# Deep Learning

Loo Tiang Kuan, Leonard

A0098795B

supervised by

Associate Professor Ji Hui

co-supervised by

Assistant Professor Alvina Goh Siew Wee

May 5, 2017

# Overview

**Session 1**
Activations
Cost
Stochastic Gradient Descent
Backpropagation
Overfitting
Regularization

**Session 2**
Initialisation
Hyper-parameters
Variants of SGD
More Regularisers

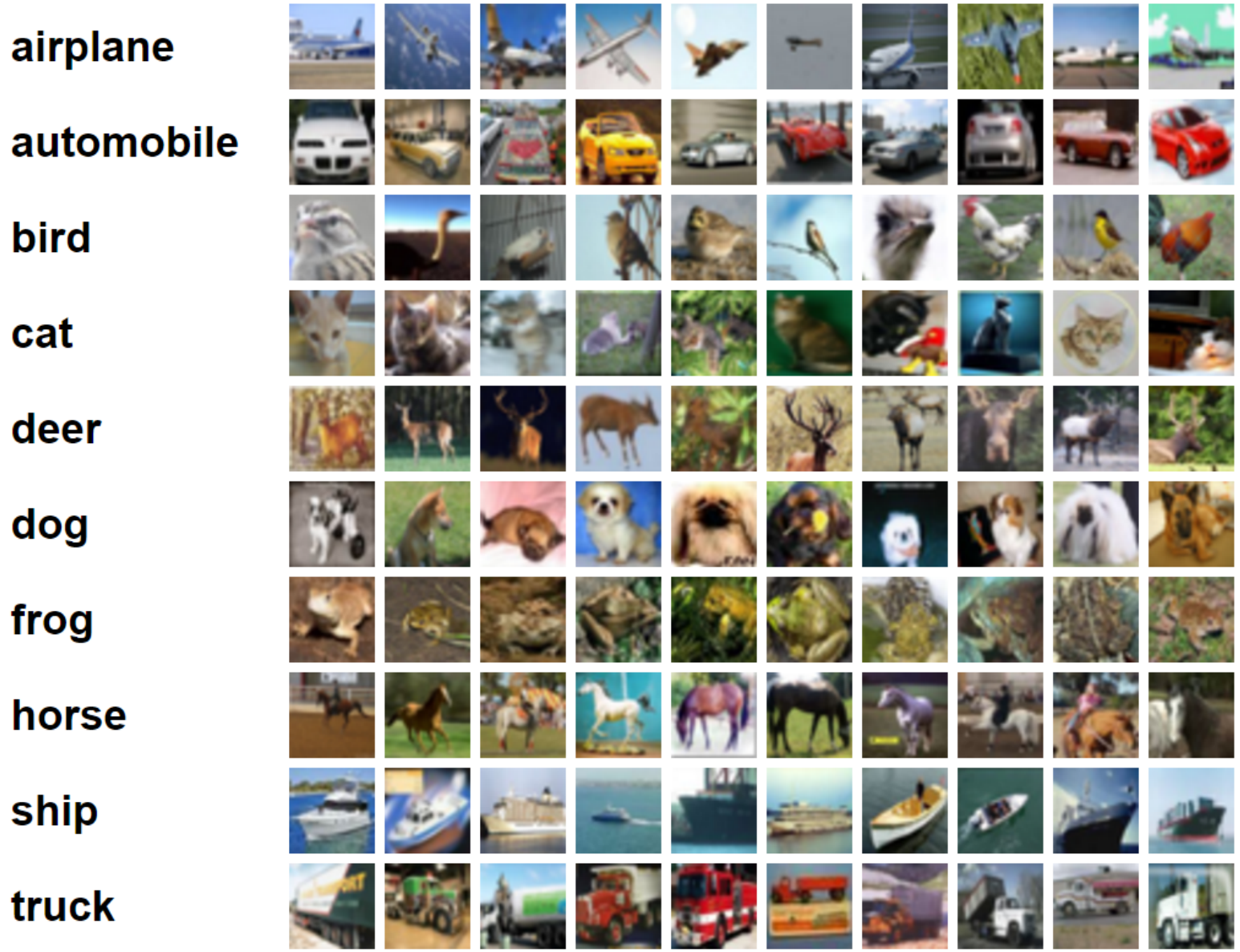
**Session 3**
Intro to ConvNets
Popular ConvNet Architectures
Denoising

**Session 4/5**
Autoencoders
Transposed Convolution
Generative Network
Generative Adversarial Network

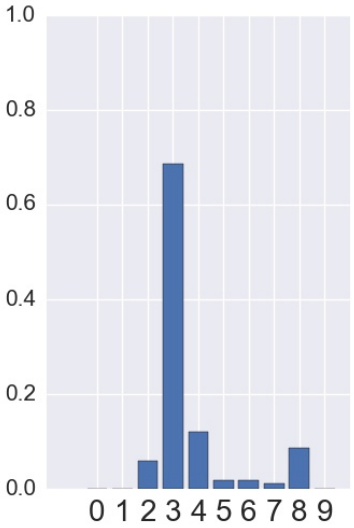
# Today's Agenda

1. CIFAR-10
2. Intro to Convolutional Neural Networks (ConvNets)
3. Popular Architectures

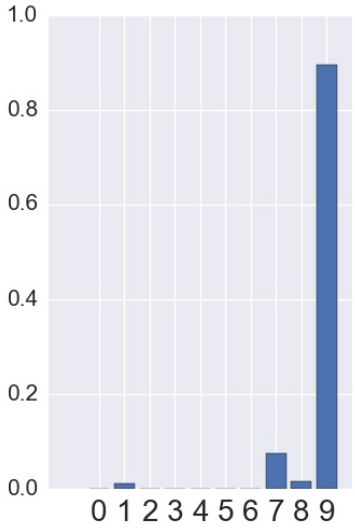
# CIFAR-10


airplane
automobile
bird
cat
deer
dog
frog
horse
ship
truck

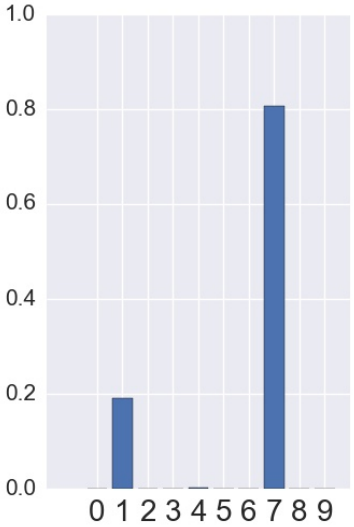
# CIFAR-10



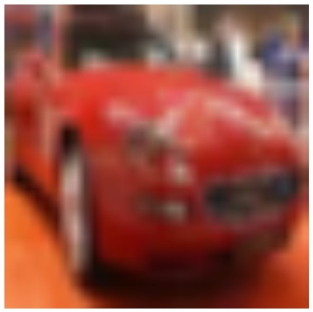

True Label: 6

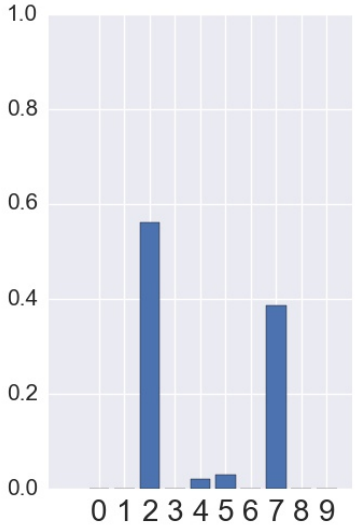



True Label: 9

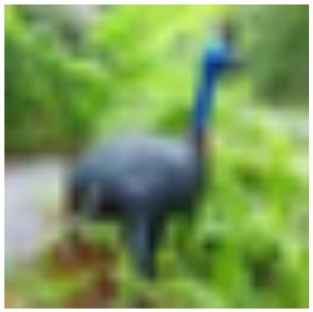



True Label: 1






True Label: 2

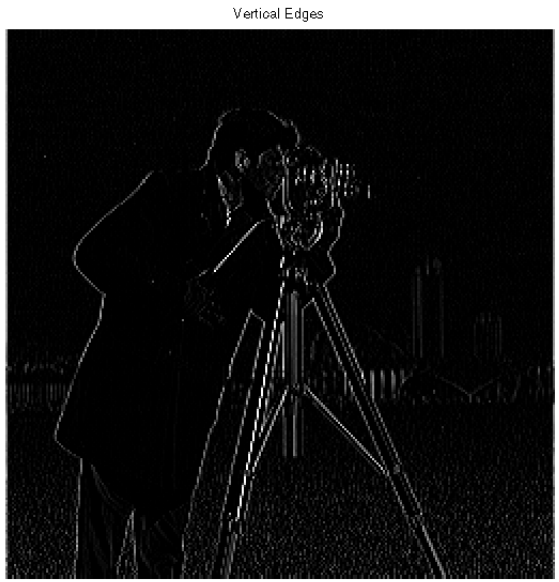


# Outline

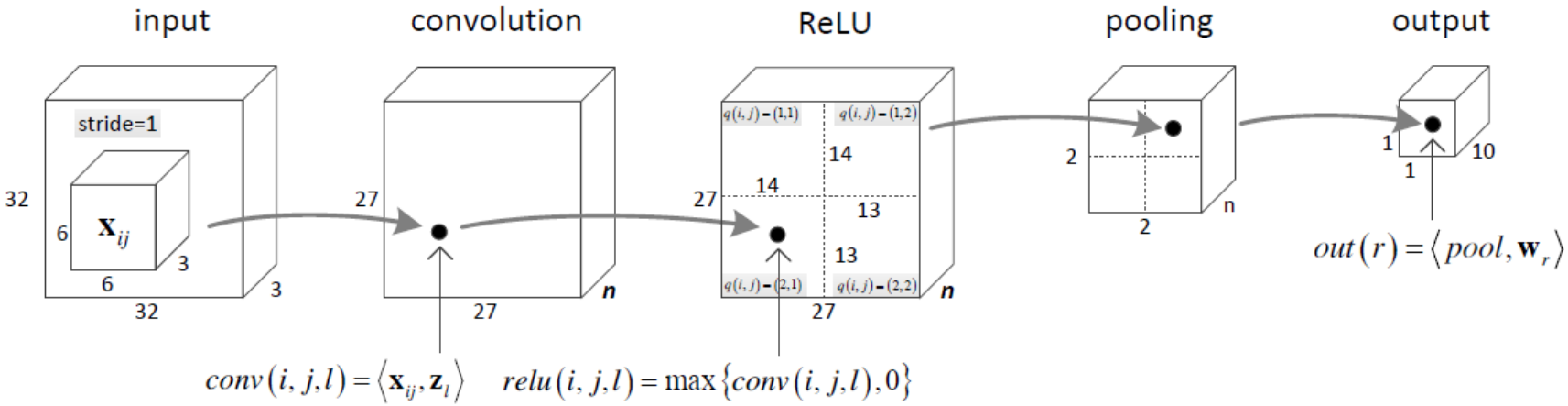
# Why ConvNets?

$$A * [-1, 1]$$

- Filters

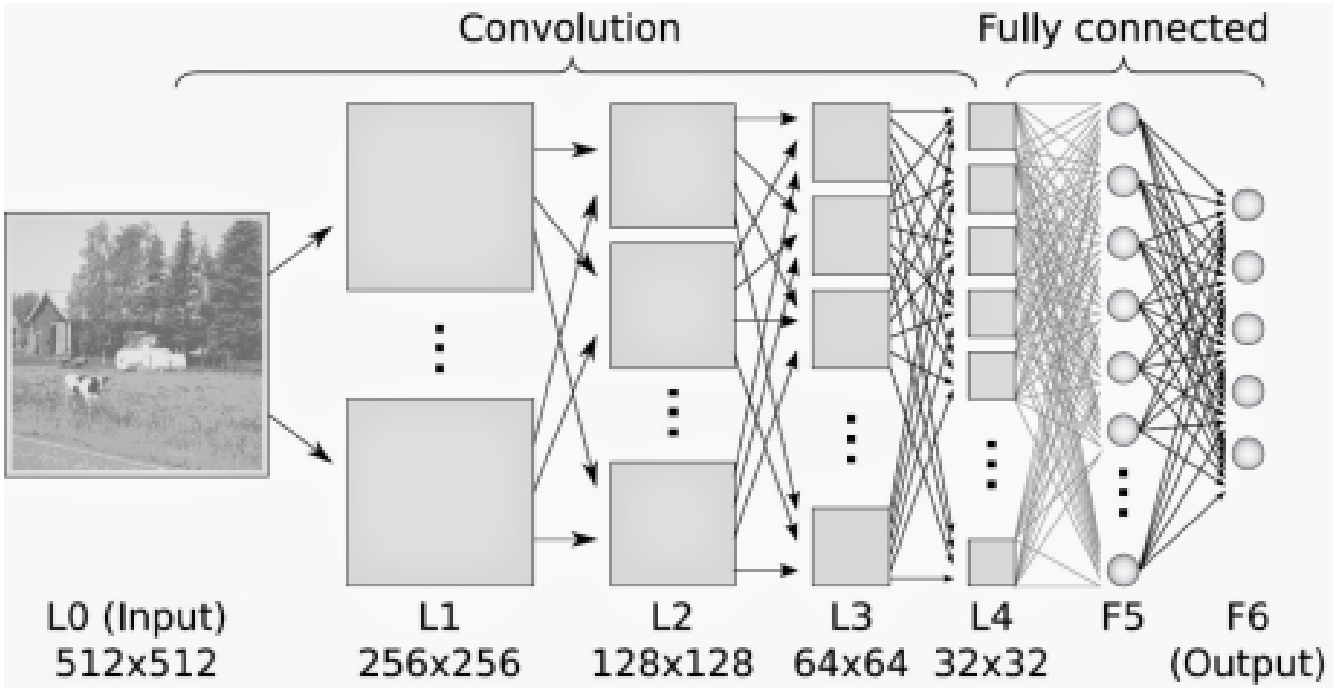
# Intro to Convolutional Neural Networks (ConvNets)




- Fully-connected layers / Dense layers
  - Assumes a given pixel has strong relationship with all other pixels.
  - Vectorized layer.
- Convolutional layers (CNN)
  - Assumes pixels only have strong relationship with its neighbourhood.
  - Matrix layer (3D).

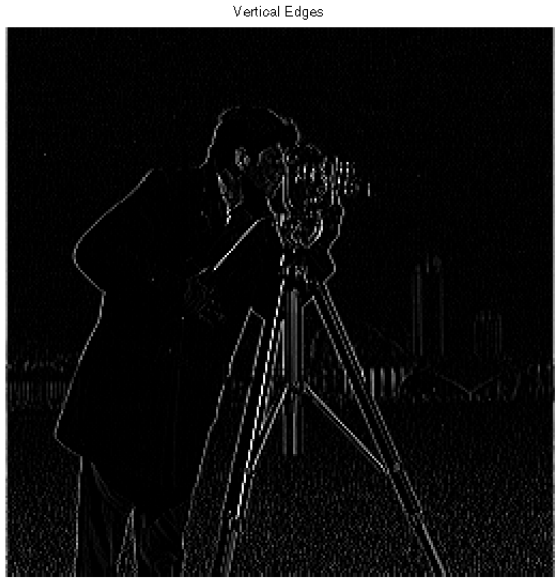
# Outline

# Architecture



# Filters

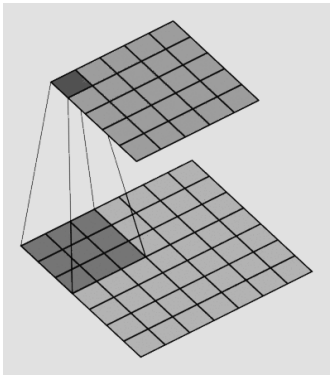
- Filters

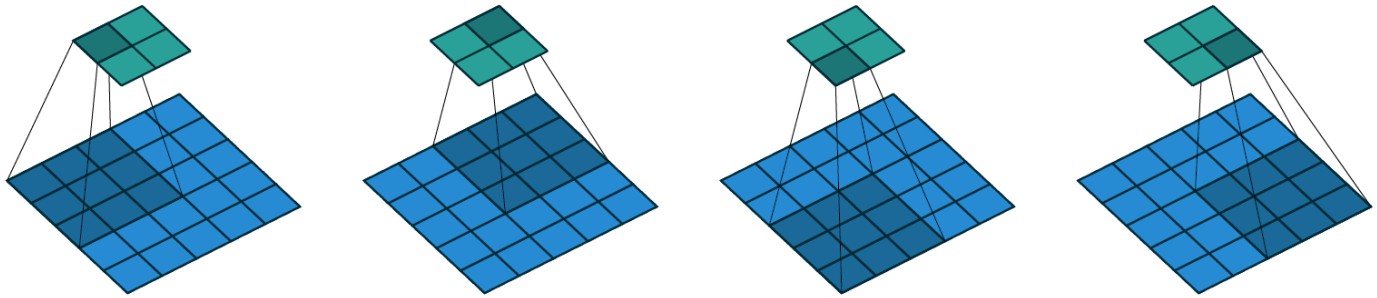

- Shared weights & biases

# Convolutional Layer

Convolution



Stride: $s$



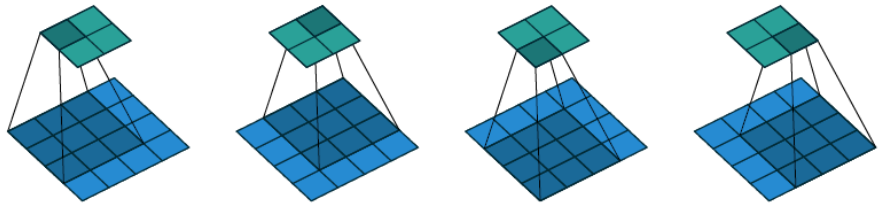
Output size: $\left\lfloor \frac{in_{size}+2\times pad_{size}-fil_{size}}{s} \right\rfloor + 1$

# Outline

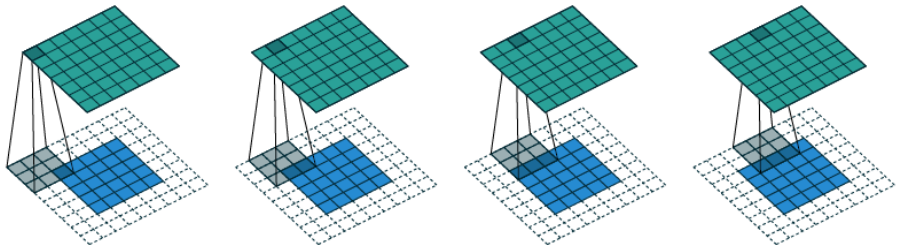
# Padding

For a $m$ x $m$ filter $f$, consider $A * f$

- Valid
  - No padding.



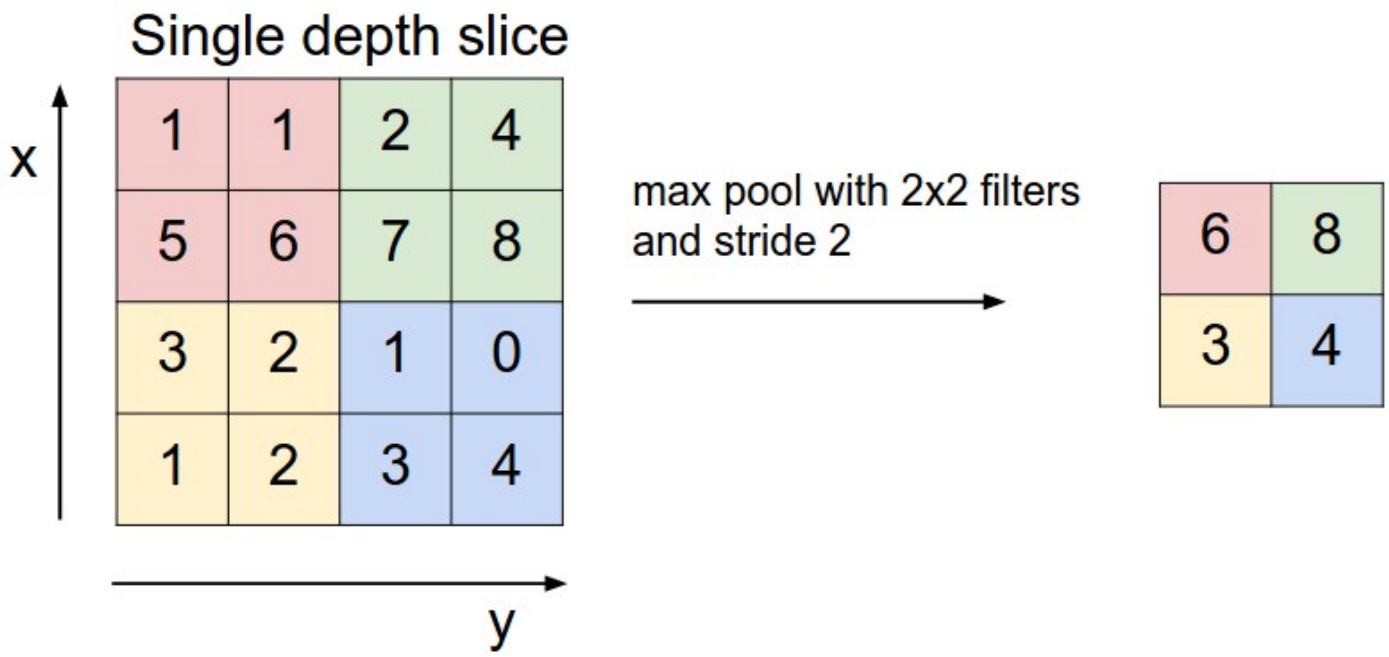
- Full
  - Pads $A$ by $m-1$ x $m-1$.



- Same
  - Pads $A$ by $\frac{m-1}{2}$ x $\frac{m-1}{2}$ to get same output size.

# Outline

# Pooling




- Functions
  - Max Pooling (Most common): Extracts the maximum activation.
  - Average Pooling
- Downsample layer size.
- Don't use if wish to retain the size.

# Visualisation

$$\text{Output size: } \left\lfloor \frac{in_{size} - pool_{size}}{s} \right\rfloor + 1$$

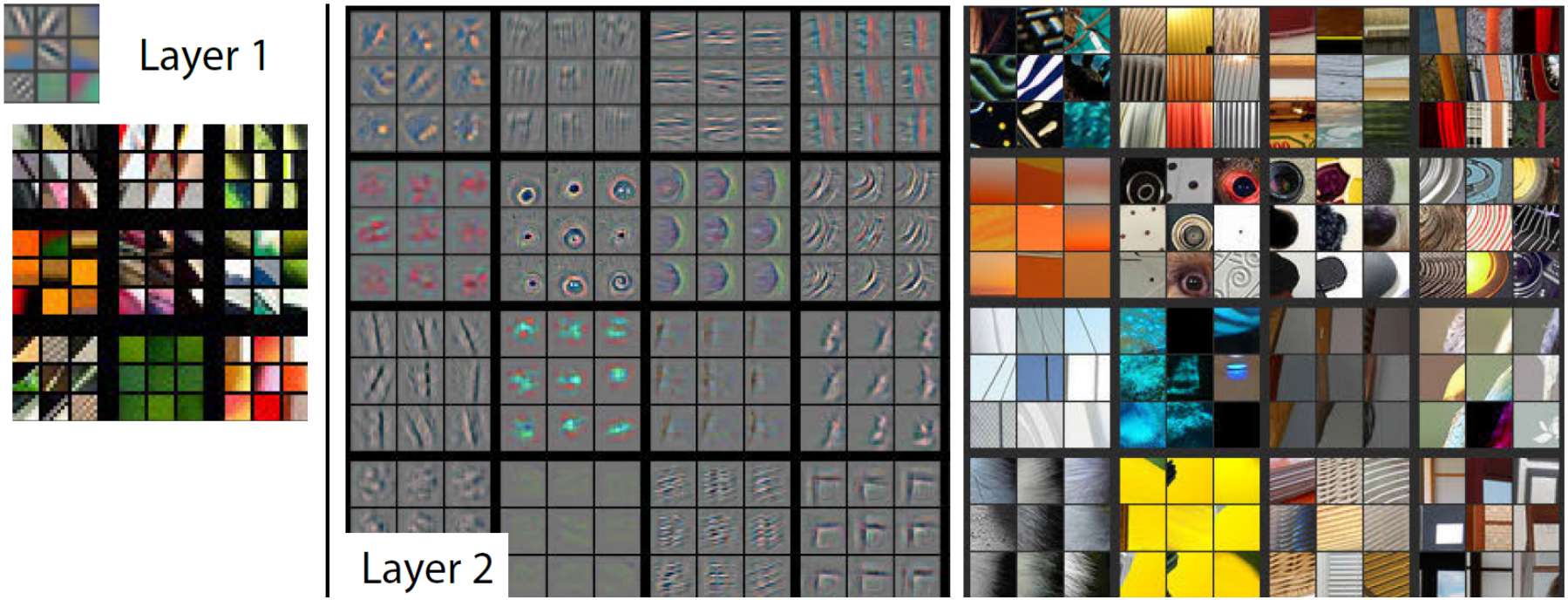
Visualisation

# Outline

# Outline

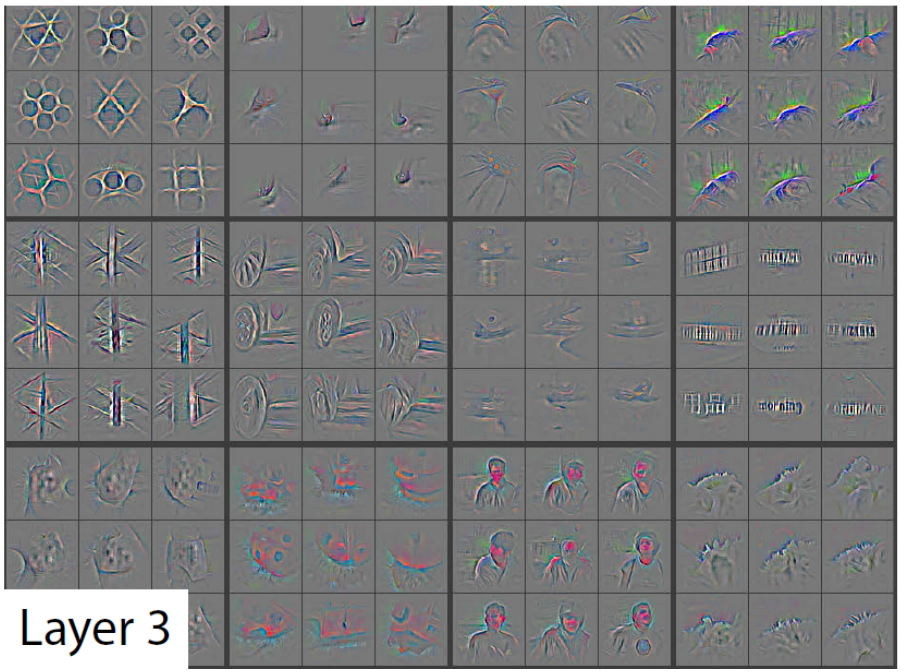
# Analysis Of Feature Maps: Filters

[ZFNet] Page 4 fig 2

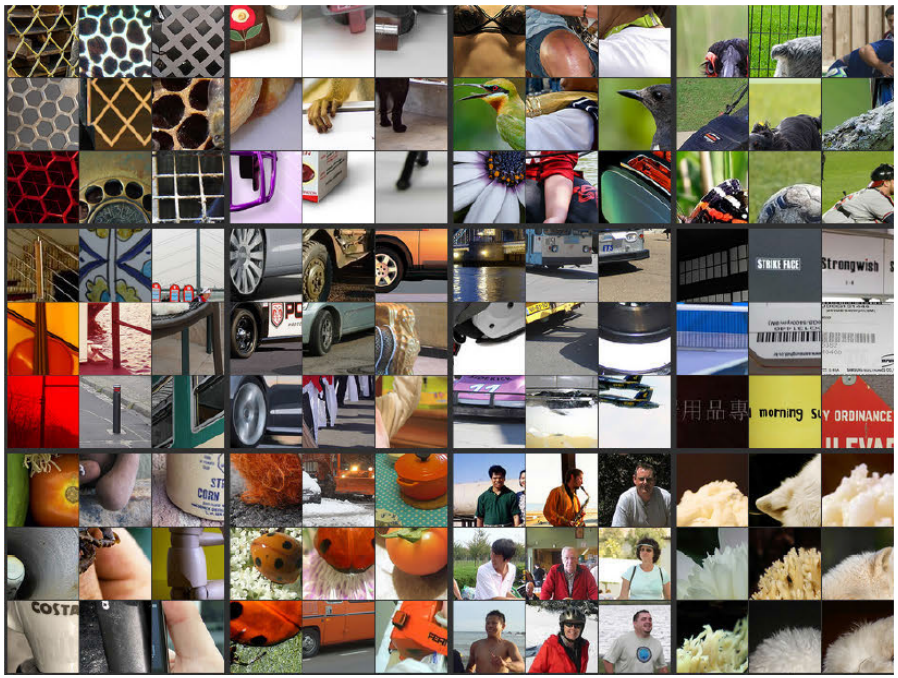



Responds to corners and other edge/color conjunctions.

# Analysis Of Feature Maps: Filters



Layer 3

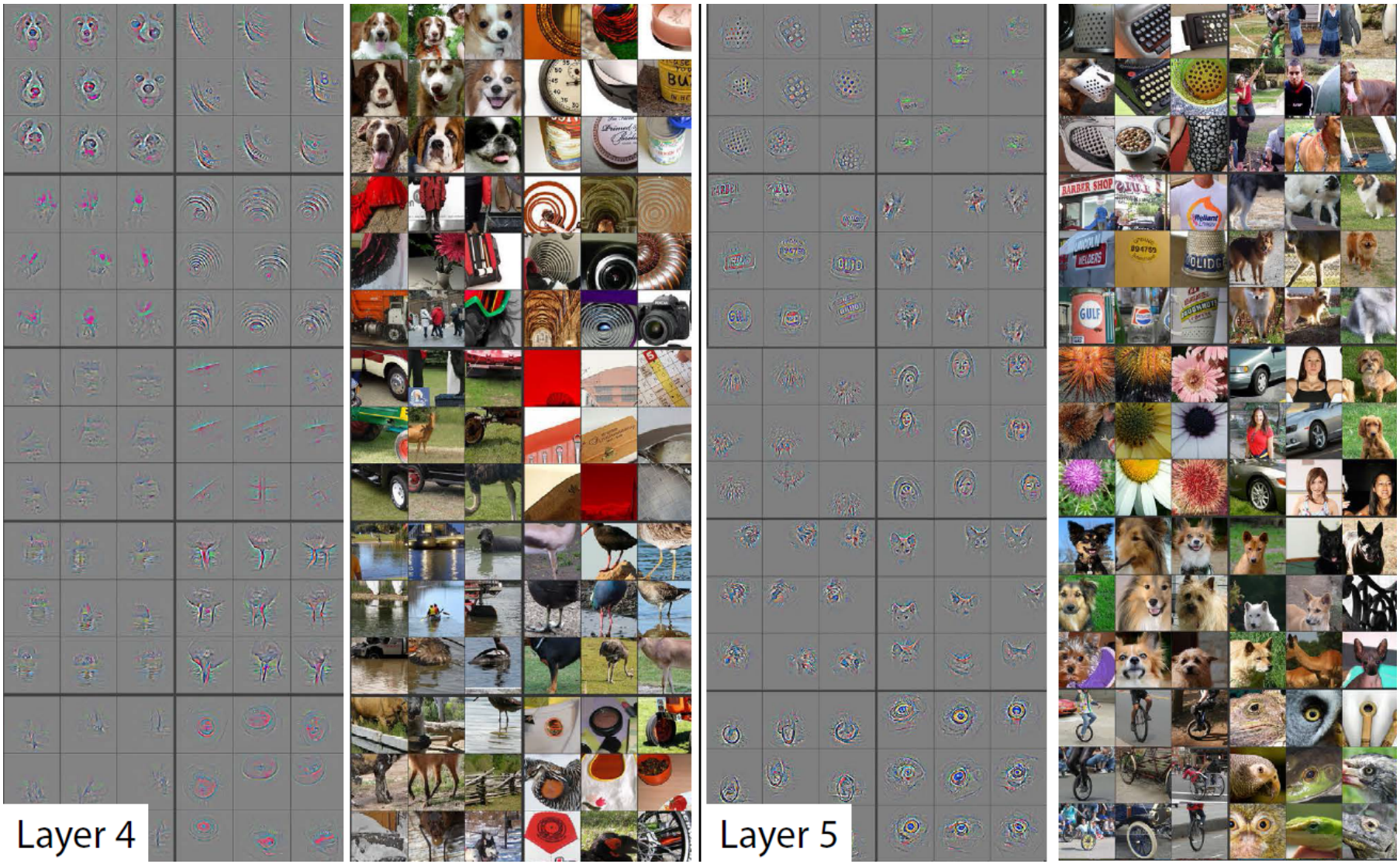


More complex invariances

Mesh patterns

Text

# Analysis Of Feature Maps: Filters

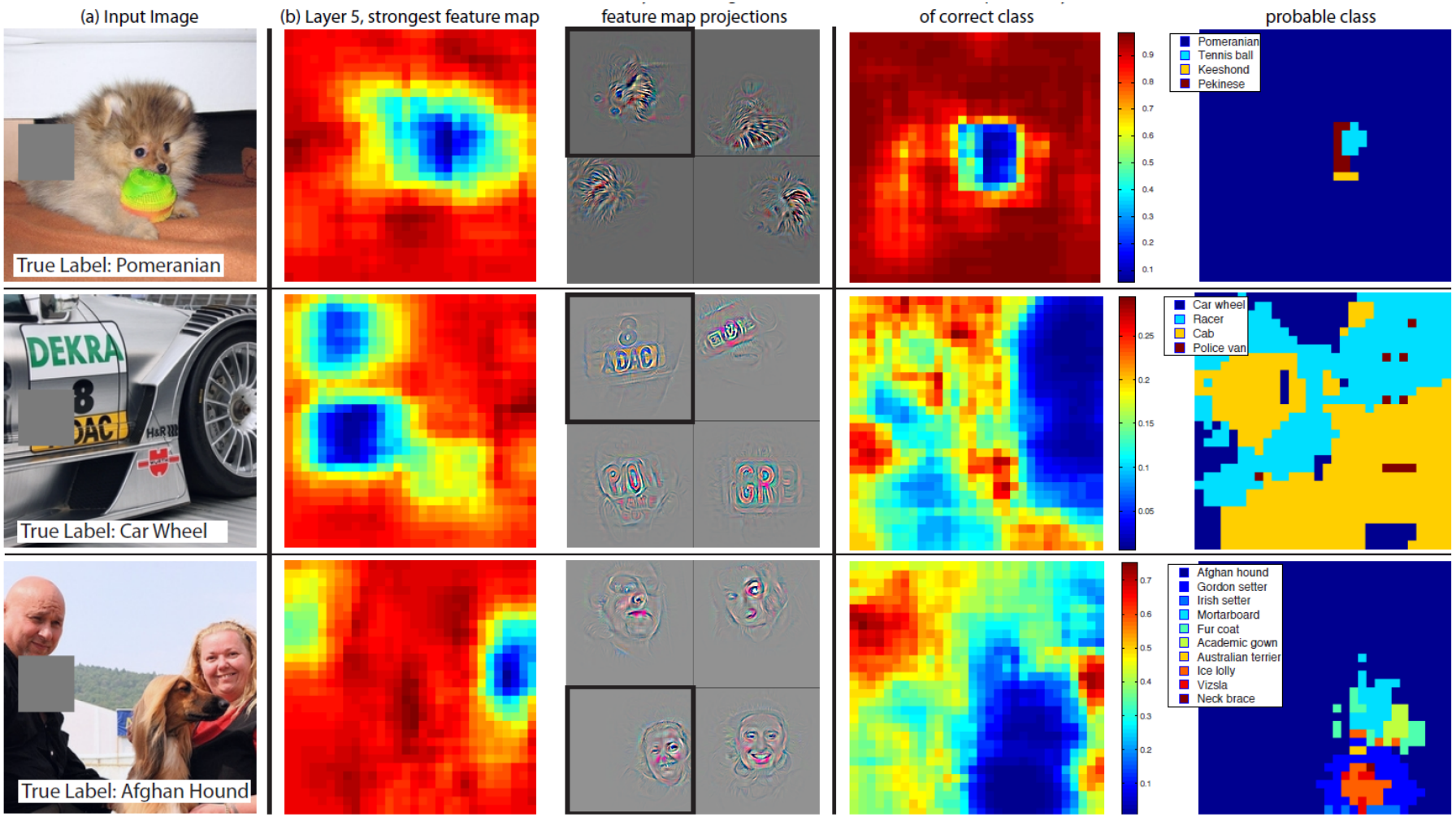



Greater Variance **Layer 4**:Class Specific; **Layer 5**:Entire objects

# Outline

# Analysis Of Feature Maps: Occlusion

## Classification in ImageNet



# Outline

# Outline

# Backpropagation in Convolution

Matrix Representation:
Consider $B = A * f$, where $A \in \mathbb{R}^{4x4}$, $f \in \mathbb{R}^{3x3}$, $B \in \mathbb{R}^{2x2}$,

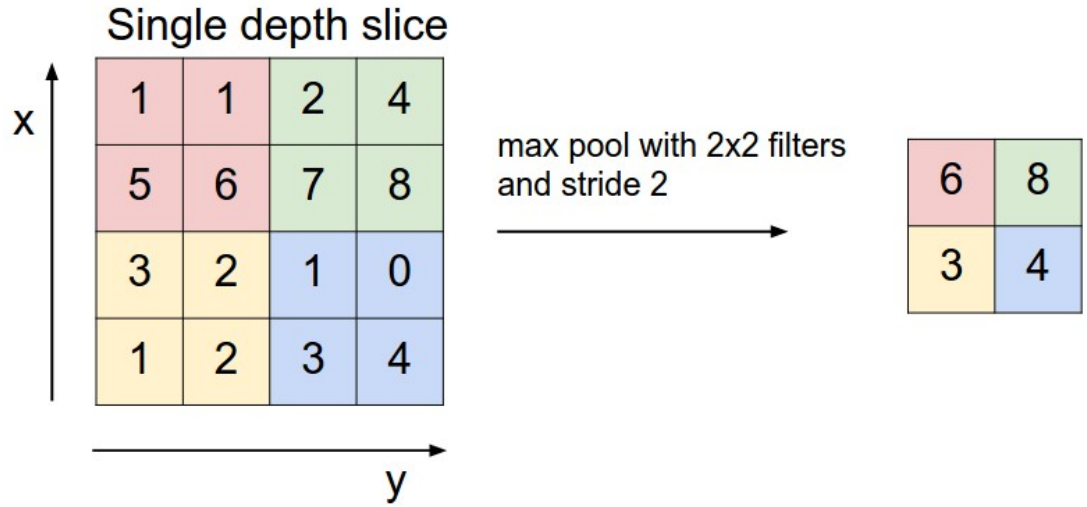
$$C = \begin{pmatrix} a & b & c & & d & e & f & & g & h & i & & & & \\ & a & b & c & & d & e & f & & g & h & i & & & \\ & & & & a & b & c & & d & e & f & & g & h & i \\ & & & & & a & b & c & & d & e & f & & g & h & i \end{pmatrix}.$$

We have, for their respective vectorised representation $\tilde{B}, \tilde{A}$,

$$\begin{aligned} & B = A * f \\ \iff & \tilde{B} = C\tilde{A} \\ \iff & a^l = g(W^l a^{l-1}) \\ \text{Recall } \delta^l = & \left( [W^{l+1}]^T \delta^{l+1} \right) \odot g'(z^l). \end{aligned}$$

# Outline

# Backpropagation in Pooling

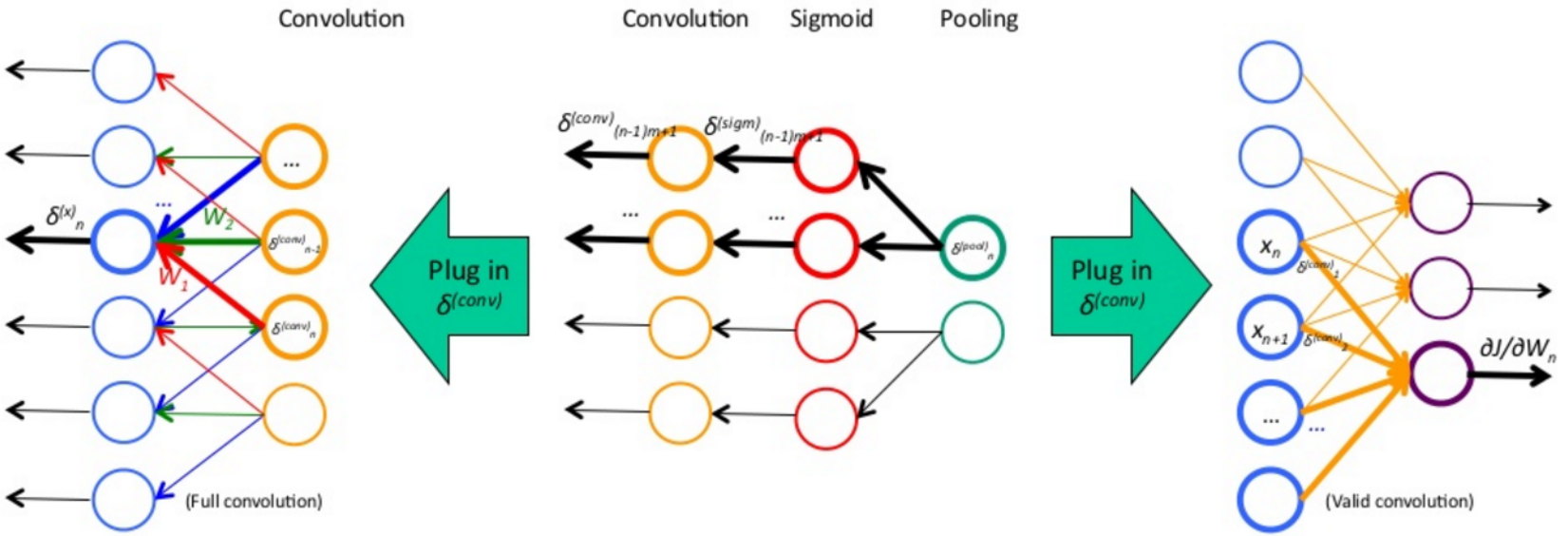



- Upsampling
- Backpass the error, spreading their error proportionately to their contributions.
- Derivatives in Pooling p(x)

| | | |
|---|---|---|
| Max pooling | $p(x) = \max_i x_i,$ | $\frac{\partial p}{\partial x} = [e_j]$, j is at the max position |
| Average pooling | $p(x) = \frac{\sum_{i=0}^{m} x_i}{m},$ | $\frac{\partial p}{\partial x} = \frac{1}{m}\mathbf{1}$ |
| Norm pooling | $p(x) = \|x\|_n,$ | $\frac{\partial p}{\partial x_k} = \frac{\|x_i\|^{n-1}}{\|x\|_n}$ |

# Backpropagation Summary

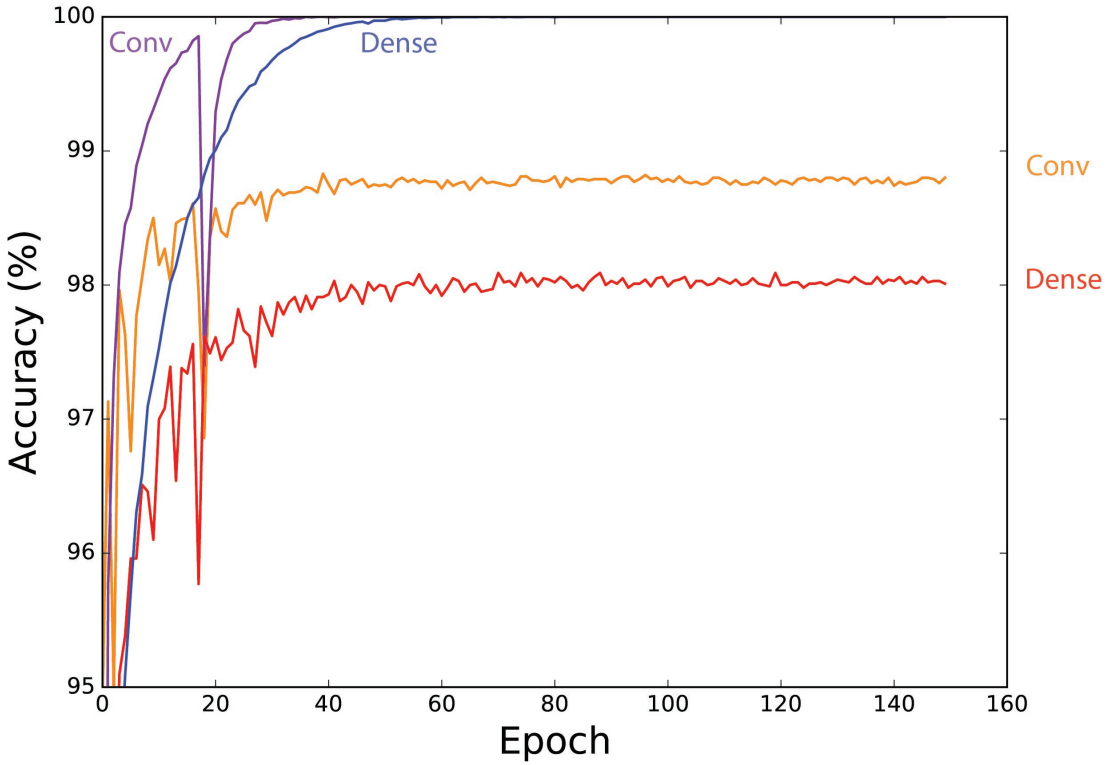
2. Propagate error signals $\delta^{(conv)}$ 1. Propagate error signals $\delta^{(pool)}$ 3. Compute gradient $\nabla_w J$




$$\delta^{(x)} = \delta^{(conv)} * \text{flip}(w) \qquad \delta^{(conv)} = \text{upsample}\left(\delta^{(pool)}, g'\right) \underbrace{\bullet f^{(sigm)} \bullet \left(1 - f^{(sigm)}\right)}_{\text{Derivative of sigmoid}} \qquad x * \delta^{(conv)} = \nabla_w J$$

# Outline

# Comparison

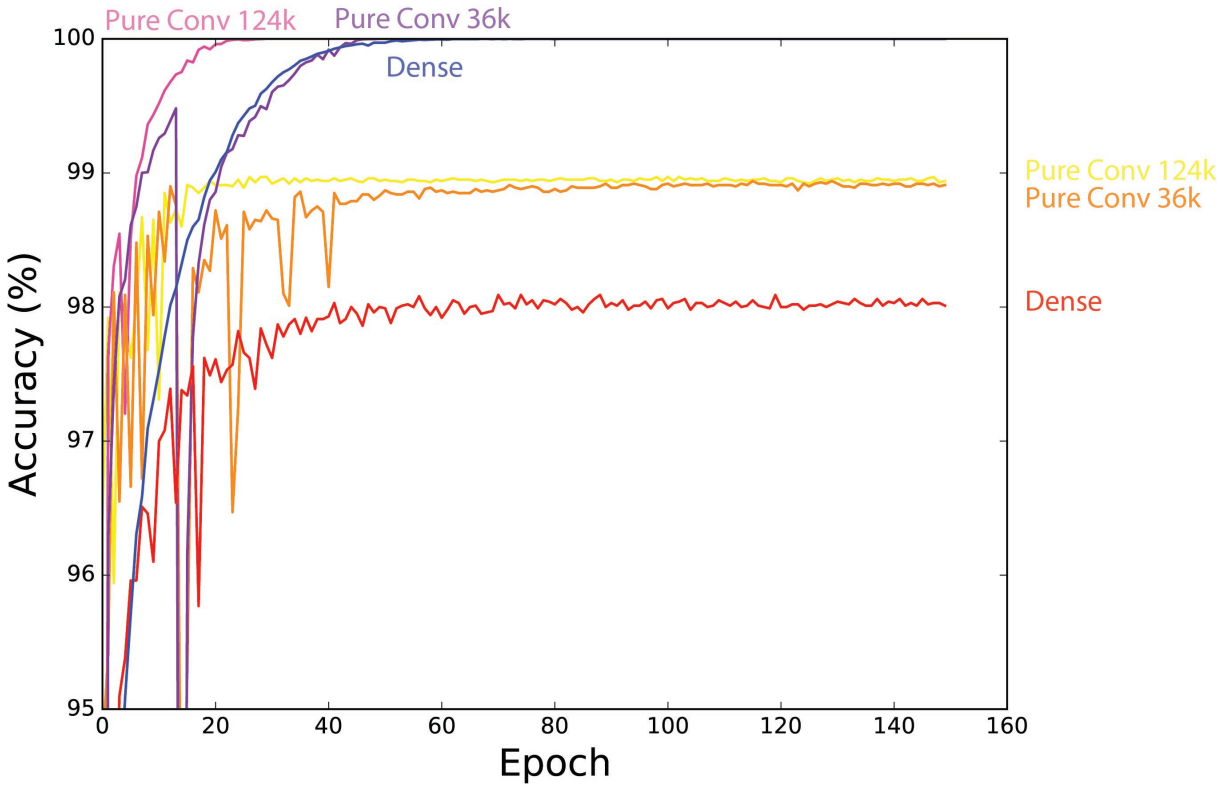



**CNN** ('same' padding)
Conv Layer 5 filters (5x5): 130 params
Pooling Layer (2x2)
Dense Layer(300): 294,300 params
Output Layer(10): 3010 params
**Total**: 300,440 params

**Traditional**
Dense Layer(280): 219,800 params
Dense Layer(280): 78,680 params
Output Layer(10): 2,810 params

**Total**: 301,290 params

# Comparison

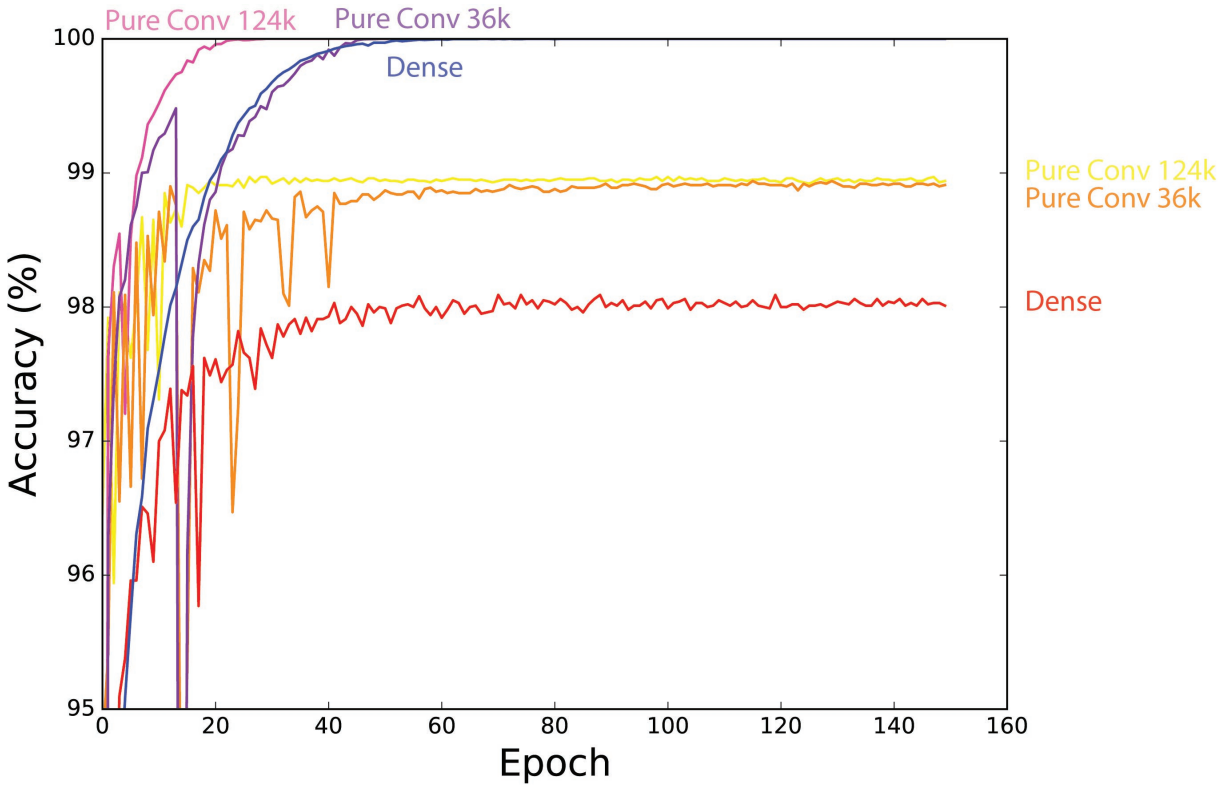



**CNN**

Conv Layer 32 filters (5x5): 832 params
Conv Layer 64 filters (5x5): 25,632 params
Conv Layer 32 filters (3x3): 9,248 params
Output Layer(10): 330 params
**Total**: 36,042 params

Pooling Layer (2x2)
Pooling Layer (2x2)

# Comparison




**CNN**

Conv Layer 64 filters (5x5): 1,664 params
Pooling Layer (2x2)
Conv Layer 64 filters (5x5): 102,464 params
Pooling Layer (2x2)
Conv Layer 32 filters (3x3): 18,464 params
Output Layer(10): 1,290 params
**Total**: 123,882 params

# Outline

1. CIFAR-10
2. Intro to Convolutional Neural Networks (ConvNets)
3. Popular Architectures
   - ImageNet
   - LeNet-5 [1998]
   - AlexNet [2012]
   - ZFNet [2013]
   - VGGNet [2014]
   - GoogLeNet [2015]
   - ResNet [2016]

# ImageNet Challenge

IMAGENET

- 1,000 object classes (categories).
- Images:
  - 1.2 M train
  - 100k test.


Cifar:(3x32x32)
ImageNet:(3x227x227)

# Outline

1. CIFAR-10
2. Intro to Convolutional Neural Networks (ConvNets)
3. Popular Architectures
   - ImageNet
   - LeNet-5 [1998]
   - AlexNet [2012]
   - ZFNet [2013]
   - VGGNet [2014]
   - GoogLeNet [2015]
   - ResNet [2016]

# LeNet-5

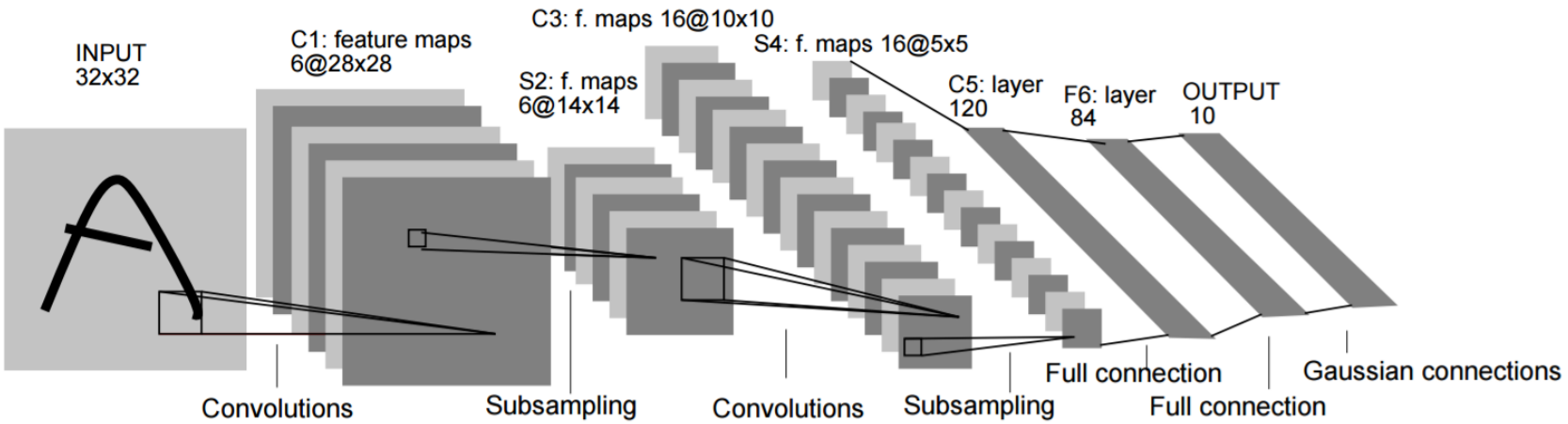


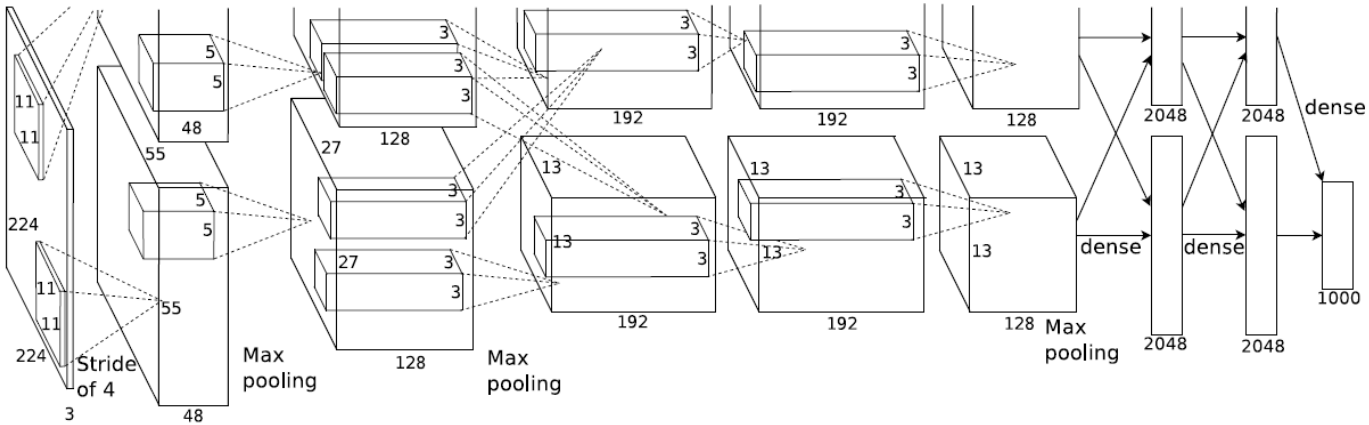

Fig. 2. Architecture of LeNet-5, a Convolutional Neural Network, here for digits recognition. Each plane is a feature map, i.e. a set of units whose weights are constrained to be identical.


LeCun, Y., Bottou, L., Bengio, Y., & Haffner, P. (1998). *"Gradient-based learning applied to document recognition. "* Proceedings of the IEEE, 86(11), 2278-2324.

# Outline

1. CIFAR-10
2. Intro to Convolutional Neural Networks (ConvNets)
3. Popular Architectures
   - ImageNet
   - LeNet-5 [1998]
   - AlexNet [2012]
   - ZFNet [2013]
   - VGGNet [2014]
   - GoogLeNet [2015]
   - ResNet [2016]

# AlexNet




[224x224x3] Input
[55x55x96] Conv1 (11x11), stride 4, pad (2x2)
[27x27x96] Pool1 (3x3), stride 2
[27x27x256] Conv2 (5x5), stride 1, pad 'same'
[13x13x256] Pool2 (3x3), stride 2
[13x13x384] Conv3 (3x3), stride 1, pad 'same'
[13x13x384] Conv4 (3x3), stride 1, pad 'same'
[13x13x384] Conv5 (3x3), stride 1, pad 'same'
[6x6x256] Pool3 (3x3), stride 2
[4096] Dense6
[4096] Dense7
[1000] Dense (Output) (Accuracy 84.6%)

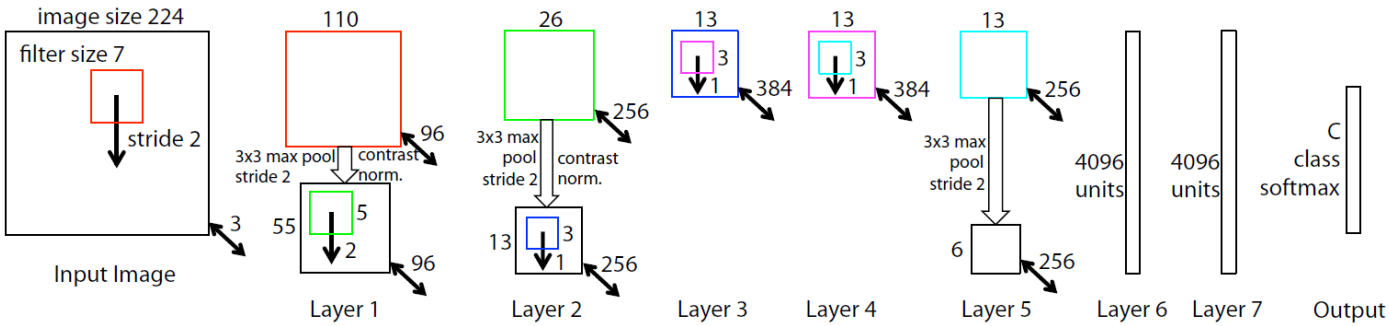
# References

Krizhevsky, A., Sutskever, I., & Hinton, G. E. (2012). *"Imagenet classification with deep convolutional neural networks."* In Advances in neural information processing systems (pp. 1097-1105).

# Outline

1. CIFAR-10
2. Intro to Convolutional Neural Networks (ConvNets)
3. Popular Architectures
   - ImageNet
   - LeNet-5 [1998]
   - AlexNet [2012]
   - ZFNet [2013]
   - VGGNet [2014]
   - GoogLeNet [2015]
   - ResNet [2016]

# ZFNet




[224x224x3] Input
[110x110x96] Conv1 (7x7), stride 2
[55x55x96] Pool1 (3x3), stride 2
[26x26x256] Conv2 (5x5), stride 2
[13x13x256] Pool2 (3x3), stride 2
[13x13x512] Conv3 (3x3), stride 1
[13x13x1024] Conv4 (3x3), stride 1
[13x13x512] Conv5 (3x3), stride 1
[6x6x256] Pool3 (3x3), stride 2
[4096] Dense6
[4096] Dense7
[1000] Dense (Output) (Accuracy 85.2%)

# Comparison for ImageNet

| **Layers** | **AlexNet** | **ZFNet** |
|---|---|---|
| InputLayer | (3, 224, 224) | (3, 224, 224) |
| Conv1 | (11x11) 34,944 | (7x7) 14,208 |
| | (96, 55, 55) | (96, 110, 110) |
| MaxPool1 | (96, 27, 27) | (96, 55, 55) |
| Conv2 (5x5) | 614,656 | 614,656 |
| | (256, 27, 27) | (256, 26, 26) |
| MaxPool2 | (256, 13, 13) | (256, 13, 13) |
| Conv3 (3x3) | 885,120 (384, 13, 13) | 1,180,160 (512, 13, 13) |
| Conv4 (3x3) | 1,327,488 (384, 13, 13) | 4,719,616 (1024, 13, 13) |
| Conv5 (3x3) | 884,992 (256, 13, 13) | 4,719,104 (512, 13, 13) |
| MaxPool3 | (256, 6, 6) | (512, 6, 6) |
| Dense6 | 37,752,832 (4096) | 75,501,568 (4096) |
| Dense7 | 16,781,312 (4096) | 16,781,312 (4096) |
| OutputLayer | 4,097,000 (1000) | 4,097,000 (1000) |
| **Total** | **62,378,344** | **107,627,624** |

# ZFNet

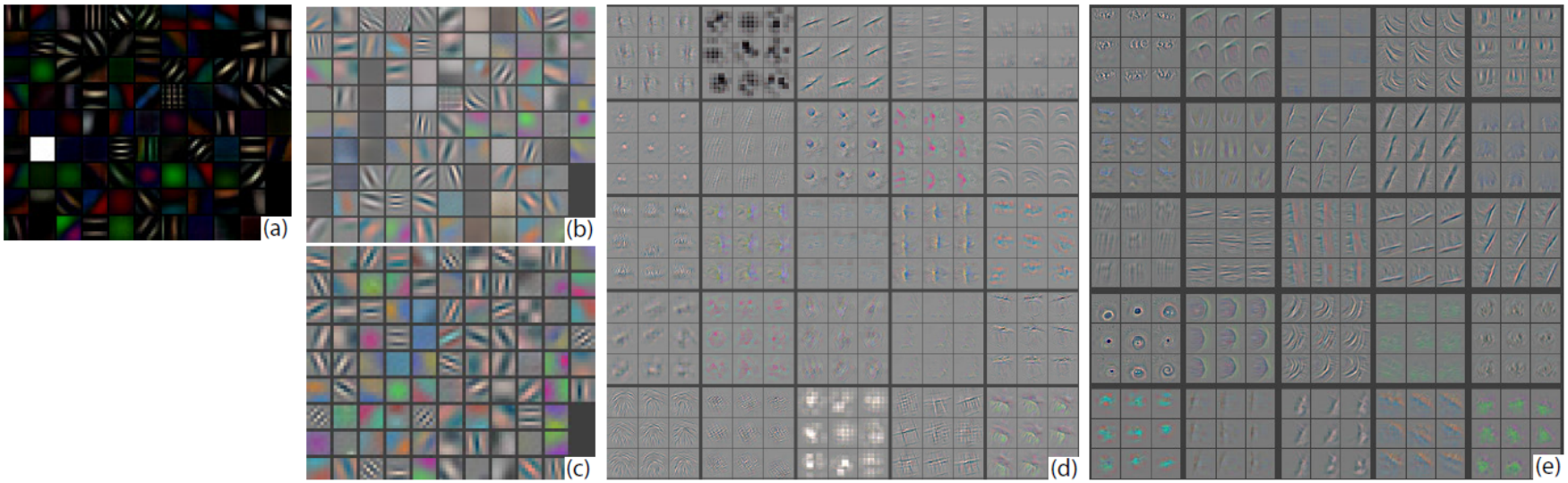



*Figure 6.* (a): 1st layer features without feature scale clipping. Note that one feature dominates. (b): 1st layer features from (Krizhevsky et al., 2012). (c): Our 1st layer features. The smaller stride (2 vs 4) and filter size (7x7 vs 11x11) results in more distinctive features and fewer "dead" features. (d): Visualizations of 2nd layer features from (Krizhevsky et al., 2012). (e): Visualizations of our 2nd layer features. These are cleaner, with no aliasing artifacts that are visible in (d).

Zeiler, M. D., & Fergus, R. (2014, September). *"Visualizing and understanding convolutional networks."* In European Conference on Computer Vision (pp. 818-833). Springer International Publishing.

# Outline

1. CIFAR-10
2. Intro to Convolutional Neural Networks (ConvNets)
3. Popular Architectures
   - ImageNet
   - LeNet-5 [1998]
   - AlexNet [2012]
   - ZFNet [2013]
   - VGGNet [2014]
   - GoogLeNet [2015]
   - ResNet [2016]

# VGGNet

Idea: ONLY use 3x3 Conv filters, and 2x2 Pooling.

Framework D

| ConvNet Configuration | | | | | |
|---|---|---|---|---|---|
| A | A-LRN | B | C | D | E |
| 11 weight layers | 11 weight layers | 13 weight layers | 16 weight layers | 16 weight layers | 19 weight layers |
| input (224 × 224 RGB image) | | | | | |
| conv3-64 | conv3-64<br>**LRN** | conv3-64<br>**conv3-64** | conv3-64<br>conv3-64 | conv3-64<br>conv3-64 | conv3-64<br>conv3-64 |
| maxpool | | | | | |
| conv3-128 | conv3-128 | conv3-128<br>**conv3-128** | conv3-128<br>conv3-128 | conv3-128<br>conv3-128 | conv3-128<br>conv3-128 |
| maxpool | | | | | |
| conv3-256<br>conv3-256 | conv3-256<br>conv3-256 | conv3-256<br>conv3-256 | conv3-256<br>conv3-256<br>**conv1-256** | conv3-256<br>conv3-256<br>**conv3-256** | conv3-256<br>conv3-256<br>conv3-256<br>**conv3-256** |
| maxpool | | | | | |
| conv3-512<br>conv3-512 | conv3-512<br>conv3-512 | conv3-512<br>conv3-512 | conv3-512<br>conv3-512<br>**conv1-512** | conv3-512<br>conv3-512<br>**conv3-512** | conv3-512<br>conv3-512<br>conv3-512<br>**conv3-512** |
| maxpool | | | | | |
| conv3-512<br>conv3-512 | conv3-512<br>conv3-512 | conv3-512<br>conv3-512 | conv3-512<br>conv3-512<br>**conv1-512** | conv3-512<br>conv3-512<br>**conv3-512** | conv3-512<br>conv3-512<br>conv3-512<br>**conv3-512** |
| maxpool | | | | | |
| FC-4096 | | | | | |
| FC-4096 | | | | | |
| FC-1000 | | | | | |
| soft-max | | | | | |

# VGGNet

- Accuracy 92.7%
- Weight Initialisation I
  - Initialised with trained network A
  - Random for intermediate layers.
- Weight Initialisation II
  - Random initialisation for all.
- (-) Used Dense instead of Global Average Pooling final layer
  - Up to 138 million parameters (Dense)
  - Only 21 million parameters (Global Average Pooling)

# VGGNet (Dense vs Global Average)

| Layer Type | Shape | Parameters |
|---|---|---|
| Input | (3, 224, 224) | |
| Conv(64) | (64, 224, 224) | 1,792 |
| Conv(64) | (64, 224, 224) | 36,928 |
| MaxPool2D | (64, 112, 112) | |
| Conv(128) | (128, 112, 112) | 73,856 |
| Conv(128) | (128, 112, 112) | 14,7584 |
| MaxPool2D | (128, 56, 56) | |
| Conv(256) | (256, 56, 56) | 295,168 |
| Conv(256) | (256, 56, 56) | 590,080 |
| Conv(256) | (256, 56, 56) | 590,080 |
| MaxPool2D | (256, 28, 28) | |
| Conv(512) | (512, 28, 28) | 1,180,160 |
| Conv(512) | (512, 28, 28) | 2,359,808 |
| Conv(512) | (512, 28, 28) | 2,359,808 |
| MaxPool2D | (512, 14, 14) | |
| Conv(512) | (512, 14, 14) | 2,359,808 |
| Conv(512) | (512, 14, 14) | 2,359,808 |
| Conv(512) | (512, 14, 14) | 2,359,808 |
| MaxPool2D | (512, 7, 7) | |
| DenseLayer | (4096) | 102,764,544 |
| DenseLayer | (4096) | 16,781,312 |
| Output | (1000) | 4,097,000 |
| **Total** | | **138,357,544** |

| Layer Type | Shape | Parameters |
|---|---|---|
| Input | (3, 224, 224) | |
| Conv(64) | (64, 224, 224) | 1,792 |
| Conv(64) | (64, 224, 224) | 36,928 |
| MaxPool2D | (64, 112, 112) | |
| Conv(128) | (128, 112, 112) | 73,856 |
| Conv(128) | (128, 112, 112) | 14,7584 |
| MaxPool2D | (128, 56, 56) | |
| Conv(256) | (256, 56, 56) | 295,168 |
| Conv(256) | (256, 56, 56) | 590,080 |
| Conv(256) | (256, 56, 56) | 590,080 |
| MaxPool2D | (256, 28, 28) | |
| Conv(512) | (512, 28, 28) | 1,180,160 |
| Conv(512) | (512, 28, 28) | 2,359,808 |
| Conv(512) | (512, 28, 28) | 2,359,808 |
| MaxPool2D | (512, 14, 14) | |
| Conv(512) | (512, 14, 14) | 2,359,808 |
| Conv(512) | (512, 14, 14) | 2,359,808 |
| Conv(512) | (512, 14, 14) | 2,359,808 |
| MaxPool2D | (512, 7, 7) | |
| GlobalPool | (512) | |
| DenseLayer | (4096) | 2,101,248 |
| Output | (1000) | 4,097,000 |
| **Total** | | **20,912,936** |

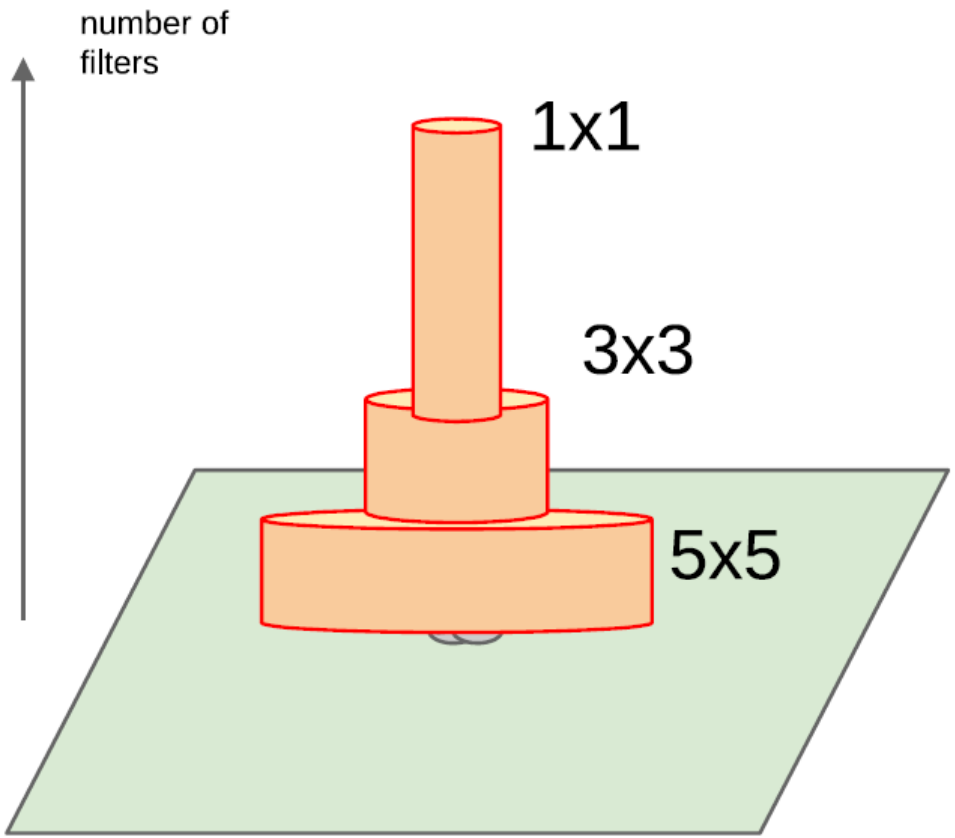
# References

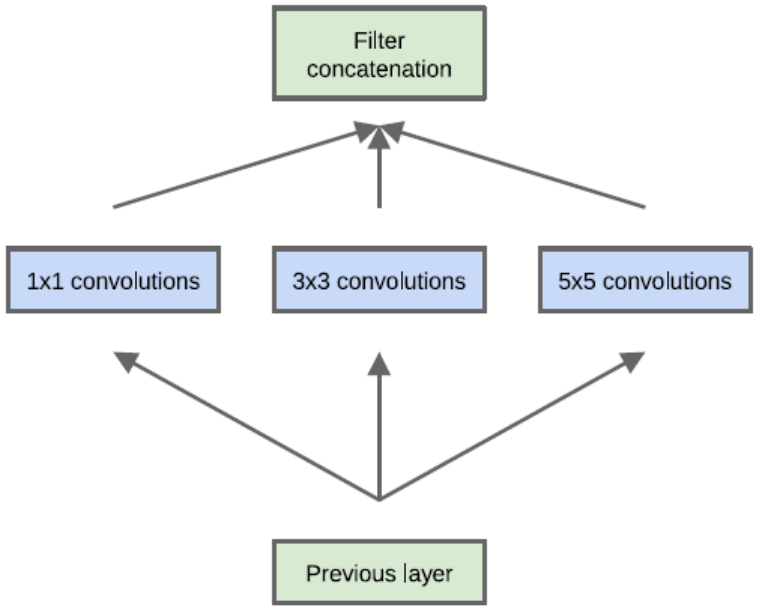
Simonyan, K., & Zisserman, A. (2014). *"Very deep convolutional networks for large-scale image recognition."* arXiv preprint arXiv:1409.1556.

# Outline

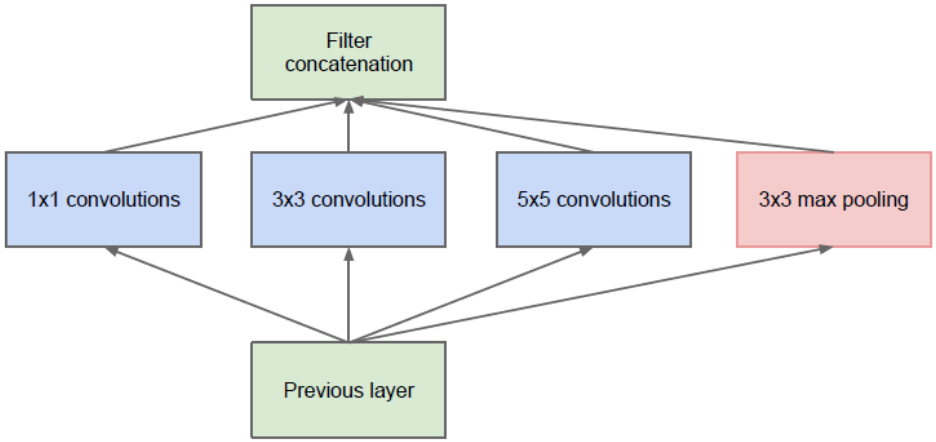
1. CIFAR-10
2. Intro to Convolutional Neural Networks (ConvNets)
3. Popular Architectures
   - ImageNet
   - LeNet-5 [1998]
   - AlexNet [2012]
   - ZFNet [2013]
   - VGGNet [2014]
   - GoogLeNet [2015]
   - ResNet [2016]

## Schematic view (naive version)

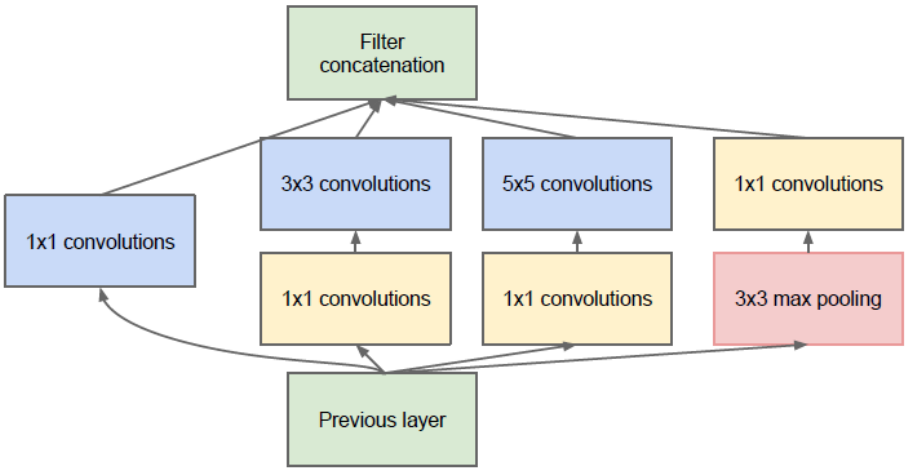


# GoogLeNet




(a) Inception module, naïve version




(b) Inception module with dimensionality reduction

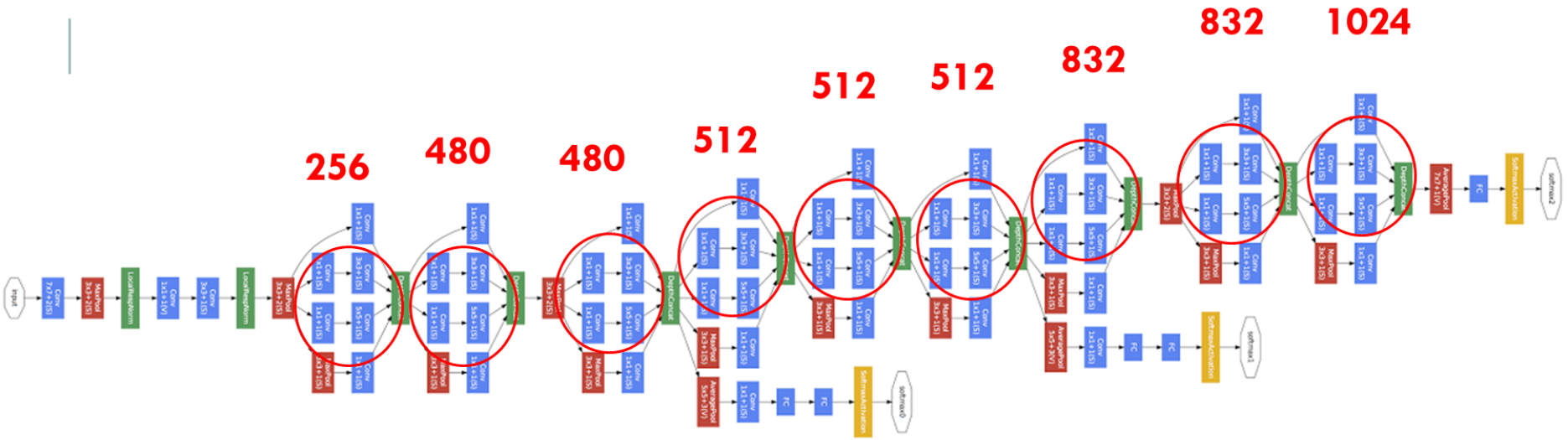
Figure 2: Inception module

# GoogLeNet

| type | patch size/ stride | output size | depth | #1×1 | #3×3 reduce | #3×3 | #5×5 reduce | #5×5 | pool proj | params | ops |
|---|---|---|---|---|---|---|---|---|---|---|---|
| convolution | 7×7/2 | 112×112×64 | 1 | | | | | | | 2.7K | 34M |
| max pool | 3×3/2 | 56×56×64 | 0 | | | | | | | | |
| convolution | 3×3/1 | 56×56×192 | 2 | | 64 | 192 | | | | 112K | 360M |
| max pool | 3×3/2 | 28×28×192 | 0 | | | | | | | | |
| inception (3a) | | 28×28×256 | 2 | 64 | 96 | 128 | 16 | 32 | 32 | 159K | 128M |
| inception (3b) | | 28×28×480 | 2 | 128 | 128 | 192 | 32 | 96 | 64 | 380K | 304M |
| max pool | 3×3/2 | 14×14×480 | 0 | | | | | | | | |
| inception (4a) | | 14×14×512 | 2 | 192 | 96 | 208 | 16 | 48 | 64 | 364K | 73M |
| inception (4b) | | 14×14×512 | 2 | 160 | 112 | 224 | 24 | 64 | 64 | 437K | 88M |
| inception (4c) | | 14×14×512 | 2 | 128 | 128 | 256 | 24 | 64 | 64 | 463K | 100M |
| inception (4d) | | 14×14×528 | 2 | 112 | 144 | 288 | 32 | 64 | 64 | 580K | 119M |
| inception (4e) | | 14×14×832 | 2 | 256 | 160 | 320 | 32 | 128 | 128 | 840K | 170M |
| max pool | 3×3/2 | 7×7×832 | 0 | | | | | | | | |
| inception (5a) | | 7×7×832 | 2 | 256 | 160 | 320 | 32 | 128 | 128 | 1072K | 54M |
| inception (5b) | | 7×7×1024 | 2 | 384 | 192 | 384 | 48 | 128 | 128 | 1388K | 71M |
| avg pool | 7×7/1 | 1×1×1024 | 0 | | | | | | | | |
| dropout (40%) | | 1×1×1024 | 0 | | | | | | | | |
| linear | | 1×1×1000 | 1 | | | | | | | 1000K | 1M |
| softmax | | 1×1×1000 | 0 | | | | | | | | |

Table 1: GoogLeNet incarnation of the Inception architecture.

# GoogLeNet: Auxiliary Classifiers




- Encourage discrimination in early layers
- Increase gradient proportion during backprop
- Regularizer

# GoogLeNet: Average Pooling

| Method | Testing Error |
|---|---|
| mlpconv + Fully Connected | 11.59% |
| mlpconv + Fully Connected + Dropout | 10.88% |
| mlpconv + Global Average Pooling | 10.41% |

- Accuracy 95.3%
- 5 million parameters



Szegedy, C., Liu, W., Jia, Y., Sermanet, P., Reed, S., Anguelov, D., ... & Rabinovich, A. (2015). *"Going deeper with convolutions."* In Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition (pp. 1-9).


[12] Lin, M., Chen, Q., & Yan, S. (2013). *"Network in network."* arXiv preprint arXiv:1312.4400.

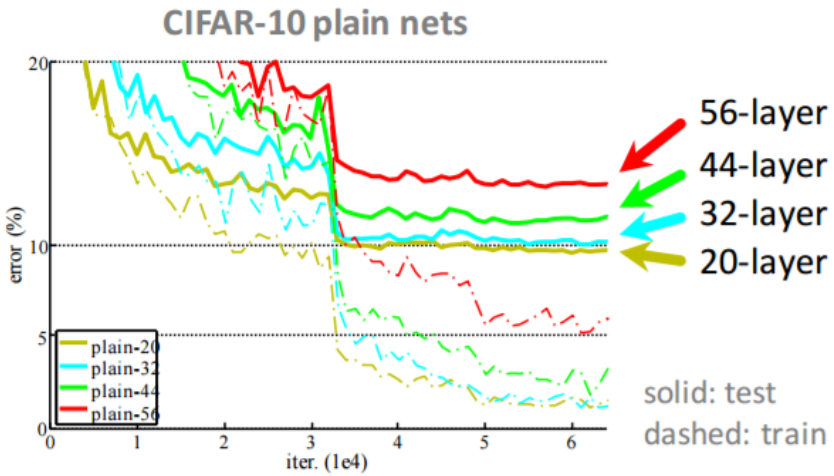
From NUS Engine Faculty!!!

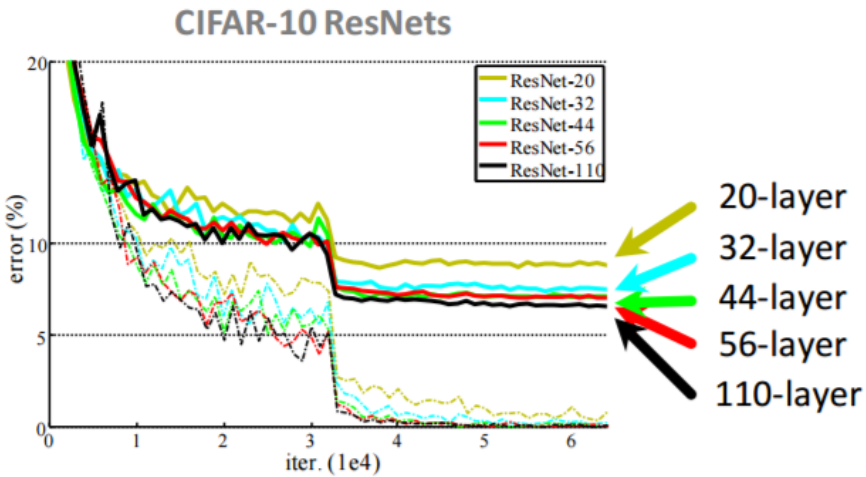
# Outline

1. CIFAR-10
2. Intro to Convolutional Neural Networks (ConvNets)
3. Popular Architectures
   - ImageNet
   - LeNet-5 [1998]
   - AlexNet [2012]
   - ZFNet [2013]
   - VGGNet [2014]
   - GoogLeNet [2015]
   - ResNet [2016]

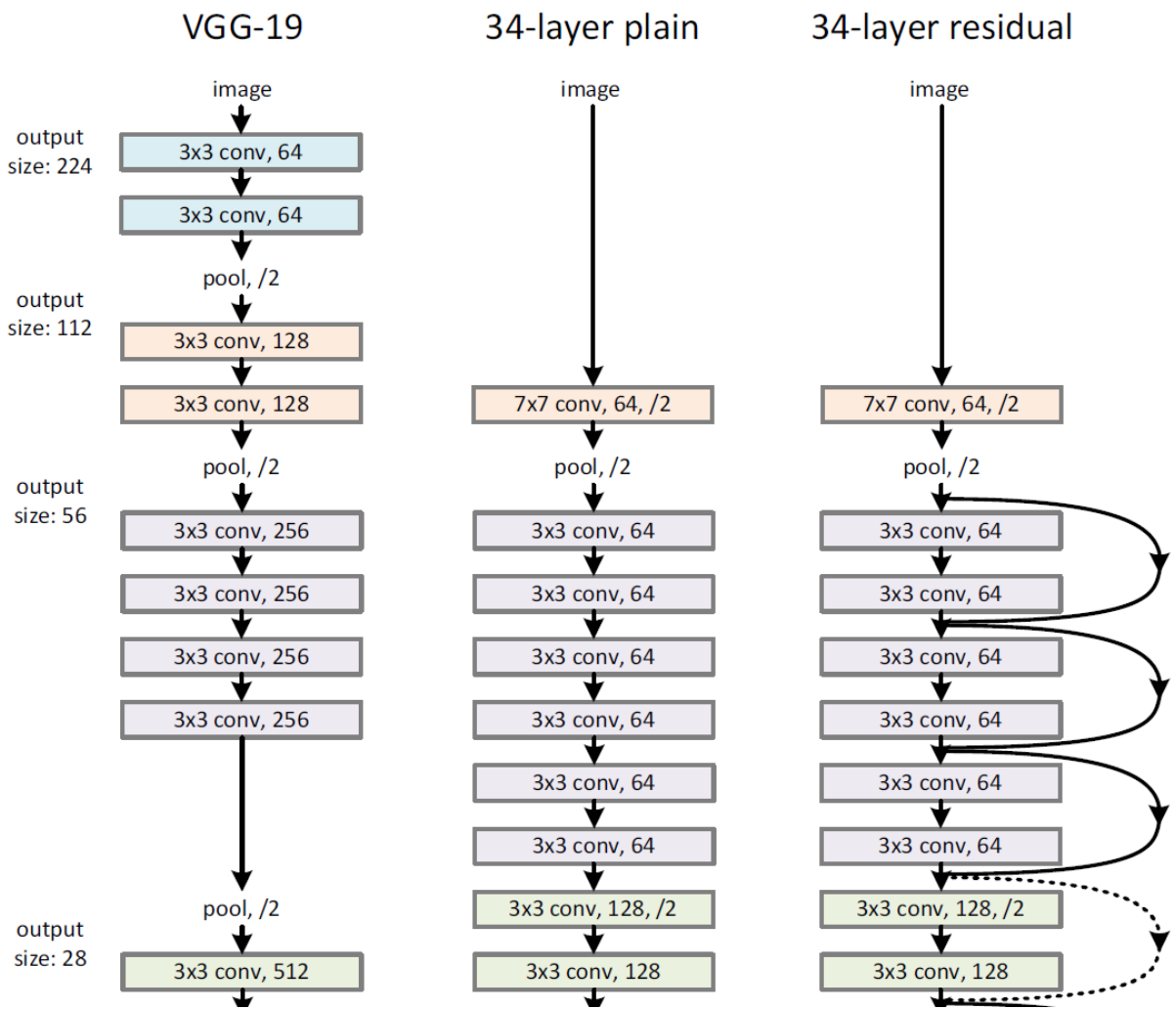
# ResNet

Idea: Deeper layers should provide better accuracy/error rates



# ResNet




Accuracy 96.4% ($>$ 95%!)

# ResNet

Motivation: Deeper layers should give less error

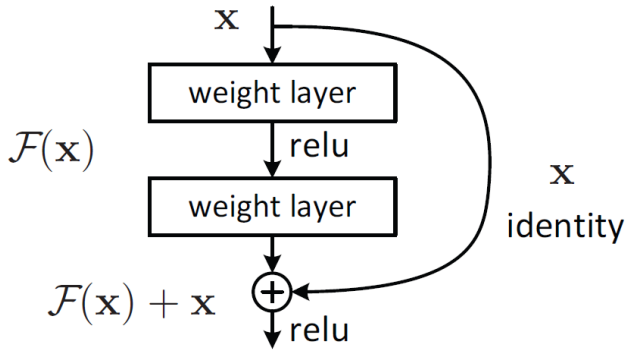
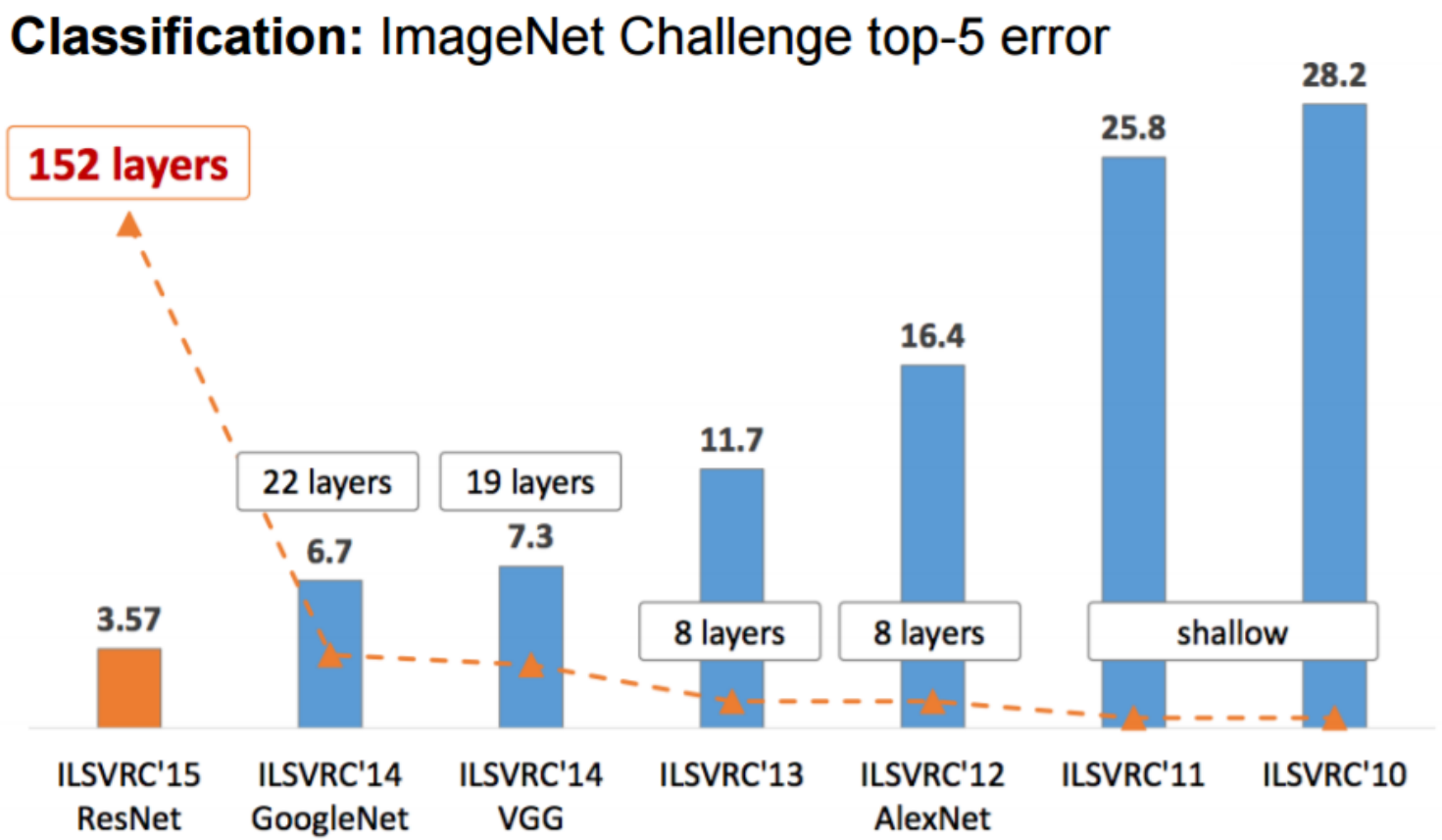



Figure 2. Residual learning: a building block.

- Difficult to learn $H(x) = F(x) + x$ directly.
- Assume $I(x)$ for "non-existent" layers.
- $F(x)$: 'perturbations from $I(x)$'.
  - If optimal $H(x)$ is indeed $I(x)$ (unlikely scenerio), weights will simply learn to be 0.

# ResNet: Depth





He, K., Zhang, X., Ren, S., & Sun, J. (2015). *"Deep residual learning for image recognition. "* arXiv preprint arXiv:1512.03385.

He, K. ICML 2016 *"Tutorial on Deep Residual Networks."*

# Thank You