LECTURE SERIES FOR DIGITAL SURVEILLANCE SYSTEMS AND APPLICATION

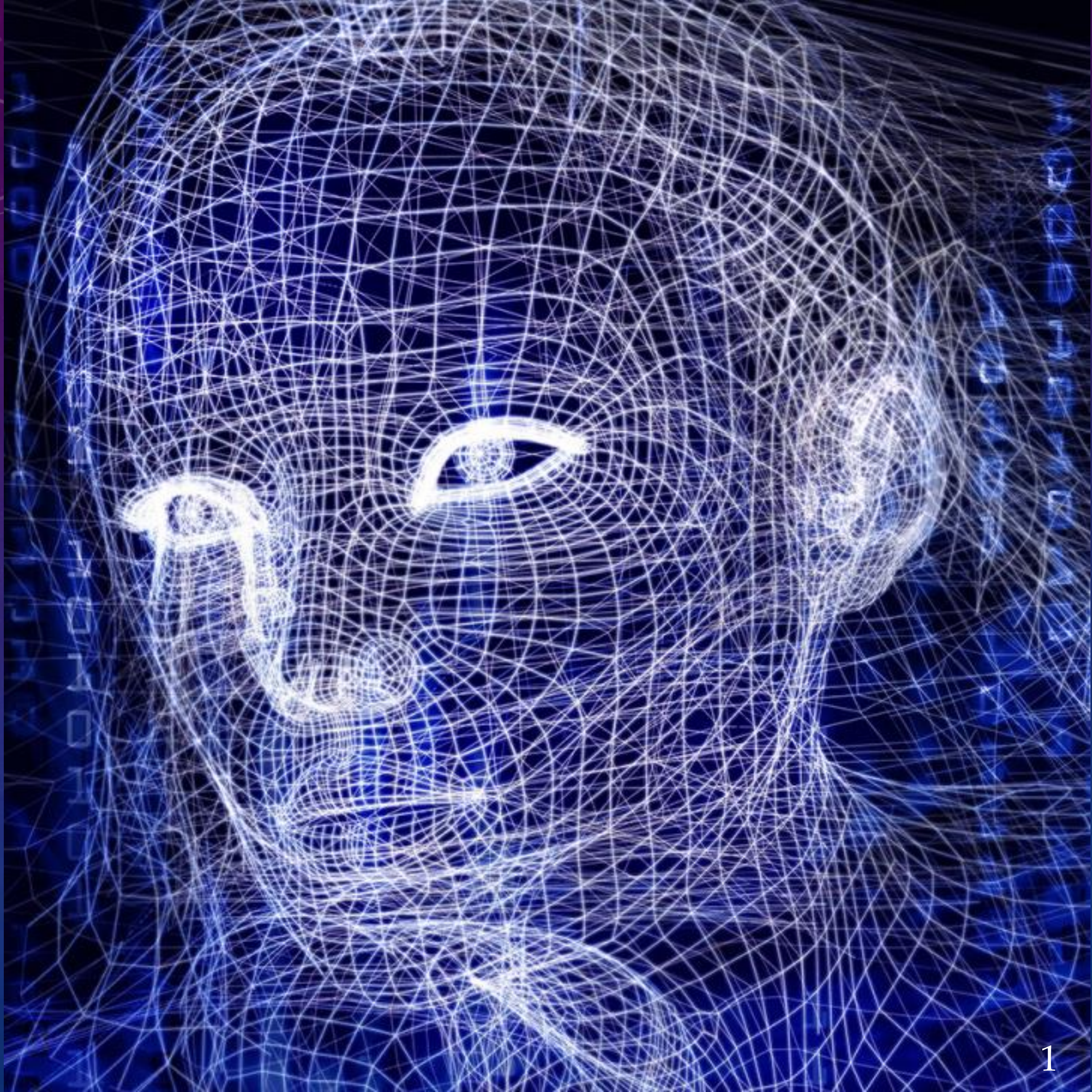
Chapter 3

Introduction to Convolution Neural Network

徐繼聖

Gee-Sern Jison Hsu

National Taiwan University of Science and Technology



Convolution Neural Networks

1) Deep Learning Software
2) CNN Architectures
   - LeNet
   - AlexNet
   - VGGNet
   - GoogleNet
   - ResNet
3) Well-Known Network
4) What Does Filter Learn?
5) Training Techniques
   - Dropout
   - Training Neural Networks
   - Batch Normalization

Deep Learning Software – Stanford University School of Engineering

https://www.youtube.com/watch?v=6SlgtELqOWc [1:17:39]

Deep Learning Software – Stanford University School of Engineering

Example: Matrix Multiplication(10:02 – 11:55)

Programming GPUs (11:56 – 14:22)

CPU vs GPU in practice (14:23 – 16:50)

CPU/GPU Communication (16:51 – 22:04)

Recall: Computational Graphs (22:05 – 22:49)

The point of deep learning frameworks (22:50 – 24:03)

Computation Graphs Numpy (24:04 – 27:37)

TensorFlow (27:38 – 50:44)

PyTorch (51:20 – 1:18:06)

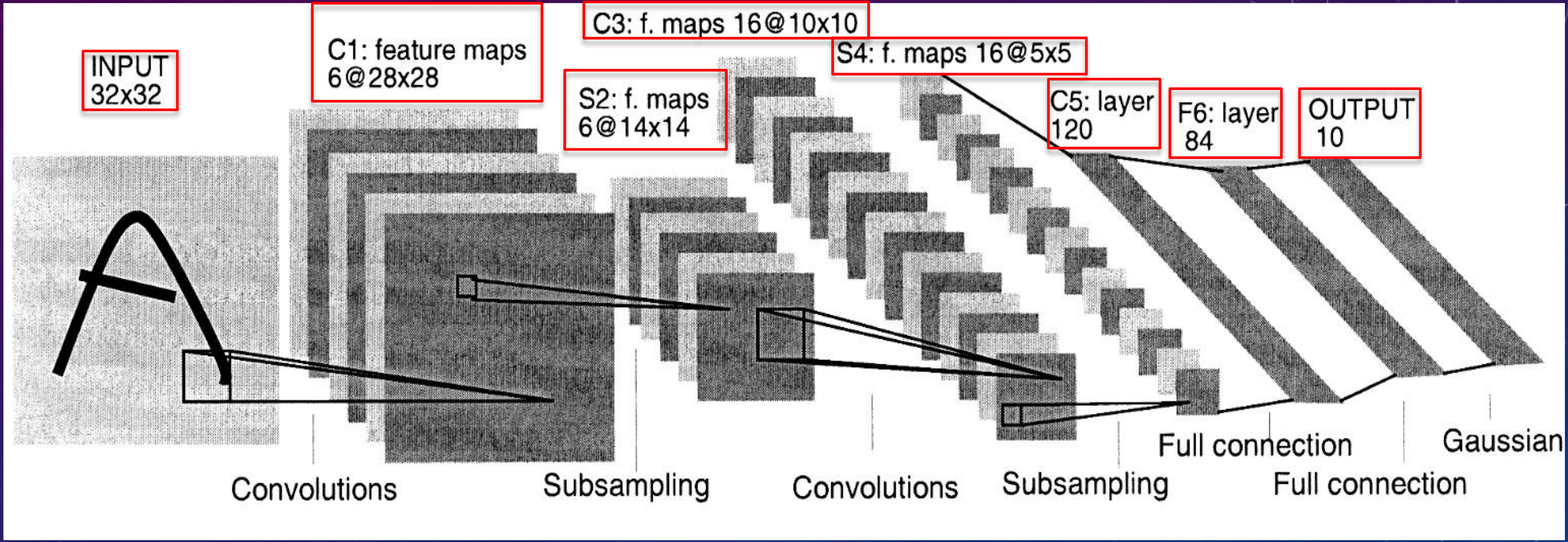
CNN Architectures – Stanford University School of Engineering

https://www.youtube.com/watch?v=DAOcjicFr1Y&list=PL3FW7Lu3i5JvHM8ljYj-zLfQRF3EO8sYv&index=9 [1:17:39]

CNN Architectures – Stanford University School of Engineering

LeNet	(02:47 – 03:15)
AlexNet	(03:16 – 15:29)
VGGNet	(15:30 – 28:37)
GoogLeNet	(28:37 – 47:23)
ResNet	(47:24 – 1:02:33)

LeNet


INPUT
32x32
C1: feature maps
6@28x28
S2: f. maps
6@14x14
C3: f. maps 16@10x10
S4: f. maps 16@5x5
C5: layer
120
F6: layer
84
OUTPUT
10
Convolutions
Subsampling
Convolutions
Subsampling
Full connection
Full connection
Gaussian

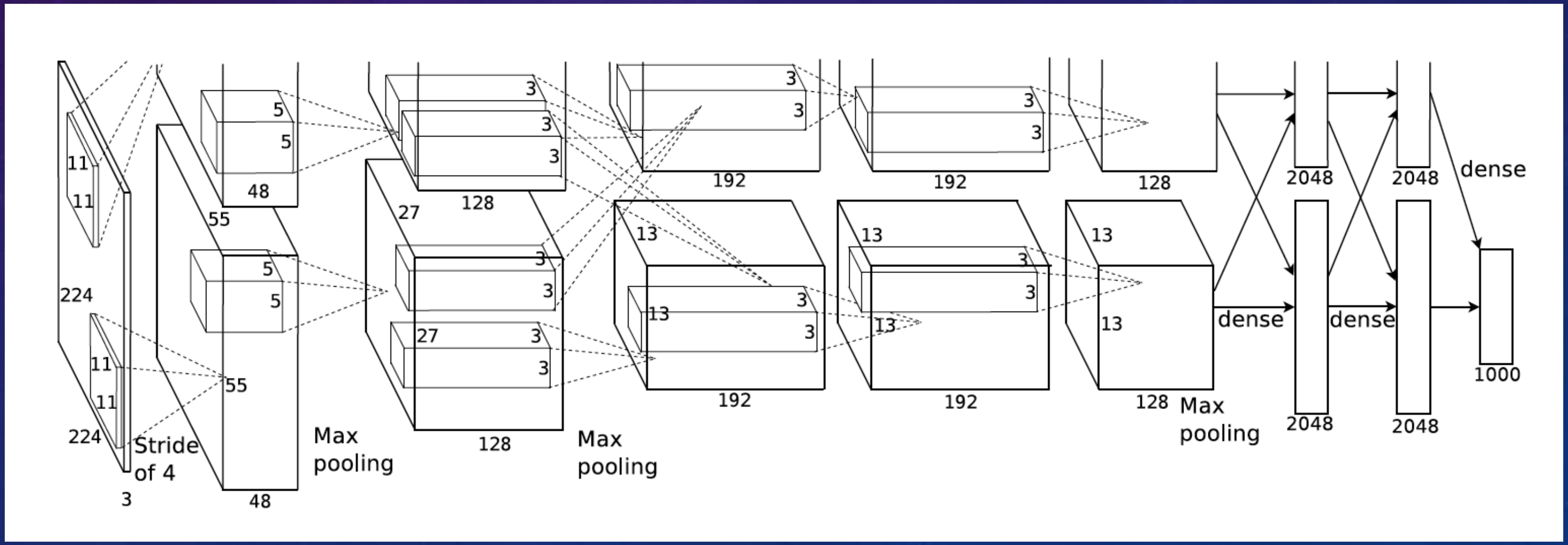
Example 3.1 Calculate Parameters

- Given a input image sized in 32×32×1, i.e., $width \times height \times channel$.
- With a convolutional kernel at 5×5×6 in "C1" layer, please compute the number of the training parameters while convolving the input image.
- In this case, we can know how many parameters need to be updated during training.

Layer Name	Input W×H×D	Kernel W×H×D/S	Output W×H×D	Params
C1: conv2d	32×32×1	5×5×6	28×28×6	1×5×5×6+6=156 weights biases
S2: pool/2	28×28×6	2×2/2	14×14×6	0
C3: conv2d	14×14×6	5×5×16	10×10×16	6×5×5×16+16 =2,416
S4: pool/2	10×10×16	2×2/2	5×5×16	0
C5: conv2d	5×5×16	5×5×120	1×1×120	16×5×5×120+120 =48,120
F6: conv2d	1×1×120	1×1×84	1×1×84	120×1×1×84+84 =10,164
F7: conv2d	1×1×84	1×1×10	1×1×10	84×1×1×10+10 =850
			Total	61,706

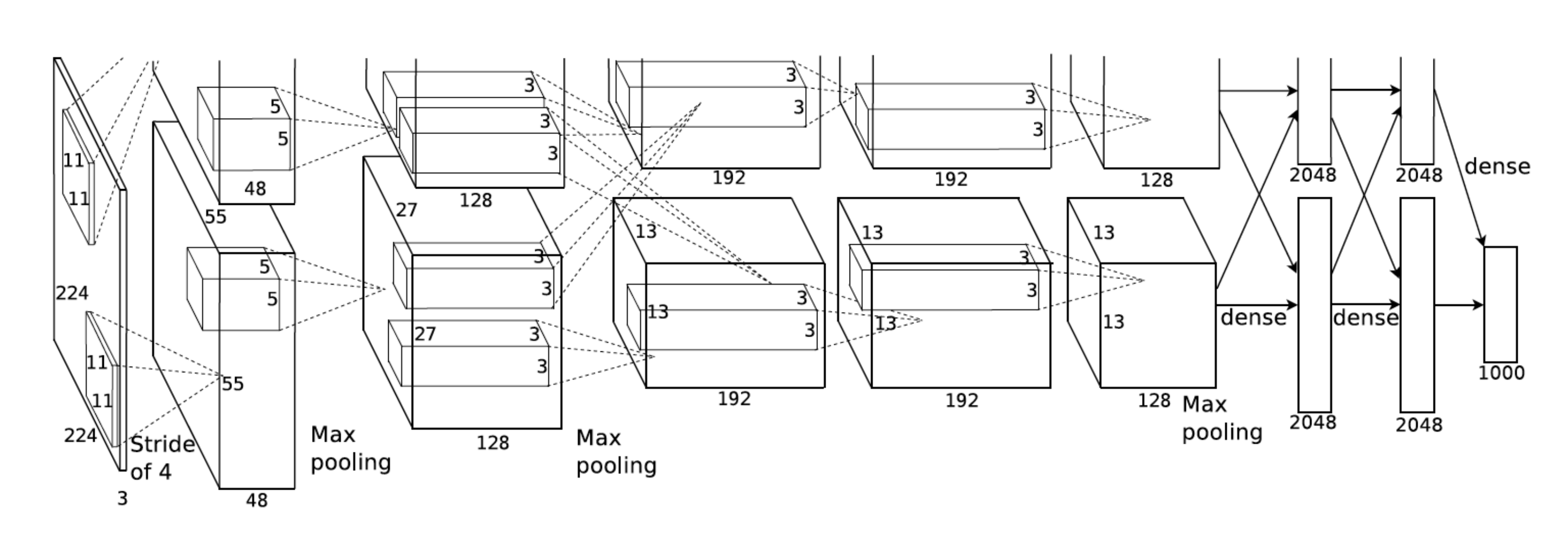
AlexNet

- Use Relu instead of Tanh to add non-linearity. It accelerates the speed by 6 times at the same accuracy.
- Use dropout instead of regularization to deal with overfitting. However the training time is doubled with the dropout rate of 0.5.
- Overlap pooling to reduce the size of network. It reduces the top-1 and top-5 error rates by 0.4% and 0.3%, repectively.



AlexNet

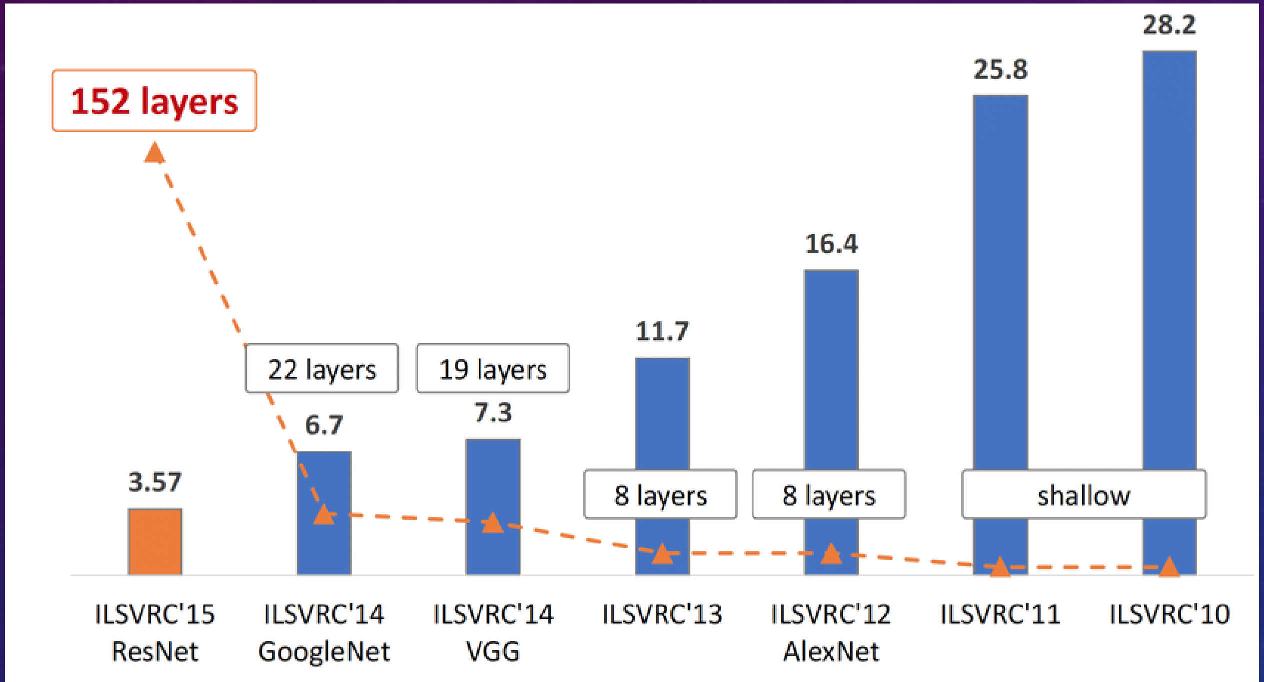
- Copy convolution layers into different GPUs; Distribute the fully connected layers into different GPUs.
- Feed one batch of training data into convolutional layers for every GPU (Data Parallel).
- Feed the results of convolutional layers into the distributed fully connected layers batch by batch (Model Parallel) When the last step is done for every GPU. Backpropogate gradients batch by batch and synchronize the weights of the convolutional layers.

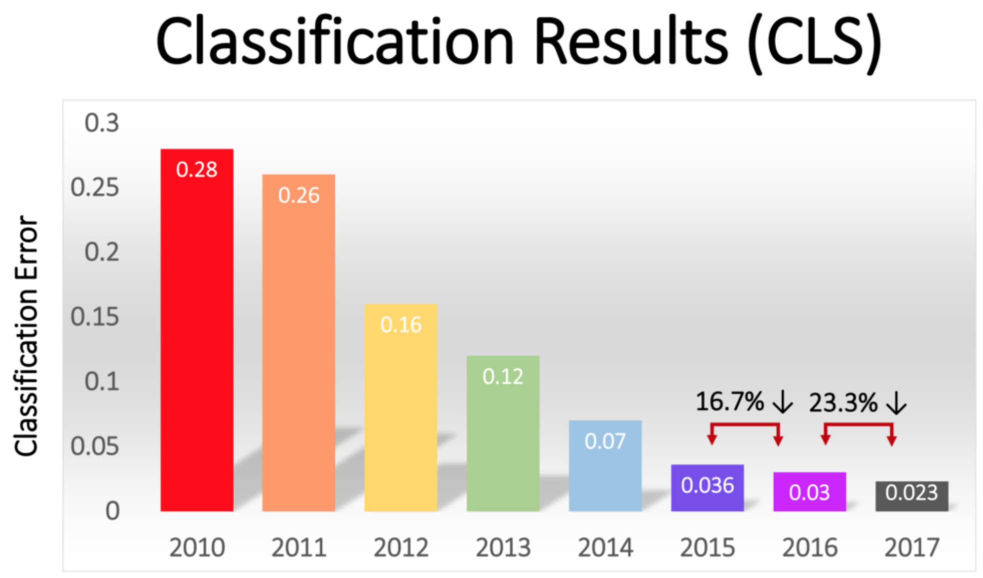


AlexNet Parameters

AlexNet Network - Structural Details													
Input			Output			Layer	Stride	Pad	Kernel size		in	out	# of Param
227	227	3	55	55	96	conv1	4	0	11	11	3	96	34944
55	55	96	27	27	96	maxpool1	2	0	3	3	96	96	0
27	27	96	27	27	256	conv2	1	2	5	5	96	256	614656
27	27	256	13	13	256	maxpool2	2	0	3	3	256	256	0
13	13	256	13	13	384	conv3	1	1	3	3	256	384	885120
13	13	384	13	13	384	conv4	1	1	3	3	384	384	1327488
13	13	384	13	13	256	conv5	1	1	3	3	384	256	884992
13	13	256	6	6	256	maxpool5	2	0	3	3	256	256	0
						fc6			1	1	9216	4096	37752832
						fc7			1	1	4096	4096	16781312
						fc8			1	1	4096	1000	4097000
Total													62,378,344

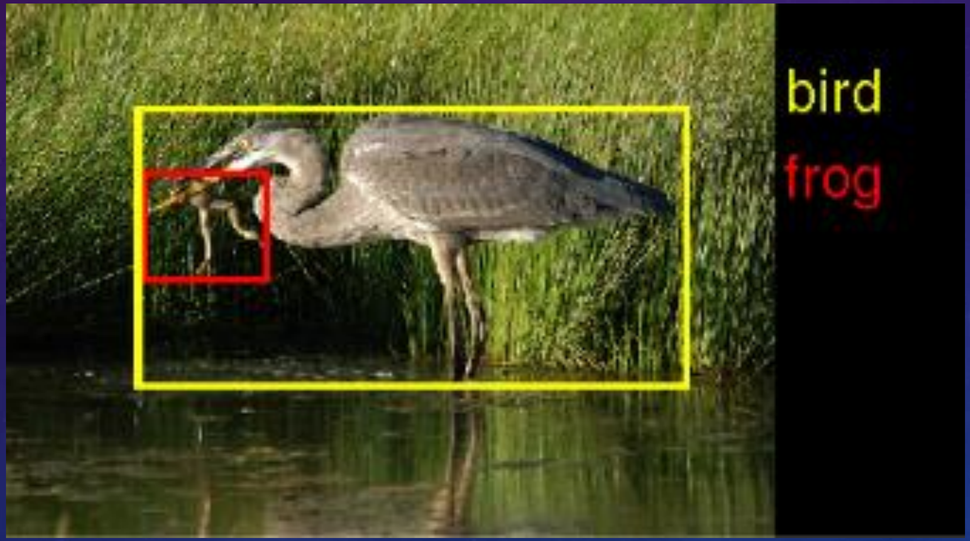
ILSVRC (ImageNet Large Scale Visual Recognition Competition)

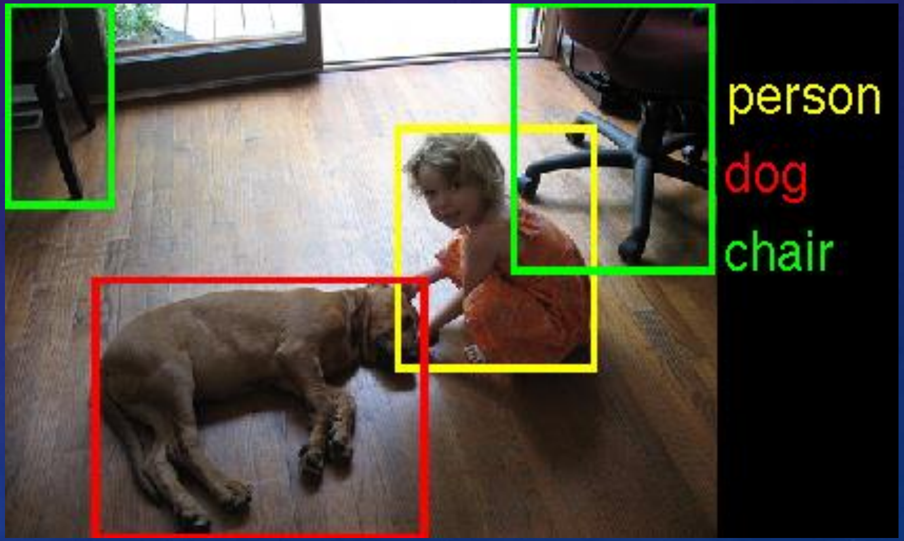




ILSVRC (ImageNet Large Scale Visual Recognition Competition)

- The characteristics of ILSVRC includes:
  - ➢ A detection challenge on fully labeled data for 200 categories of objects
  - ➢ An image classification plus object localization challenge with 1000 categories.

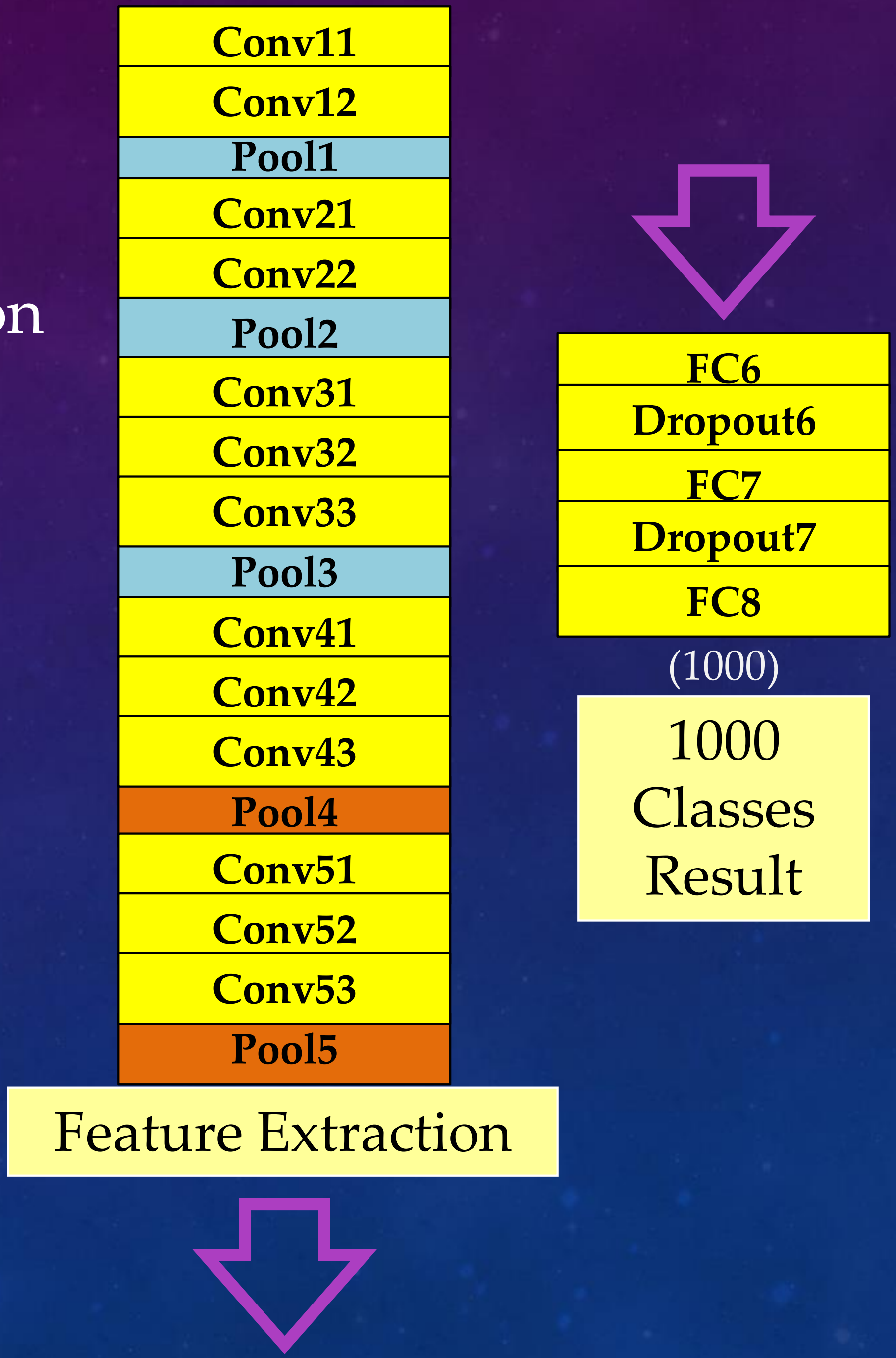




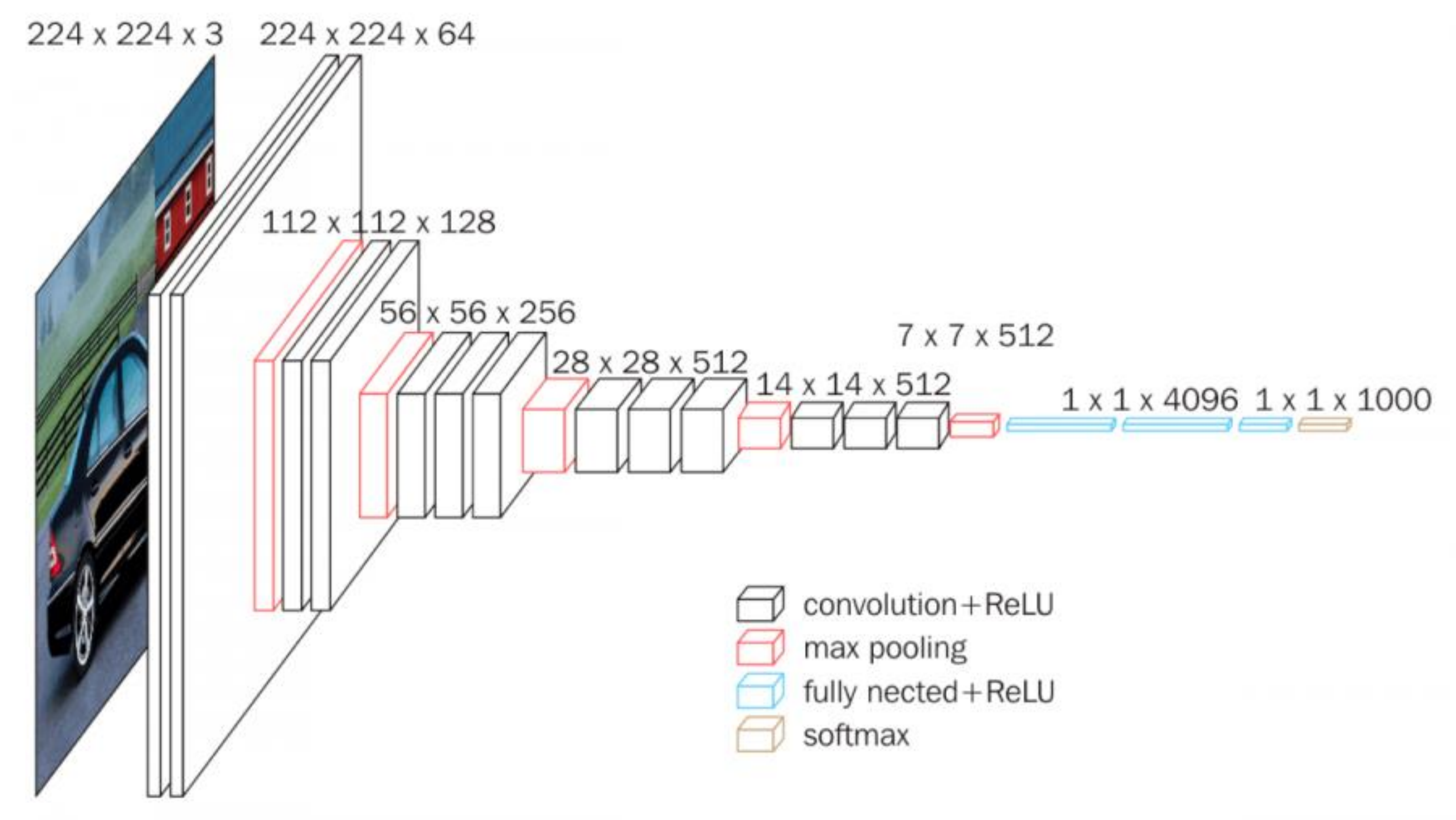
http://image-net.org/challenges/LSVRC/2014/

VGGNet

- Task: 1000 Objects on ImageNet Competition
- Layer
  - Convolutional layer
  - Max pooling layer
  - Dropout layer
  - Fully connected layer



ILSVRC-2014 VGG Model



VGG Parameters

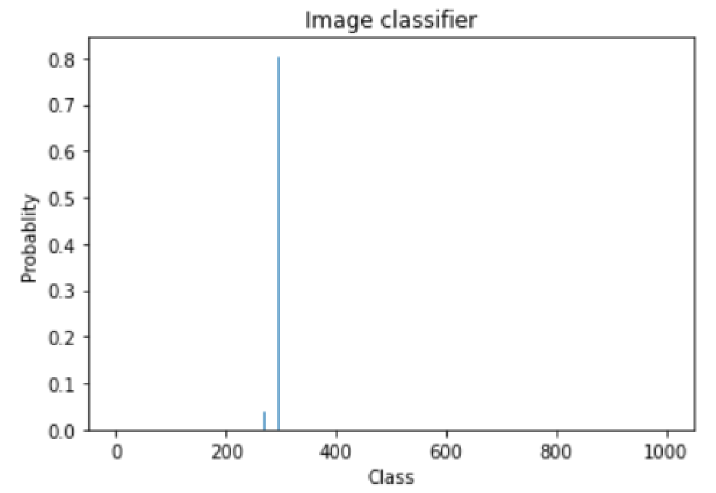
Layer (type)	Output Shape	Param #	Connected to
input_1 (InputLayer)	(None, 224, 224, 3)	0	
block1_conv1 (Convolution2D)	(None, 224, 224, 64)	1792	input_1[0][0]
block1_conv2 (Convolution2D)	(None, 224, 224, 64)	36928	block1_conv1[0][0]
block1_pool (MaxPooling2D)	(None, 112, 112, 64)	0	block1_conv2[0][0]
block2_conv1 (Convolution2D)	(None, 112, 112, 128)	73856	block1_pool[0][0]
block2_conv2 (Convolution2D)	(None, 112, 112, 128)	147584	block2_conv1[0][0]
block2_pool (MaxPooling2D)	(None, 56, 56, 128)	0	block2_conv2[0][0]
block3_conv1 (Convolution2D)	(None, 56, 56, 256)	295168	block2_pool[0][0]
block3_conv2 (Convolution2D)	(None, 56, 56, 256)	590080	block3_conv1[0][0]
block3_conv3 (Convolution2D)	(None, 56, 56, 256)	590080	block3_conv2[0][0]
block3_pool (MaxPooling2D)	(None, 28, 28, 256)	0	block3_conv3[0][0]
block4_conv1 (Convolution2D)	(None, 28, 28, 512)	1180160	block3_pool[0][0]
block4_conv2 (Convolution2D)	(None, 28, 28, 512)	2359808	block4_conv1[0][0]
block4_conv3 (Convolution2D)	(None, 28, 28, 512)	2359808	block4_conv2[0][0]
block4_pool (MaxPooling2D)	(None, 14, 14, 512)	0	block4_conv3[0][0]
block5_conv1 (Convolution2D)	(None, 14, 14, 512)	2359808	block4_pool[0][0]
block5_conv2 (Convolution2D)	(None, 14, 14, 512)	2359808	block5_conv1[0][0]
block5_conv3 (Convolution2D)	(None, 14, 14, 512)	2359808	block5_conv2[0][0]
block5_pool (MaxPooling2D)	(None, 7, 7, 512)	0	block5_conv3[0][0]
flatten (Flatten)	(None, 25088)	0	block5_pool[0][0]
fc1 (Dense)	(None, 4096)	102764544	flatten[0][0]
fc2 (Dense)	(None, 4096)	16781312	fc1[0][0]
predictions (Dense)	(None, 1000)	4097000	fc2[0][0]

Total params: 138357544

Example 3.2 Use VGGNet pretrained on ImageNet

- Please download the "3-2_VGGNet_ImageNet.zip" from the Moodle and unzip it.
- Follow the instruction in "How_to_Use_Colab.pdf" to upload the .ipynb file to Colab.
- Upload the "3-2_VGGNet_ImageNet.ipynb" and "imagenet1000_clsidx_to_labels.txt" to the Google Colab.
- Compare the probability of the images downloaded from Internet.

Original image: ice_bear.jpg



Probability of the classes

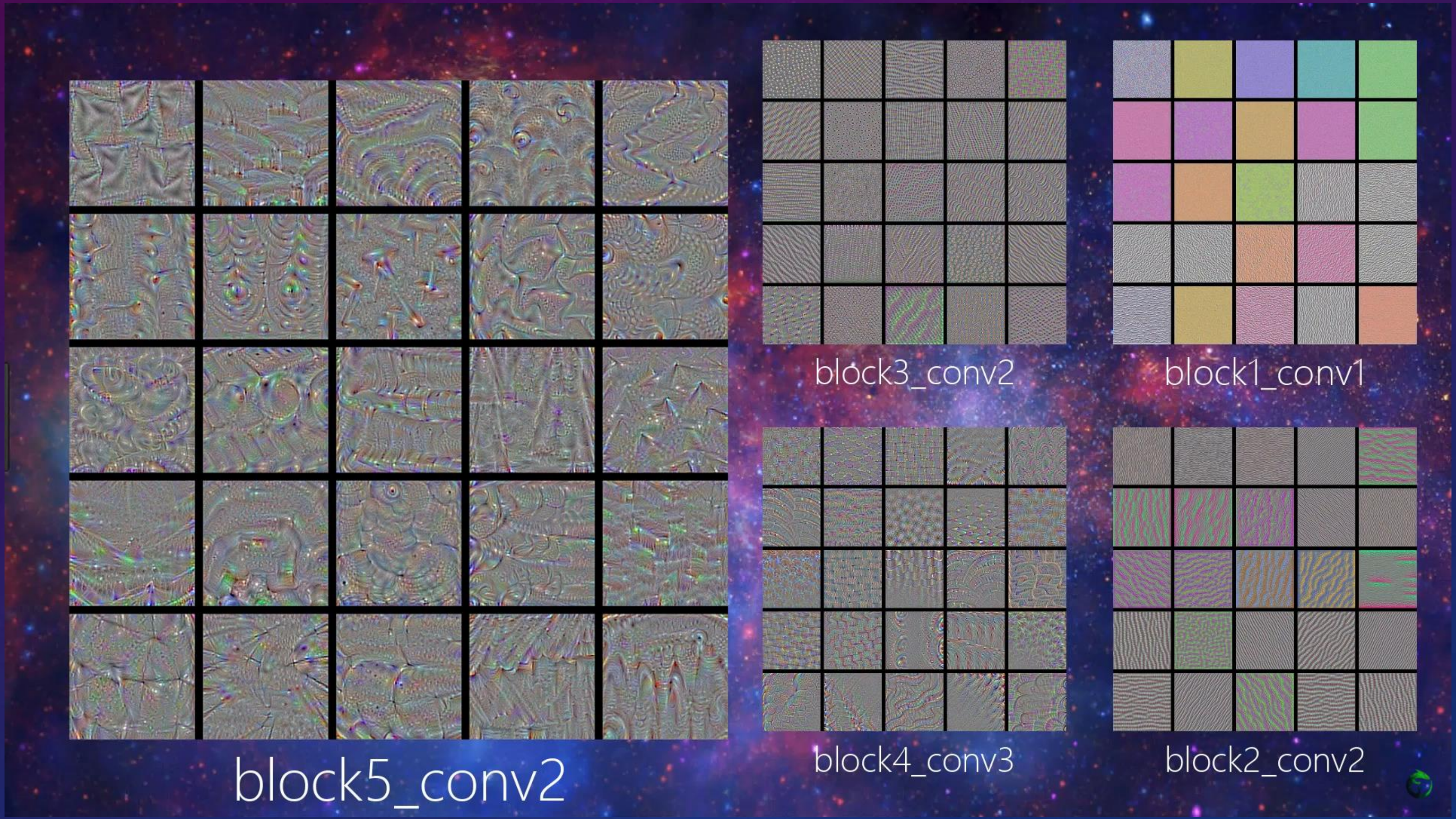
```
TOP_1
Probablity:0.804244875907898
Predicted: 'ice bear

TOP_2
Probablity:0.14214567840099335
Predicted: 'Arctic fox

TOP_3
Probablity:0.03769978880882263
Predicted: 'white wolf
```

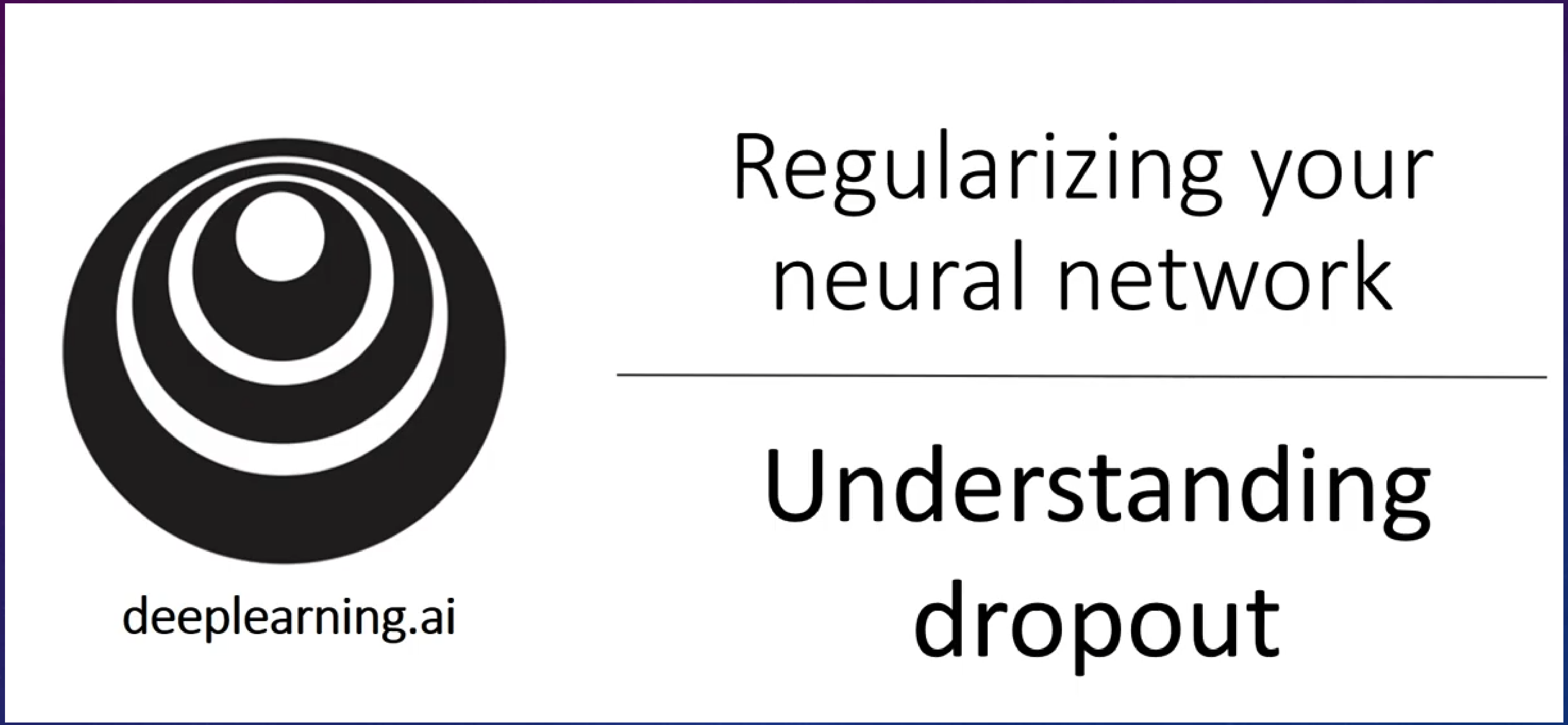
Predicted class : ice bear

What does Filter Learn?



https://www.youtube.com/watch?v=cNBBNAxC8l4 [6:08]

Training Techniques - Understanding Dropout

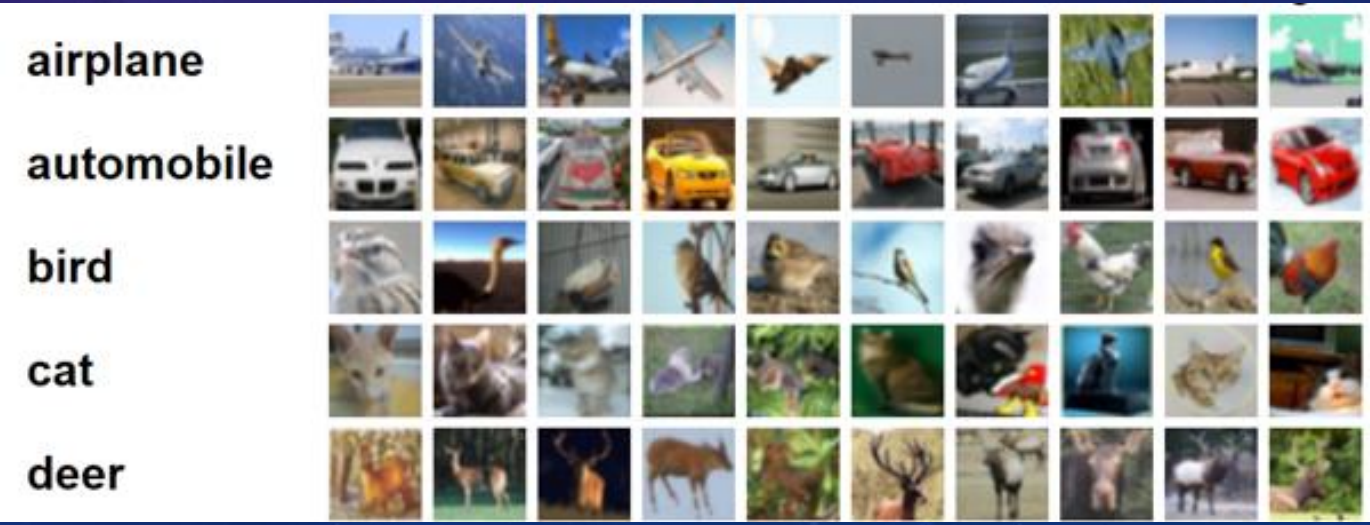


https://www.youtube.com/watch?v=ARq74QuavAo [7:04]

Example 3.3 Train VGGNet on CIFAR100

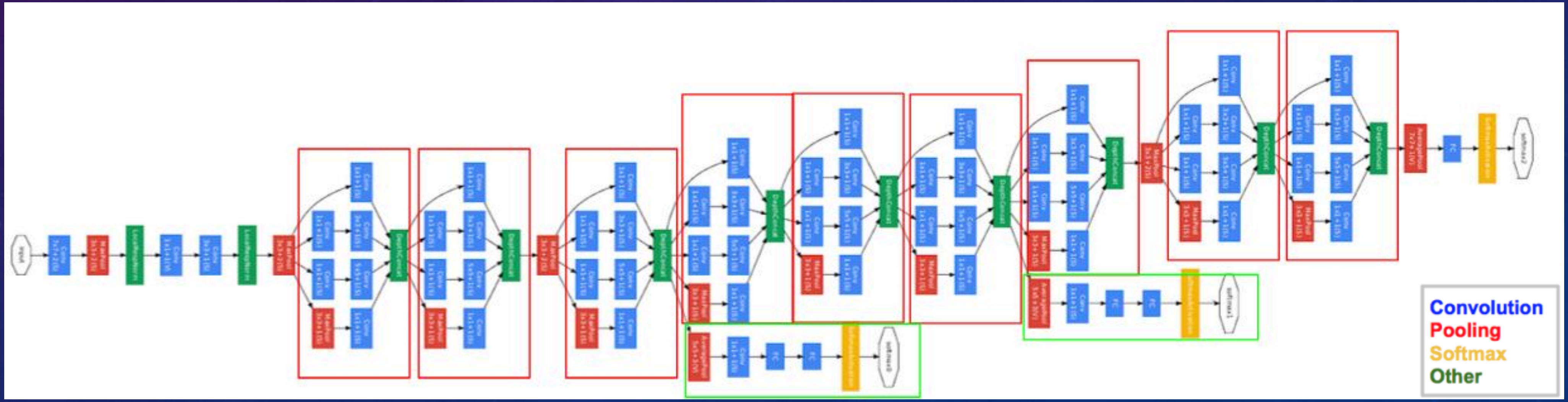
- Please download the “3-3_VGGNet_CIFAR100.zip” from the Moodle and unzip it.
- Upload the “3-3_VGGNet_CIFAR100.ipynb” to the Google Colab.
- Use the VGG-16 model pretrained on ImageNet to train the CIFAR-100 dataset with the following parameters: input size = 32 (color image), batch size = 256, learning rate=0.001.
- Please search for the images of following categories: apple, dolphin and dog.
- Given these images as input to your trained model, what are the probabilities in the output layer?

CIFAR 100 dataset contains 60000 images and consists of 100 class



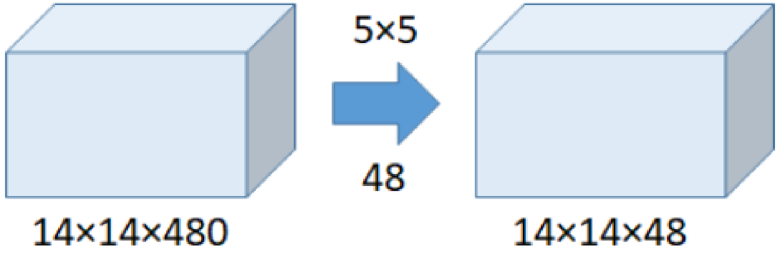
GoogleNet

- 1×1 Convolution at the middle of the network
  - ➢ In GoogleNet, 1×1 convolution is used as a dimension reduction module to reduce the computation.
- Global average pooling is used at the end of the network instead of using fully connected layers
- Inception module



GoogleNet

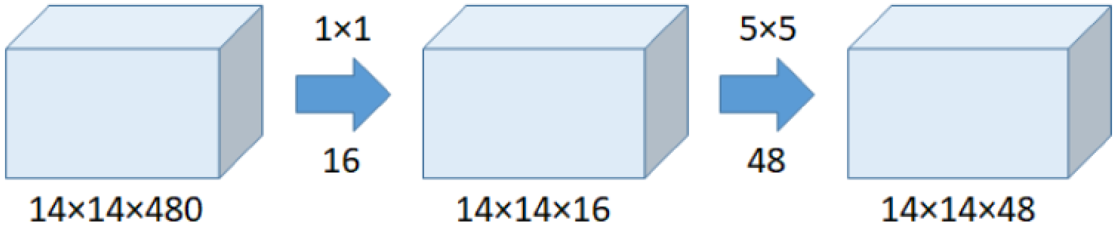
- 1×1 Convolution at the middle of the network
  - ➢ In GoogleNet, 1×1 convolution is used as a dimension reduction module to reduce the computation.



Without the Use of 1×1 Convolution

Number of operations = $(14\times14\times48)\times(5\times5\times480) = 112.9M$

With the use of 1×1 convolution:



With the Use of 1×1 Convolution

Number of operations for $1\times1 = (14\times14\times16)\times(1\times1\times480) = 1.5M$

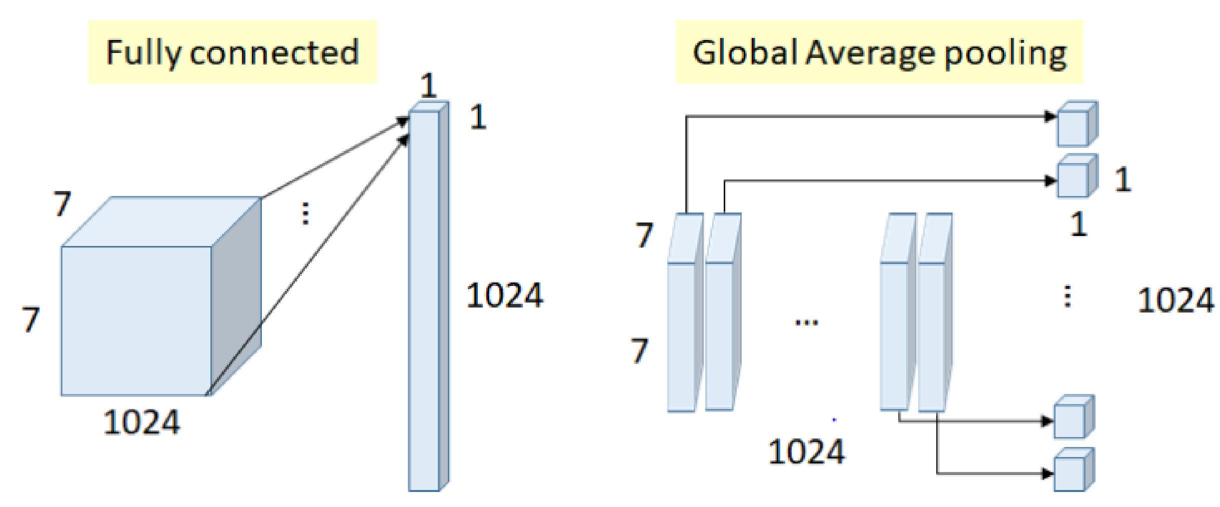
Number of operations for $5\times5 = (14\times14\times48)\times(5\times5\times16) = 3.8M$

Total number of operations = 1.5M + 3.8M = 5.3M

which is much much smaller than 112.9M !!!!!!!!!!!!!!!

GoogleNet

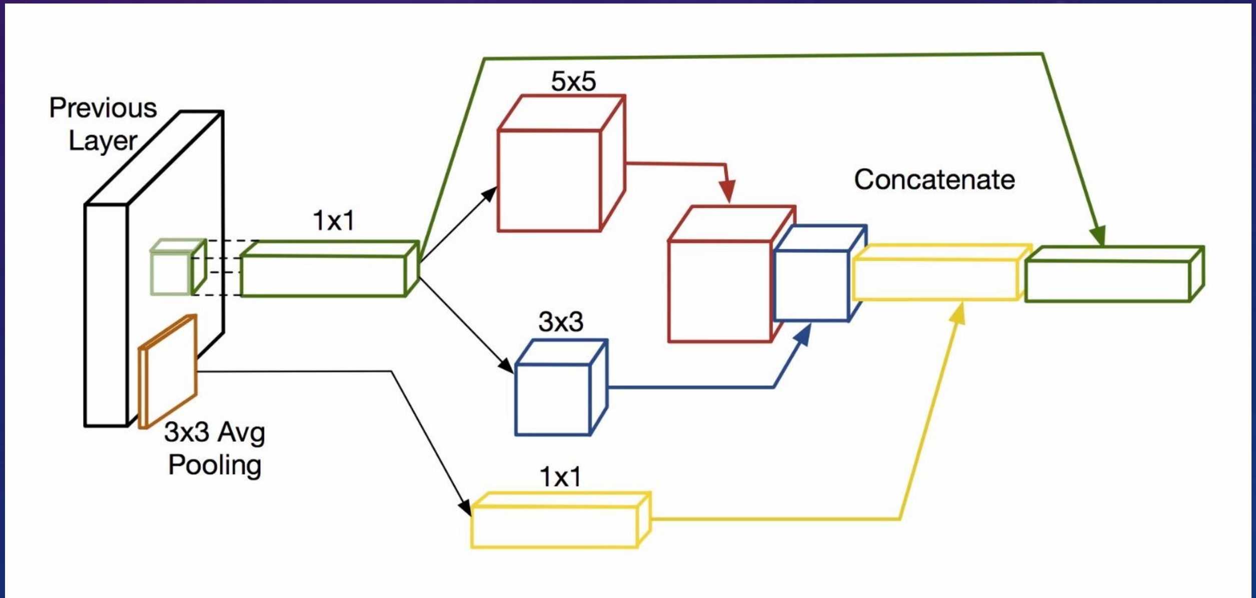
- Number of weights (connections) above = 7×7×1024×1024 = 51.3M
- In GoogleNet, global average pooling is used nearly at the end of network by averaging each feature map from 7×7 to 1×1, as in the figure above.
- Number of weights = 0
- And authors found that a move from FC layers to average pooling improved the top-1 accuracy by about 0.6%.



Fully Connected Layer VS Global Average Pooling

GoogleNet

- 1×1 conv, 3×3 conv, 5×5 conv, and 3×3 max pooling are done altogether for the previous input, and stack together again at output. When image's coming in, different sizes of convolutions as well as max pooling are tried. Then different kinds of features are extracted.
- After that, all feature maps at different paths are concatenated together as the input of the next module.

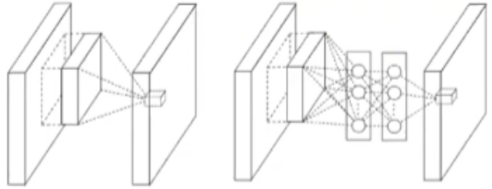


GoogleNet - Network In Network

Boulder

Network In Network

1) MLP Convolution Layers (1x1 Convolutions)



(a) Linear convolution layer (b) Mlpconv layer

$$f^1_{i,j,k_1} = \max({w^1_{k_1}}^T x_{i,j} + b_{k_1}, 0).$$

Input patch centered at location (i, j)

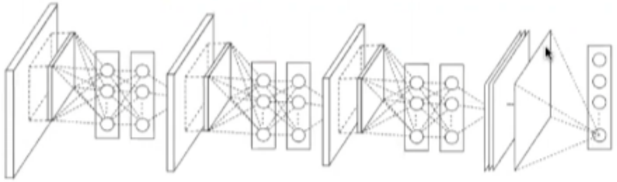
ReLU activation function

$$\vdots$$

$$f^n_{i,j,k_n} = \max({w^n_{k_n}}^T f^{n-1}_{i,j} + b_{k_n}, 0).$$

k_ℓ is used to index the channels of the feature map

2) Global Average Pooling

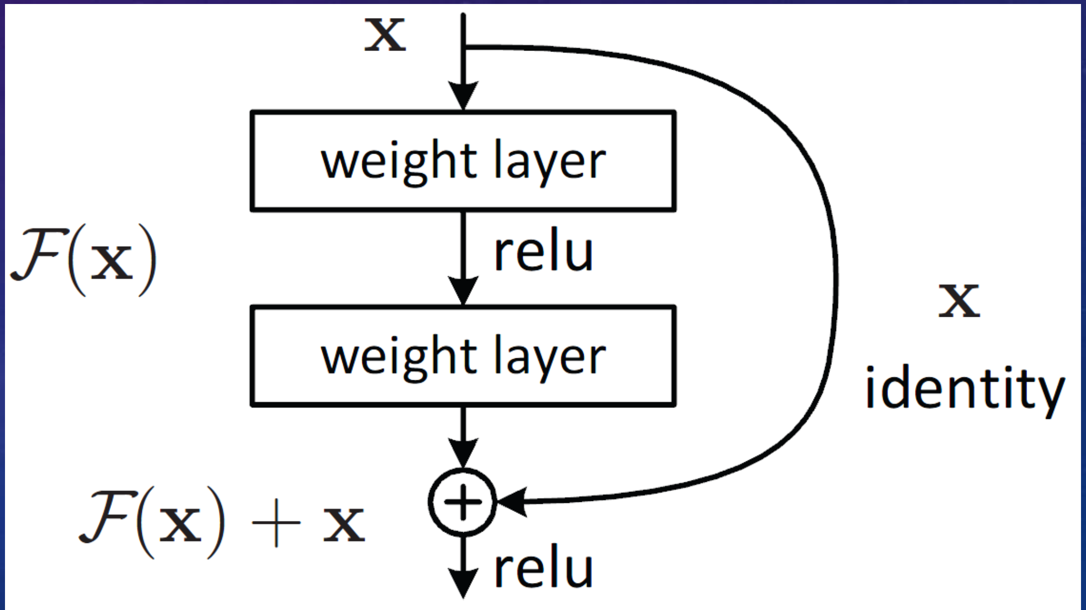


Lin, Min, Qiang Chen, and Shuicheng Yan. "Network in network." *arXiv preprint arXiv:1312.4400* (2013).

https://www.youtube.com/watch?v=XUB_L7hxSeU [8:37]

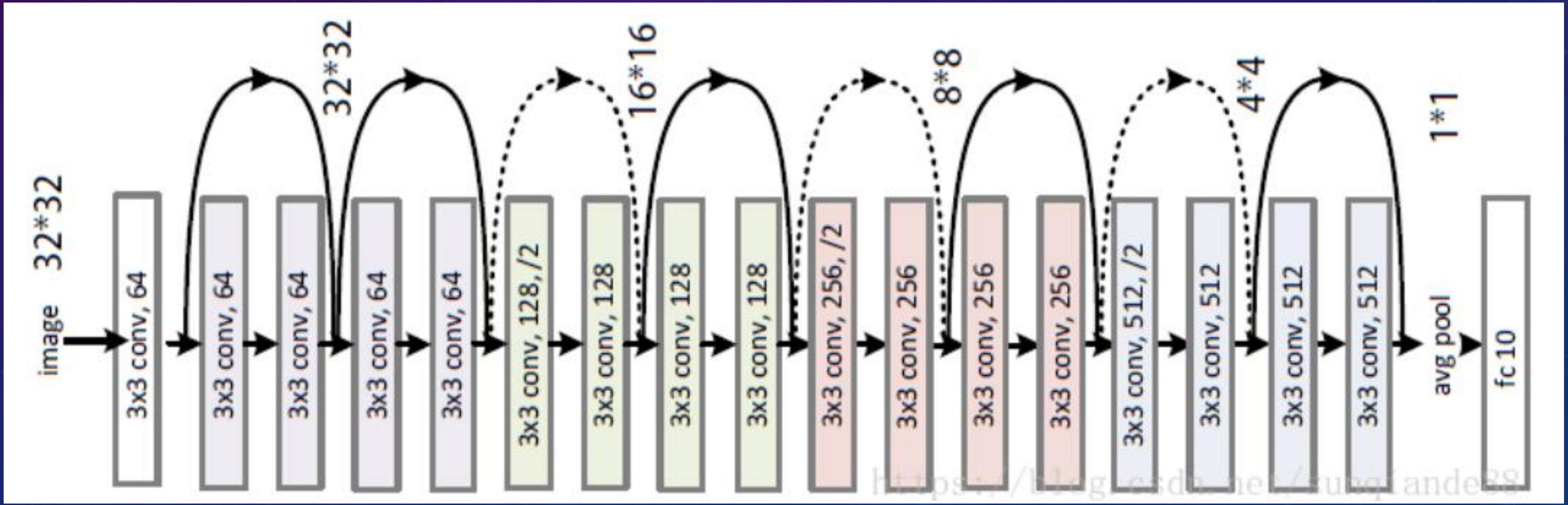
ResNet

- Since AlexNet, the state-of-the-art CNN architecture is going deeper and deeper.
- However, increasing network depth does not work by simply stacking layers together. Deep networks are hard to train because of the notorious vanishing gradient problem
- The core idea of ResNet is introducing a so-called "identity shortcut connection" that skips one or more layers, as shown in the following figure:



ResNet18

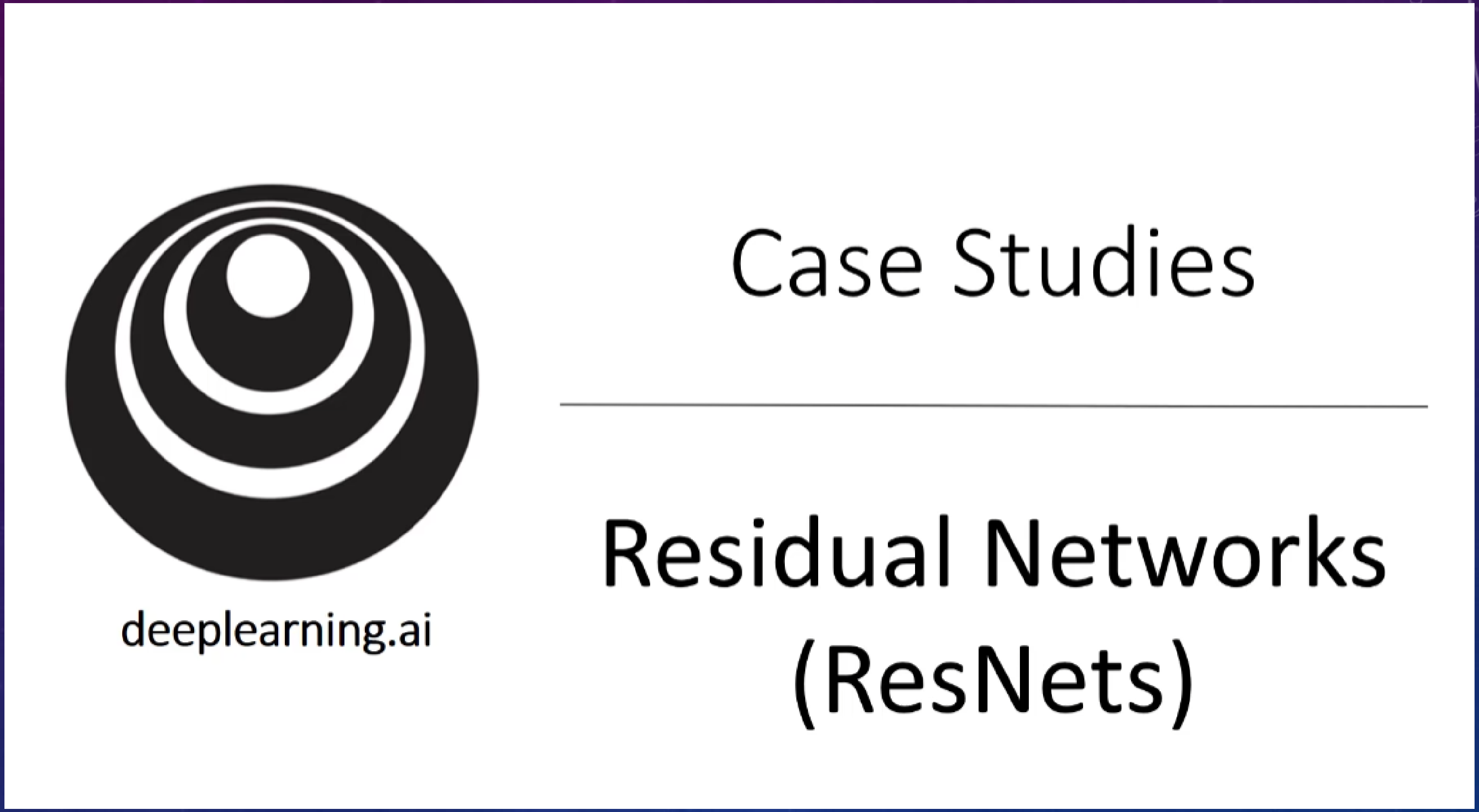
- Since AlexNet, the state-of-the-art CNN architecture is going deeper and deeper.
- Increasing network depth does not work by simply stacking layers together. Deep networks are hard to train because of the notorious vanishing gradient problem
- The core idea of ResNet is introducing a so-called "identity shortcut connection" that skips one or more layers, as shown in the following figure:



ResNet18 Parameters

ResNet18 - Structural Details														
#	Input Image			output			Layer	Stride	Pad	Kernel		in	out	Param
1	227	227	3	112	112	64	conv1	2	1	7	7	3	64	9472
	112	112	64	56	56	64	maxpool	2	0.5	3	3	64	64	0
2	56	56	64	56	56	64	conv2-1	1	1	3	3	64	64	36928
3	56	56	64	56	56	64	conv2-2	1	1	3	3	64	64	36928
4	56	56	64	56	56	64	conv2-3	1	1	3	3	64	64	36928
5	56	56	64	56	56	64	conv2-4	1	1	3	3	64	64	36928
6	56	56	64	28	28	128	conv3-1	2	0.5	3	3	64	128	73856
7	28	28	128	28	28	128	conv3-2	1	1	3	3	128	128	147584
8	28	28	128	28	28	128	conv3-3	1	1	3	3	128	128	147584
9	28	28	128	28	28	128	conv3-4	1	1	3	3	128	128	147584
10	28	28	128	14	14	256	conv4-1	2	0.5	3	3	128	256	295168
11	14	14	256	14	14	256	conv4-2	1	1	3	3	256	256	590080
12	14	14	256	14	14	256	conv4-3	1	1	3	3	256	256	590080
13	14	14	256	14	14	256	conv4-4	1	1	3	3	256	256	590080
14	14	14	256	7	7	512	conv5-1	2	0.5	3	3	256	512	1180160
15	7	7	512	7	7	512	conv5-2	1	1	3	3	512	512	2359808
16	7	7	512	7	7	512	conv5-3	1	1	3	3	512	512	2359808
17	7	7	512	7	7	512	conv5-4	1	1	3	3	512	512	2359808
	7	7	512	1	1	512	avg pool	7	0	7	7	512	512	0
18	1	1	512	1	1	1000	fc					512	1000	513000
Total														11,511,784

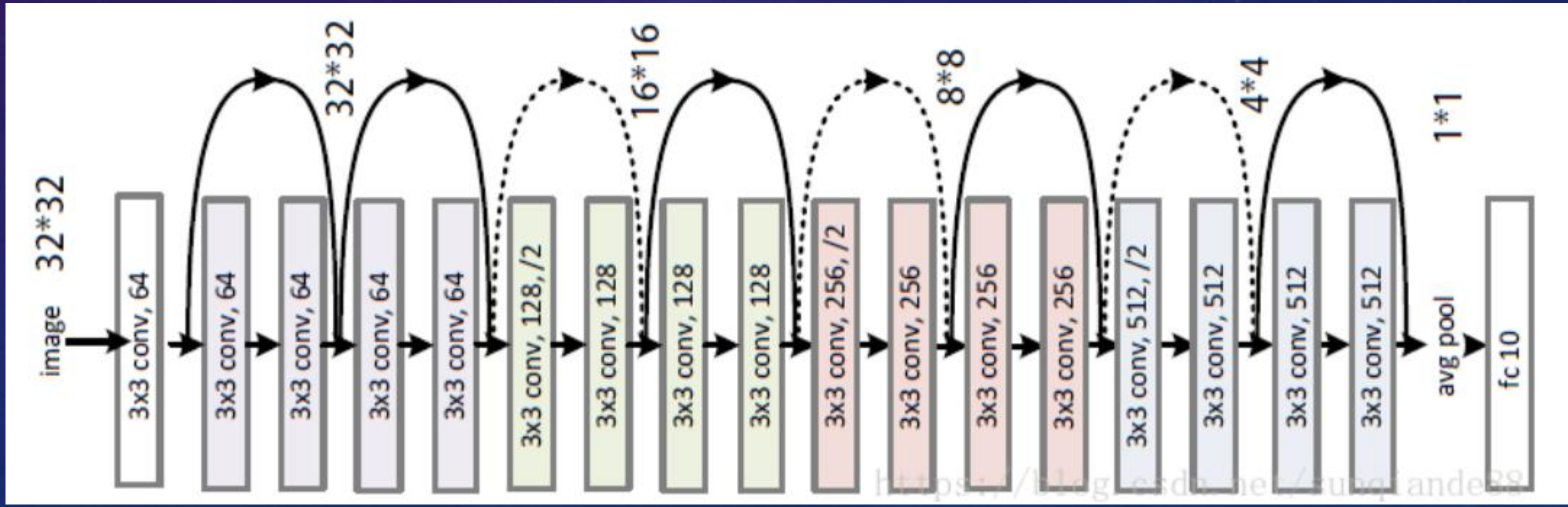
ResNet



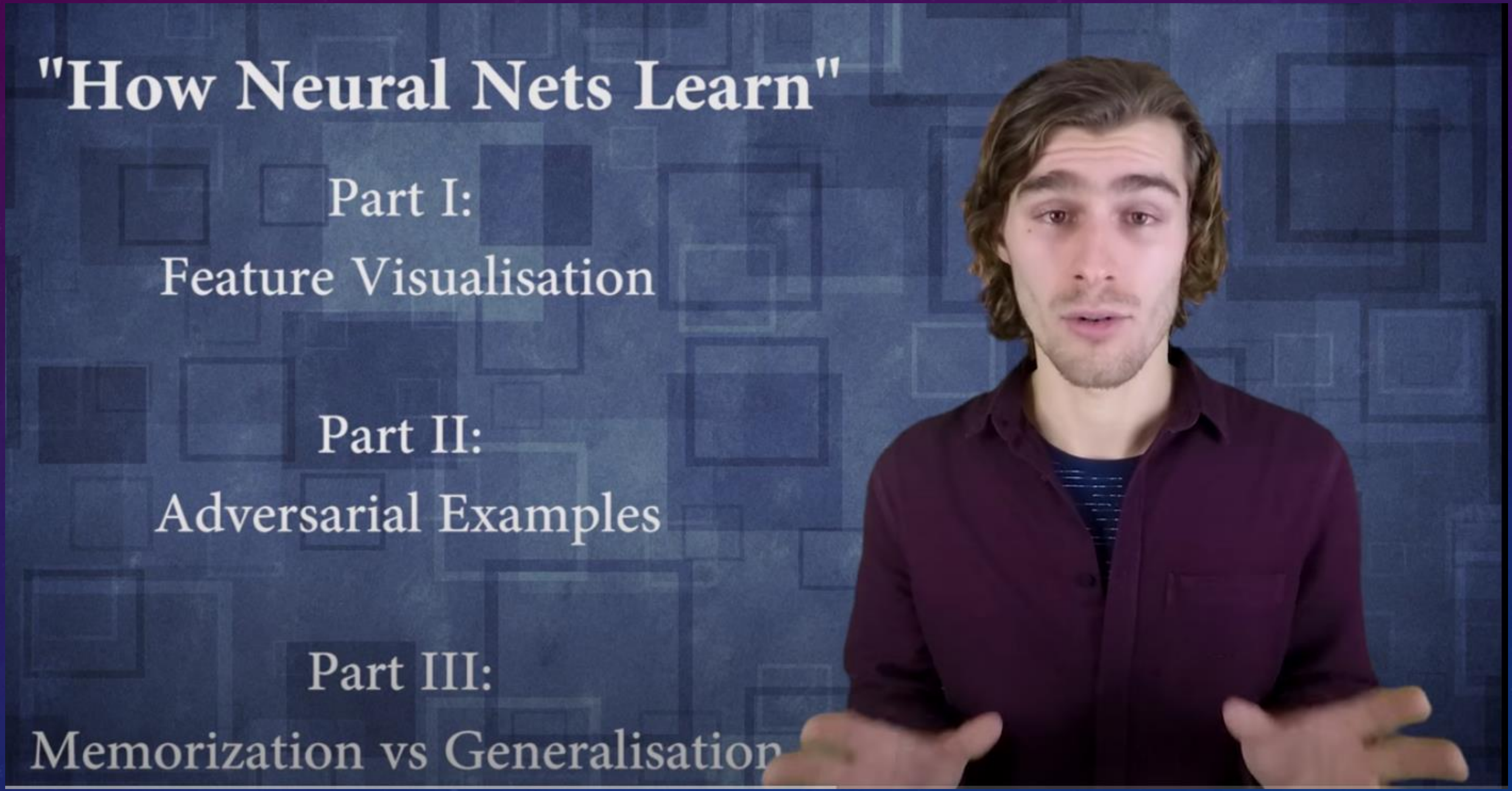
https://www.youtube.com/watch?v=ZILIbUvp5lk [07:07]

Example 3.4 Train ResNet on CIFAR-100

- Please download the "3-4_ResNet_CIFAR100.zip" from the Moodle and unzip it.
- Upload the "3-4_ResNet_CIFAR100.ipynb" to the Google Colab.
- Use the ResNet model pretrained on ImageNet to train the CIFAR-100 dataset with the following parameters: input size = 32 (color image), batch size = 64, learning rate=0.001.
- Please use the same images as of the previous examples of apple, dolphin and dog.
- Given these images as input to your trained model, what are the probabilities in the output layer?



How neural networks learn - Part I: Feature Visualization



https://www.youtube.com/watch?v=McgxRxi2Jqo&feature=youtu.be&ab_channel=ArxivInsights [0:00 - 08:35]

Example 3.5 Feature Map Visualization

- Please download the "3-5_Feature_map_visualization.zip" from the Moodle, which is built on the VGG-16 model pretrained on the ImageNet.
- Upload the "3-5_Feature_map_visualization.ipynb" and "imagenet1000_clsidx_to_labels.txt" to the Google Colab.
- Use the cat image as the input, please show the feature maps extracted from the layer-index-5 (red bbox in the right figure below).

Original image: cat.jpg

Feature map: cat_feature_5.jpg

```
Sequential(
  (0): Conv2d(3, 64, kernel_size=(3, 3), stride=(1, 1), padding=(1, 1))
  (1): ReLU(inplace=True)
  (2): Conv2d(64, 64, kernel_size=(3, 3), stride=(1, 1), padding=(1, 1))
  (3): ReLU(inplace=True)
  (4): MaxPool2d(kernel_size=2, stride=2, padding=0, dilation=1, ceil_mode=False)
  (5): Conv2d(64, 128, kernel_size=(3, 3), stride=(1, 1), padding=(1, 1))
  (6): ReLU(inplace=True)
  (7): Conv2d(128, 128, kernel_size=(3, 3), stride=(1, 1), padding=(1, 1))
  (8): ReLU(inplace=True)
  (9): MaxPool2d(kernel_size=2, stride=2, padding=0, dilation=1, ceil_mode=False)
```

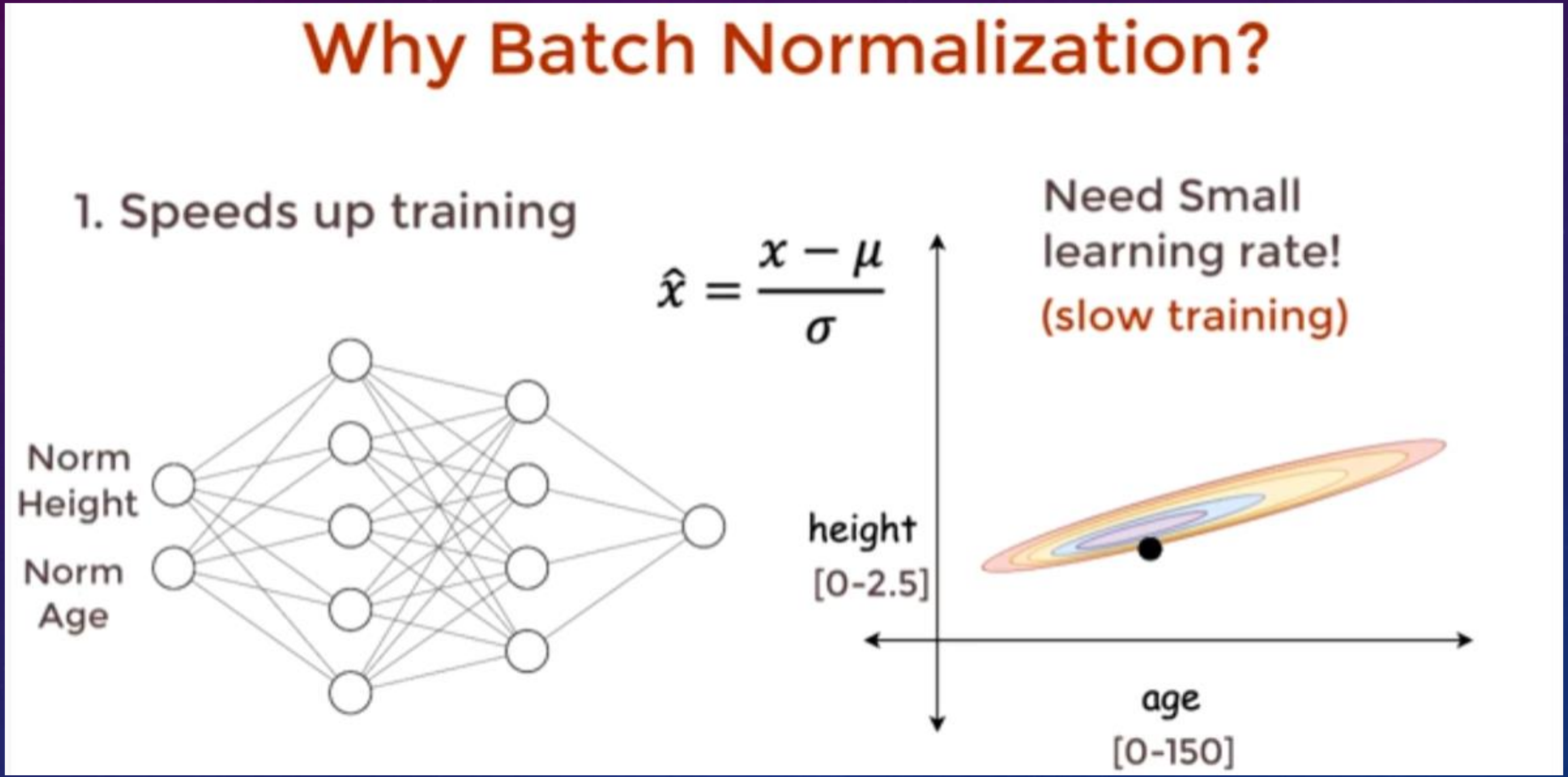
Part of VGG-16 structure.

Example 3.6 Feature Comparison

- Please download the "3-6_Feature_Comparison.zip" from the Moodle, which is built on the VGG-16 model pretrained on the ImageNet.
- Upload the "3-6_Feature_Compare.ipynb" to the Google Colab.
- Given two dog images as inputs, the layer index-5 of VGG-16 is used to extract the feature representations, and the cosine similarity between two representations is used to measure of how similar of the inputs.
- Please show the feature maps specified in Ex.3.5, and the similarity score of two given images.

Cosine similarity = 0.701

Batch Normalization explained



https://www.youtube.com/watch?v=DtEq44FTPM4 [8:48]

Training Neural Networks – Stanford University School of Engineering

https://www.youtube.com/watch?v=wEoyxE0GP2M [1:20:19]

Training Neural Networks – Stanford University School of Engineering

Activation Functions (04:54 – 48:57)

Data Preprocessing (27:34 – 48:57)

Weight Initialization (34:24 – 48:57)

Batch Normalization (48:58 – 1:04:38)

Babysitting the Learning process (1:04:39 – 1:10:15)

Hyperparameter Optimization (1:10:16 – 1:19:49)

Batch Normalization Mathematically

- Batch Normalization manipulates the layer inputs by calculating a batch's mean and variance. The data is then scaled and shifted.
- Therefore, Batch Normalization is a special kind of preprocessing. The mathematical procedure can be seen on the right.

Input: Values of x over a mini-batch: $\mathcal{B} = \{x_{1\ldots m}\}$;
Parameters to be learned: γ, β

Output: $\{y_i = \mathrm{BN}_{\gamma,\beta}(x_i)\}$

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

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

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

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

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

Advantages of Batch Normalization

- Improves gradient flow through the network.
- Allows higher learning rates.
- Reduces the strong dependence on initialization.
- Regularizes the model.

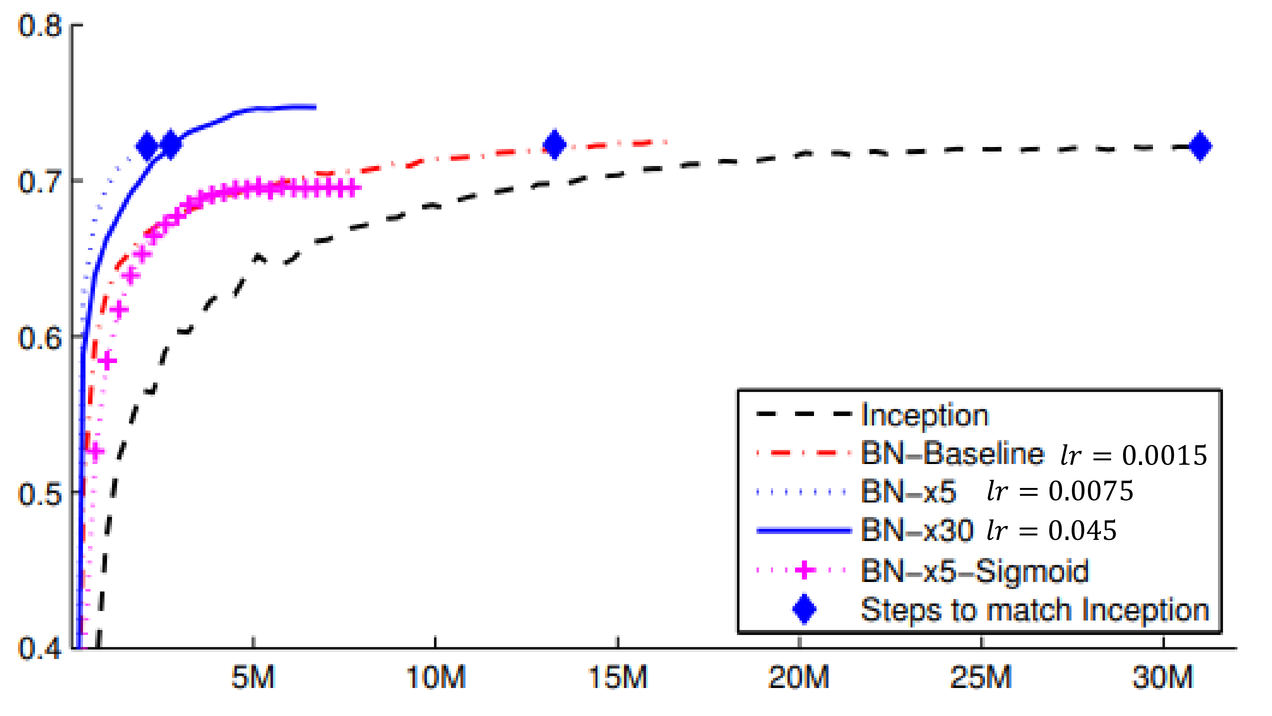
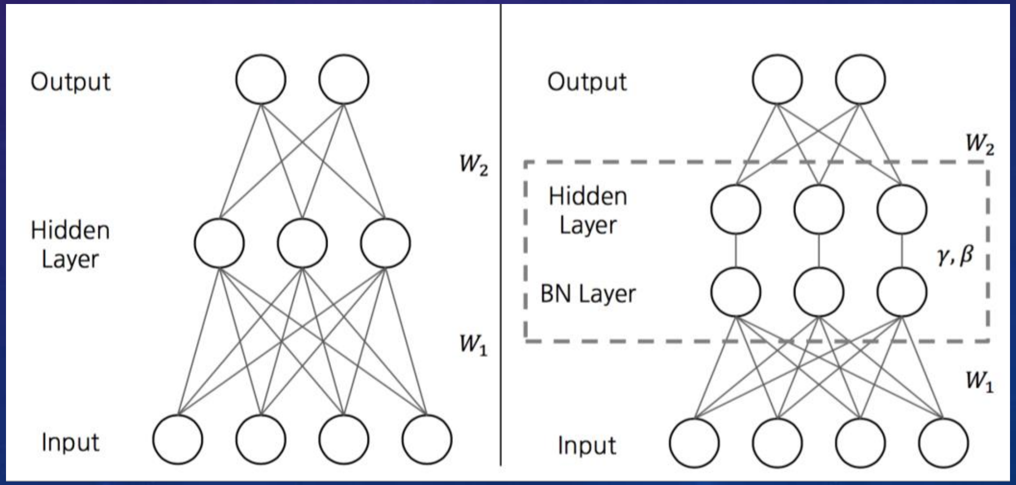


Figure 2. Single crop validation accuracy of Inception and its batch-normalized variants, vs. the number of training steps.

Image from "Batch Normalization: Accelerating Deep Network Training by Reducing Internal Covariate Shift"

Example 3.7 Batch Normalization

- Please download the “3-7_Batch_Normalization.zip” from the Moodle, which is built on the VGG-16 model pretrained on the ImageNet.
- Upload the “3-7_Batch_Normalization.ipynb” to the Google Colab.
- Use the VGG-16 pretrained model with/without batch normalization to retrain the CIFAR-100 dataset with the following parameters: input size = 32 (color image), batch size = 64, learning rate=0.001.
- Please show the accuracies of first five epochs, and compare the accuracy with and without using batch normalization.



https://meet.google.com/qex-scyh-hvz