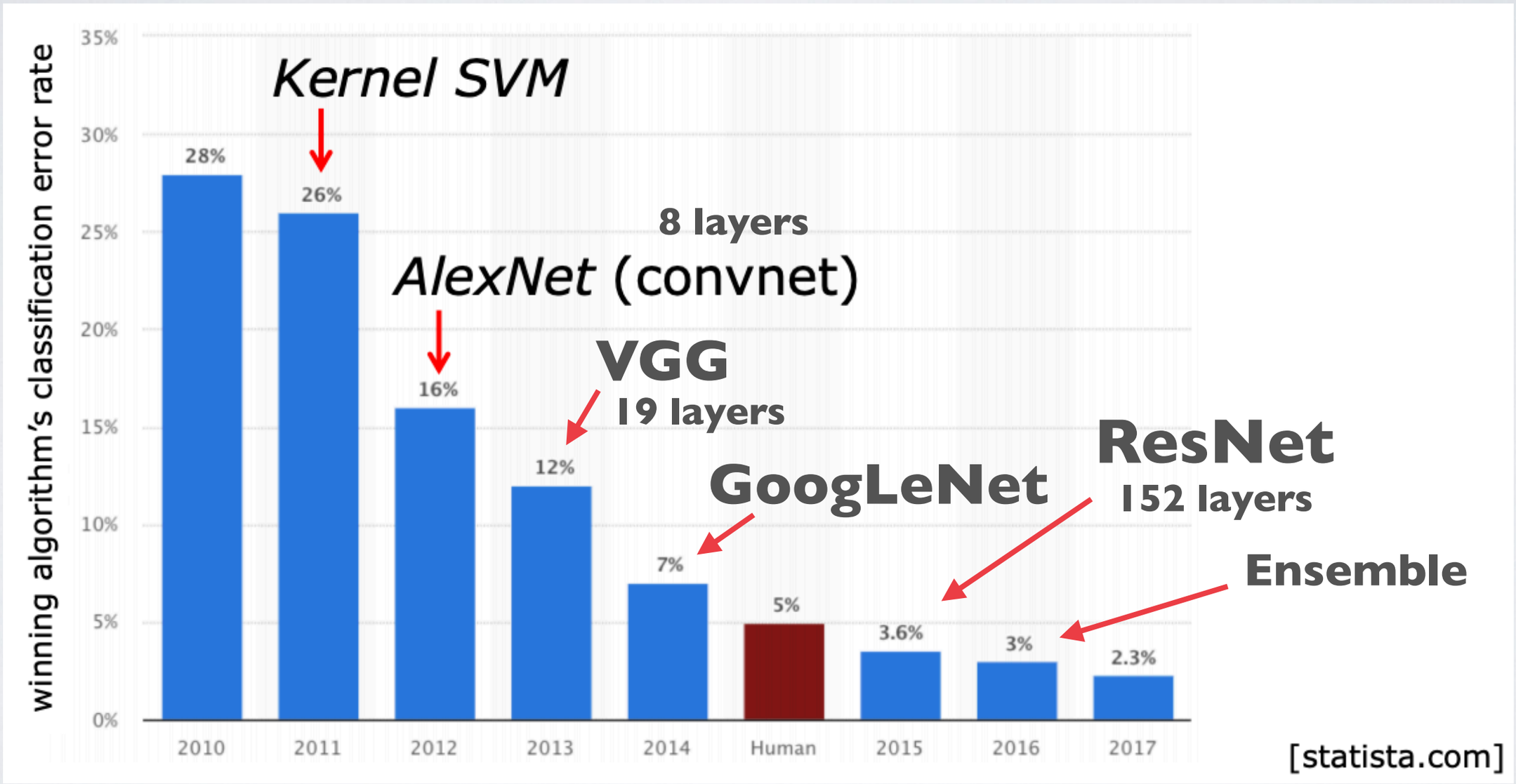
# DENSENET

## 121 LAYERS

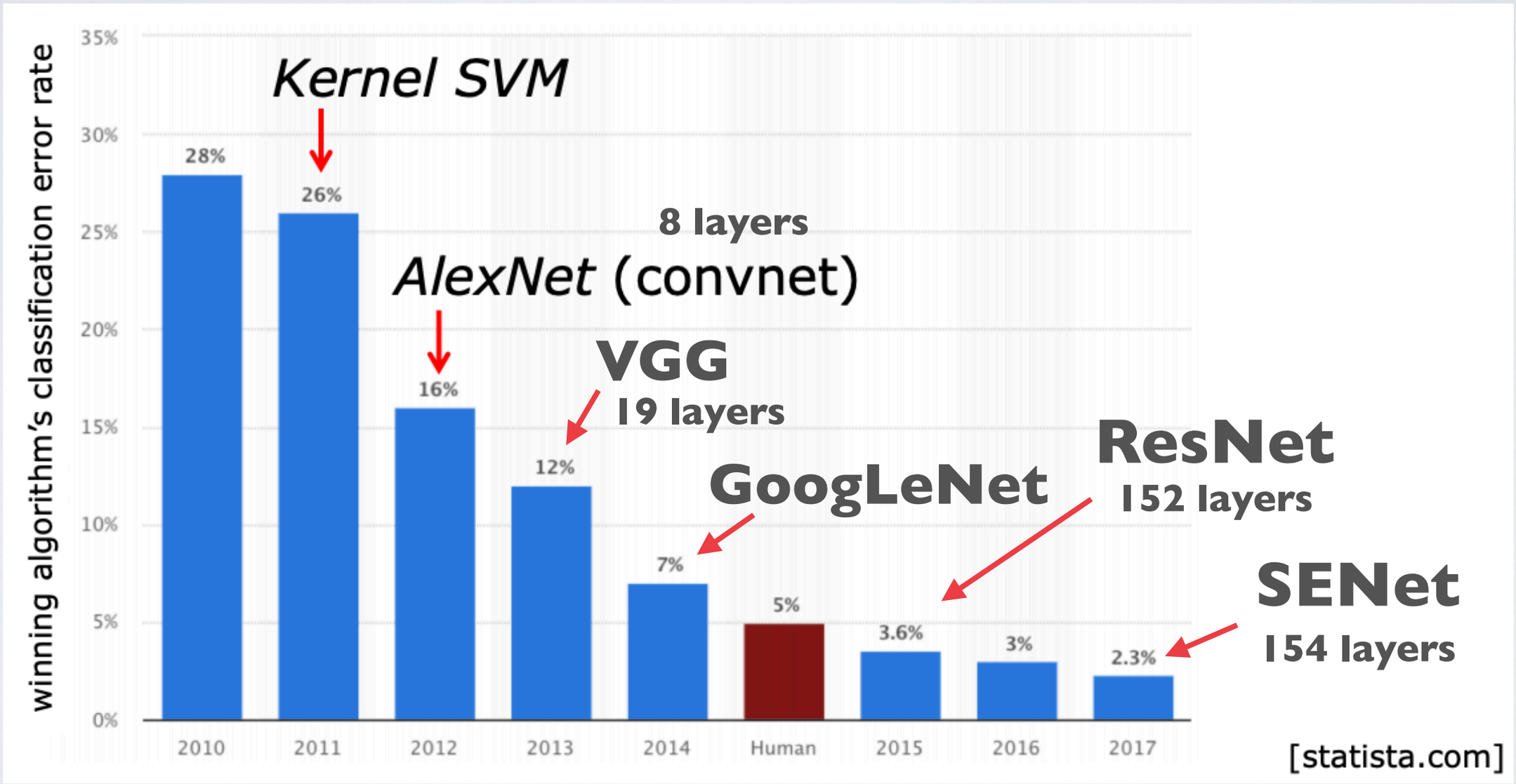



Huang et al, "Densely Connected," 2016

# IMAGENET HISTORY



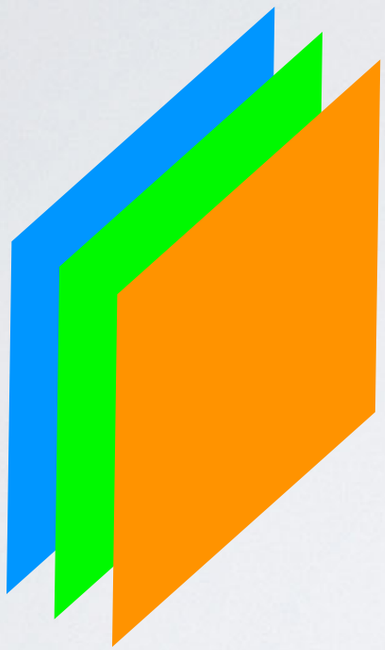
# IMAGENET HISTORY



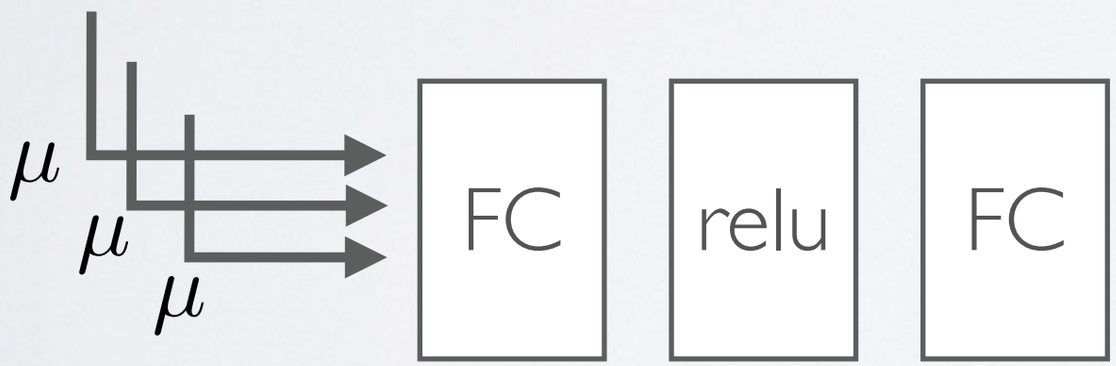
# SENET

## squeeze and excite network

Hu et al 2017

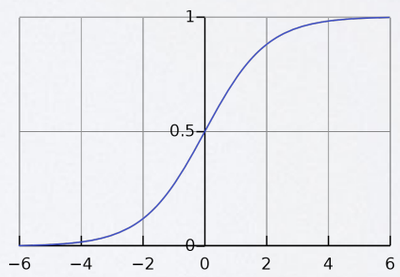


SENet block

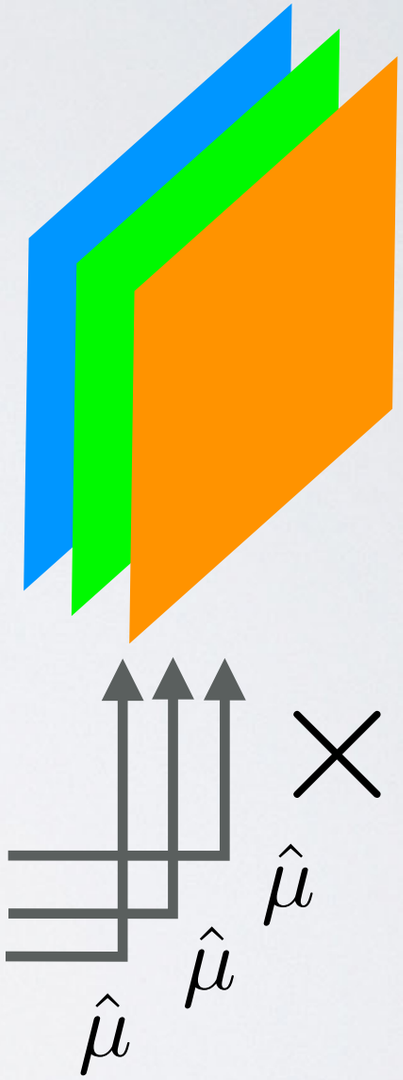



**"squeeze"**

global average pooling




sigmoid

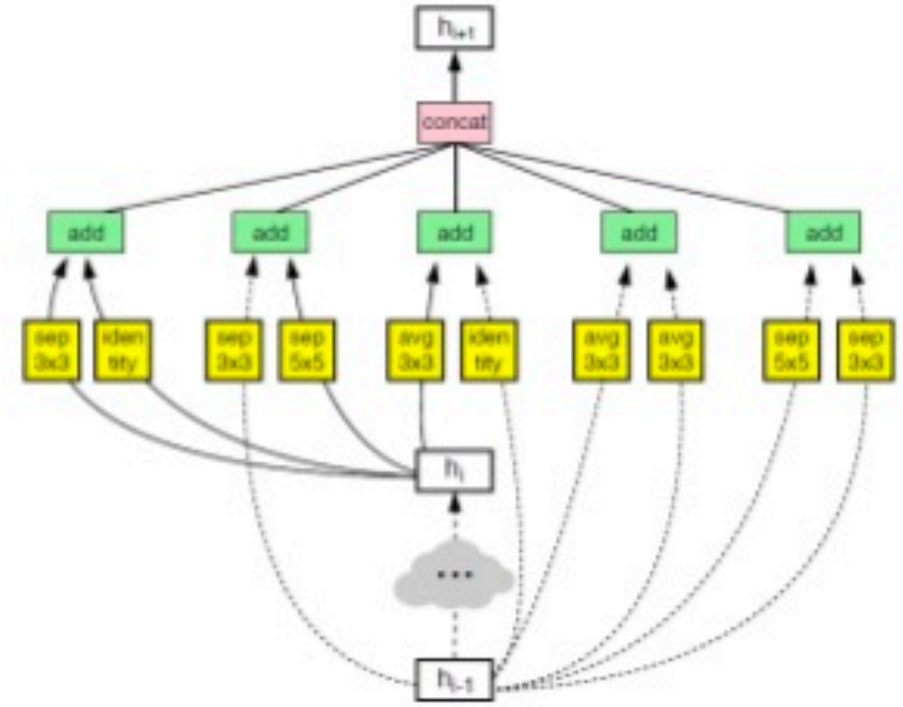



**"excite"**

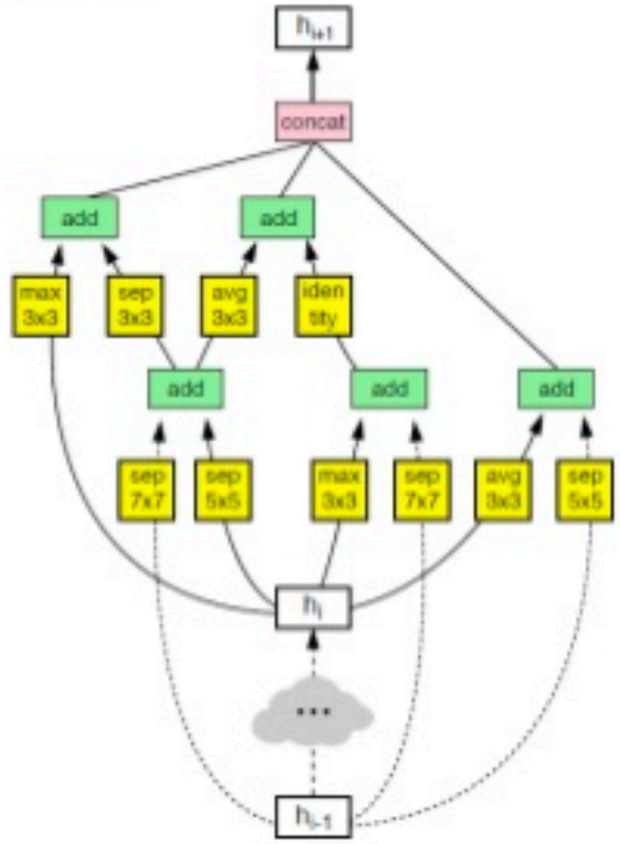
multiply

# AUTO ML

## Best Architecture (NASNet-A)

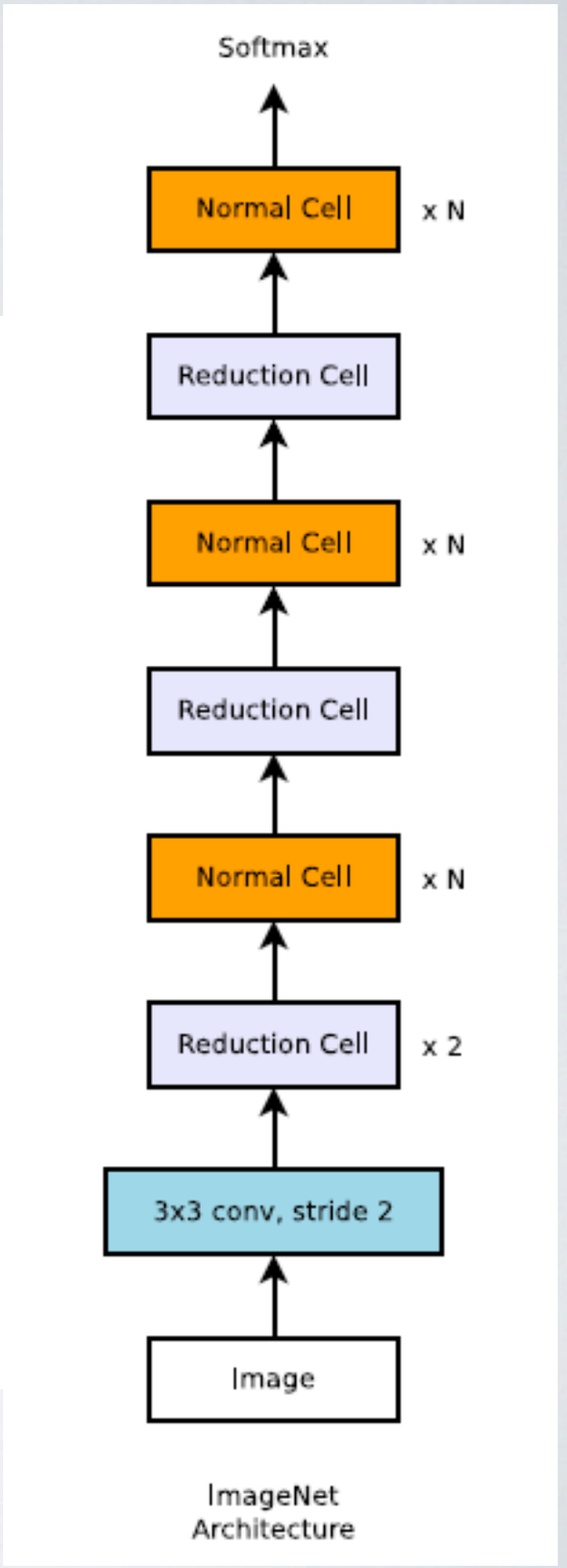



*Normal Cell*




*Reduction Cell*

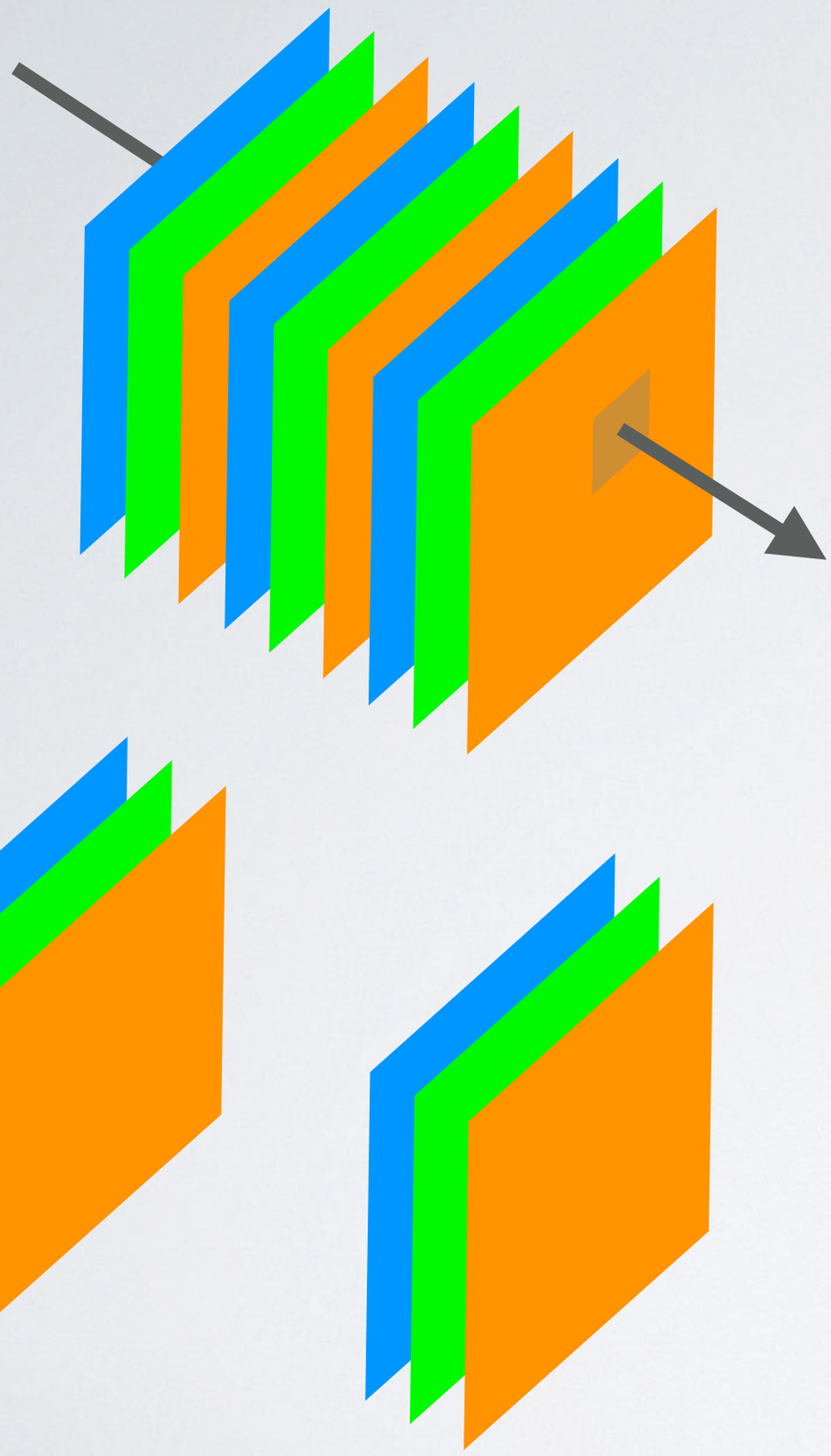
**Zoph et al, 2017**




ImageNet Architecture

# CONTROLLING COMPUTATIONAL COST

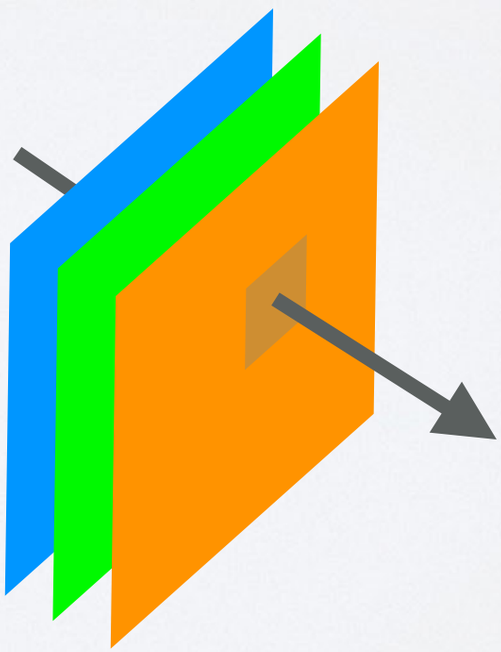
# GROUPED CONVOLUTION



Feature cost

$$k^2 \times C$$





Feature cost

$$k^2 \times (C/g)$$

# SEPARABLE CONV

MobileNet
2017

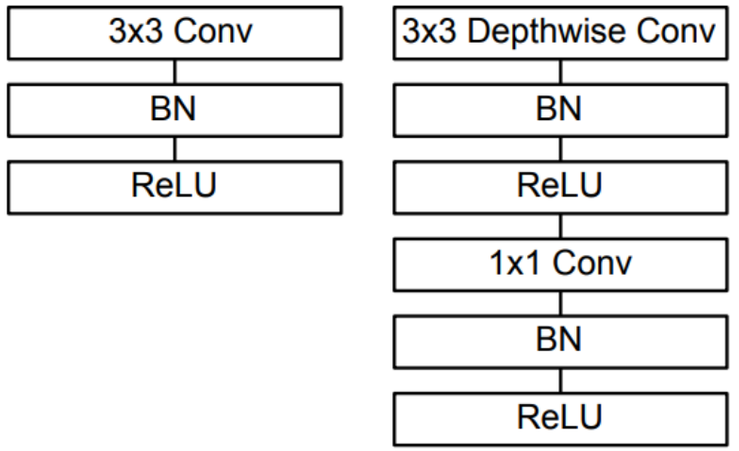
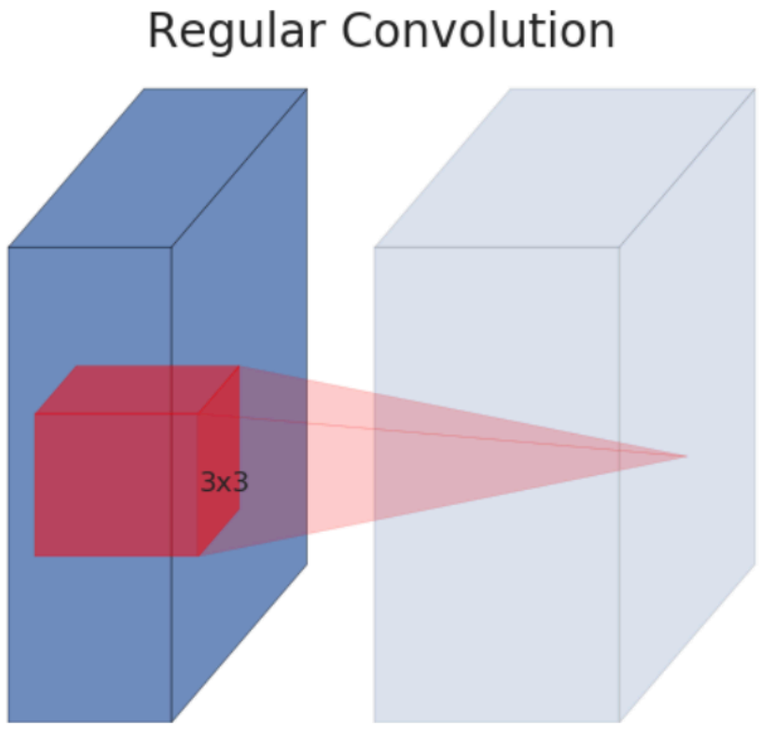
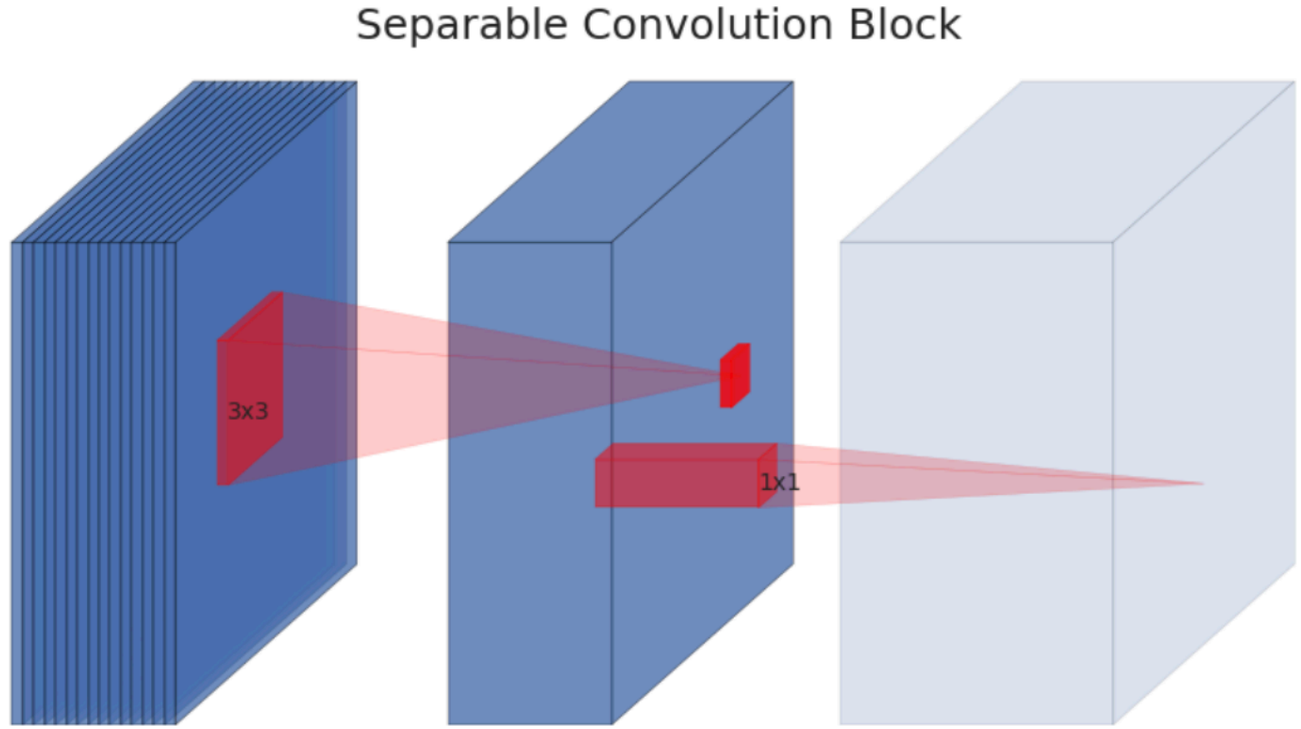



Figure 3. Left: Standard convolutional layer with batchnorm and ReLU. Right: Depthwise Separable convolutions with Depthwise and Pointwise layers followed by batchnorm and ReLU.



# BOTTLENECKS

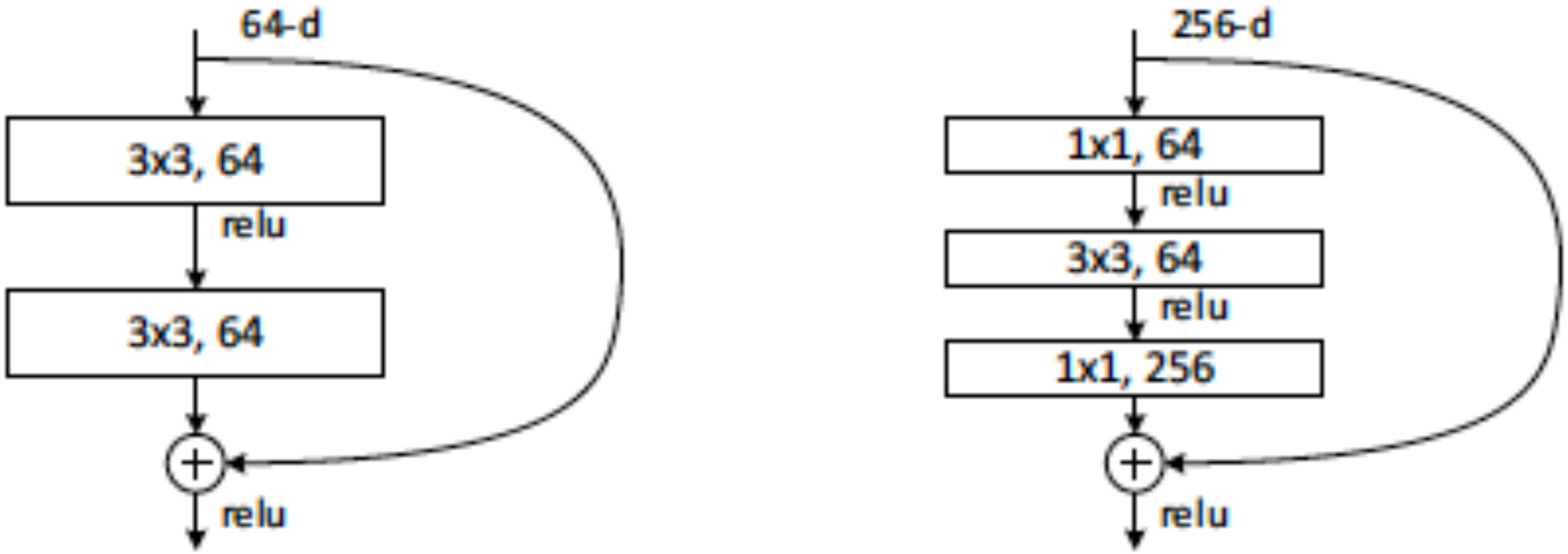



Figure 5. A deeper residual function $\mathcal{F}$ for ImageNet. Left: a building block (on 56×56 feature maps) as in Fig. 3 for ResNet-34. Right: a "bottleneck" building block for ResNet-50/101/152.

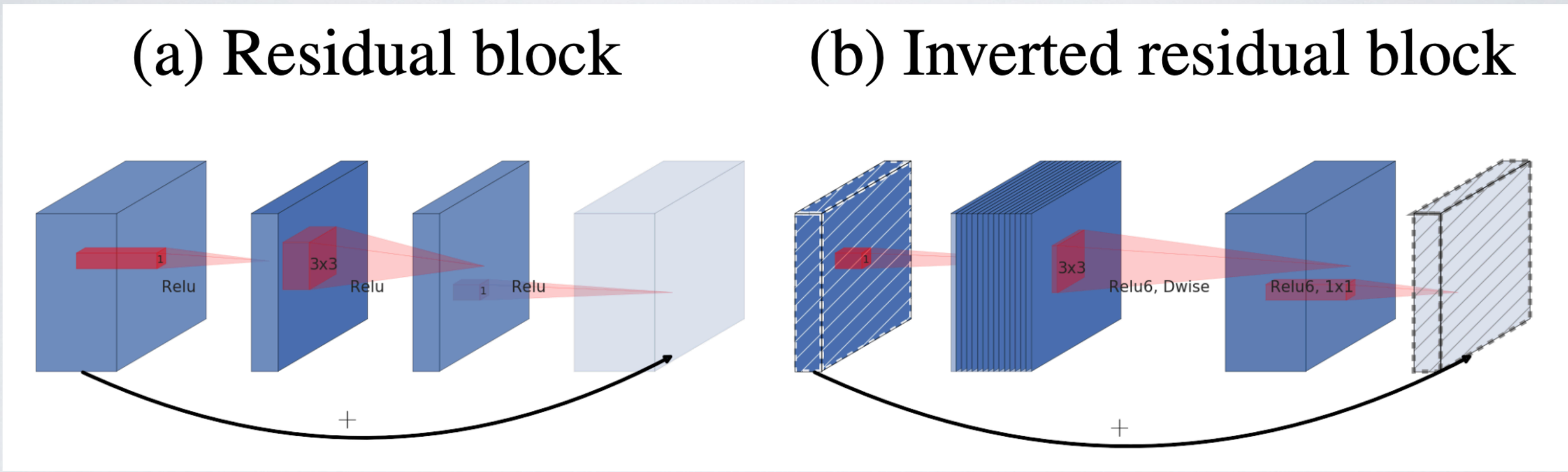


Stolen from He 2015

# INVERTED BOTTLENECK

MobileNetv2

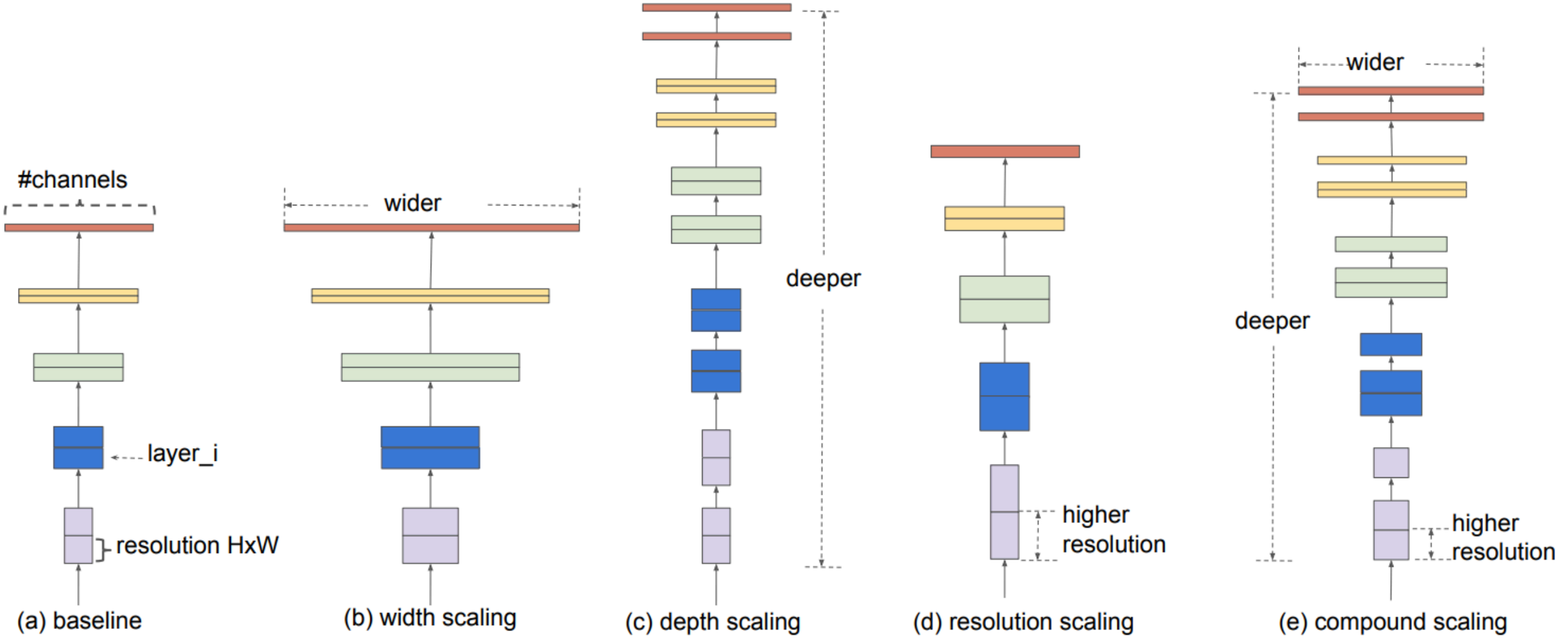
2019




Hashed boxed have no non-linearity

# EfficientNet

2020



# EfficientNet

2020

Optimize scaling parameters to get best performance for fixed cost

Do architecture search once on small models, then scale it up

$$\text{depth: } d = \alpha^\phi$$
$$\text{width: } w = \beta^\phi$$
$$\text{resolution: } r = \gamma^\phi$$
$$\text{s.t. } \alpha \cdot \beta^2 \cdot \gamma^2 \approx 2$$
$$\alpha \geq 1, \beta \geq 1, \gamma \geq 1$$

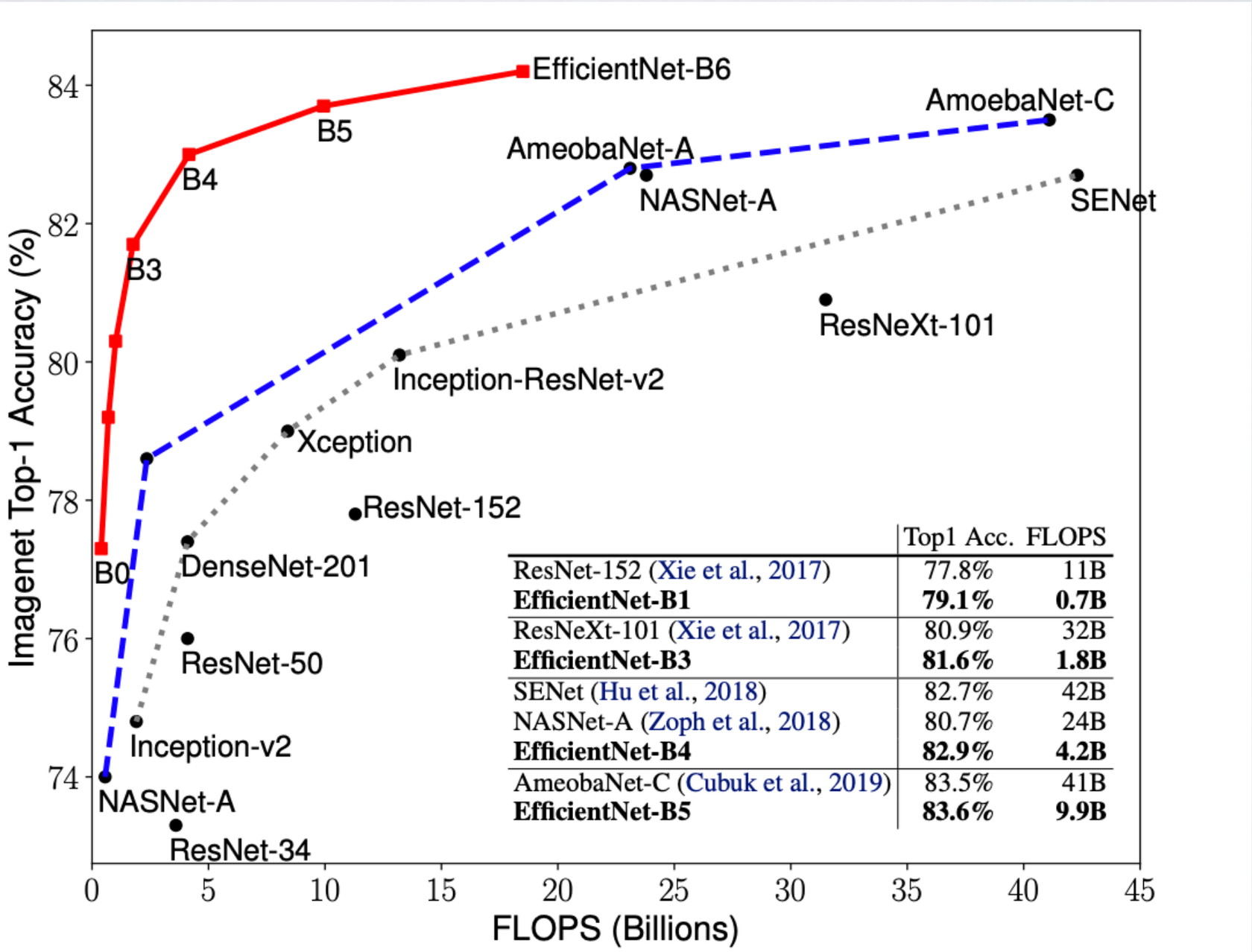
EfficientNet-B0

Based on MobileNet

| Stage $i$ | Operator $\hat{\mathcal{F}}_i$ | Resolution $\hat{H}_i \times \hat{W}_i$ | #Channels $\hat{C}_i$ | #Layers $\hat{L}_i$ |
|---|---|---|---|---|
| 1 | Conv3x3 | $224 \times 224$ | 32 | 1 |
| 2 | MBConv1, k3x3 | $112 \times 112$ | 16 | 1 |
| 3 | MBConv6, k3x3 | $112 \times 112$ | 24 | 2 |
| 4 | MBConv6, k5x5 | $56 \times 56$ | 40 | 2 |
| 5 | MBConv6, k3x3 | $28 \times 28$ | 80 | 3 |
| 6 | MBConv6, k5x5 | $14 \times 14$ | 112 | 3 |
| 7 | MBConv6, k5x5 | $14 \times 14$ | 192 | 4 |
| 8 | MBConv6, k3x3 | $7 \times 7$ | 320 | 1 |
| 9 | Conv1x1 & Pooling & FC | $7 \times 7$ | 1280 | 1 |

# EfficientNet

2020




| | Top1 Acc. | FLOPS |
|---|---|---|
| ResNet-152 (Xie et al., 2017) | 77.8% | 11B |
| **EfficientNet-B1** | **79.1%** | **0.7B** |
| ResNeXt-101 (Xie et al., 2017) | 80.9% | 32B |
| **EfficientNet-B3** | **81.6%** | **1.8B** |
| SENet (Hu et al., 2018) | 82.7% | 42B |
| NASNet-A (Zoph et al., 2018) | 80.7% | 24B |
| **EfficientNet-B4** | **82.9%** | **4.2B** |
| AmeobaNet-C (Cubuk et al., 2019) | 83.5% | 41B |
| **EfficientNet-B5** | **83.6%** | **9.9B** |

*Figure 5.* **FLOPS vs. ImageNet Accuracy –** Similar to Figure 1 except it compares FLOPS rather than model size.

# Revisiting ResNets

2021

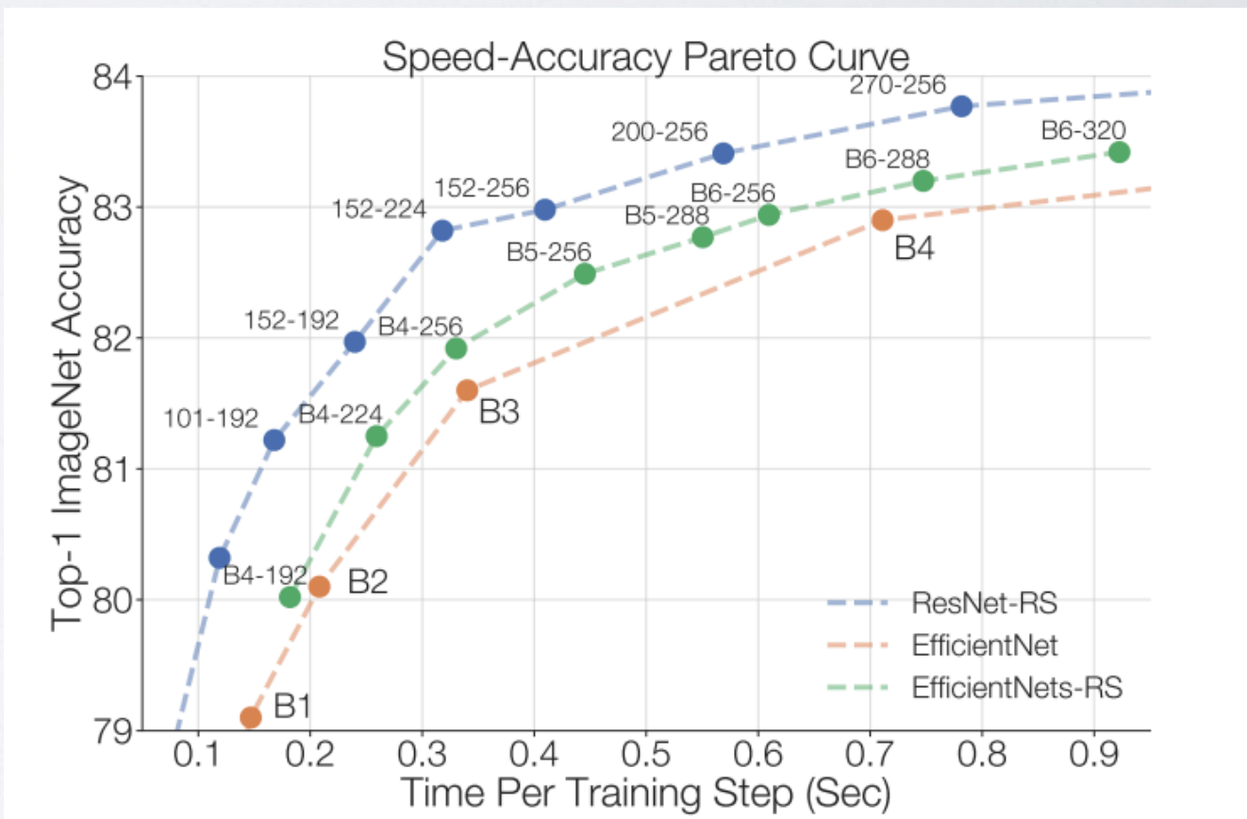
Train (modified) ResNets with modern tricks

The resolution scaling of EfficientNet is inefficient

ResNets are faster than EffNets
despite having more FLOPs

| Model | RS-350 | ENet-B6 | RS-420 | ENet-B7 |
|---|---|---|---|---|
| Resolution | 256 | 528 | 320 | 600 |
| Top-1 Acc. | **84.0** | **84.0** | 84.4 | **84.7** |
| Params (M) | 164 | 43 (3.8x) | 192 | 66 (2.9x) |
| FLOPs (B) | 69 | 38 (1.8x) | 128 | 74 (1.7x) |
| TPU-v3 | | | | |
| Latency (s) | 1.1 | 3.0 (2.7x) | 2.1 | 6.0 (2.9x) |
| Memory (GB) | 7.3 | 16.6 (2.3x) | 15.5 | 28.3 (1.8x) |
| V100 | | | | |
| Latency (s) | 4.7 | 15.7 (3.3x) | 10.2 | 29.9 (2.8x) |

ResNet-RS trains faster then EffNet



# CONV NET SECRET SAUCE

# FLEXIBILITY

$x$

$w_1$

$w_i$

$y$

**One's hot label**

0 0 1 0

ball car hat

**grumpy cat**

$$f(x; w) \rightarrow y$$

image weights

# FLEXIBILITY

$x$

$w_1$

$w_i$

$y$

**One's hot label**

0 0 1 0

ball car hat

**grumpy cat**

$$f(x; w) \rightarrow y$$

Neural loss function

$$L(w) = \min_w \sum_i \| f(x_i; w) - y_i \|^2$$

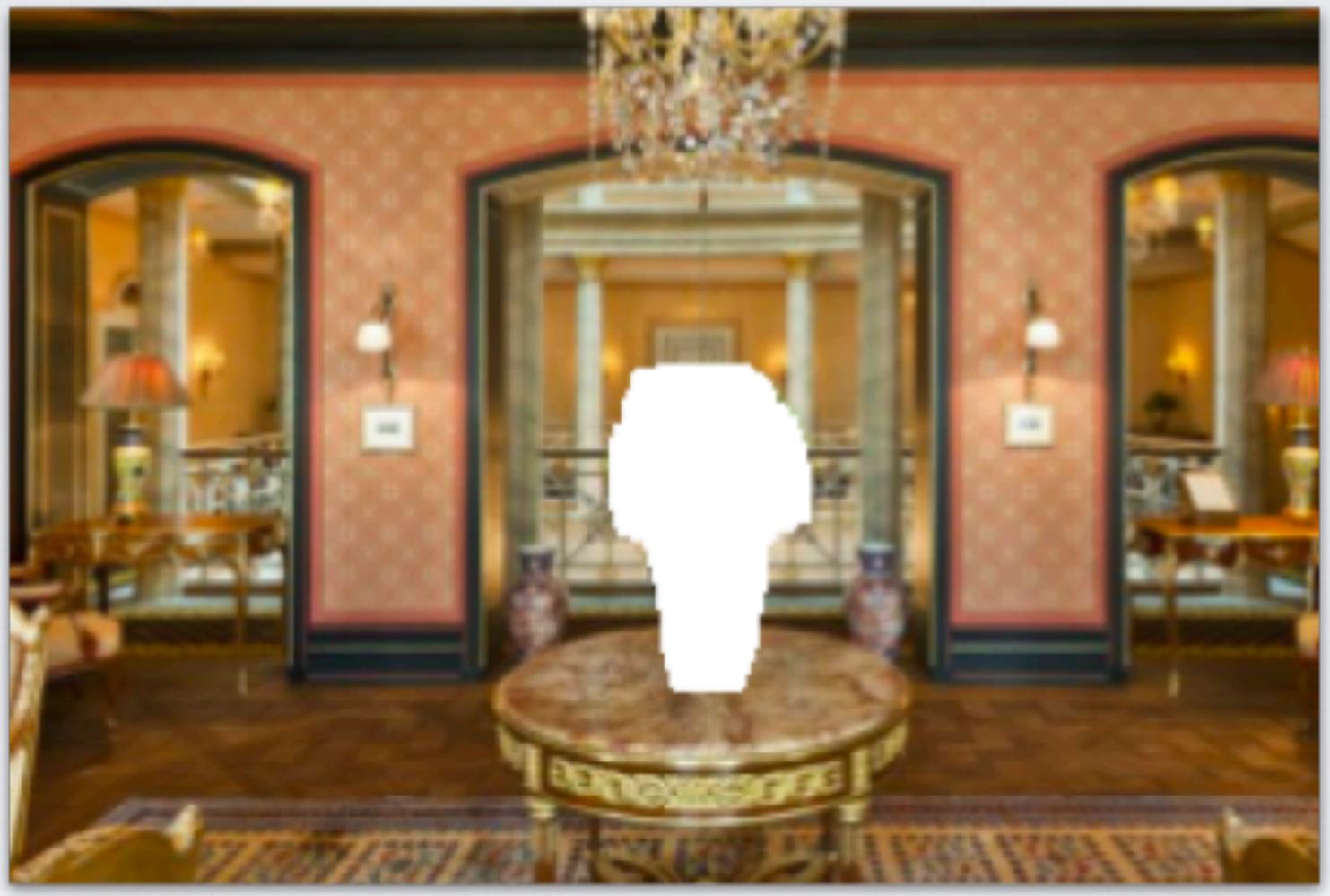
# IMPLICIT BIAS

**Neural net loss landscapes have bad minima**

**But we don’t find them**

**Neural nets “like” to learn natural image behavior**

# IMPLICIT BIAS



Corrupted

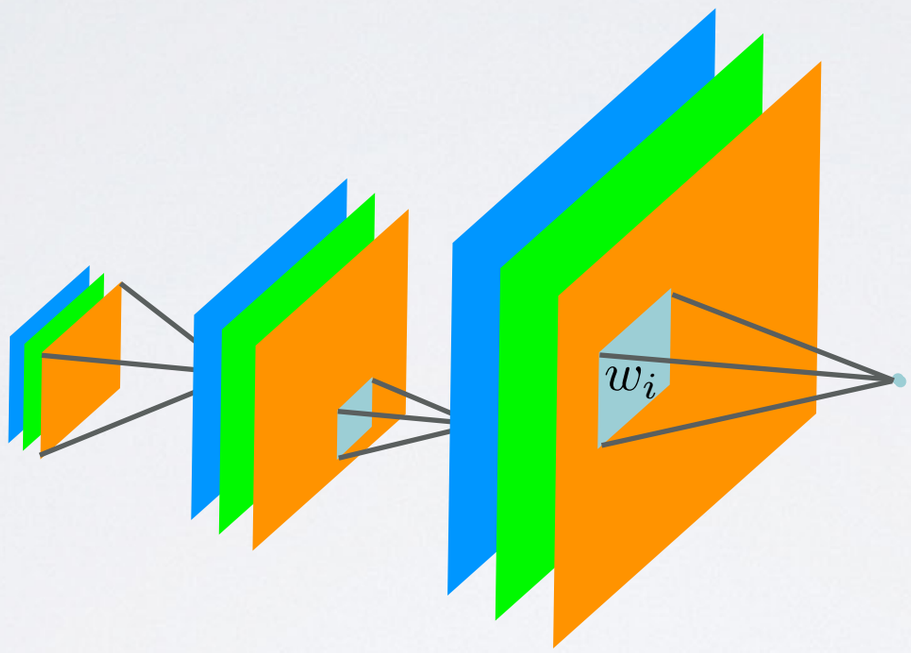


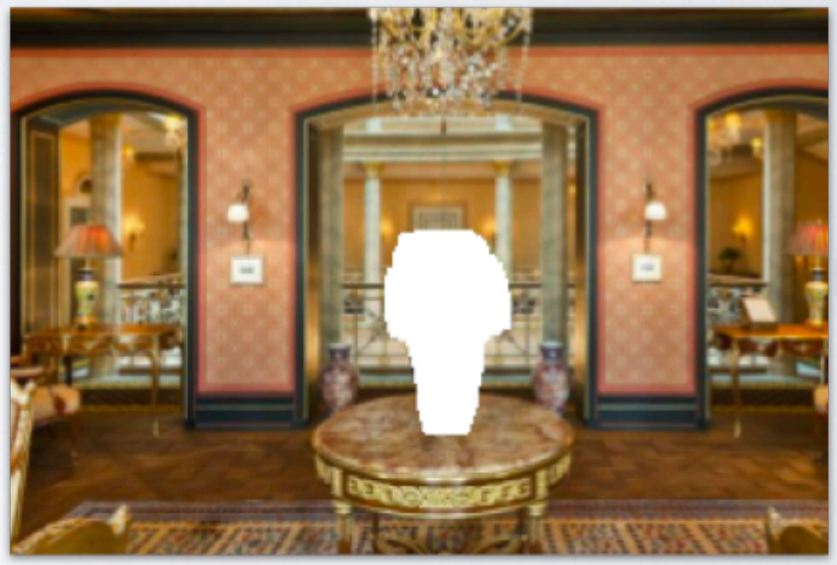
**Ulyanov, “Deep Image Prior”**

# IMPLICIT BIAS

**random vector (frozen)**





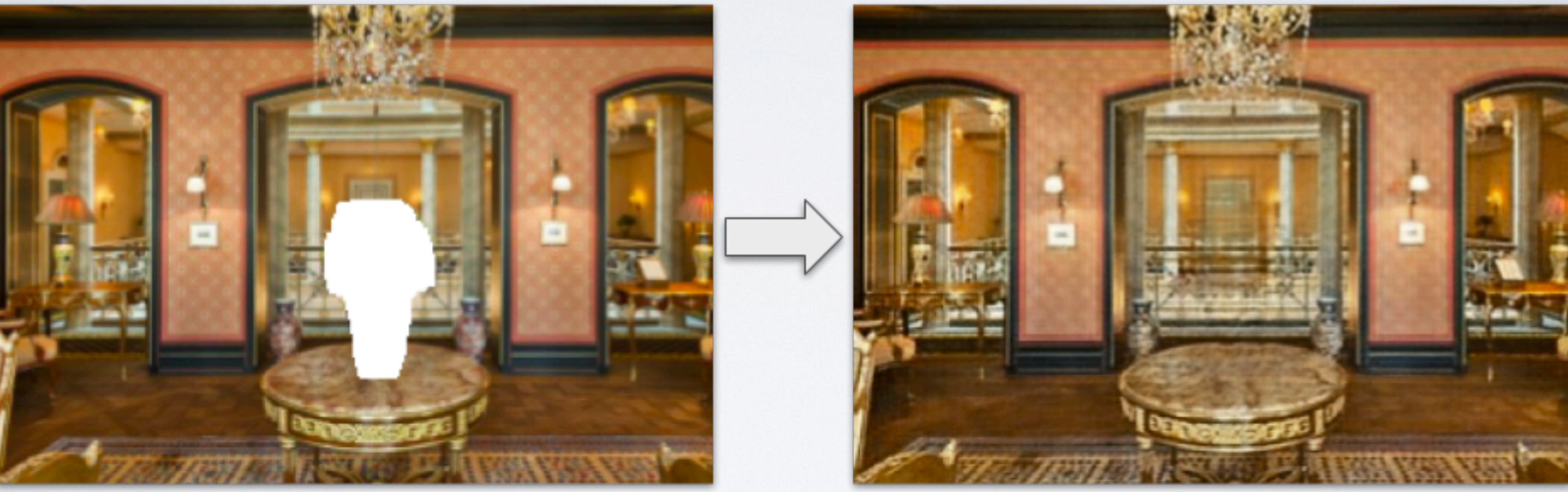
Corrupted

$$\min_{\text{network params}} \ell(\text{network output}, \text{known pixels})$$



**Ulyanov, “Deep Image Prior”**

# IMPLICIT BIAS

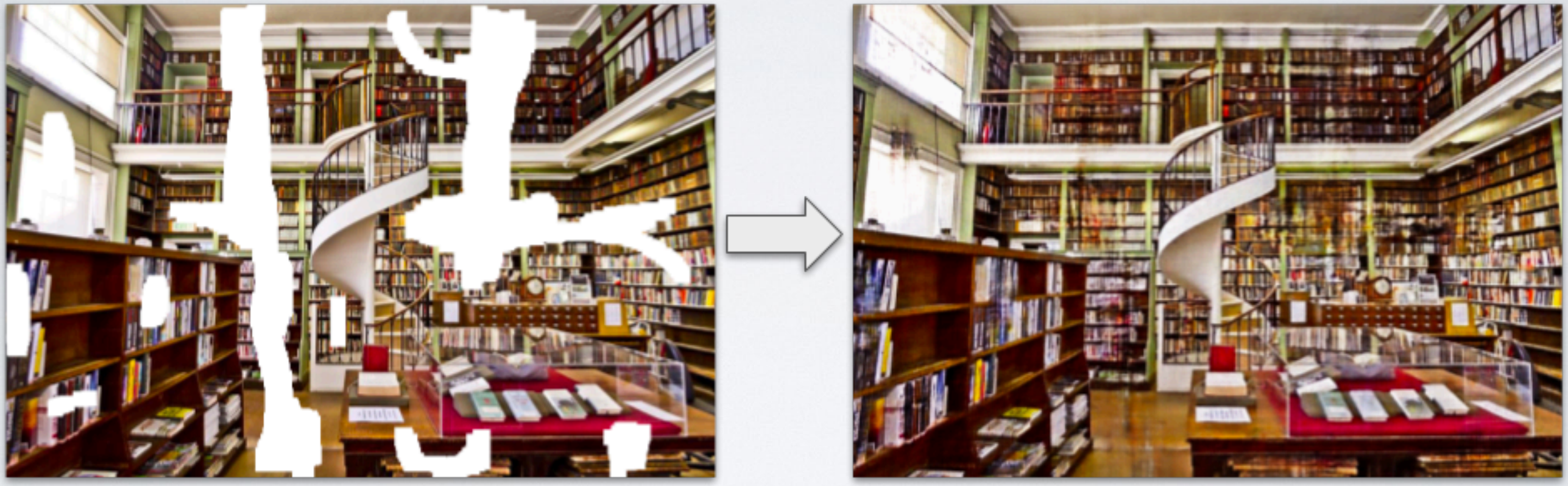





**Ulyanov, "Deep Image Prior"**

# IMPLICIT BIAS

**Inpainting**



**Corrupted** **Deep image prior**



**Ulyanov, “Deep Image Prior”**

# TRANSFORMERS?

Not in this lecture