Unit 7_04 Convnet Case Studies

TFIP-AI Artificial Neural Networks and Deep Learning

Deep learning architecture

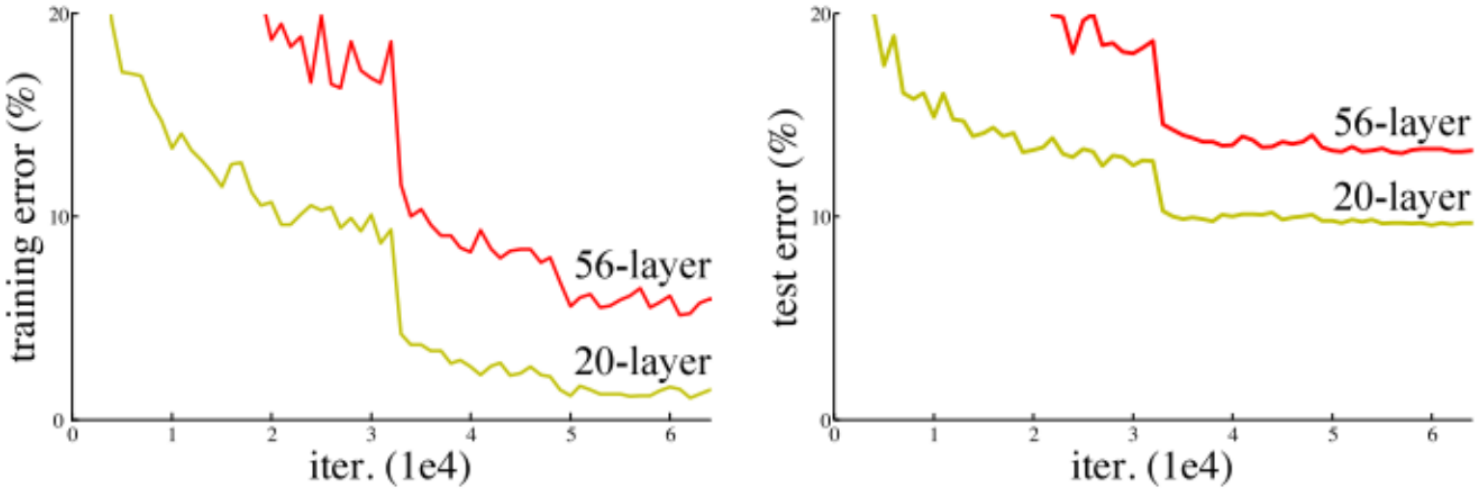
- Deep learning architecture
  - LeNet 5
  - **AlexNet** @ the LSVRC2012 classification contest
  - **VGG16, VGG19 – 19 layers**
  - GoogleNet(inception_v1) 22 layers
  - **Deep Residual Network**
- ResNet makes it possible to train up to hundreds or even thousands of layers and still achieves compelling performance.

Resnet

Convnet Case Studies

Motivation

Increasing network depth does not work by simply stacking layers together. As the network goes deeper, its performance gets saturated or even starts degrading rapidly.



increasing network depth leads to worse performance

ResNet - idea

- When net starts degrading, net with less layers performs better than a deeper net, why not passing feature maps from early layer onto the deeper layer directly
- It's called identity shortcut connection or identity mapping

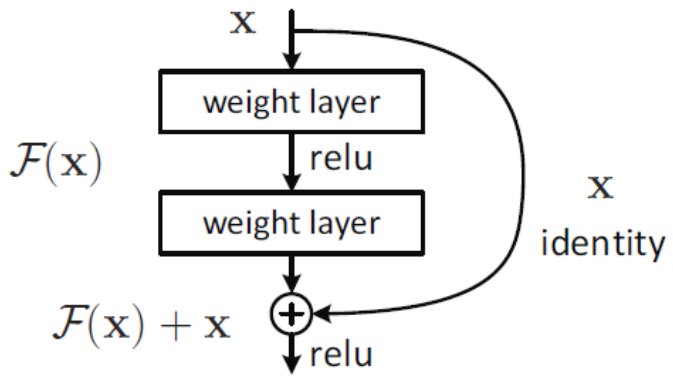
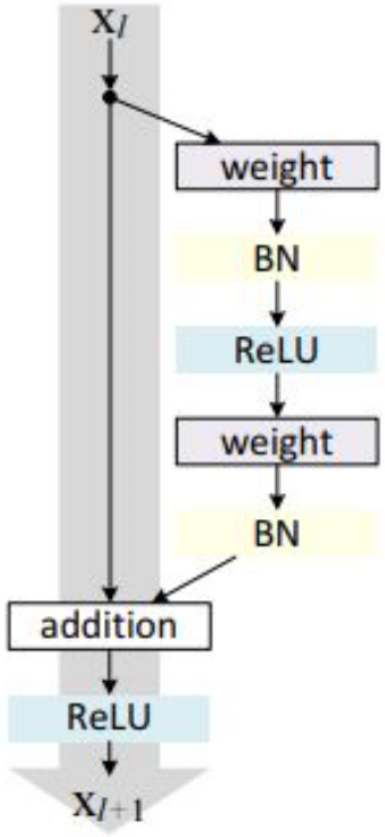


Figure 2. Residual learning: a building block.

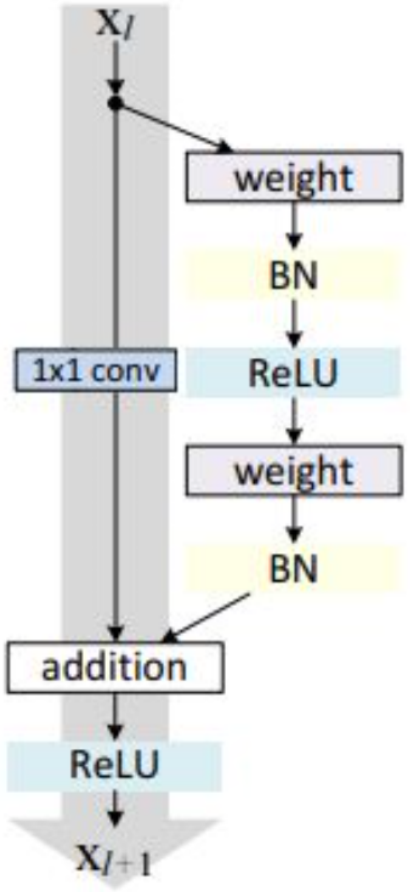
Residual block



$$x_{l+1} = x_l + \mathcal{F}(x_l, W_l)$$

- Weight
  - Convolution
- BN
  - batch normalization
  - To avoid gradient vanishing

Residual block



Predicted

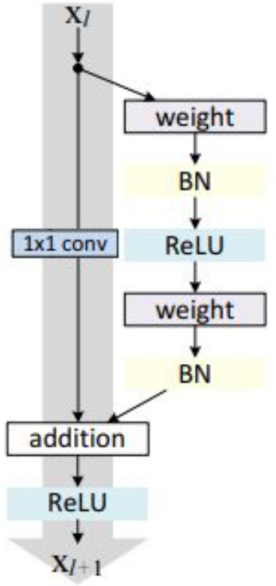
Observed

Residual

$$x_{l+1} = h(x_l) + \mathcal{F}(x_l, W_l)$$

- Number of feature map in level l is not equal to number of feature maps in level $l + 1$
- $h(x)$
  - 1×1conv

Resnet_block_v1, code in keras




```
def res_block_v1(x, input_filter, output_filter):
    res_x = Conv2D(kernel_size=(3,3), filters=output_filter, strides=1, padding='same'
    res_x = BatchNormalization()(res_x)
    res_x = Activation('relu')(res_x)
    res_x = Conv2D(kernel_size=(3,3), filters=output_filter, strides=1, padding='same')(res_x)
    res_x = BatchNormalization()(res_x)
    if input_filter == output_filter:
        identity = x
    else:
        identity = Conv2D(kernel_size=(1,1), filters=output_filter, strides=1, padding='same')(x)
    x = keras.layers.add([identity, res_x])
    output = Activation('relu')(x)
    return output
```

Batch Normalize

Input: Values of x over a mini-batch: $\mathcal{B} = \{x_{1...m}\}$;
Parameters to be learned: γ, β
Output: $\{y_i = \mathrm{BN}_{\gamma,\beta}(x_i)\}$

$$\mu_{\mathcal{B}} \leftarrow \frac{1}{m}\sum_{i=1}^{m} x_i \qquad \text{// mini-batch mean}$$

$$\sigma^2_{\mathcal{B}} \leftarrow \frac{1}{m}\sum_{i=1}^{m}(x_i - \mu_{\mathcal{B}})^2 \qquad \text{// mini-batch variance}$$

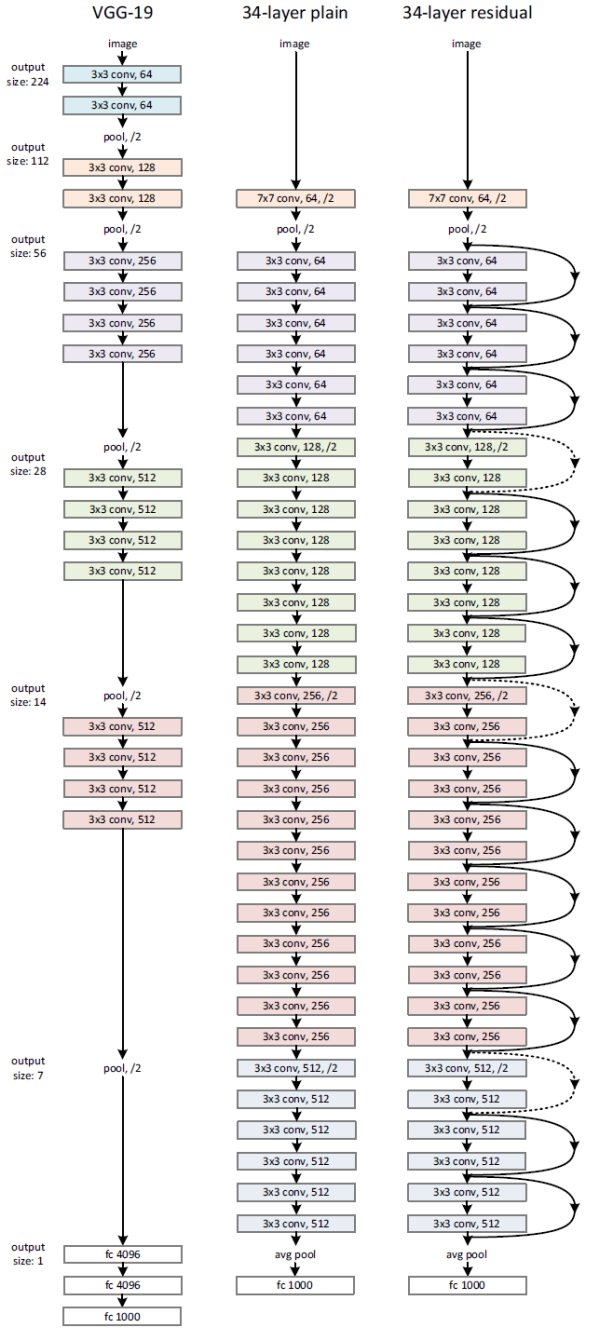
$$\widehat{x}_i \leftarrow \frac{x_i - \mu_{\mathcal{B}}}{\sqrt{\sigma^2_{\mathcal{B}} + \epsilon}} \qquad \text{// normalize}$$

$$y_i \leftarrow \gamma\widehat{x}_i + \beta \equiv \mathrm{BN}_{\gamma,\beta}(x_i) \qquad \text{// scale and shift}$$

Algorithm 1: Batch Normalizing Transform, applied to activation x over a mini-batch.

Build up a ResNet

- Step 1: Build up a plain deep net
- Step 2: Insert identity mapping between the specified two layers



Stacked resnet block to build a ResNet

```
def resnet_v1(x):
    x = Conv2D(kernel_size=(3,3), filters=16, strides=1, padding='same', activation='relu')(x)
    x = res_block_v1(x, 16, 16)
    x = res_block_v1(x, 16, 32)
    x = Flatten()(x)
    outputs = Dense(10, activation='softmax', kernel_initializer='he_normal')(x)
    return outputs
```

Residual block

$$y_l = h(x_l) + \mathcal{F}(x_l, W_l)$$

$$x_{l+1} = f(y_l)$$

$$x_{l+1} = x_l + \mathcal{F}(x_l, W_l)$$

- $h(x)$
  - 1×1 conv or identity mapping
- $f(x)$
  - Activation function

- Assuming
- $h(x)$ is identity mapping
  - i.e. $h(x) = x$
- $f(x)$ is also identity mapping

Gradient of loss function

- If both functions are identity mapping, for a deeper level $L > l$

$$x_L = x_l + \sum_{i=1}^{L-1} \mathcal{F}(x_i, W_i) \qquad x_{l+1} = x_l + \mathcal{F}(x_l, W_l)$$

- The gradient $$\frac{\partial \varepsilon}{\partial x_l} = \frac{\partial \varepsilon}{\partial x_L}\frac{\partial x_L}{\partial x_l} = \frac{\partial \varepsilon}{\partial x_L}(1 + \frac{\partial}{\partial x_l}\sum_{i=1}^{L-1}\mathcal{F}(x_i, W_i)) = \frac{\partial \varepsilon}{\partial x_L} + \frac{\partial \varepsilon}{\partial x_L}\frac{\partial}{\partial x_l}\sum_{i=1}^{L-1}\mathcal{F}(x_i, W_i)$$

- From the feed-forward and back-propagation
  - When the two assumptions applied, the info passes back and forth between early level and deeper level

Is identity mapping the best?

- Assuming $h(x) = \lambda_l x_l$

- Residual Block, level l

$$x_{l+1} = \lambda_l x_l + \mathcal{F}(x_l, W_l)$$

- Level L

$$x_L = (\prod_{i=l}^{L-1} \lambda_l) x_l + \sum_{i=l}^{L-1} ((\prod_{i=l}^{L-1}) \mathcal{F}(x_l, W_l)$$

Is identity mapping the best?

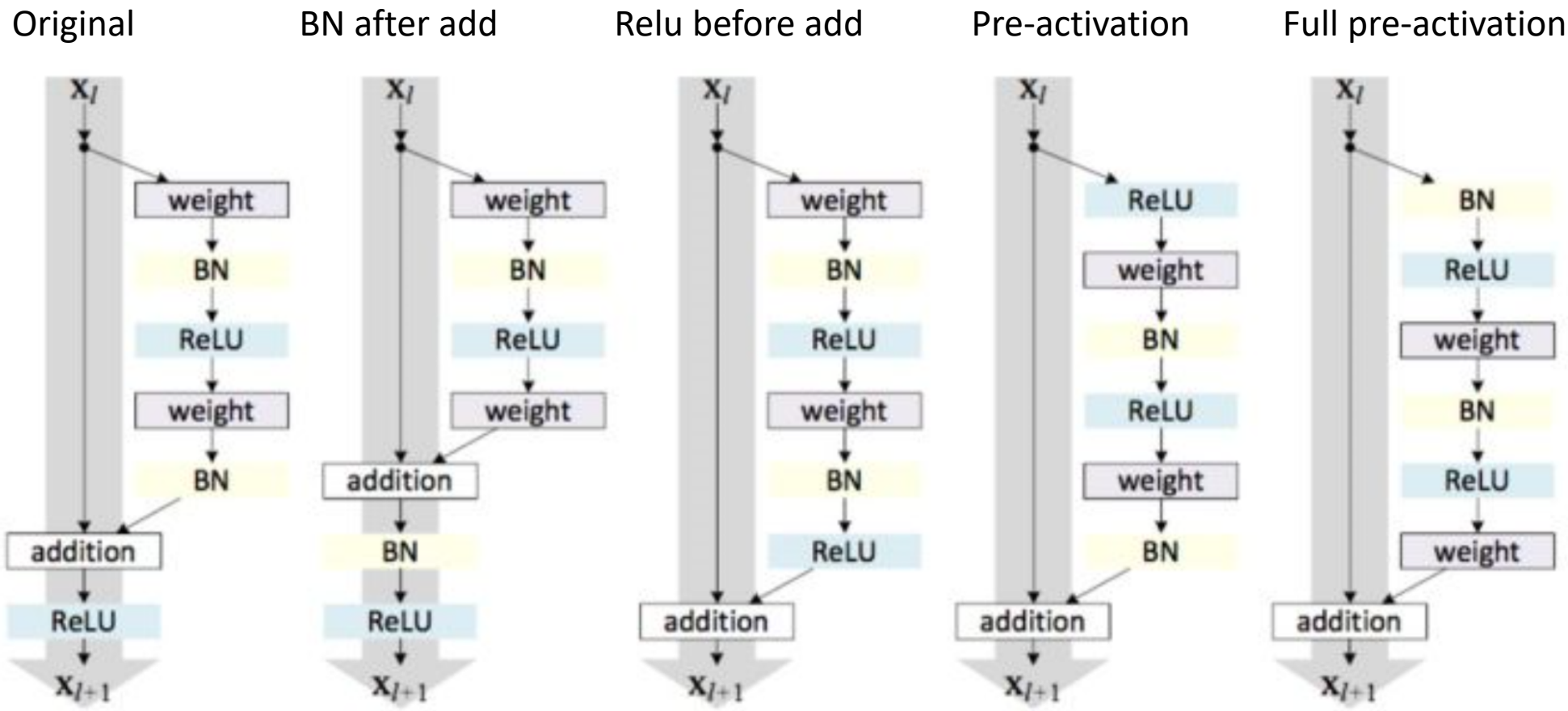
$$x_L = (\prod_{i=l}^{L-1} \lambda_l) x_l + \sum_{i=l}^{L-1} ((\prod_{i=l}^{L-1}) \mathcal{F}(x_l, W_l)$$

- The gradient of first term of x_L, let $x'L = (\prod i = l^{L-1} \lambda_l) x_l$

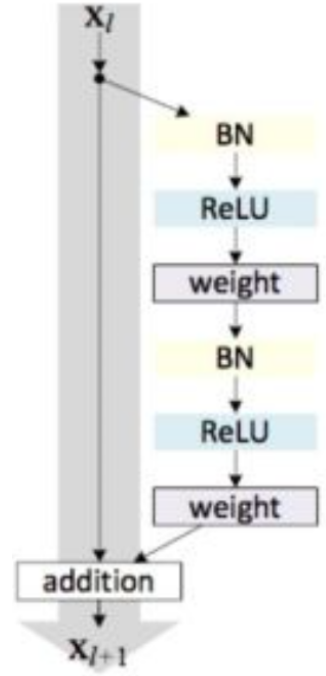
$$\frac{\partial \varepsilon}{\partial x_l} = \frac{\partial \varepsilon}{\partial x' L} (\prod i = l^{L-1} \lambda_i)$$

- From the partial differentiation, we find
  - When $\lambda \neq 1$ → gradient exploding or vanishing → only when $\lambda = 1$, i.e. identity mapping could avoid it.

Move the activation function inside residual



Resnet_block_v2, code in keras

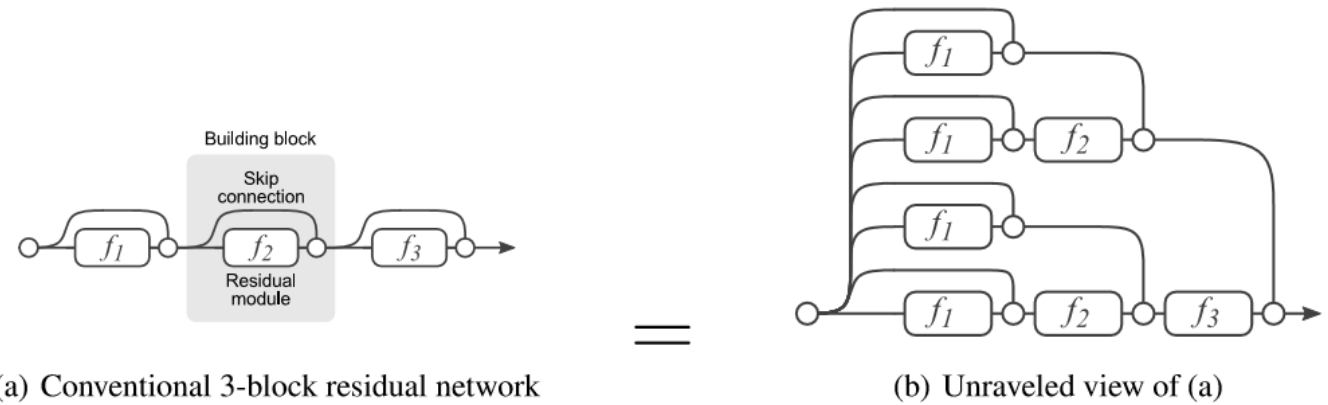



```
def res_block_v2(x, input_filter, output_filter):
    res_x = BatchNormalization()(x)
    res_x = Activation('relu')(res_x)
    res_x = Conv2D(kernel_size=(3,3), filters=output_filter, strides=1, padding='same')(res_x)
    res_x = BatchNormalization()(res_x)
    res_x = Activation('relu')(res_x)
    res_x = Conv2D(kernel_size=(3,3), filters=output_filter, strides=1, padding='same')(res_x)
    if input_filter == output_filter:
        identity = x
    else:
        identity = Conv2D(kernel_size=(1,1), filters=output_filter, strides=1, padding='same')(x)
    output= keras.layers.add([identity, res_x])
    return output
```

Build up a ResNet

```
def resnet_v2(x):
    x = Conv2D(kernel_size=(3,3), filters=16 , strides=1, padding='same', activation='relu')(x)
    x = res_block_v2(x, 16, 16)
    x = res_block_v2(x, 16, 32)
    x = BatchNormalization()(x)
    y = Flatten()(x)
    outputs = Dense(10, activation='softmax', kernel_initializer='he_normal')(y)
    return outputs
```

ResNet – from a prospective of ensemble



(a) Conventional 3-block residual network

(b) Unraveled view of (a)

$$y_i \equiv f_i(y_{i-1}) + y_{i-1},$$

$$\begin{aligned} y_3 &= y_2 + f_3(y_2) \\ &= [y_1 + f_2(y_1)] + f_3(y_1 + f_2(y_1)) \\ &= \big[y_0 + f_1(y_0) + f_2(y_0 + f_1(y_0))\big] + f_3\big(y_0 + f_1(y_0) + f_2(y_0 + f_1(y_0))\big) \end{aligned}$$

ResNet – from a prospective of ensemble

The unravelled view reveals that:

First, Residual networks can be viewed as a collection of many paths, instead of a single ultra deep network.

Second, although these paths are trained jointly, they do not strongly depend on each other.

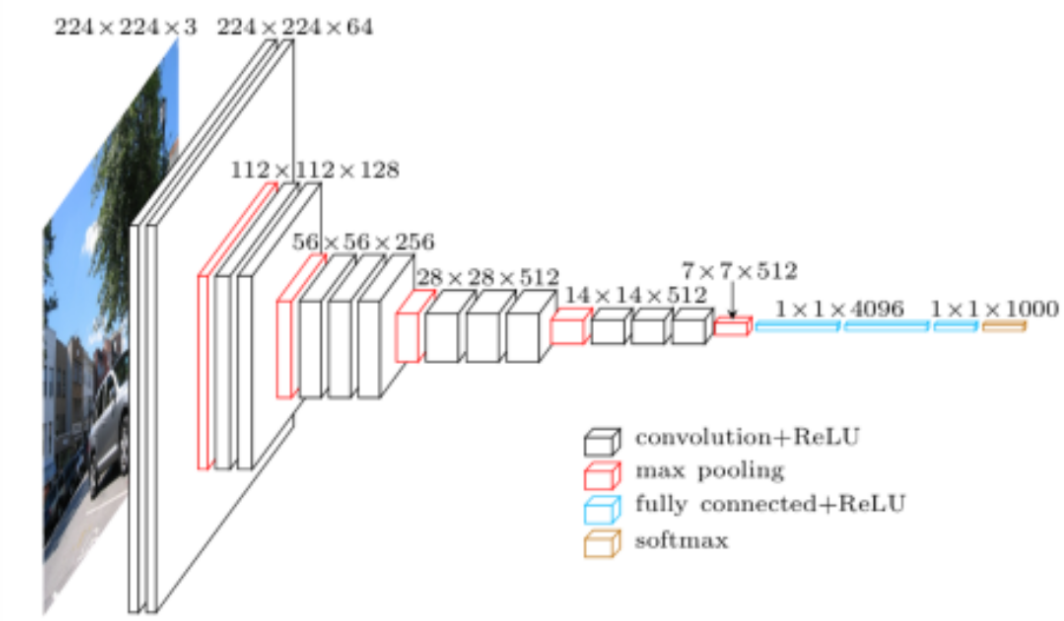
Finally, the paths through the network that contribute gradient during training are shorter than expected.

VGG 16

Convnet Case Studies

VGG

- VGG is a convolutional neural network model proposed by K. Simonyan and A. Zisserman from the University of Oxford in the paper "Very Deep Convolutional Networks for Large-Scale Image Recognition"



VGG16, VGG19

Simonyan and Zisserman found training VGG16 and VGG19 challenging (**painfully slow**), so in order to make training easier, they first trained *smaller* versions of VGG with less weight layers (columns A and C) first.

The smaller networks converged and were then used as *initializations* for the larger, deeper networks

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 **LRN**	conv3-64 **conv3-64**	conv3-64 conv3-64	conv3-64 conv3-64	conv3-64 conv3-64
maxpool					
conv3-128	conv3-128	conv3-128 **conv3-128**	conv3-128 conv3-128	conv3-128 conv3-128	conv3-128 conv3-128
maxpool					
conv3-256 conv3-256	conv3-256 conv3-256	conv3-256 conv3-256	conv3-256 conv3-256 **conv1-256**	conv3-256 conv3-256 **conv3-256**	conv3-256 conv3-256 conv3-256 **conv3-256**
maxpool					
conv3-512 conv3-512	conv3-512 conv3-512	conv3-512 conv3-512	conv3-512 conv3-512 **conv1-512**	conv3-512 conv3-512 **conv3-512**	conv3-512 conv3-512 conv3-512 **conv3-512**
maxpool					
conv3-512 conv3-512	conv3-512 conv3-512	conv3-512 conv3-512	conv3-512 conv3-512 **conv1-512**	conv3-512 conv3-512 **conv3-512**	conv3-512 conv3-512 conv3-512 **conv3-512**
maxpool					
FC-4096					
FC-4096					
FC-1000					
soft-max					

VGG 16

VGG 19

AlexNet

Convnet Case Studies

CNN for object recognition on ImageNet challenge

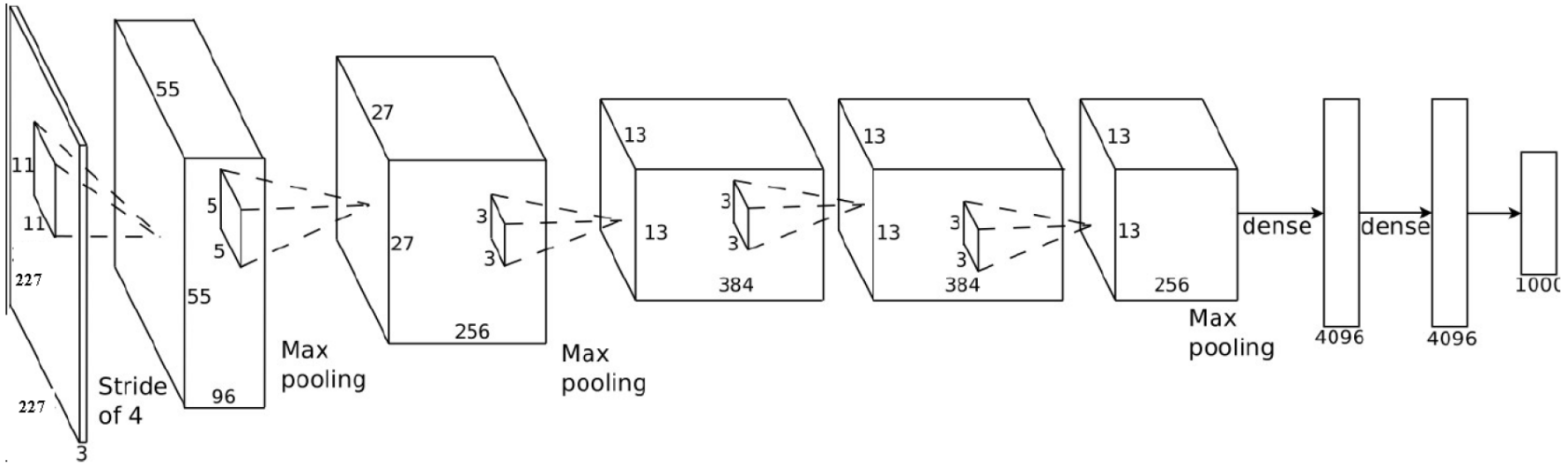
- Krizhevsky, Sutskever, and Hinton, NIPS 2012

- Trained on one million images of 1,000 categories
  - 2 GPU, 2 GB RAM on each GPU, 5GB of system memory
  - Training lasts for one week

- Google and Baidu announced their new visual search engines with the same technology six months later

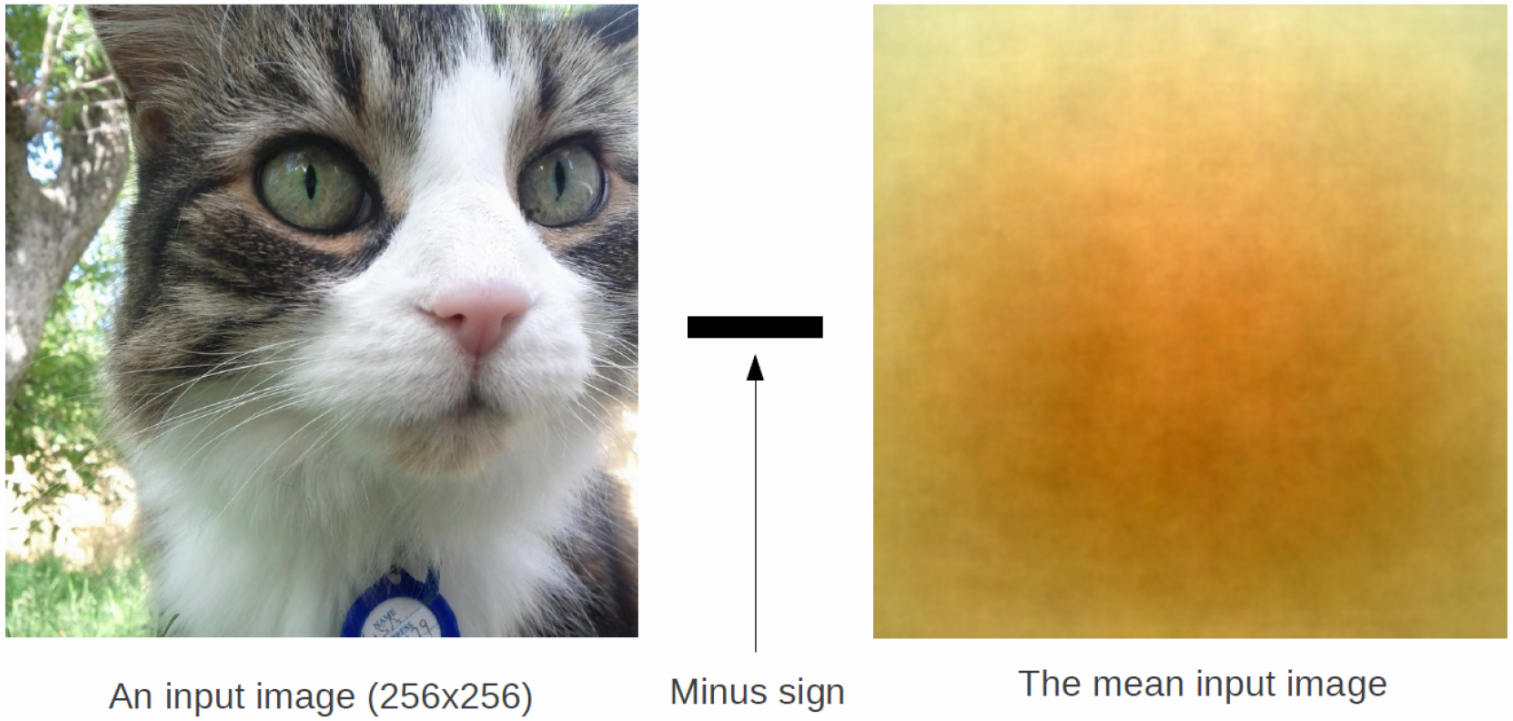
AlexNet

- 5 convolutional layers and 2 fully connected layers
- Max-pooling layers follow first, second, and fifth convolutional layers
  - Number of neurons: 253440, 186624, 64896, 64896, 43246, 4096, 4096 1000
- 650,000 neurons, 60,000,000 parameters, 630,000,000 connections



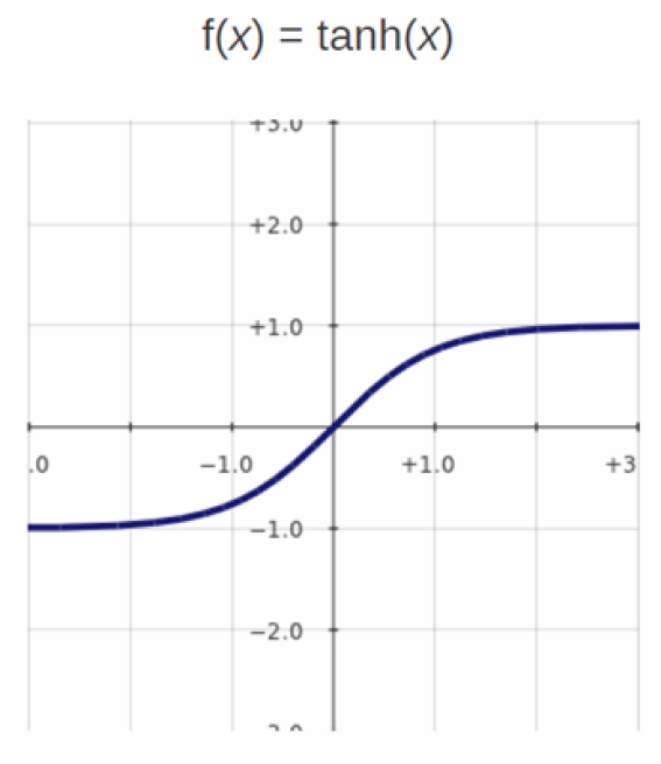
Technical details - normalization

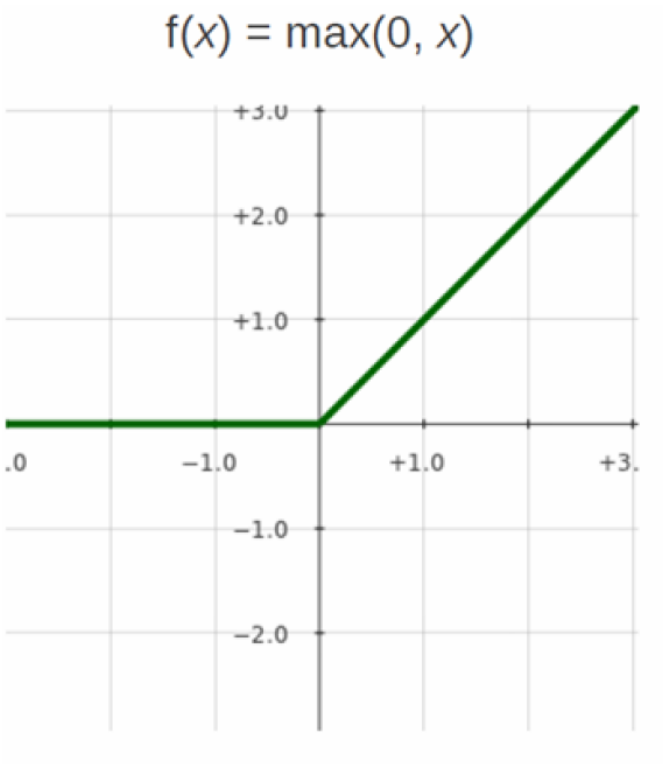
- Normalize the input by subtracting the mean image on the training set



Technical details – activation function

- Activation function



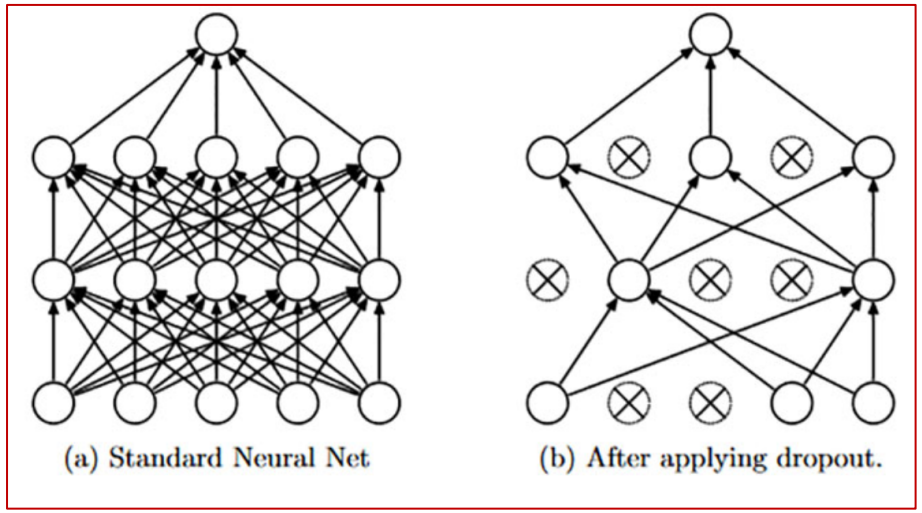


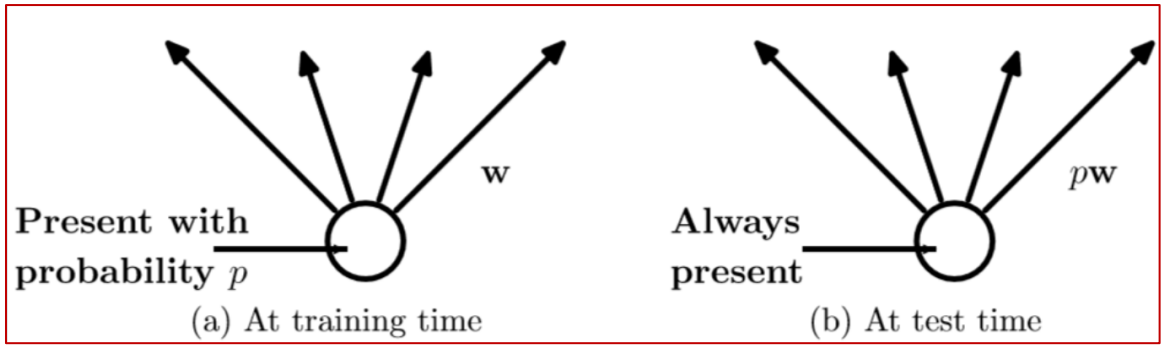
Technical details – data augmentation

- The net has 60M real-valued parameters and 650,000 neurons
- It overfits a lot. 224 by 224 image regions are randomly extracted from 256 images, and also their horizontal reflections

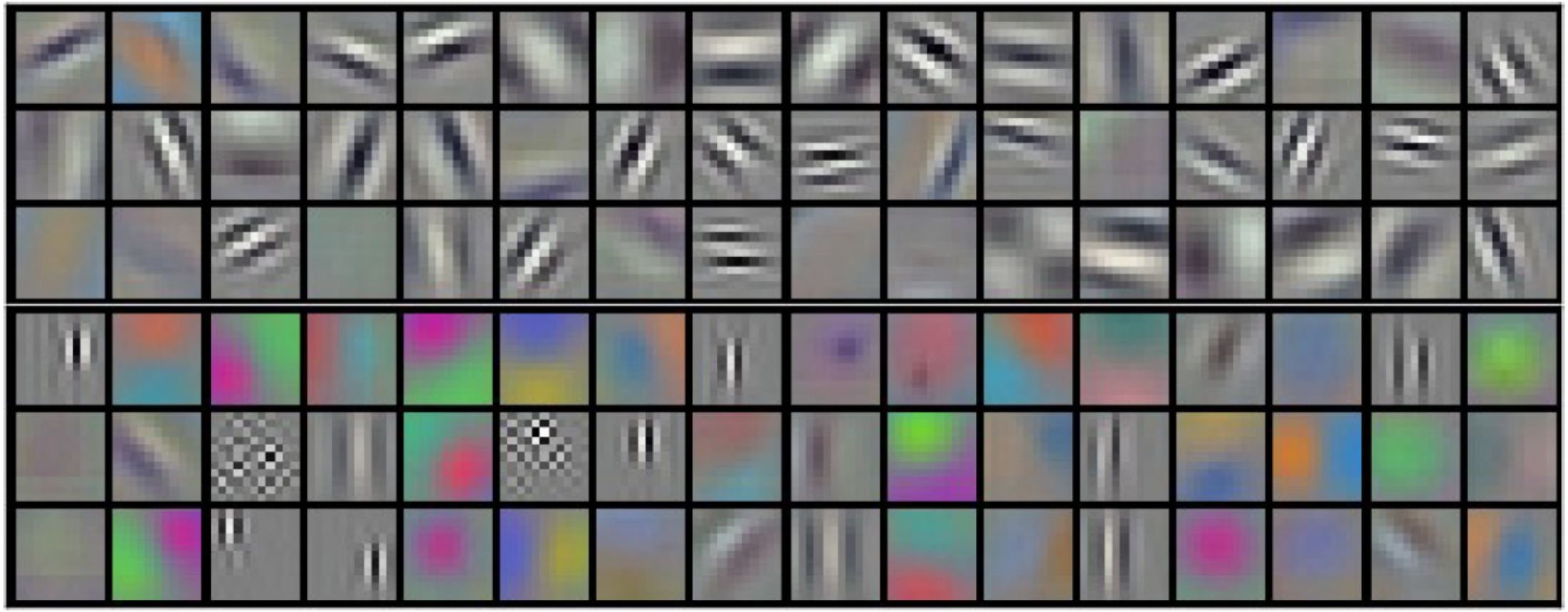
Technical details - dropout

- Independently set each hidden unit activity to zero with 0.5 probability
- Do this in the two globally-connected hidden layers at the net's output

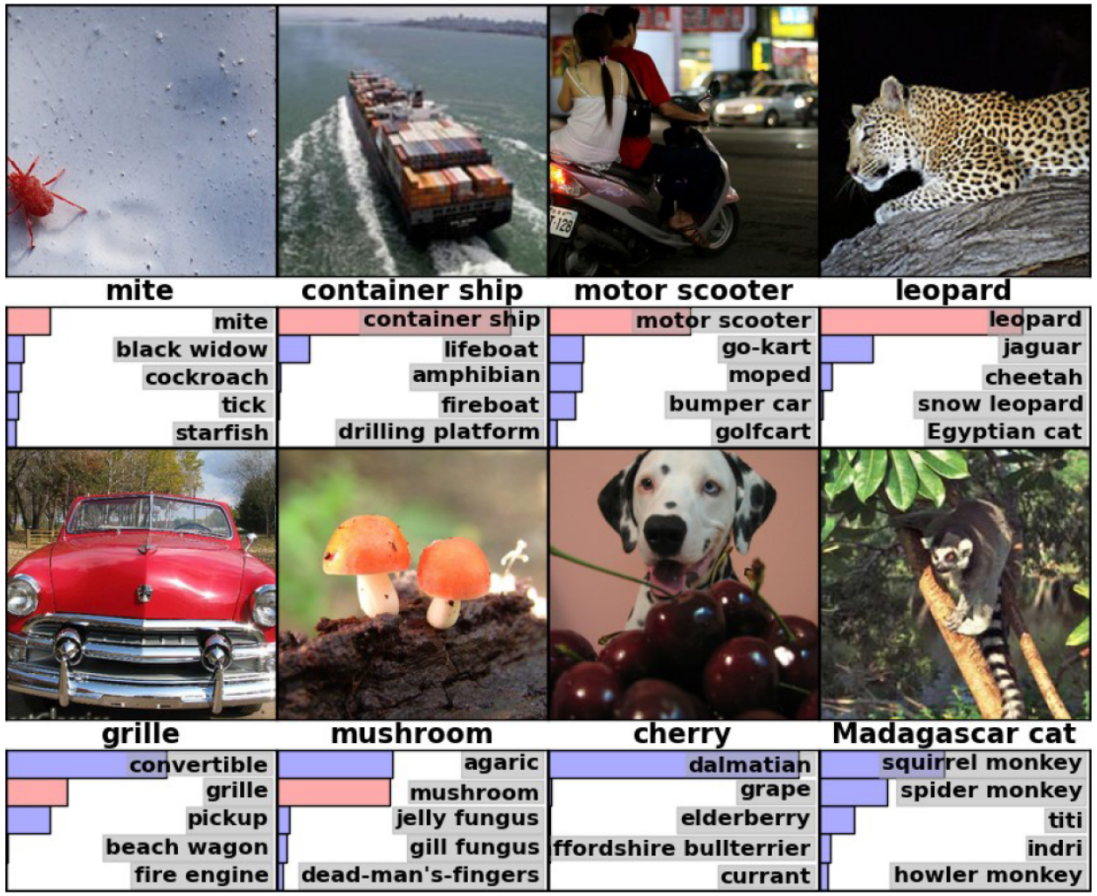




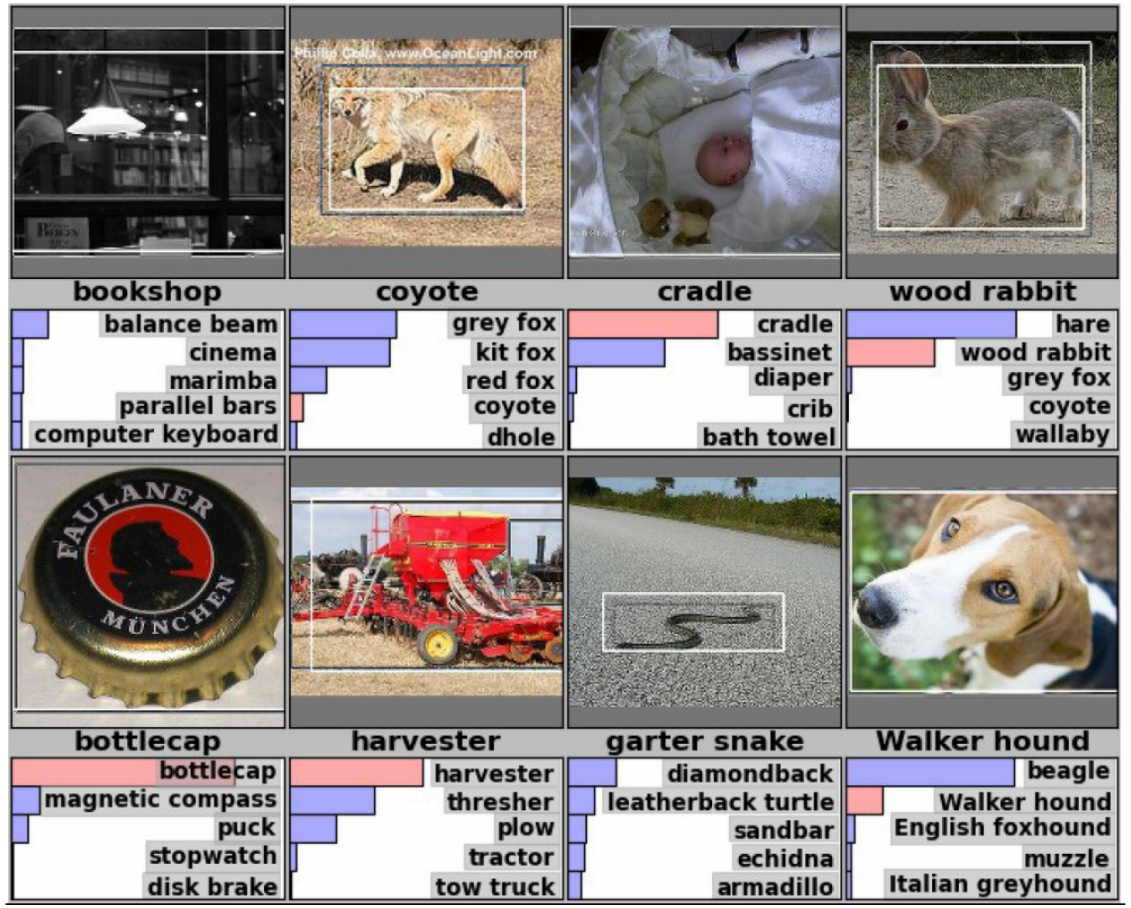
Learned low-level filters



Classification result



Detection result



Choose a pre-trained Convnet

- A pre-trained model which closely resembles your training data helps immensely

Model	Size	Top-1 Accuracy	Top-5 Accuracy	Parameters	Depth
Xception	88 MB	0.790	0.945	22,910,480	126
VGG16	528 MB	0.713	0.901	138,357,544	23
VGG19	549 MB	0.713	0.900	143,667,240	26
ResNet50	98 MB	0.749	0.921	25,636,712	-
ResNet101	171 MB	0.764	0.928	44,707,176	-
ResNet152	232 MB	0.766	0.931	60,419,944	-
ResNet50V2	98 MB	0.760	0.930	25,613,800	-
ResNet101V2	171 MB	0.772	0.938	44,675,560	-
ResNet152V2	232 MB	0.780	0.942	60,380,648	-
InceptionV3	92 MB	0.779	0.937	23,851,784	159
InceptionResNetV2	215 MB	0.803	0.953	55,873,736	572
MobileNet	16 MB	0.704	0.895	4,253,864	88
MobileNetV2	14 MB	0.713	0.901	3,538,984	88
DenseNet121	33 MB	0.750	0.923	8,062,504	121
DenseNet169	57 MB	0.762	0.932	14,307,880	169
DenseNet201	80 MB	0.773	0.936	20,242,984	201
NASNetMobile	23 MB	0.744	0.919	5,326,716	-
NASNetLarge	343 MB	0.825	0.960	88,949,818	-
EfficientNetB0	29 MB	-	-	5,330,571	-
EfficientNetB1	31 MB	-	-	7,856,239	-
EfficientNetB2	36 MB	-	-	9,177,569	-
EfficientNetB3	48 MB	-	-	12,320,535	-
EfficientNetB4	75 MB	-	-	19,466,823	-
EfficientNetB5	118 MB	-	-	30,562,527	-
EfficientNetB6	166 MB	-	-	43,265,143	-
EfficientNetB7	256 MB	-	-	66,658,687	-

The top-1 and top-5 accuracy refers to the model's performance on the ImageNet validation dataset.