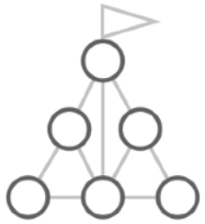


Deep Learning - Convolutional Neural Network (CNN)

Itthi Chatnuntawech

Fully connected layer
(Dense)

Convolutional layer

Conv1D, 2D, 3D, …
separable Conv

Optimizer

SGD
Adam
RMSprop

Pooling layer

max-pooling
average-pooling

Deep Learning

Evaluation metric

accuracy
F1-score
AUC
confusion matrix

Activation function

sigmoid
softmax
ESP (swish)
ReLU

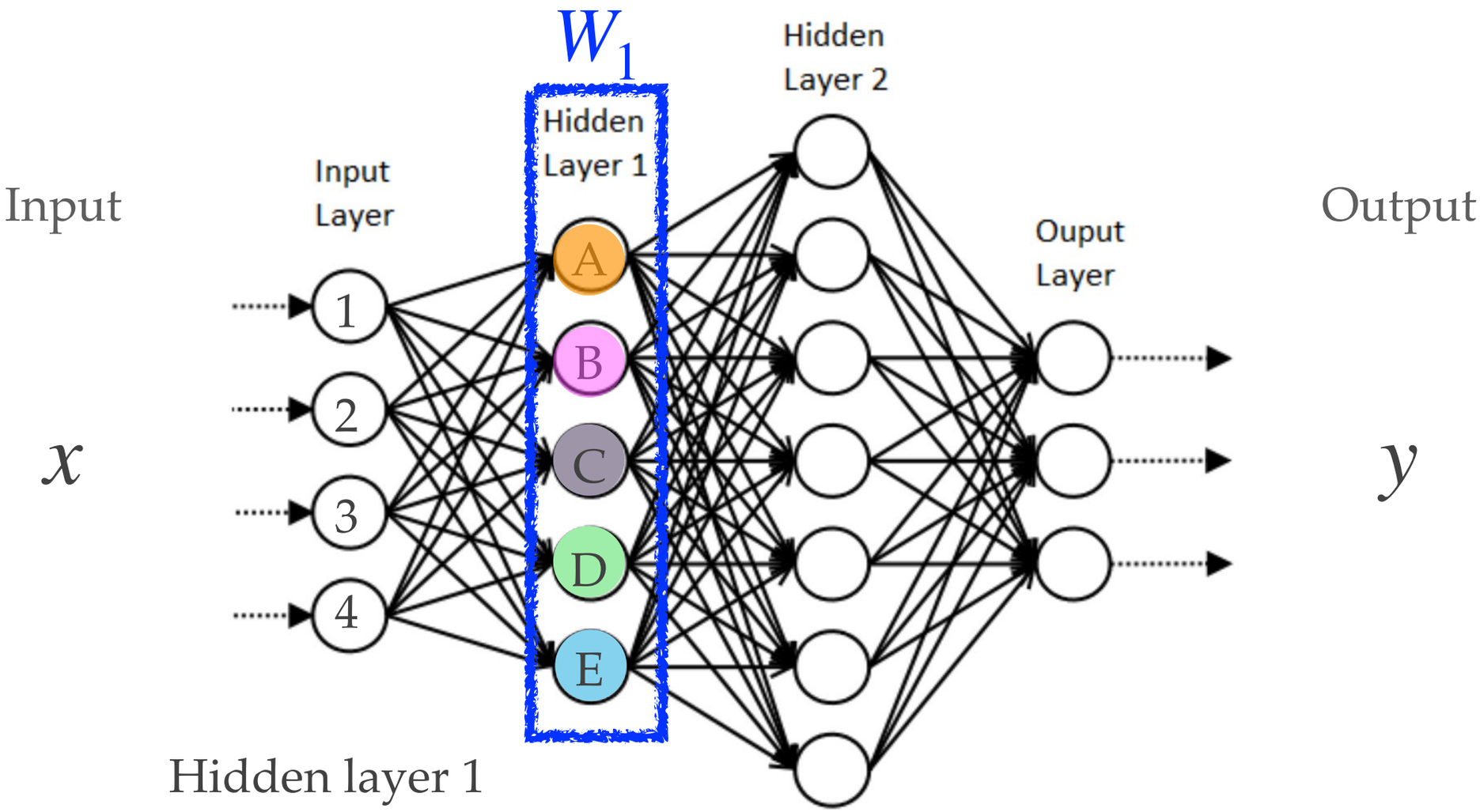
Loss function

categorical crossentropy
binary crossentropy
mean squared error
mean absolute error

Regularization

Dropout
Data augmentation
l_1, l_2 regularizations

- Combine basic components to build a neural network
  - More components → "More" representative power



Hidden layer 1

(ignore the biases for simplicity)

$$\begin{bmatrix} out_A \\ out_B \\ out_C \\ out_D \\ out_E \end{bmatrix} = a_1(W_1 x) = a_1\left(\begin{bmatrix} w_{A,1} & w_{A,2} & w_{A,3} & w_{A,4} \\ w_{B,1} & w_{B,2} & w_{B,3} & w_{B,4} \\ w_{C,1} & w_{C,2} & w_{C,3} & w_{C,4} \\ w_{D,1} & w_{D,2} & w_{D,3} & w_{D,4} \\ w_{E,1} & w_{E,2} & w_{E,3} & w_{E,4} \end{bmatrix} \begin{bmatrix} x_1 \\ x_2 \\ x_3 \\ x_4 \end{bmatrix} \right)$$

# of weights for hidden layer 1
= 4 x 5 = 20

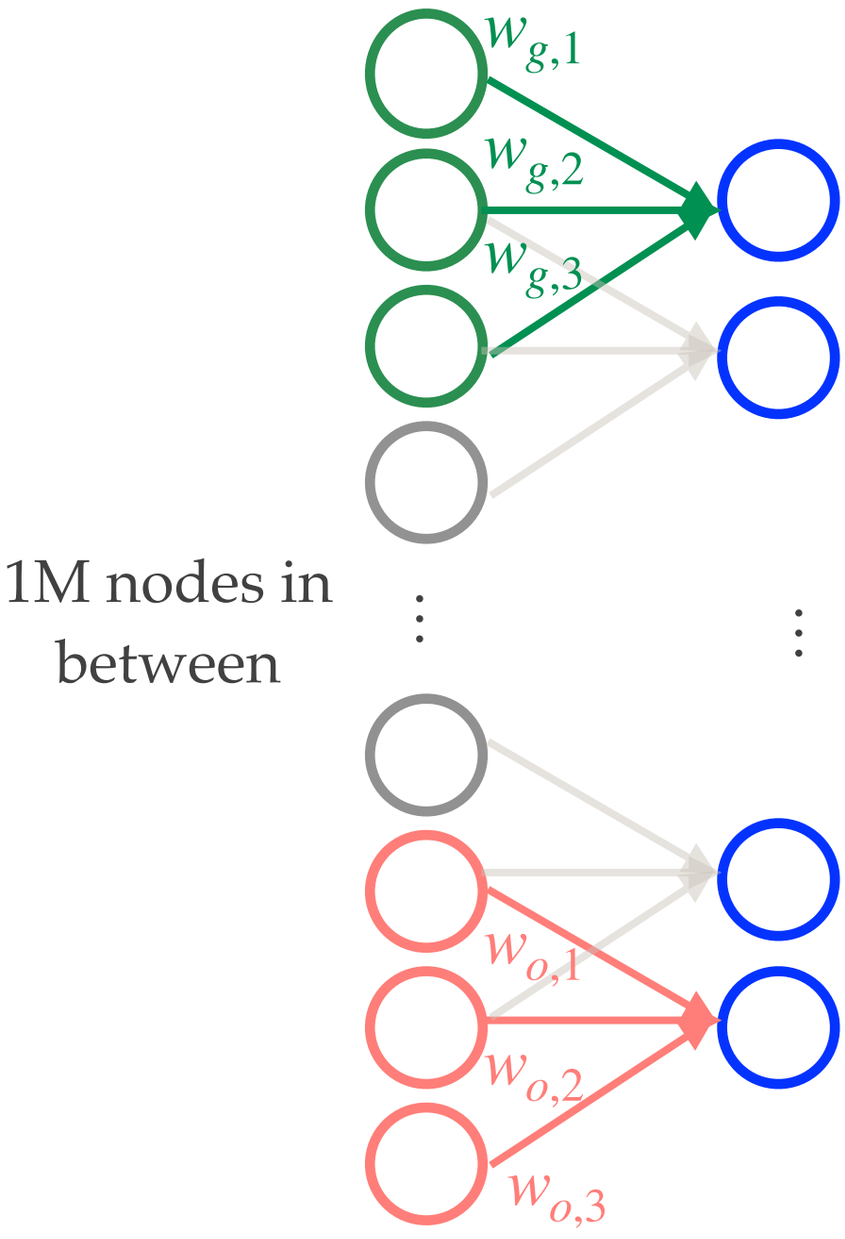
# of weights per layer = # of incoming nodes x # of hidden nodes

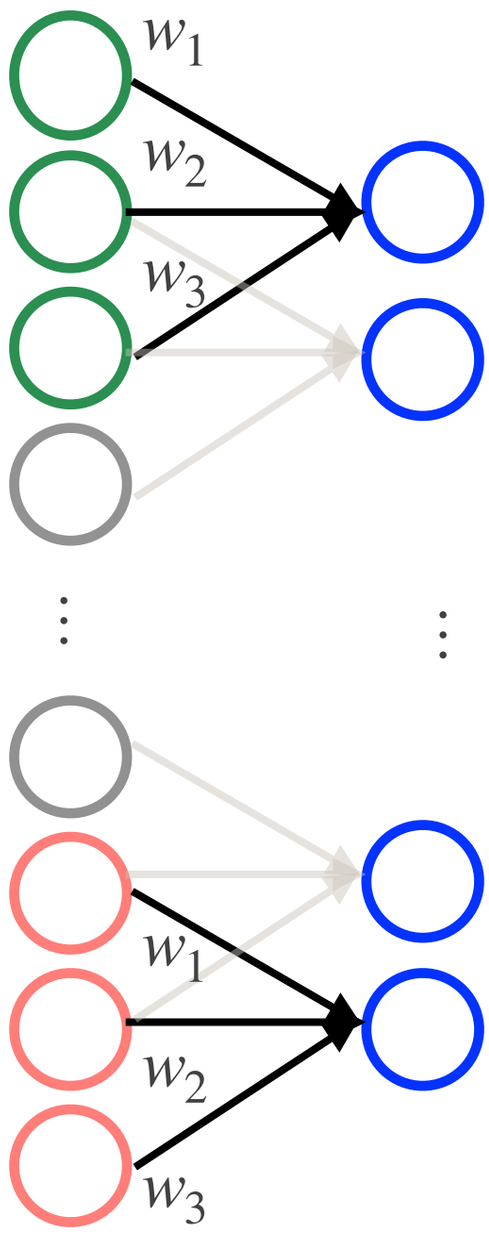
What if our input dimension is 256x256=65,536 and we use 1024 hidden nodes?

67 M trainable parameters for just one layer!

Do we need that many parameters?

- Would it be the case that
  - Only neighboring neurons talk to each other?
  - Furthermore, they talk to their neighbors in the same way?

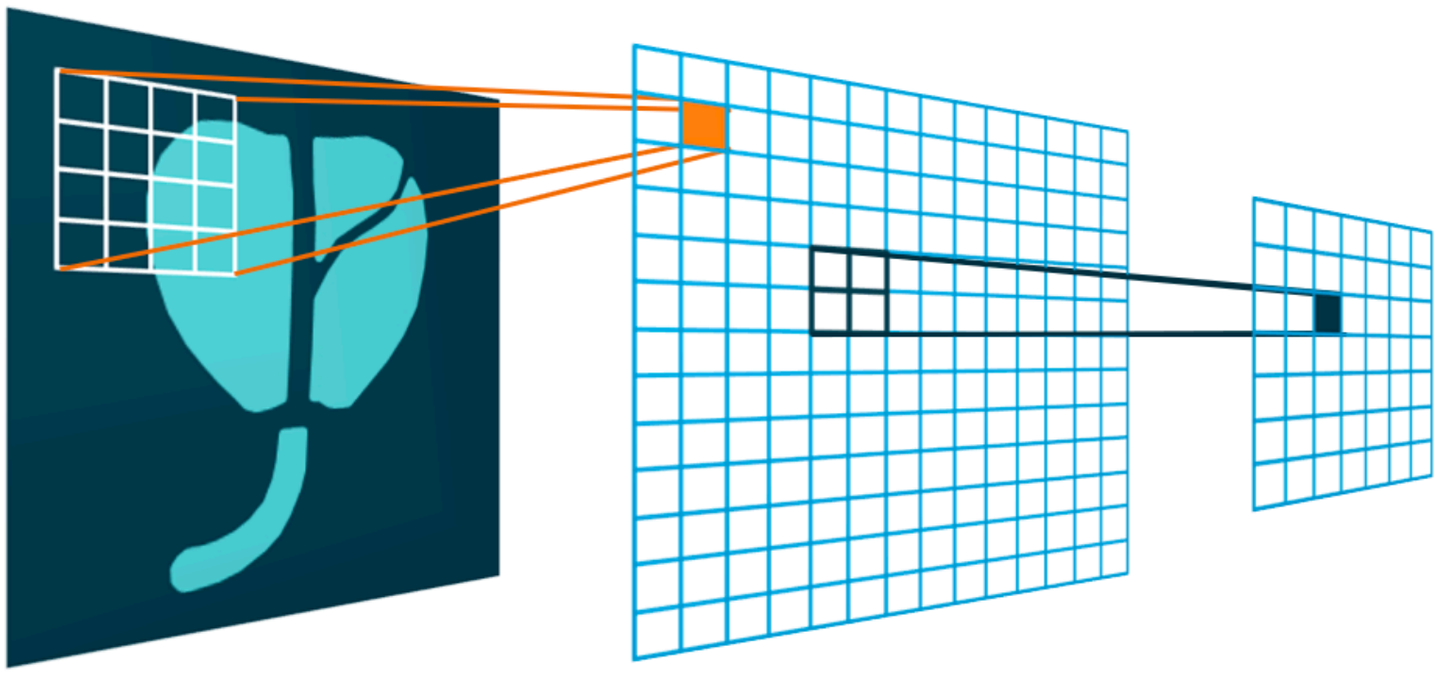




Much fewer parameters!

2D Convolution

- Similarly, for images, maybe only neighboring pixels talk to each other?



2D convolution 2D pooling

Deep learning radiology: the secret of convolutional neural networks

2D Convolution

$$y[n_1, n_2] = x[n_1, n_2] * h[n_1, n_2] = \sum_{k_1=-\infty}^{\infty} \sum_{k_2=-\infty}^{\infty} x[k_1, k_2] h[n_1 - k_1, n_2 - k_2]$$

2D Conv

0*0 + 2*1 + 1*0 + 0*1 + 1*1 + 1*1 + -3*0 + -1*1 + 1*0

0	2	1	1	0
0	1	1	1	0
-3	-1	1	0	1
0	6	0	0	1
0	4	1	7	0

x

Input

0	1	0
1	1	1
0	1	0

h

Filter / kernel

defines the relationship between neighboring pixels

	3			

y

2D Convolution

$$y[n_1, n_2] = x[n_1, n_2] * h[n_1, n_2] = \sum_{k_1=-\infty}^{\infty} \sum_{k_2=-\infty}^{\infty} x[k_1, k_2] h[n_1 - k_1, n_2 - k_2]$$

2D Conv

2*0 + 1*1 + 1*0 + 1*1 + 1*1 + 1*1 + -1*0 + 1*1 + 0*0

0	2	1	1	0
0	1	1	1	0
-3	-1	1	0	1
0	6	0	0	1
0	4	1	7	0

x

Input

0	1	0
1	1	1
0	1	0

h

Filter / kernel

defines the relationship between neighboring pixels

	3	5

y

2D Convolution

$$y[n_1, n_2] = x[n_1, n_2] * h[n_1, n_2] = \sum_{k_1=-\infty}^{\infty} \sum_{k_2=-\infty}^{\infty} x[k_1, k_2] h[n_1 - k_1, n_2 - k_2]$$

2D Conv

1*0 + 1*1 + 0*0 + 1*1 + 1*1 + 0*1 + 1*0 + 0*1 + 1*0

0	2	1	1	0
0	1	1	1	0
-3	-1	1	0	1
0	6	0	0	1
0	4	1	7	0

x

Input

0	1	0
1	1	1
0	1	0

h

Filter / kernel

defines the relationship between neighboring pixels

	3	5	3

y

2D Convolution

$$y[n_1, n_2] = x[n_1, n_2] * h[n_1, n_2] = \sum_{k_1=-\infty}^{\infty} \sum_{k_2=-\infty}^{\infty} x[k_1, k_2] h[n_1 - k_1, n_2 - k_2]$$

2D Conv

0	2	1	1	0
0	1	1	1	0
-3	-1	1	0	1
0	6	0	0	1
0	4	1	7	0

x

Input

0	1	0
1	1	1
0	1	0

h

Filter / kernel

defines the relationship between neighboring pixels

	3	5	3	
	4	1	3	
	9	8	8	

y

smaller output

2D Convolution

Zero-padding

0	0	0	0	0	0	0
0	0	2	1	1	0	0
0	0	1	1	1	0	0
0	-3	-1	1	0	1	0
0	0	6	0	0	1	0
0	0	4	1	7	0	0
0	0	0	0	0	0	0

x

Input

0	1	0
1	1	1
0	1	0

h

Filter / kernel

defines the relationship between neighboring pixels

	3	5	3	
	4	1	3	
	9	8	8	

y

smaller output

2D Convolution

Zero-padding

0	0	0	0	0	0	0
0	0	2	1	1	0	0
0	0	1	1	1	0	0
0	-3	-1	1	0	1	0
0	0	6	0	0	1	0
0	0	4	1	7	0	0
0	0	0	0	0	0	0

x

Input

0	1	0
1	1	1
0	1	0

h

Filter / kernel

defines the relationship between neighboring pixels

2	4	5	3	1
-2	3	5	3	2
-4	4	1	3	2
3	9	8	8	2
4	11	12	8	8

y

output of the same size

2D Convolution

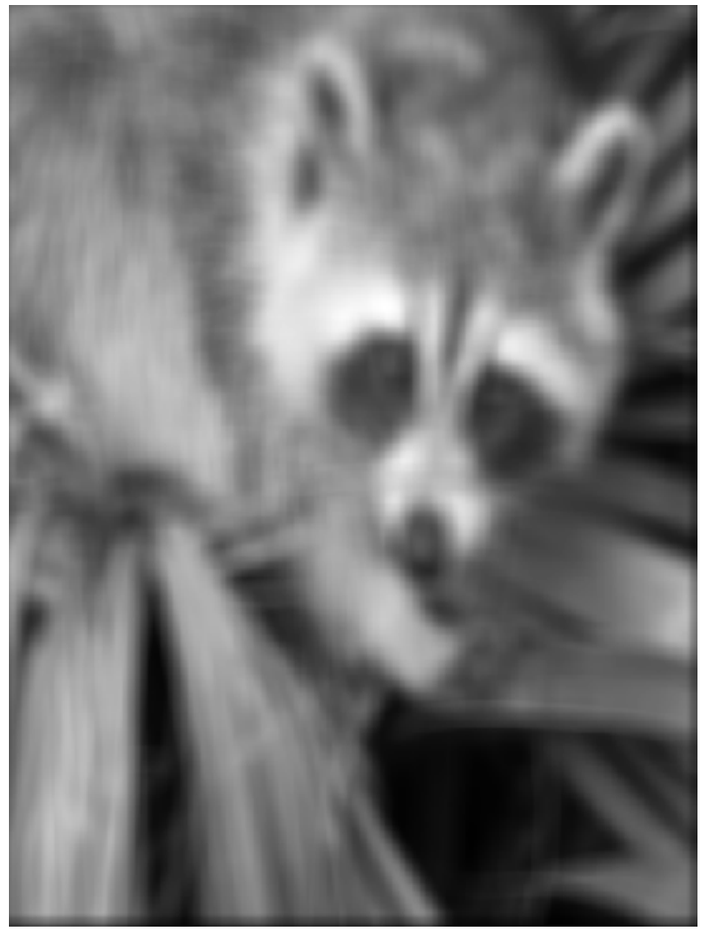
- Low-pass Filter

*

1	1	1	1	1	1	1
1	1	1	1	1	1	1
1	1	1	1	1	1	1
1	1	1	1	1	1	1
1	1	1	1	1	1	1
1	1	1	1	1	1	1
1	1	1	1	1	1	1

Filter / kernel

=



2D Convolution

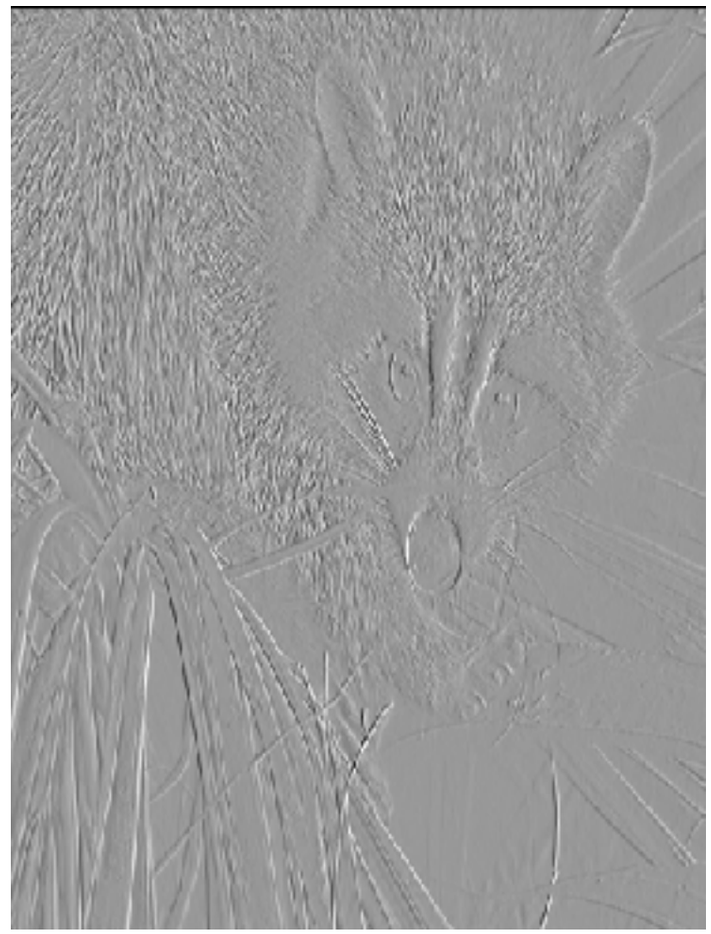
- "Gradient image"

*

-1	1
-1	1

Filter / kernel

=



2D Convolution

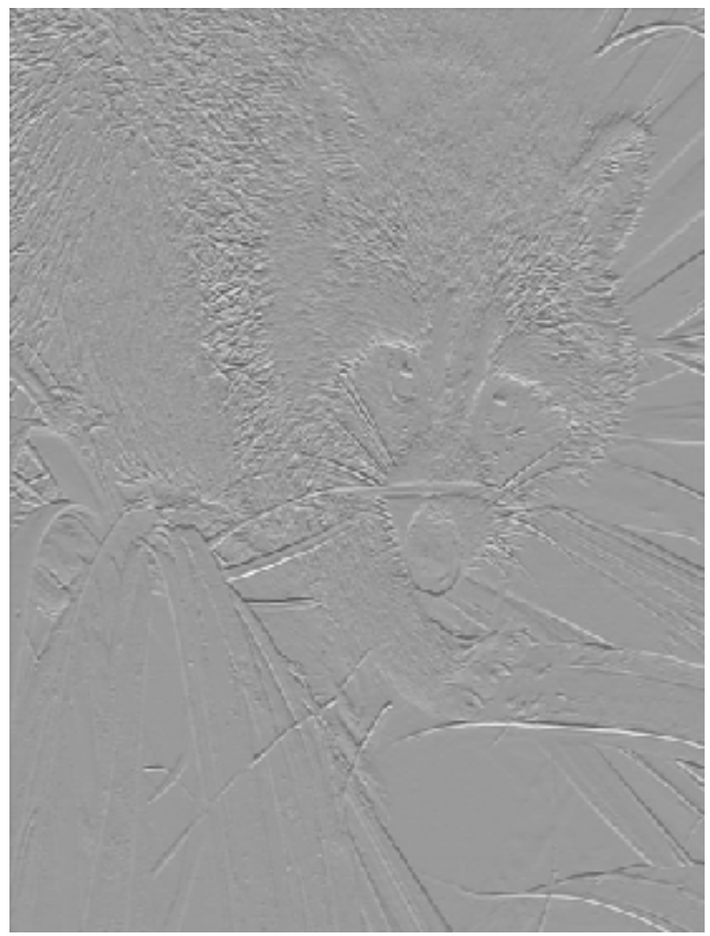
- "Gradient image"

*

-1	-1
1	1

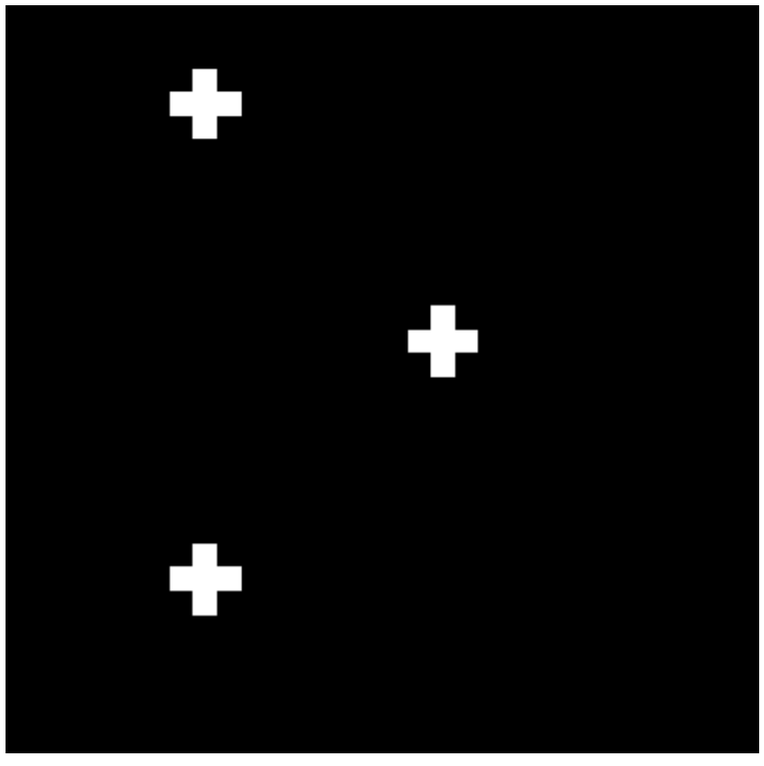
Filter / kernel

=



2D Convolution

- "Plus-sign Detector"

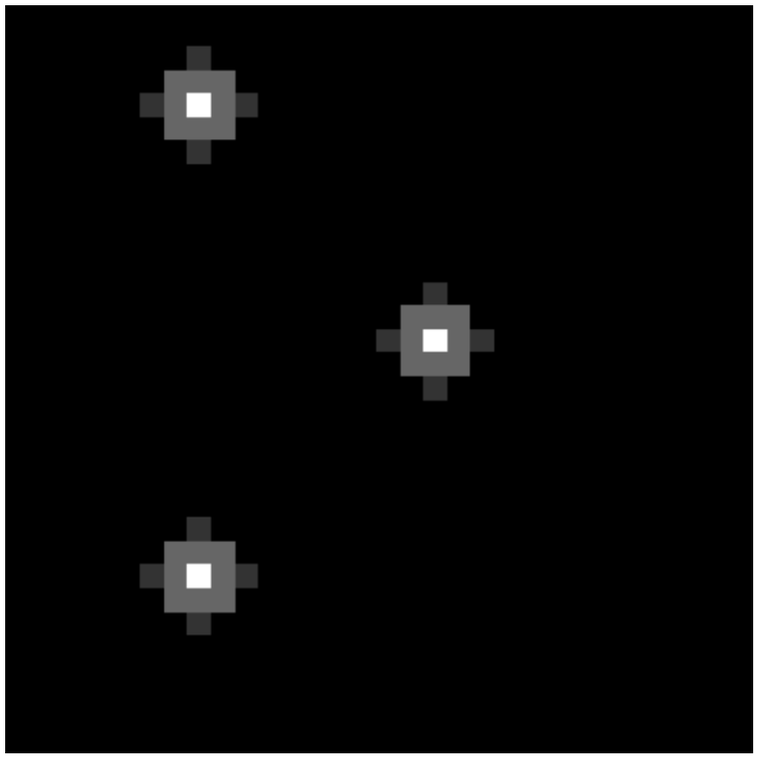


*

0	1	0
1	1	1
0	1	0

Filter / kernel

=



2D Convolution

- Different filters/kernels process images differently
  - Low-pass filter (blurring/denoising)
  - High-pass filter (sharpening)
  - Shape detector
  - Refer to Image Kernels for a good demo

complementary information

- Instead of defining the filter ourselves, we let neural network come up with good ones for us

w_{11}	w_{12}	w_{13}
w_{21}	w_{22}	w_{23}
w_{31}	w_{32}	w_{33}

Filter/kernel
defines the relationship
between neighboring pixels

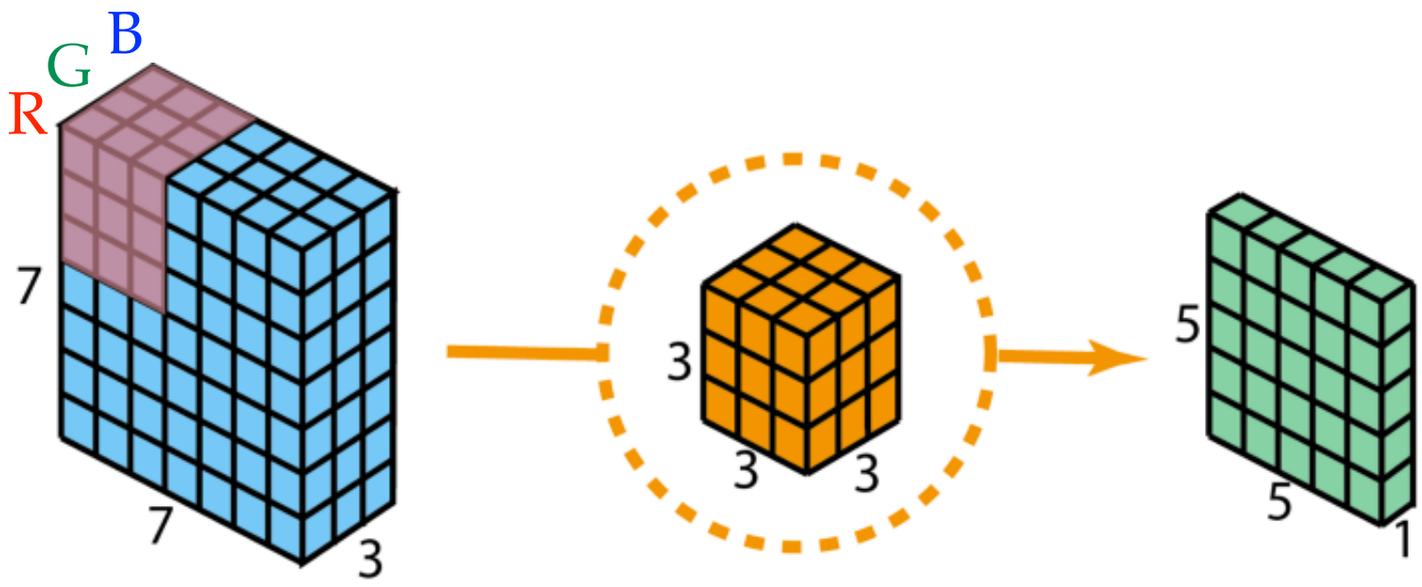
```
model.add(Conv2D(32, (3, 3), padding="same", activation="relu"))
```

# of filters filter size

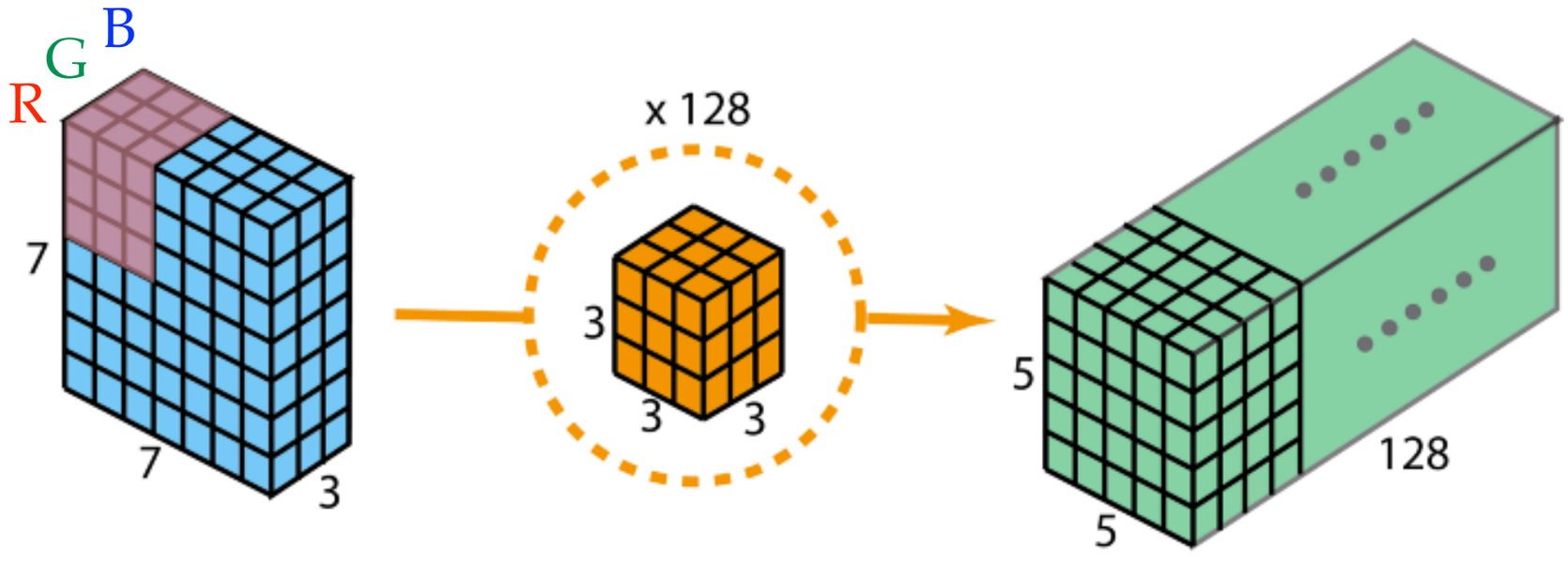
2D Convolution

- 2D convolution with more than one channels

1 filter

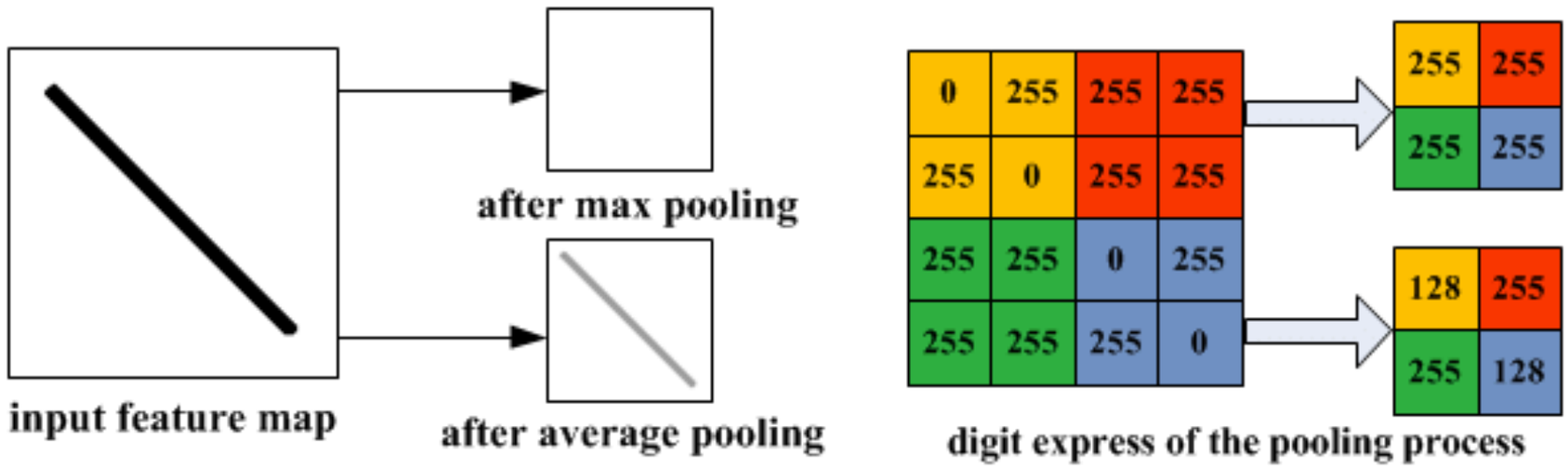


128 filters

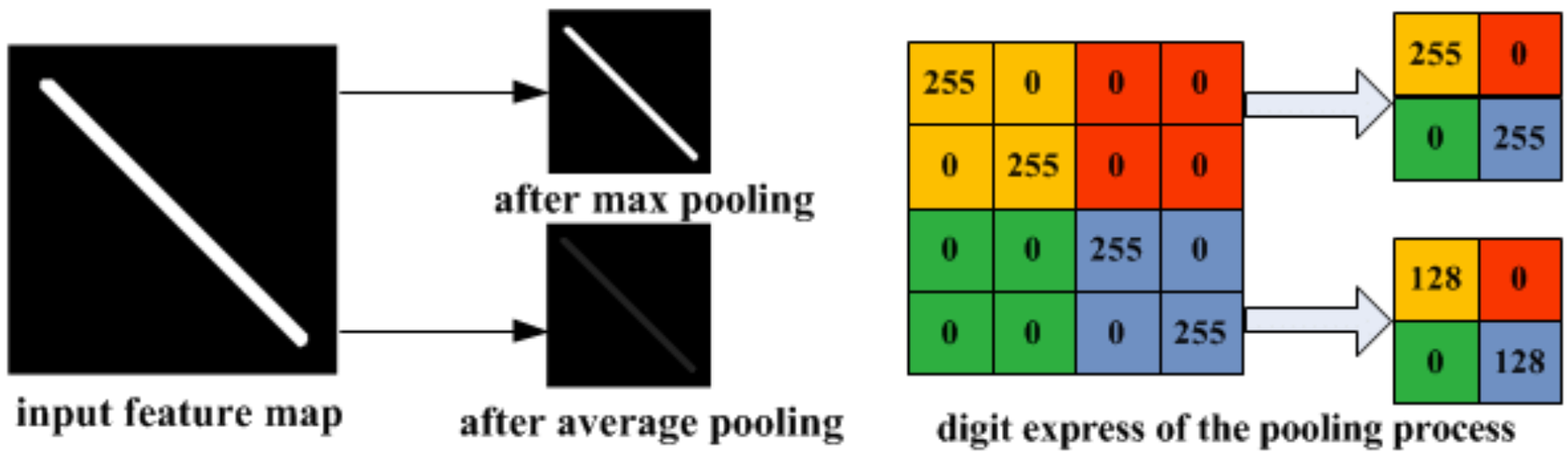


A Comprehensive Introduction to Different Types of Convolutions in Deep Learning

Pooling Layers



(a) Illustration of max pooling drawback



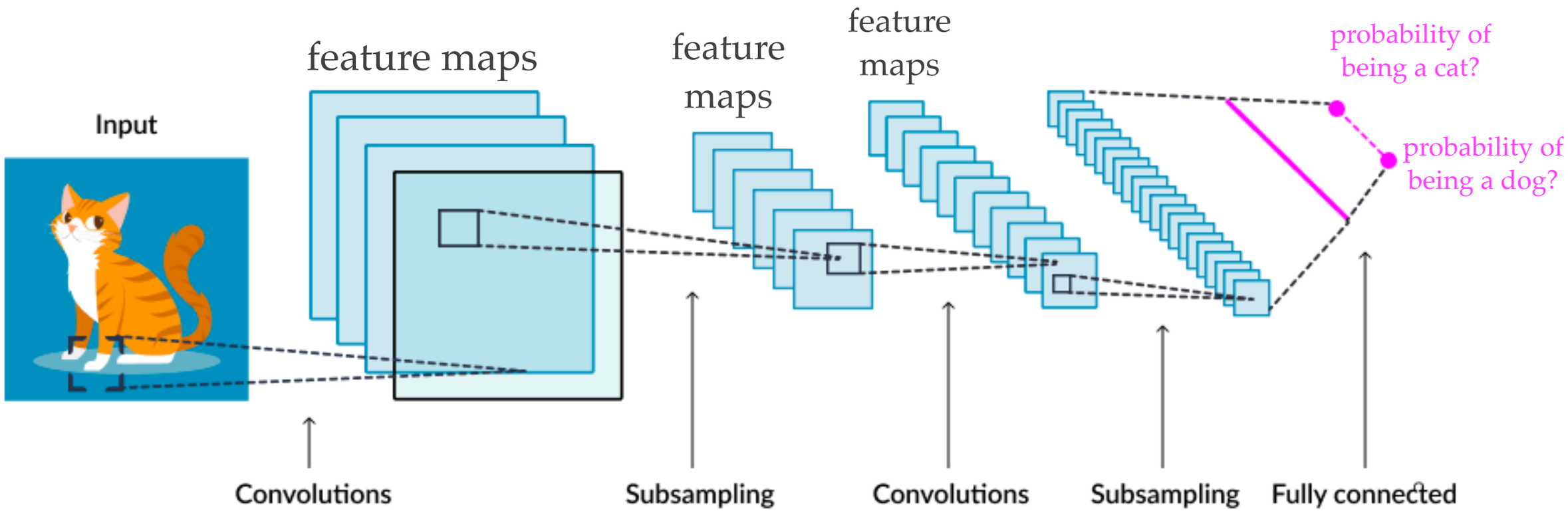
(b) Illustration of average pooling drawback

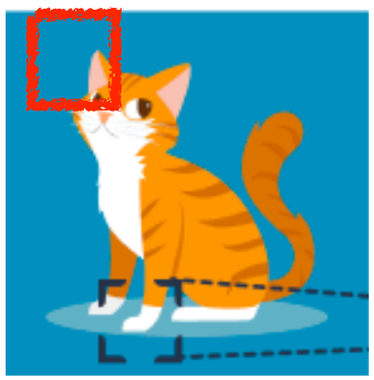
```
model.add(layers.MaxPooling2D((2,2)))
```

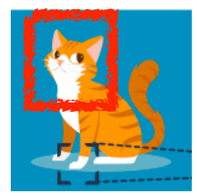
Mixed Pooling for Convolutional Neural Networks

keywords: feature maps, activation maps, receptive field, backbone

Convolutional Neural Network (CNN)





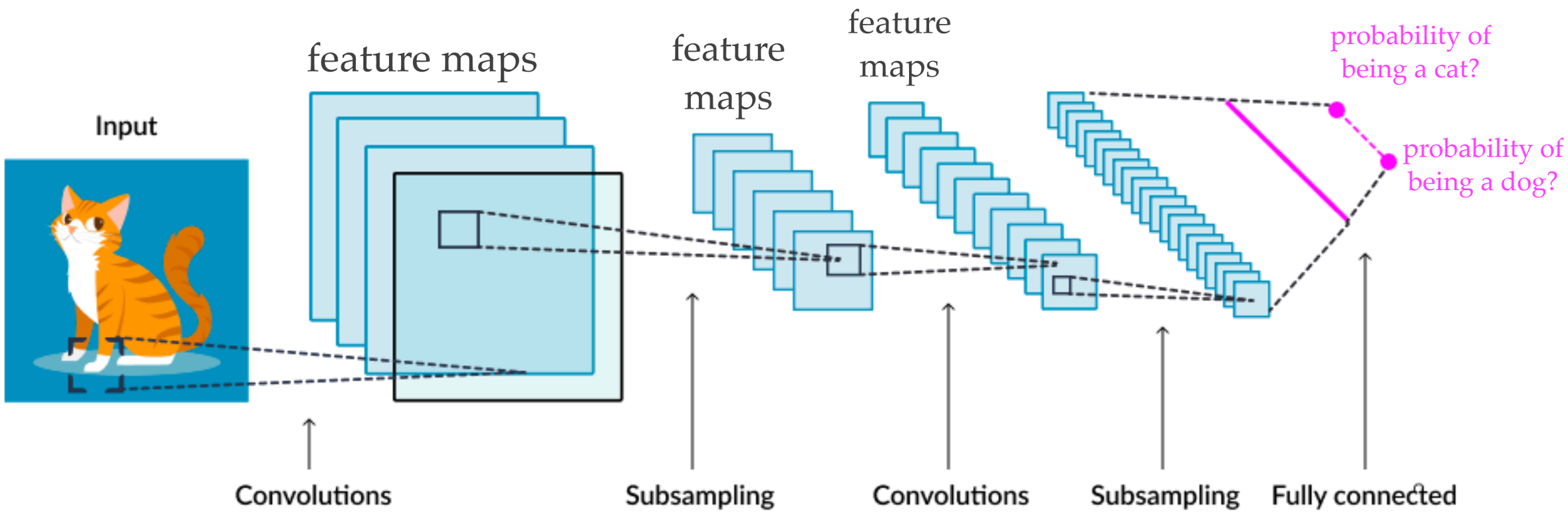


Same filter size (e.g., 3 x 3 filter), but different coverages for different "resolutions"

Disadvantages of CNN models

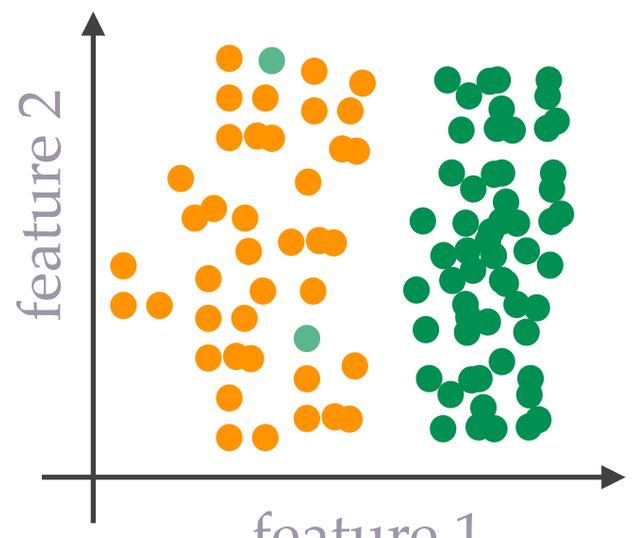
keywords: feature maps, activation maps, receptive field, backbone

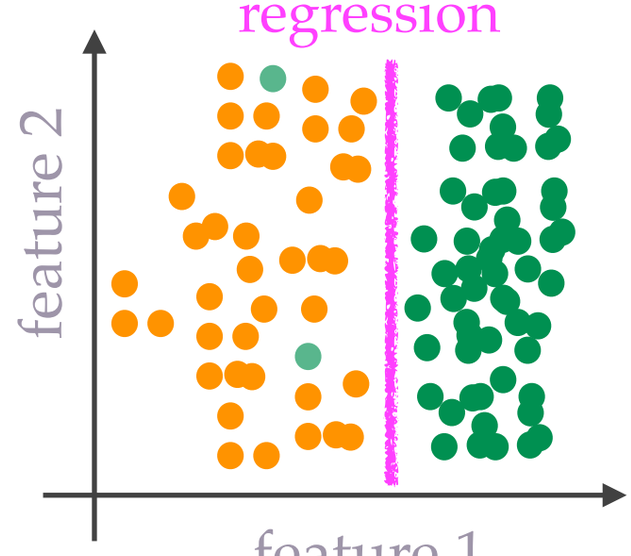
Convolutional Neural Network (CNN)



Feature extraction

Classification/ regression





Disadvantages of CNN models

Fully connected layer
(Dense)

Optimizer

SGD
Adam
RMSprop

Convolutional layer

Conv1D, 2D, 3D, …
separable Conv

Deep Learning

Evaluation metric

accuracy
F1-score
AUC
confusion matrix

Pooling layer

max-pooling
average-pooling

Activation function

sigmoid
softmax
ESP (swish)
ReLU

Loss function

categorical crossentropy
binary crossentropy
mean squared error
mean absolute error

Regularization

Dropout
Data augmentation
l_1, l_2 regularizations

- Combine basic components to build a neural network
  - More components → "More" representative power

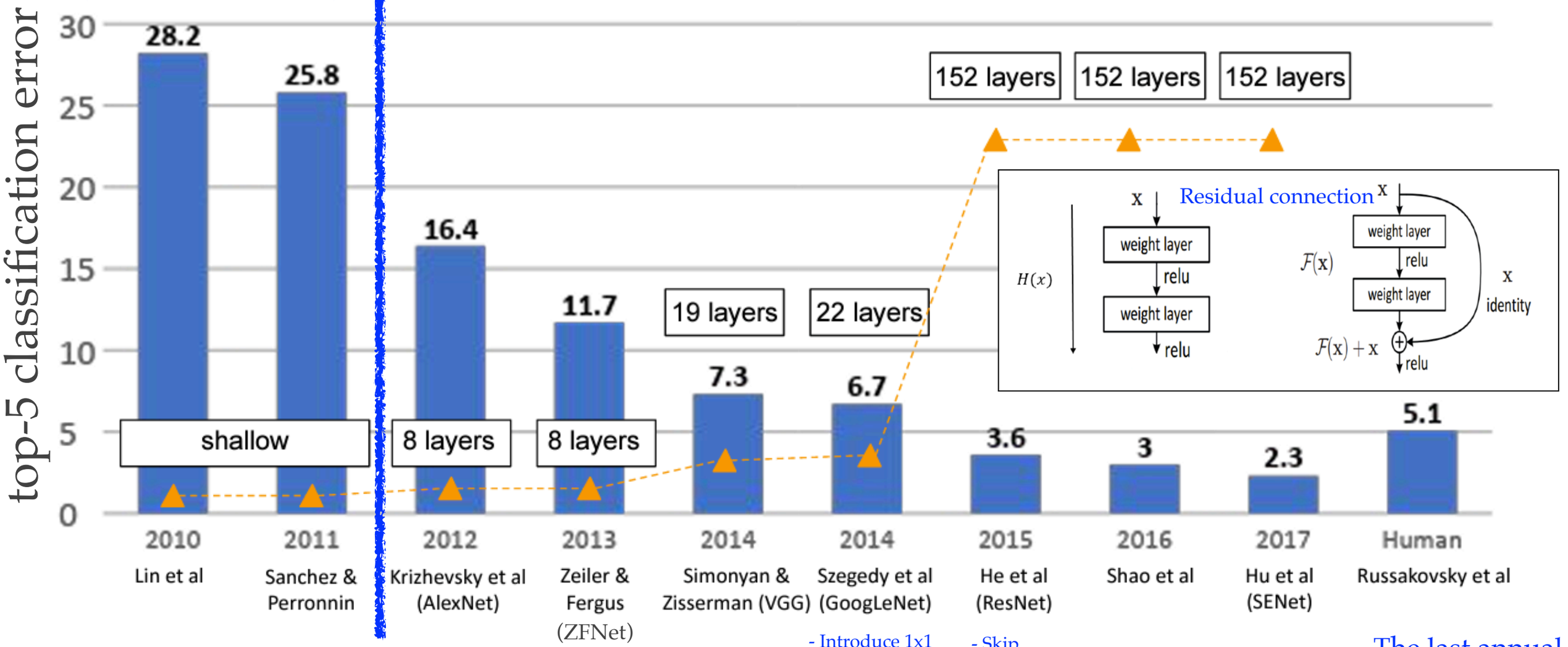
More networks: Xception, ResNeXt, DenseNet, MobileNet, NASNet, EfficientNet, SqueezeNet, Feature Pyramid Network

ImageNet

ImageNet Large Scale Visual Recognition Challenge (ILSVRC) winners

The SVM era (traditional ML)

The deep learning era



- Conv
- Pooling
- Dense
- Dropout
- ReLU
- SGD with momentum

Optimized "AlexNet" in terms of # filter, kernel sizes etc.

19 layers with filter size = 3x3

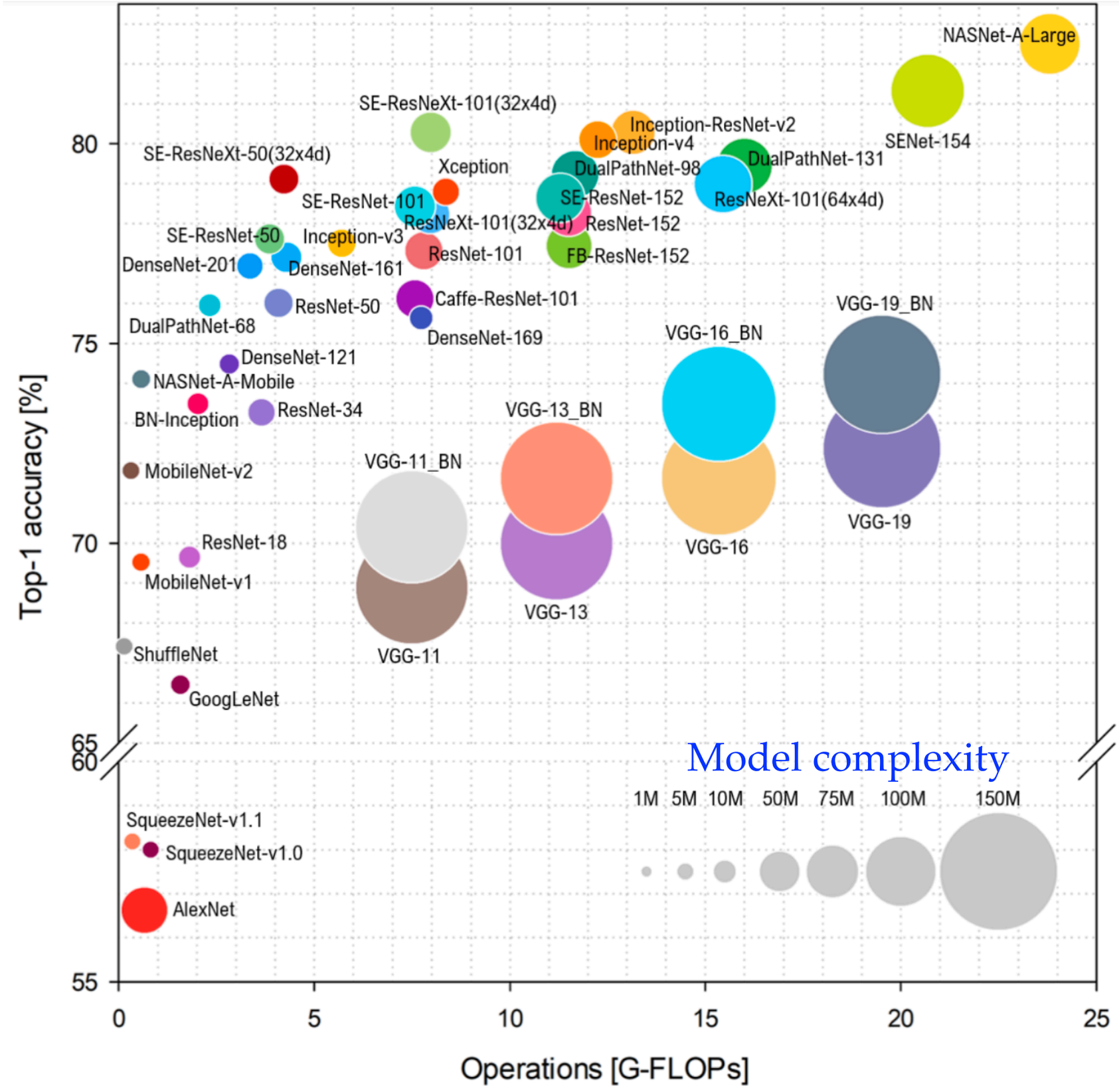
- Introduce 1x1 Conv to reduce the dimensions
- Multiple losses to help with gradients
- Inception module

- Skip connection to help with the vanishing gradient problem
- Batch normalization

- Network ensemble

- Squeeze-and-Excitation (SE) block
- Adaptive feature map reweighting

The last annual ImageNet competition was held in 2017

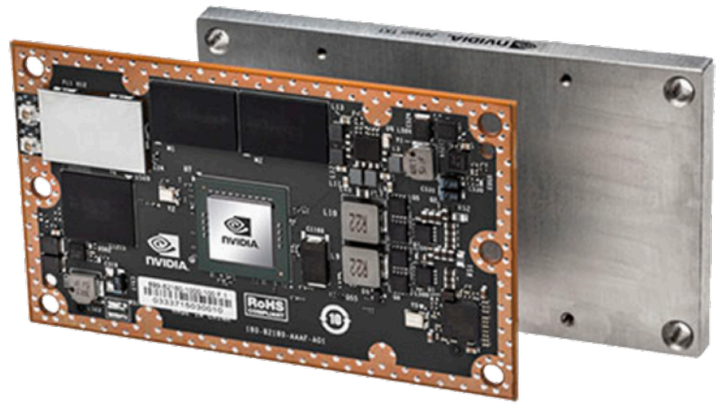


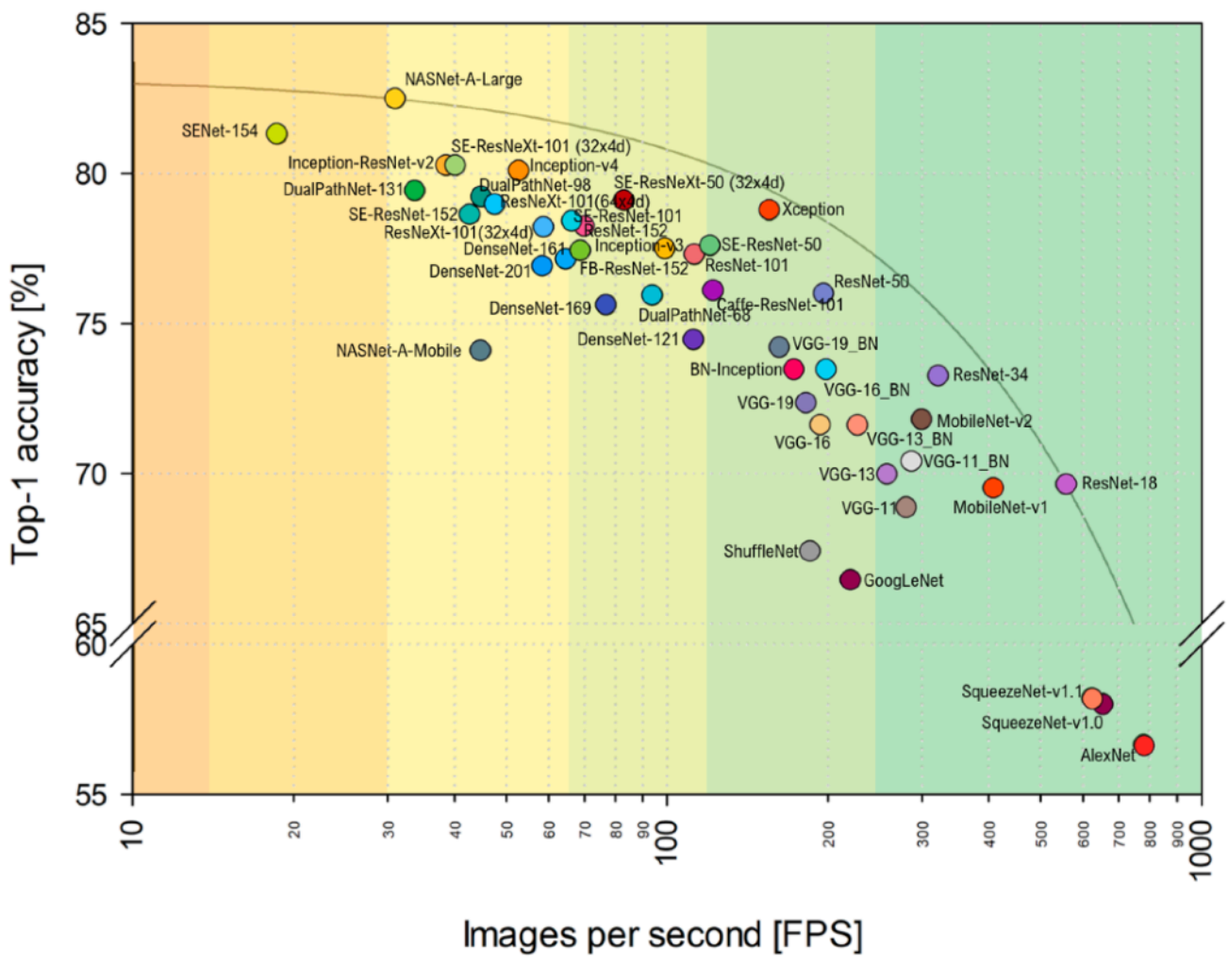
Floating-point operations (FLOPs) required for a single forward pass

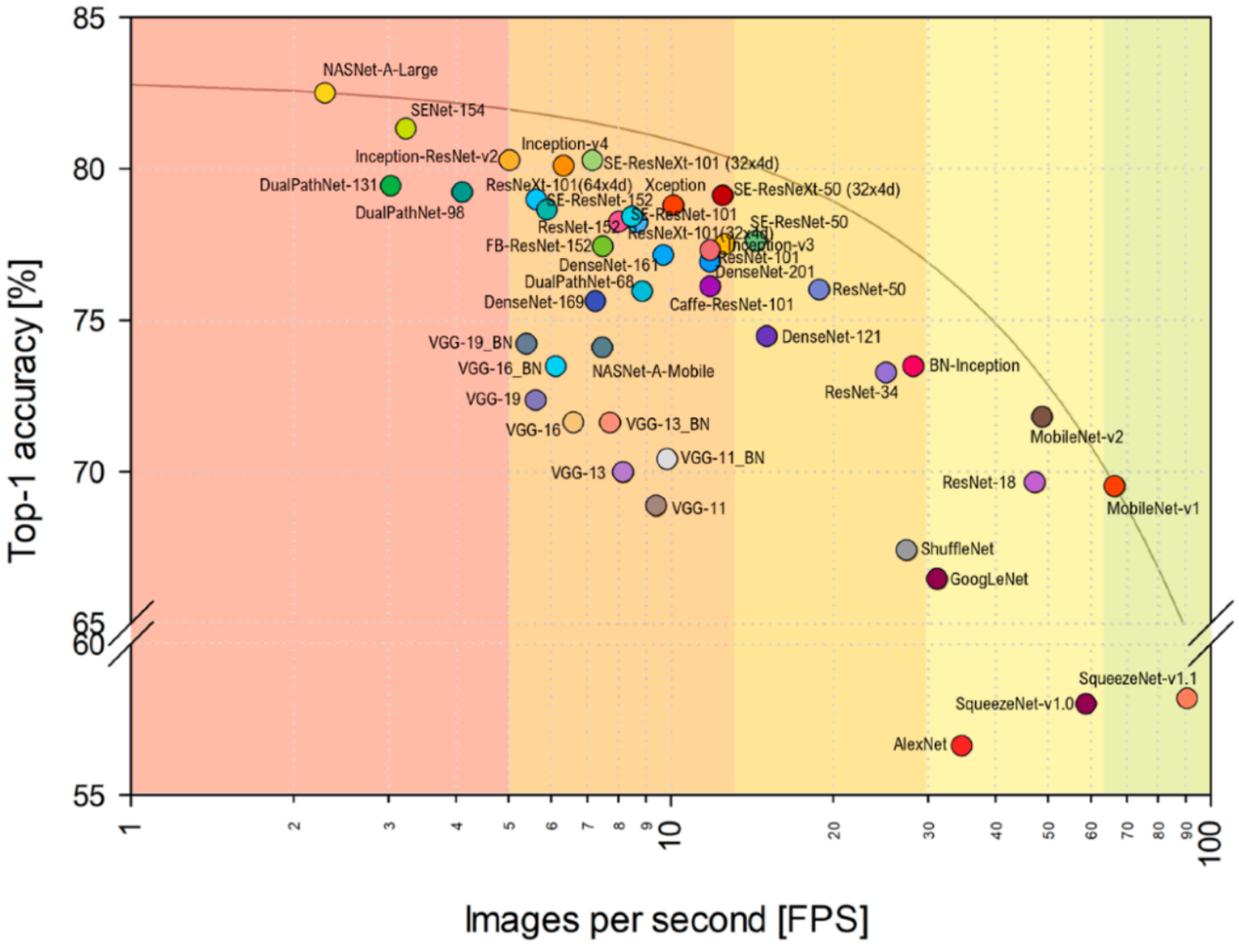
Bianco, Simone, et al. "Benchmark analysis of representative deep neural network architectures." IEEE access 6 (2018): 64270-64277.

Titan Xp

Jetson TX1



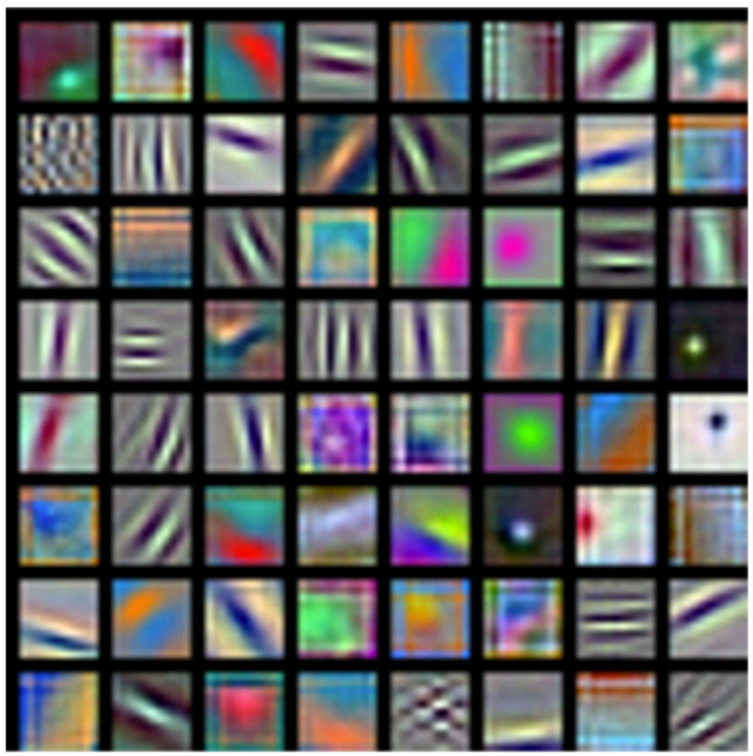




Bianco, Simone, et al. "Benchmark analysis of representative deep neural network architectures." IEEE access 6 (2018): 64270-64277.

Understanding CNN - Filter Visualization

Learned weights of the filters at the first layer

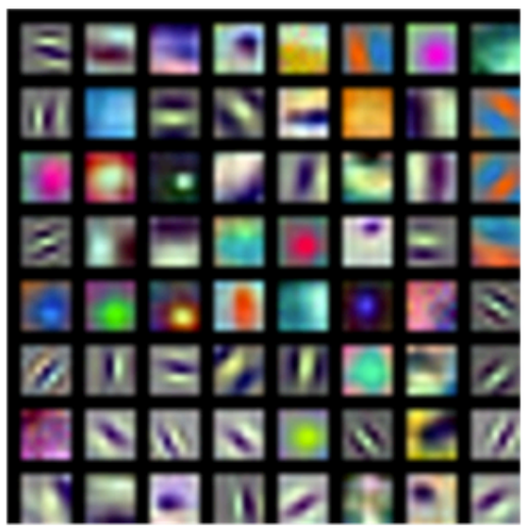


AlexNet:
64 x 3 x 11 x 11

64 filters filter size = 11 x 11
with 3 channels

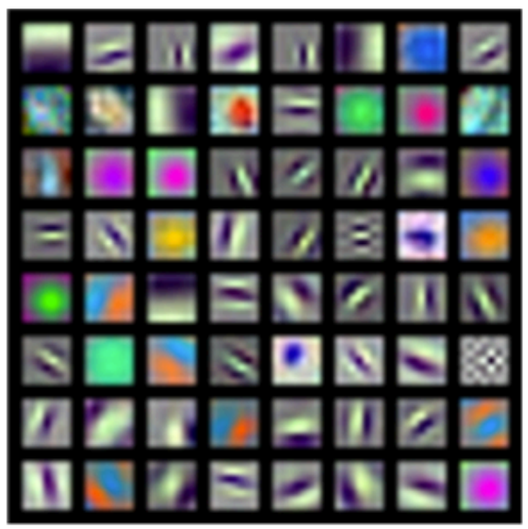
What these filters look for (through template matching/inner product)
- oriented edges
- opposing colors

Similar to human visual system!



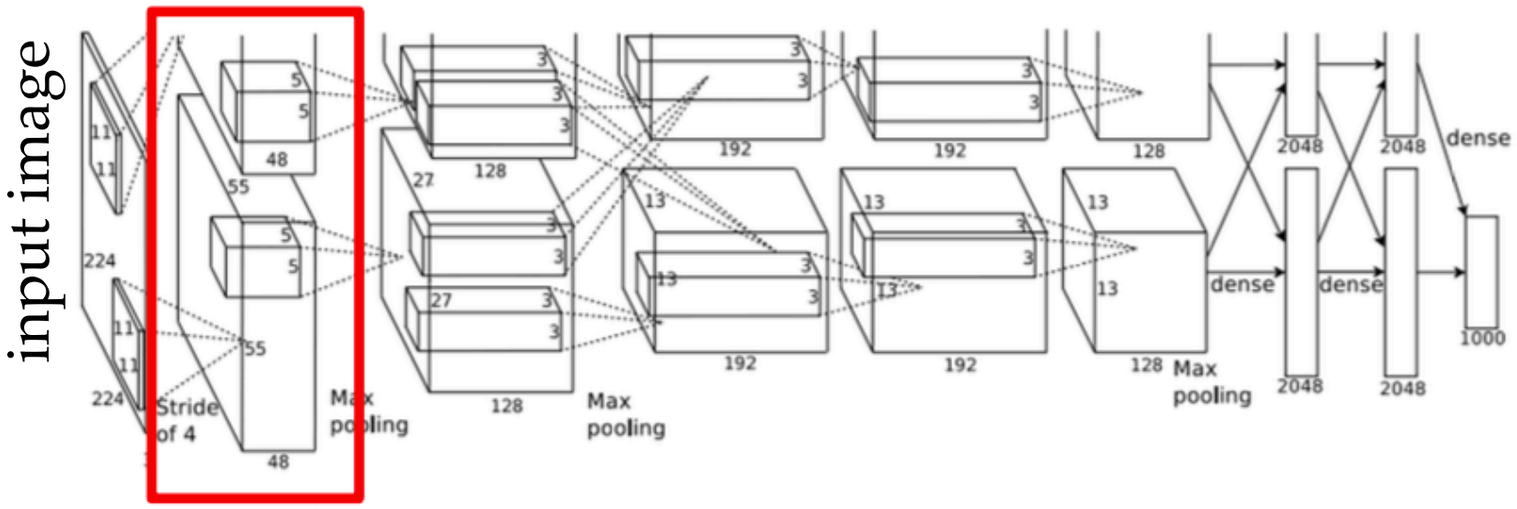
ResNet-18:
64 x 3 x 7 x 7

ResNet-101:
64 x 3 x 7 x 7



DenseNet-121:
64 x 3 x 7 x 7

AlexNet



conv1

Understanding CNN - Filter Visualization

Visualize the filters/kernels (raw weights)

We can visualize filters at higher layers, but not that interesting

(these are taken from ConvNetJS CIFAR-10 demo)

layer 1 weights

16 x 3 x 7 x 7

layer 2 weights

20 x 16 x 7 x 7

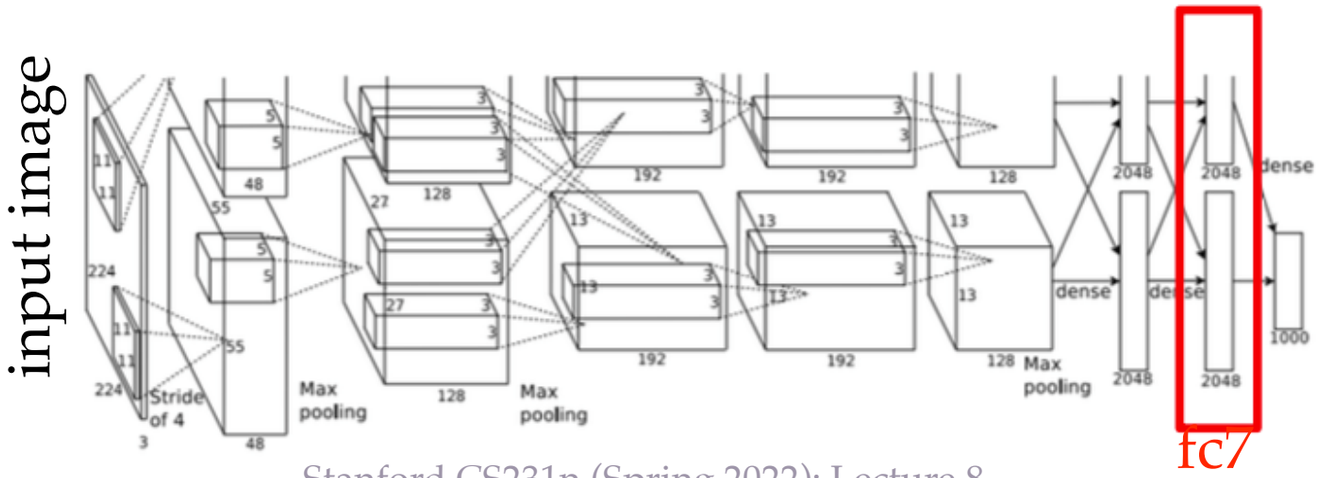
layer 3 weights

20 x 20 x 7 x 7

The weights of conv2 tell us what type of the activation patterns after conv1 that would cause conv2 to maximally activate. These filters are not connected directly to the input image, so it is not directly interpretable.

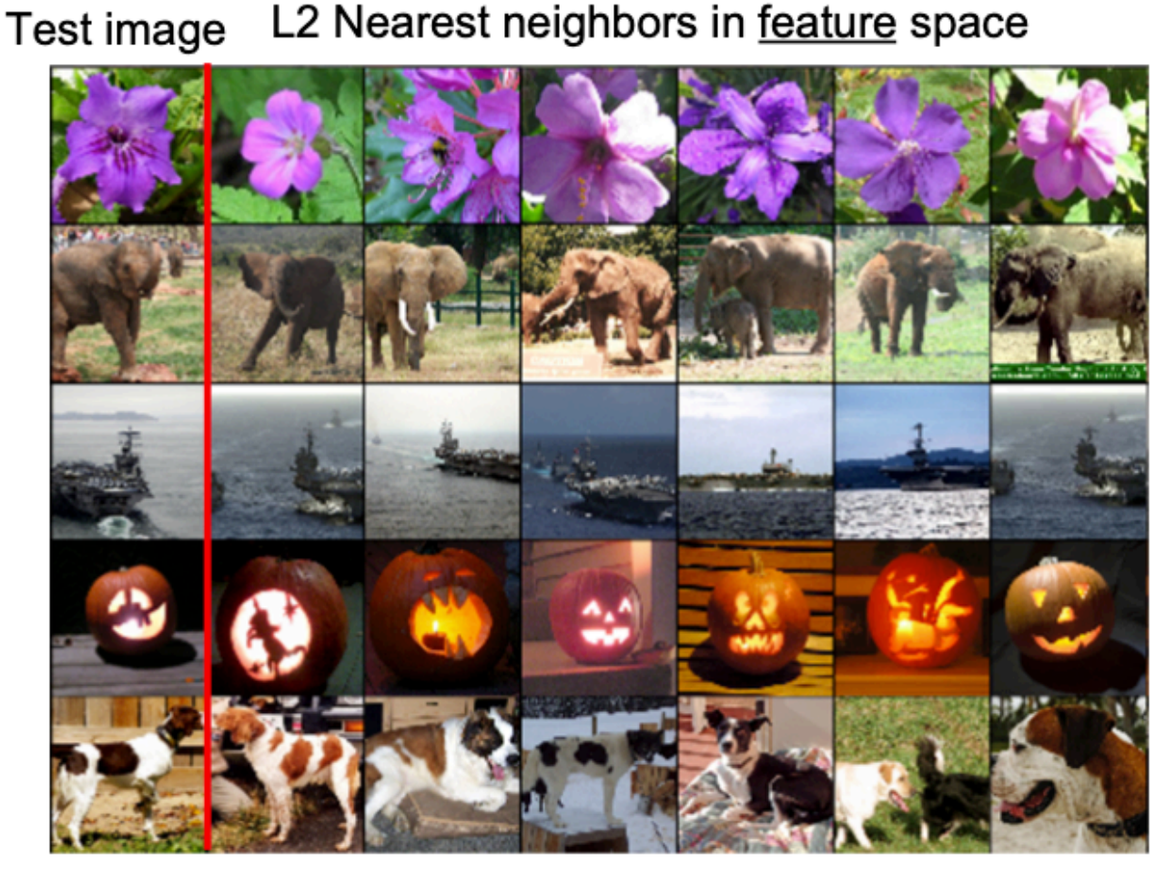
We need more complicated techniques to get a sense of what is going on.

Understanding CNN - Last Layer

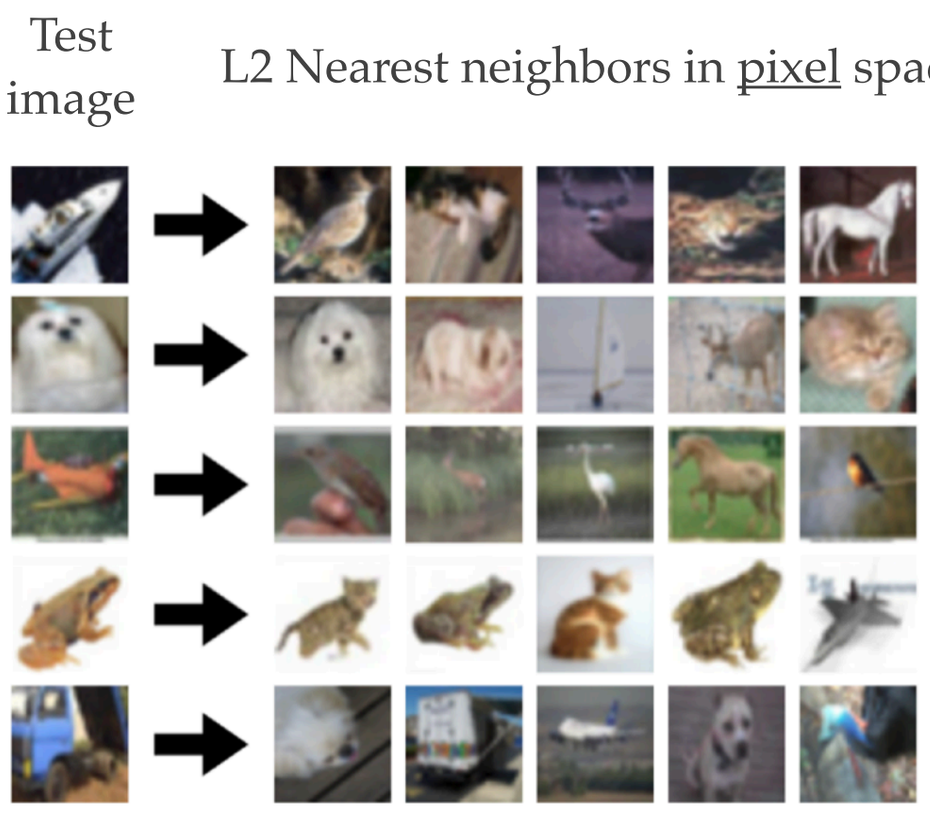


Stanford CS231n (Spring 2022): Lecture 8.

output of fc 7 = feature vector (last layer) of size 4096

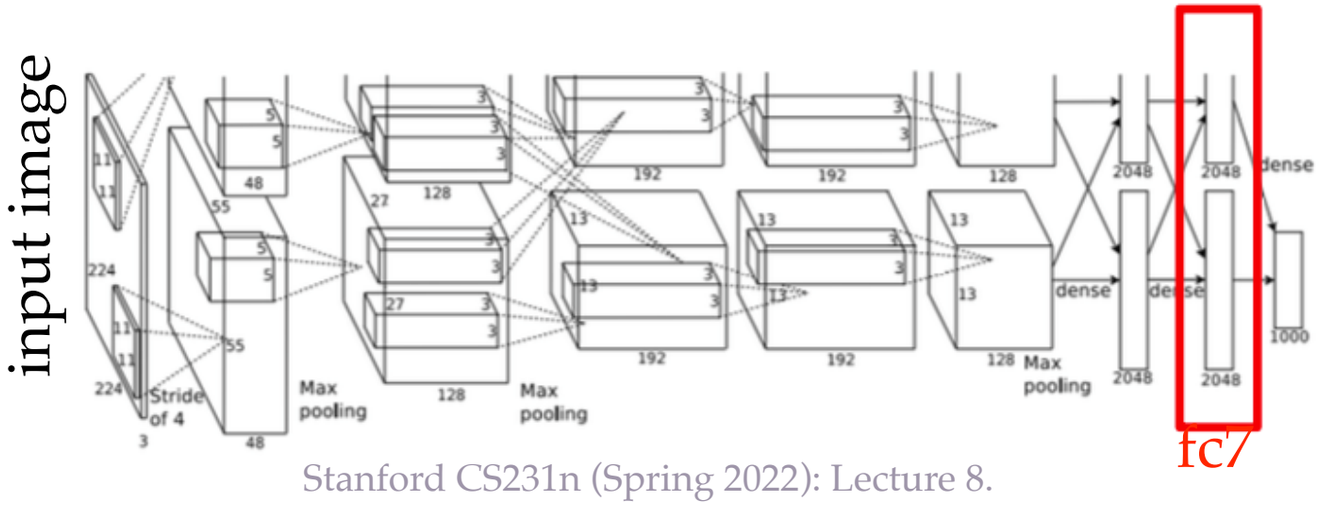


Krizhevsky, Alex, Ilya Sutskever, and Geoffrey E. Hinton. "Imagenet classification with deep convolutional neural networks." Advances in neural information processing systems 25 (2012).



Stanford CS231n (Spring 2022): Lecture 8.

Understanding CNN - Last Layer



Stanford CS231n (Spring 2022): Lecture 8.

output of fc 7 = feature vector (last layer) of size 4096

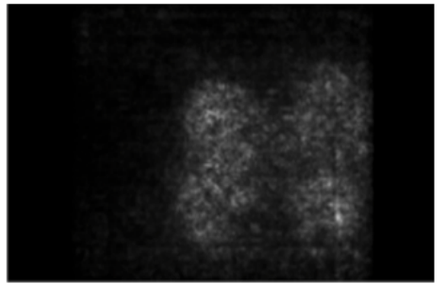
Perform a machine learning algorithm to the output of fc7

Barnes-Hut t-SNE

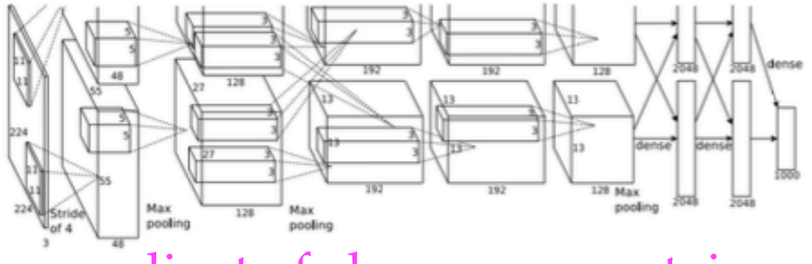
https://cs.stanford.edu/people/karpathy/cnnembed/

Understanding CNN - Other Techniques

Saliency Maps



Forward pass: Compute probabilities



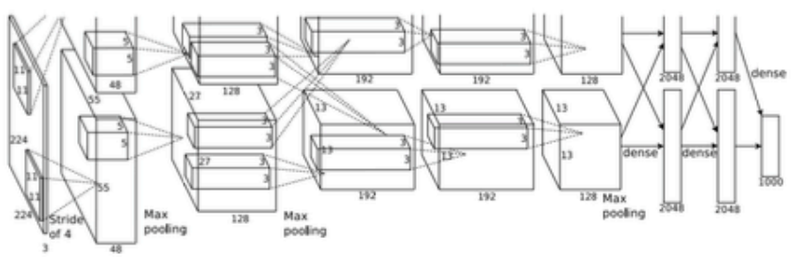
Dog

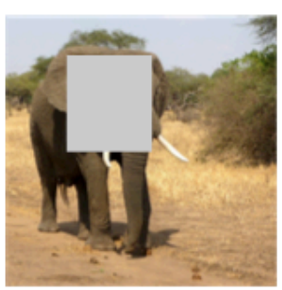
gradient of class score w.r.t. image pixels

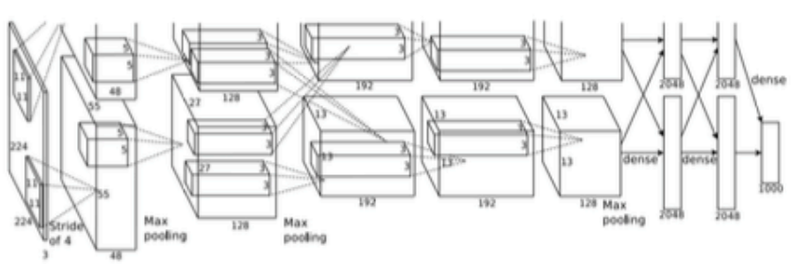
Simonyan, Karen, Andrea Vedaldi, and Andrew Zisserman. "Deep inside convolutional networks: Visualising image classification models and saliency maps." arXiv preprint arXiv:1312.6034 (2013).

Occlusion Experiments

Mask part of the image before feeding to CNN, check how much predicted probabilities change

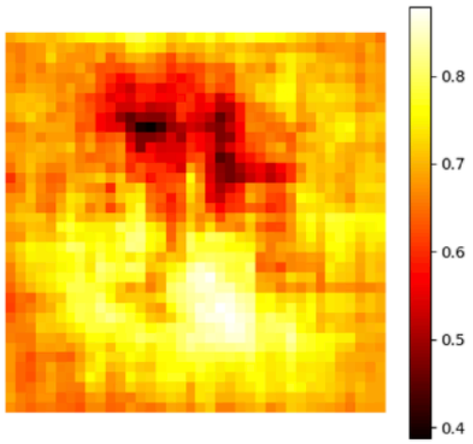






Zeiler, Matthew D., and Rob Fergus. "Visualizing and understanding convolutional networks." European conference on computer vision. Springer, Cham, 2014.

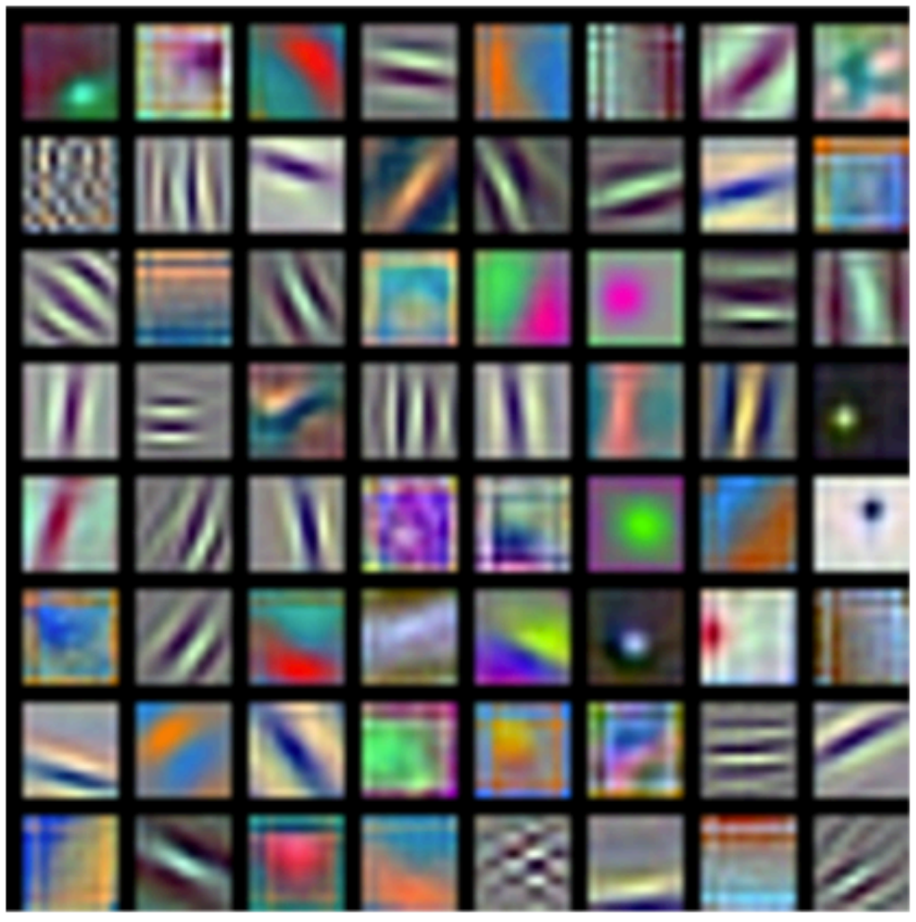
African elephant, Loxodonta africana



…and many more such as

- Guided Backpropagation
- Grad-CAM, Score-CAM, HiResCAM
- Attention map visualization

Trained CNN as a Feature Extractor



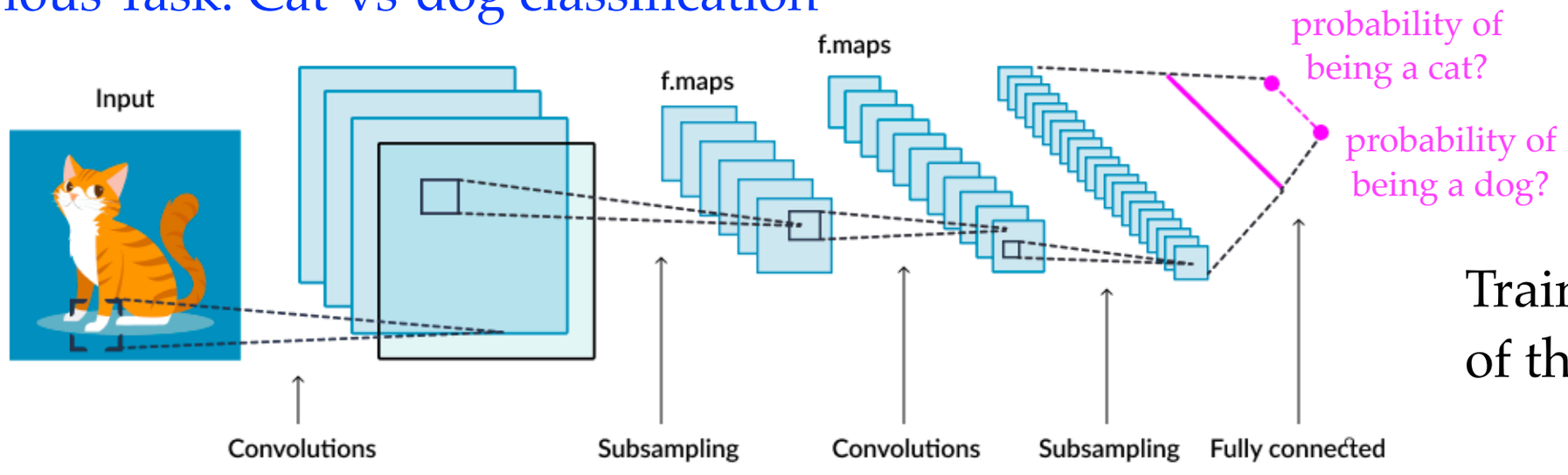
What these filters look for (through template matching/inner product)

- oriented edges
- opposing colors

These patterns are important for processing other types of images as well, so we can reuse trained weights when we encounter a related task or the same task but with a different dataset

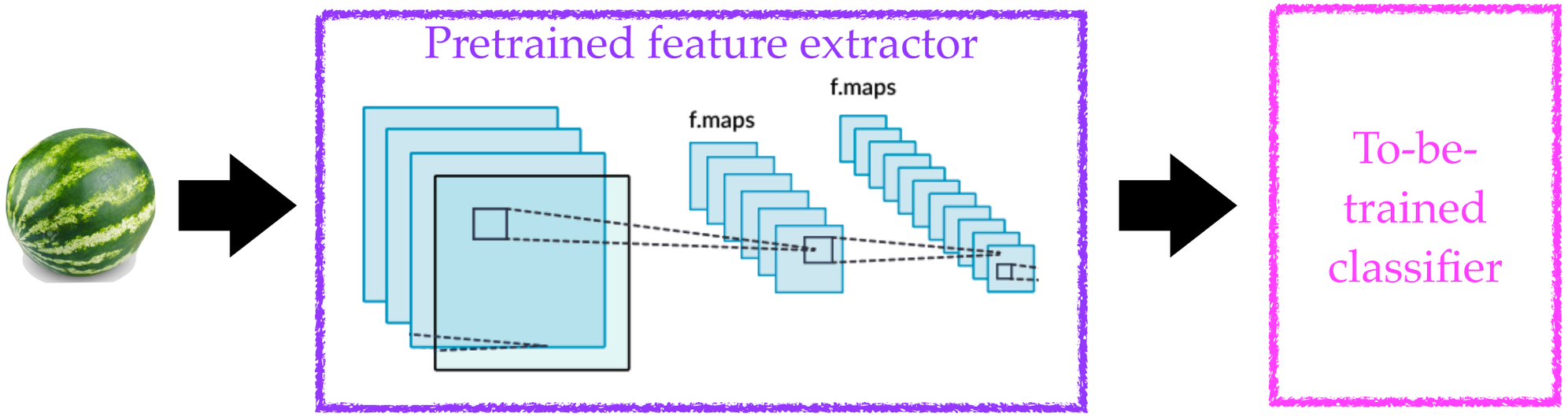
Trained CNN as a Feature Extractor

Previous Task: Cat-vs-dog classification



Train all the weights of this network

New Task: Watermelon-vs-orange classification



Take parts of the trained network and use them as a feature extractor

Only train the weights here

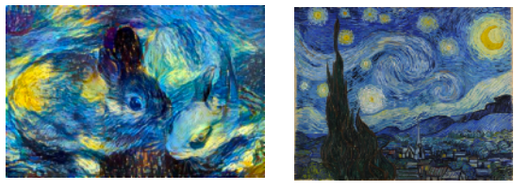
Transfer Learning and Fine-Tuning

ImageNet (1000 classes)

14M images

AI-vs-Artists (2 classes)

Assume small dataset



More "reusable"

Less "reusable"

layer 1

layer 2

⋮

layer L-1

layer L

FC-4096

FC-4096

FC-1000

Clone these layers along with their weights and do not modify the weights

Replace the classification layer and train it

layer 1

layer 2

⋮

layer L-1

layer L

FC-4096

FC-4096

FC-2

Transfer Learning and Fine-Tuning

ImageNet (1000 classes)

14M images

More "reusable"

layer 1

layer 2

⋮

layer L-1

layer L

FC-4096

FC-4096

FC-1000

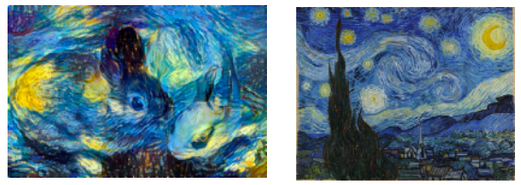
Less "reusable"

Clone these layers along with their weights and do not modify the weights

Replace the classification layer and train it

AI-vs-Artists (2 classes)

Assume small dataset



layer 1

layer 2

⋮

layer L-1

layer L

FC-4096

FC-4096

FC-2

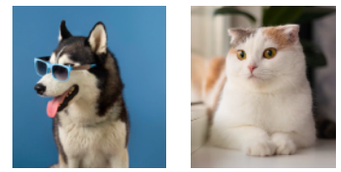
Clone these layers along with their weights and do not modify the weights

Clone these layers along with their weights and retrain them with an initially low learning rate

Replace these layers and train them

Dog-vs-cat (2 classes)

Assume dataset of moderate size



layer 1

layer 2

⋮

layer L-1

layer L

FC-4096

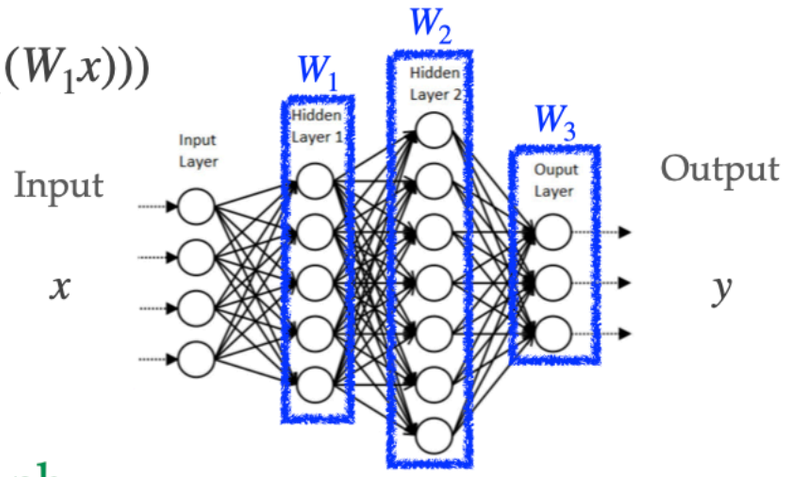
FC-n_filters

FC-2

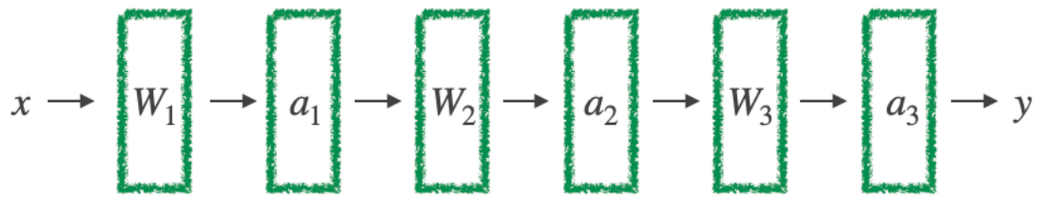
Outline

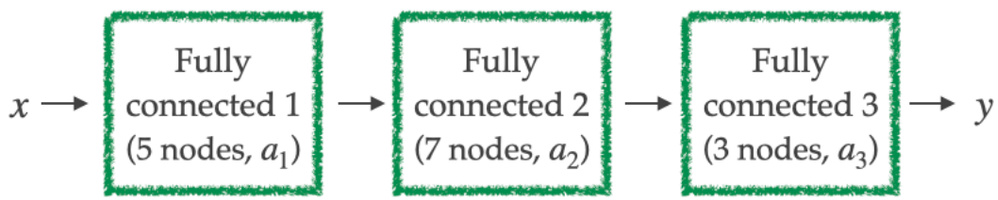
- Deep Learning Components
  - Fully connected layer
  - Computation graph
  - Activation functions
  - Loss function
  - Model optimization
    - Gradient descent
    - Backpropagation
    - Stochastic gradient descent (SGD)
  - Regularization
    - ℓ_2 and ℓ_1 regularization
    - Dropout
    - Batch normalization
  - Convolutional layer
  - Pooling layer
- Convolutional Neural Network (CNN)

$$y = a_3(W_3 a_2(W_2 a_1(W_1 x)))$$

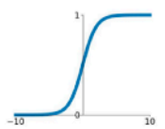


Computation graph

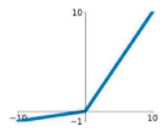




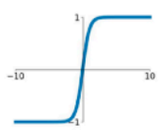
Sigmoid
$\sigma(x) = \frac{1}{1+e^{-x}}$



Leaky ReLU
$\max(0.1x, x)$

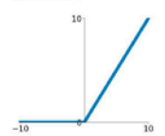


tanh
$\tanh(x)$

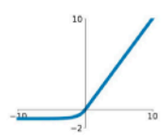


Maxout
$\max(w_1^T x + b_1, w_2^T x + b_2)$

ReLU
$\max(0, x)$

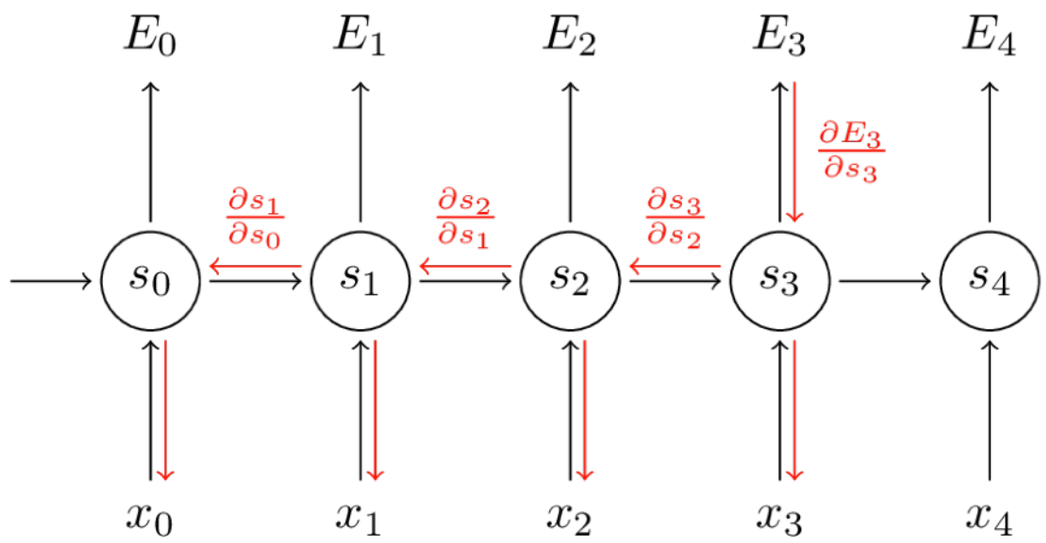


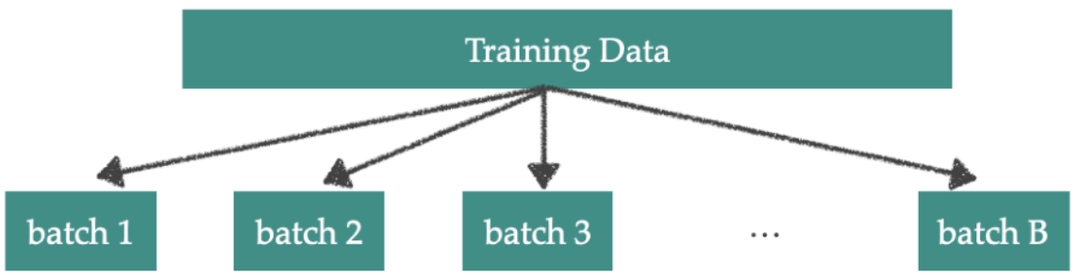
ELU
$\begin{cases} x & x \geq 0 \\ \alpha(e^x - 1) & x < 0 \end{cases}$



Outline

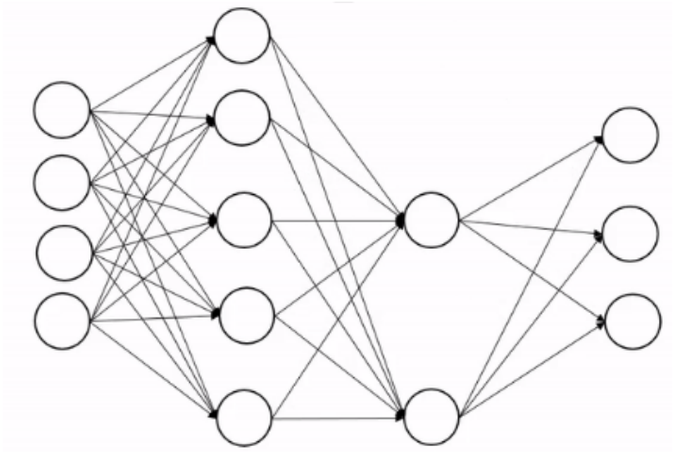
- Deep Learning Components
  - Fully connected layer
  - Computation graph
  - Activation functions
  - Loss function
  - Model optimization
    - Gradient descent
    - Backpropagation
    - Stochastic gradient descent (SGD)
  - Regularization
    - ℓ_2 and ℓ_1 regularization
    - Dropout
    - Batch normalization
  - Convolutional layer
  - Pooling layer
- Convolutional Neural Network (CNN)

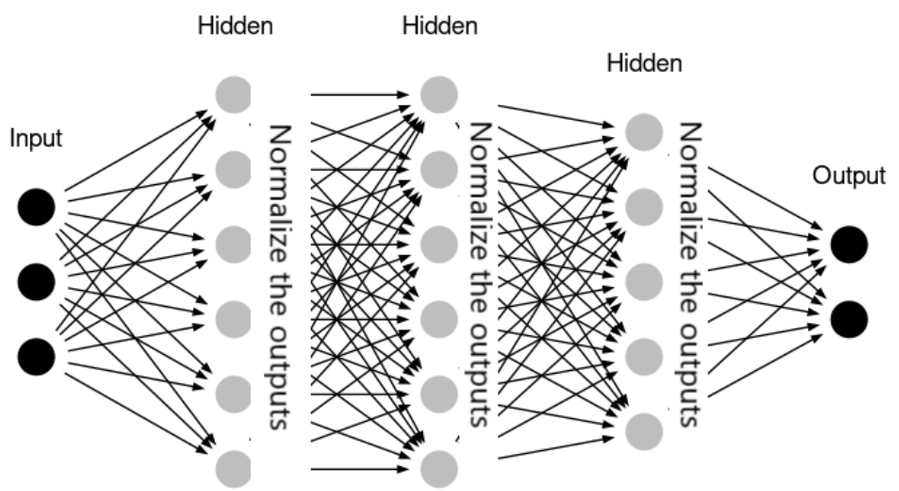


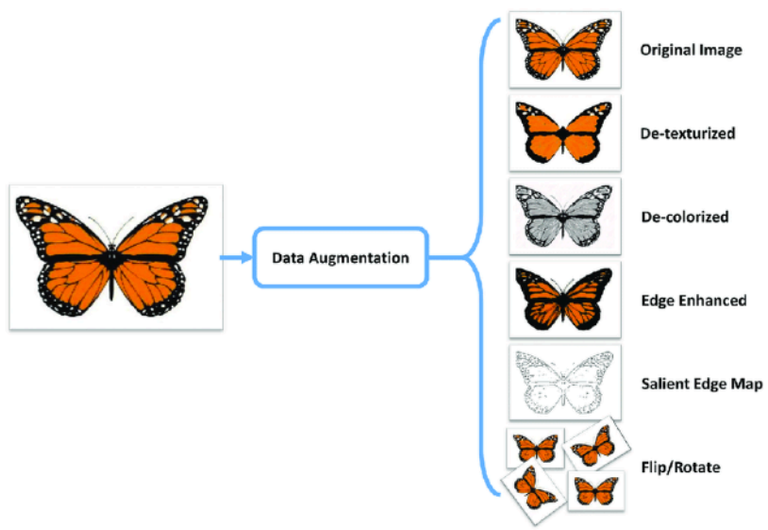


Outline

- Deep Learning Components
  - Fully connected layer
  - Computation graph
  - Activation functions
  - Loss function
  - Model optimization
    - Gradient descent
    - Backpropagation
    - Stochastic gradient descent (SGD)
  - Regularization
    - ℓ_2 and ℓ_1 regularization
    - Dropout
    - Batch normalization
  - Convolutional layer
  - Pooling layer
- Convolutional Neural Network (CNN)

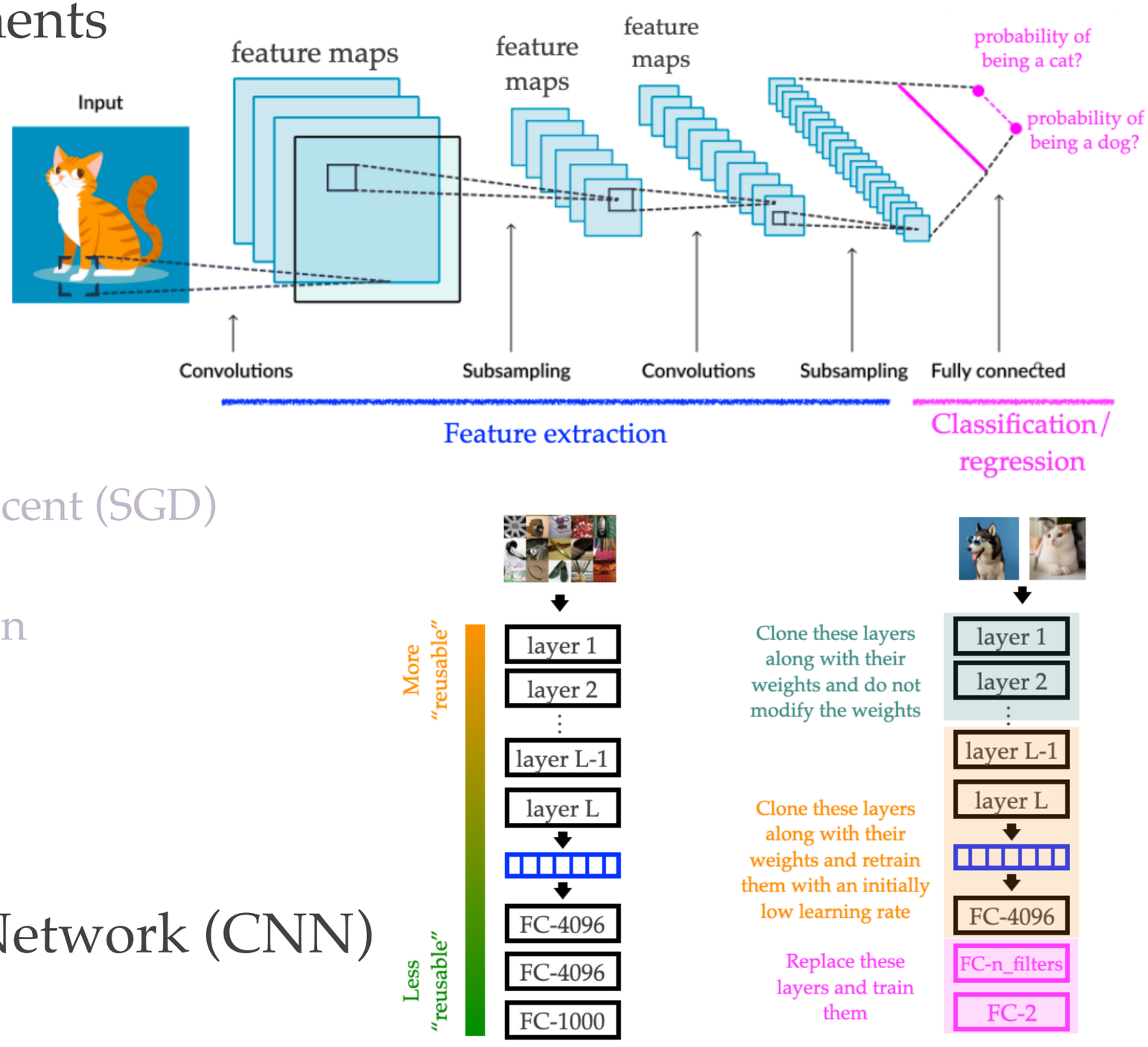


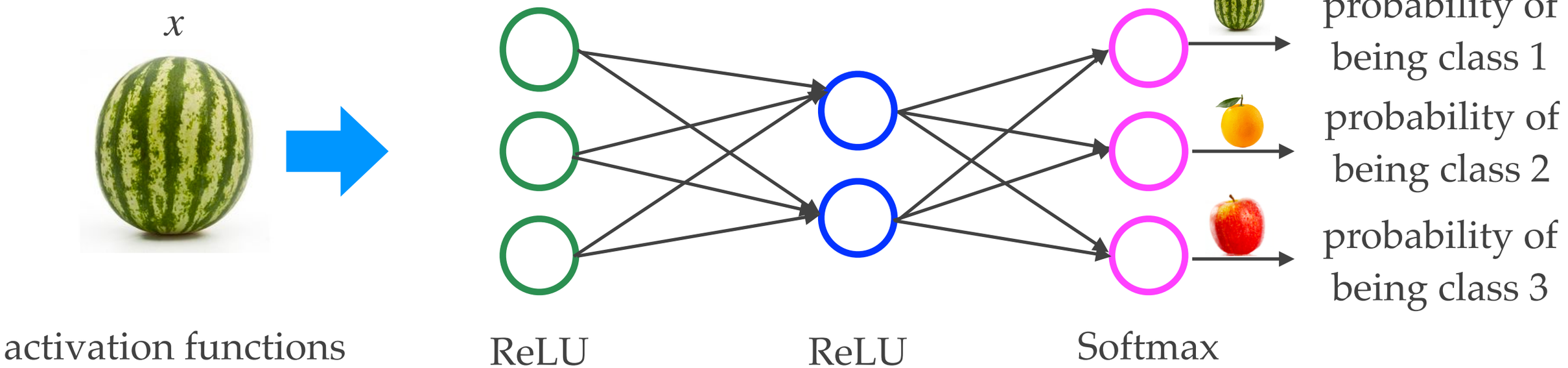




Outline

- Deep Learning Components
  - Fully connected layer
  - Computation graph
  - Activation functions
  - Loss function
  - Model optimization
    - Gradient descent
    - Backpropagation
    - Stochastic gradient descent (SGD)
  - Regularization
    - ℓ_2 and ℓ_1 regularization
    - Dropout
    - Batch normalization
  - Convolutional layer
  - Pooling layer
- Convolutional Neural Network (CNN)



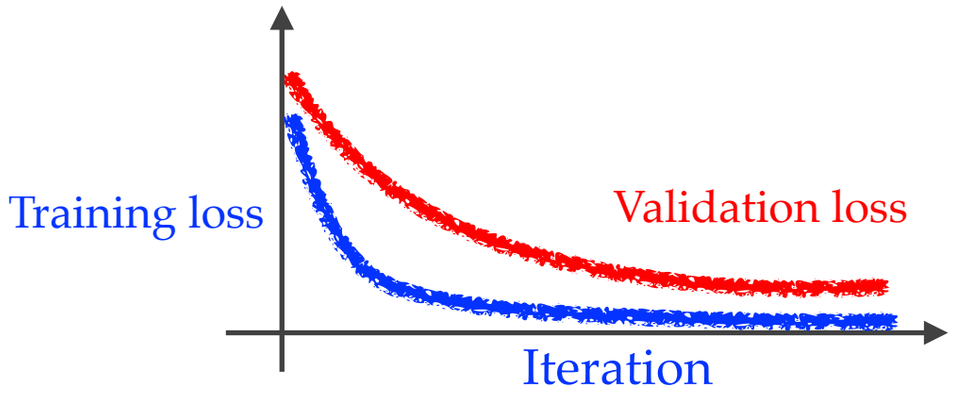



```
# Import necessary modules
import tensorflow as tf
from tensorflow import keras
from tensorflow.keras import layers

# Create the model
model = keras.Sequential()
model.add(layers.Dense(3, activation="relu"))
model.add(layers.Dense(2, activation="relu"))
model.add(layers.Dense(3, activation="softmax"))

# Compile the model
model.compile(
    optimizer='adam',
    loss=tf.keras.losses.CategoricalCrossentropy(),
)

# Train the model for 100 epochs with a batch size of 32
model.fit(x_train, y_train, batch_size=32, epochs=100, validation_data=(x_val,y_val))
```



Model

Loss function and optimizer