AI 기반 영상 데이터 분석 실습
- Day 3 -

Course overview

Course overview

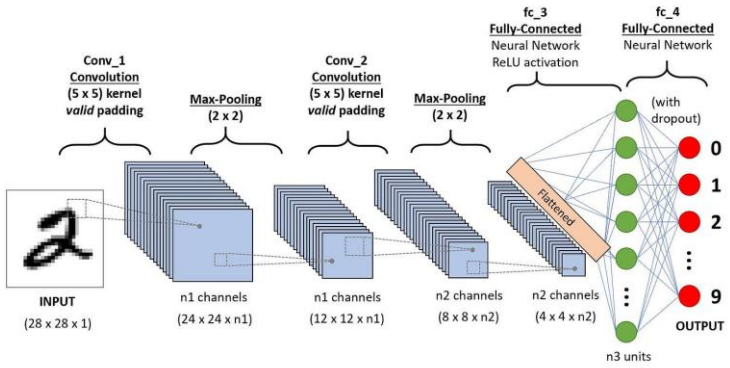
Day	Morning session (이론)	Afternoon Lab session (실습)
Jan. 19 (Mon)	• 인공지능 개요 • Vision task 소개 • MNIST Classification review	• GitHub 활용법 • 주제 선정 • 데이터 수집
Jan. 20 (Tue)	• 영상처리 기초 • 이미지 전처리 기법	• 전처리, 증강, 시각화 • Dataset split • Data set class & Data loading
Jan. 21 (Wed)	• Fundamentals on CNN • Basic architectures	• CNN architecture 구현 (Resnet)
Jan. 22 (Thu)	• Weight update (Gradient descent, Optimizers) • Loss monitoring	• Torch model • 아키텍처 선택 및 구현 (Resnet + @) • 모델 학습 및 하이퍼파라미터 튜닝
Jan. 26 (Mon)	• Model evaluation • Grad-CAM	• 모델 평가 • Grad-CAM 실습
Jan. 27 (Tue)	• Github에 프로젝트 업로드 • 프로젝트 발표	

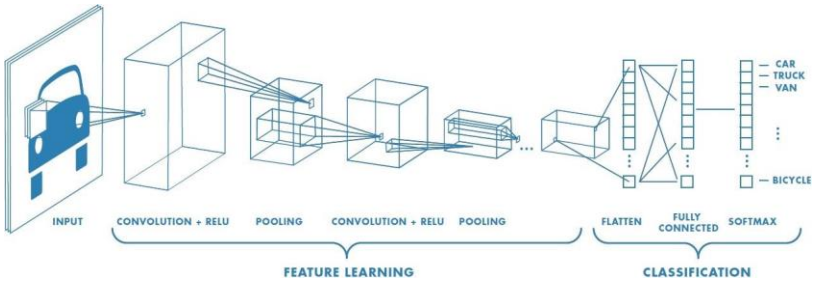
CNN Architecture Overview

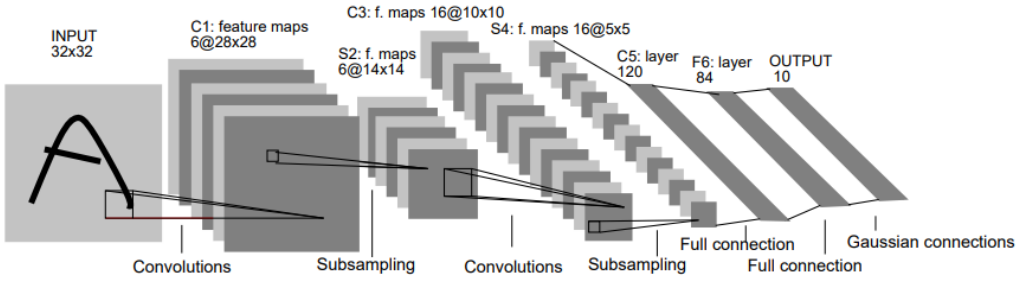
Convolutional Neural Networks

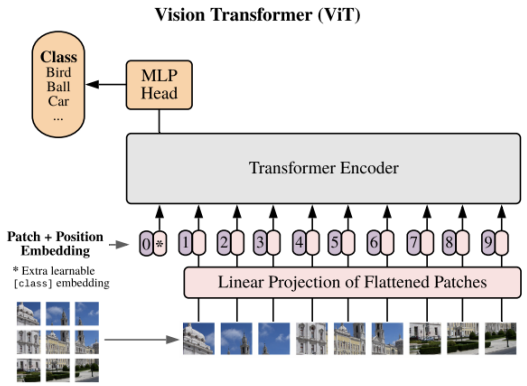
Definition of CNN

- Convolutional neural networks are neural networks that use convolution in place of general matrix multiplication (or weighted summation) in at least one of their layers.





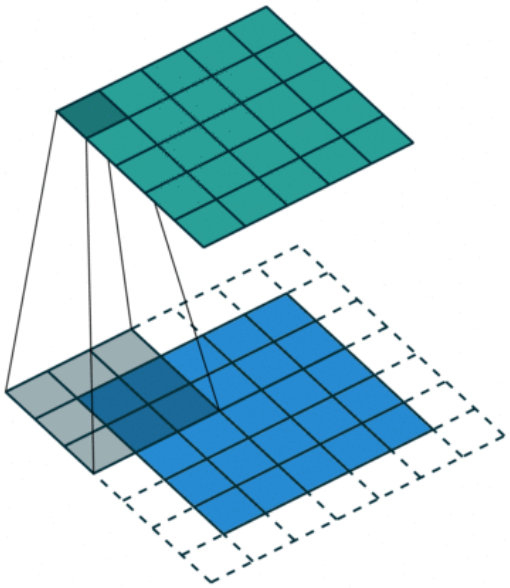


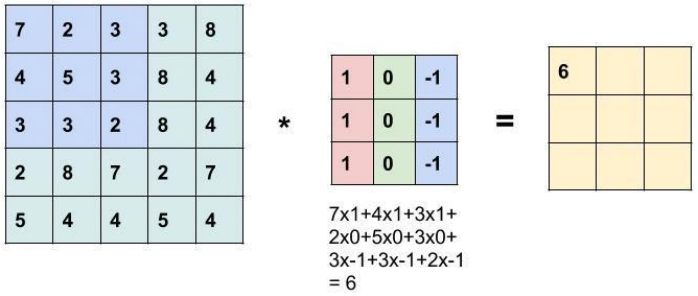


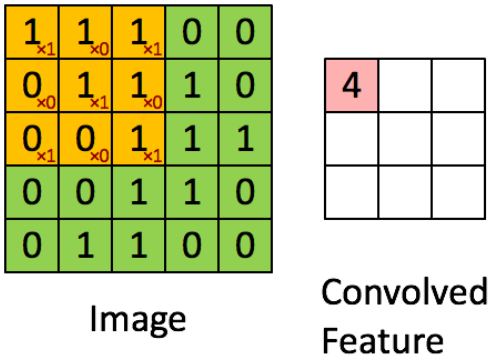
Convolutional Neural Networks

Basic operation of CNN

- Unlike basic neural networks that rely on weighted summation as their core operation, CNNs utilize the **convolution** operation.







Convolutional Neural Networks

Convolutional Neural Networks

- Very similar to ordinary Neural Networks
  - Composed of neurons with learnable weights and biases
  - Each neuron performs a dot product + non-linearity activation
  - Map raw image pixels → class scores through a differentiable score function
  - Trained with a loss function
  - Use the same optimization and training tricks as regular neural networks

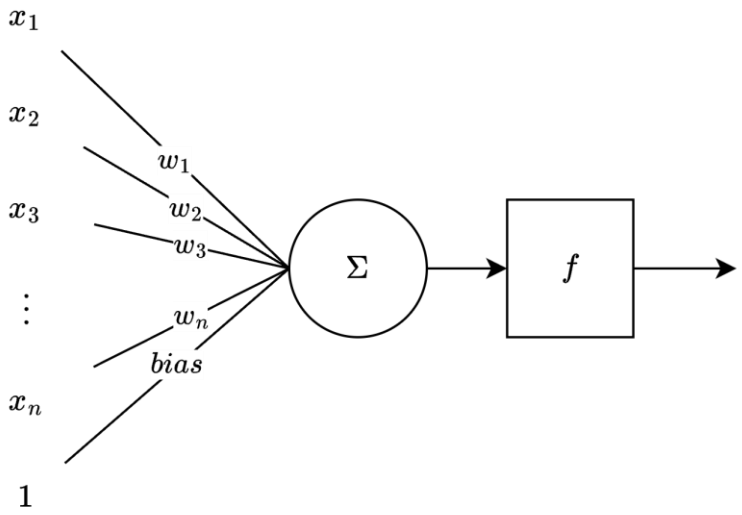
Convolutional Neural Networks

Convolutional Neural Networks

- **Key difference**
  - CNNs make the explicit assumption that inputs are images
  - CNN architectures
    - Encode spatial structure of images
    - Reduce the number of parameters
    - Improve **computational efficiency** during forward propagation

Functioning of artificial neuron

Artificial neuron (Neuron, Perceptron, Node)



$$y = f\left(\sum_i w_i x_i + bias\right)$$

$$y = f(\boldsymbol{w}^T \boldsymbol{x})$$

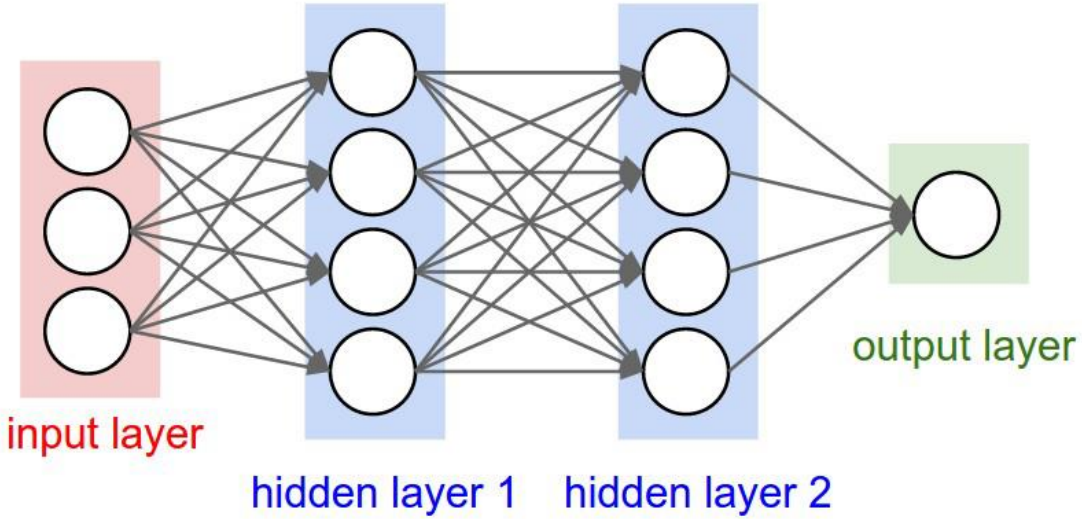
$$\boldsymbol{x} = [x_1, \ldots, x_n, 1]$$

$$\boldsymbol{w} = [w_1, \ldots, w_n, bias]$$

CNN Architecture Overview

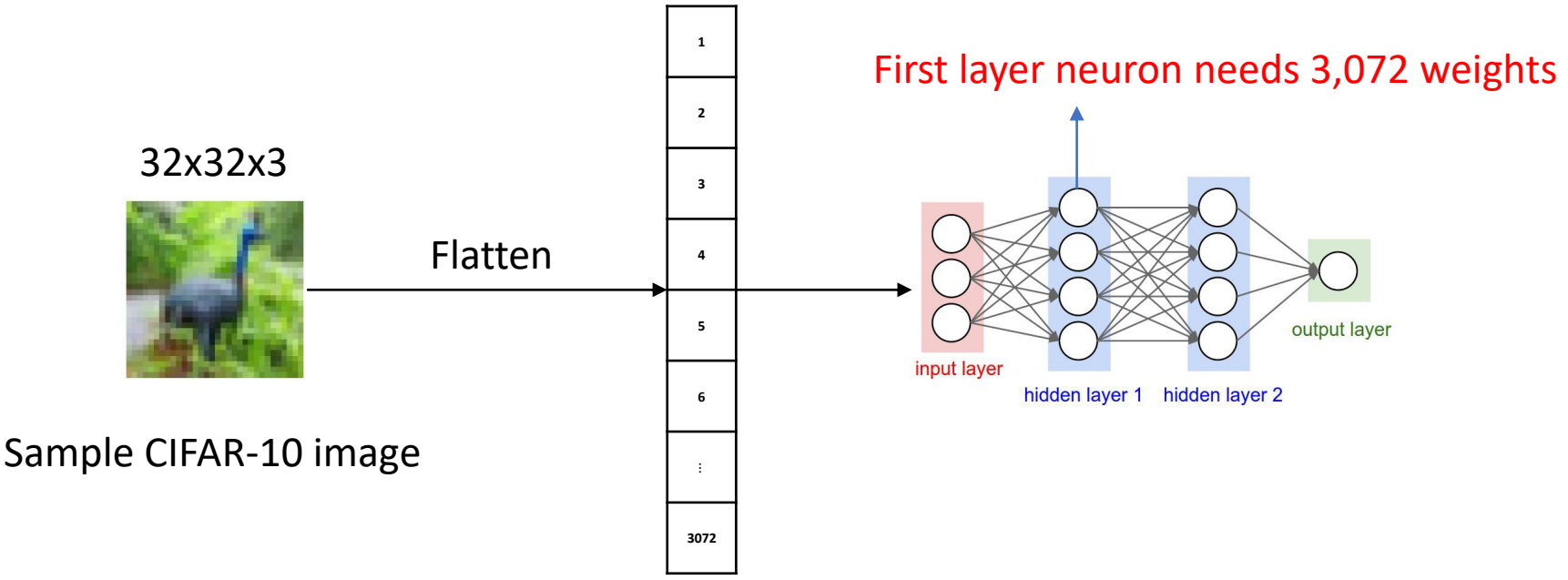
Regular Neural Networks

- Take a single vector input and pass it through fully connected layers
- Each neuron is connected to all neurons in the previous layer
- Neurons within a layer work independently (no shared connections)
- The last fully connected layer is the output layer that gives class scores



CNN Architecture Overview

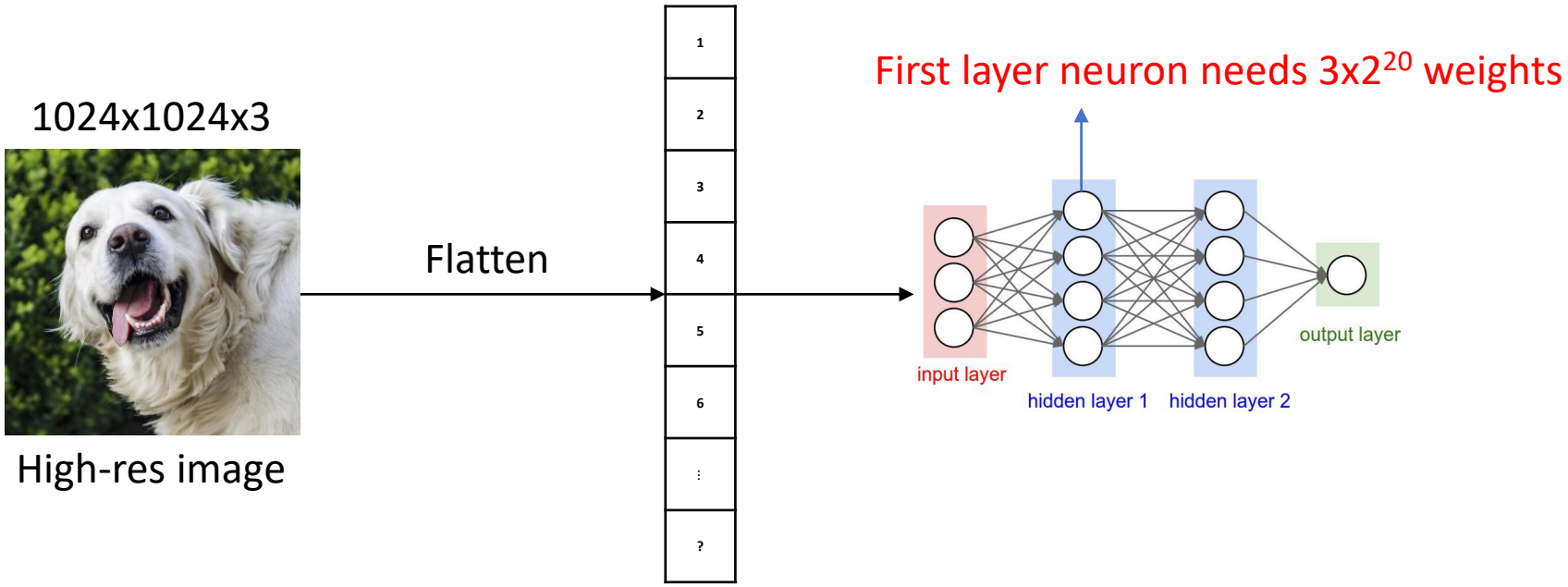
Regular Nets Don’t Scale to Images



Sample CIFAR-10 image

CNN Architecture Overview

Regular Nets Don't Scale to Images



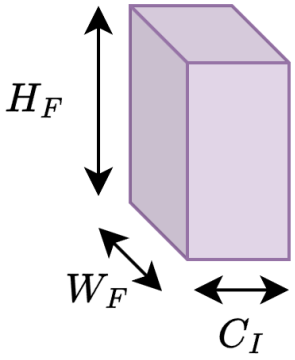
CNN Architecture Overview

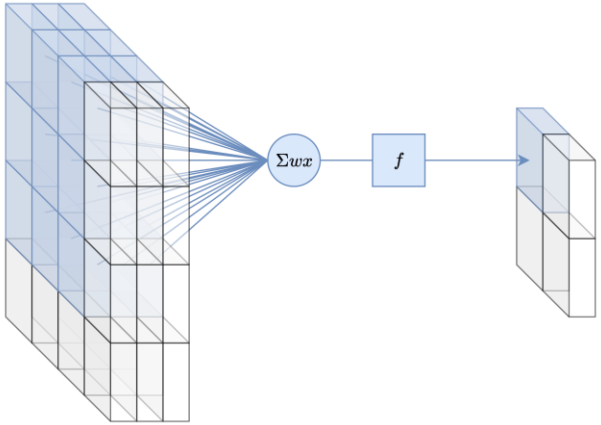
Regular Nets Don’t Scale to Images

- **Fully connected structure** causes
  - Too many parameters overfitting
  - Inefficient computation
- CNNs solve this by using **local connectivity + parameter sharing**

CNN Architecture Overview

CNNs Use 3D Volumes of Neuron weights





- CNN layers arrange neurons' weights in 3D volumes: width × height × depth
  - Neuron's weights → weights of convolutional filter
- **Each neuron connects to only a small local region** (not fully connected)
- Output volume gradually shrinks to 1×1×#classes (e.g., 1×1×10 for CIFAR-10)
- This design reduces parameters and preserves spatial structure

Layers used to build CNN

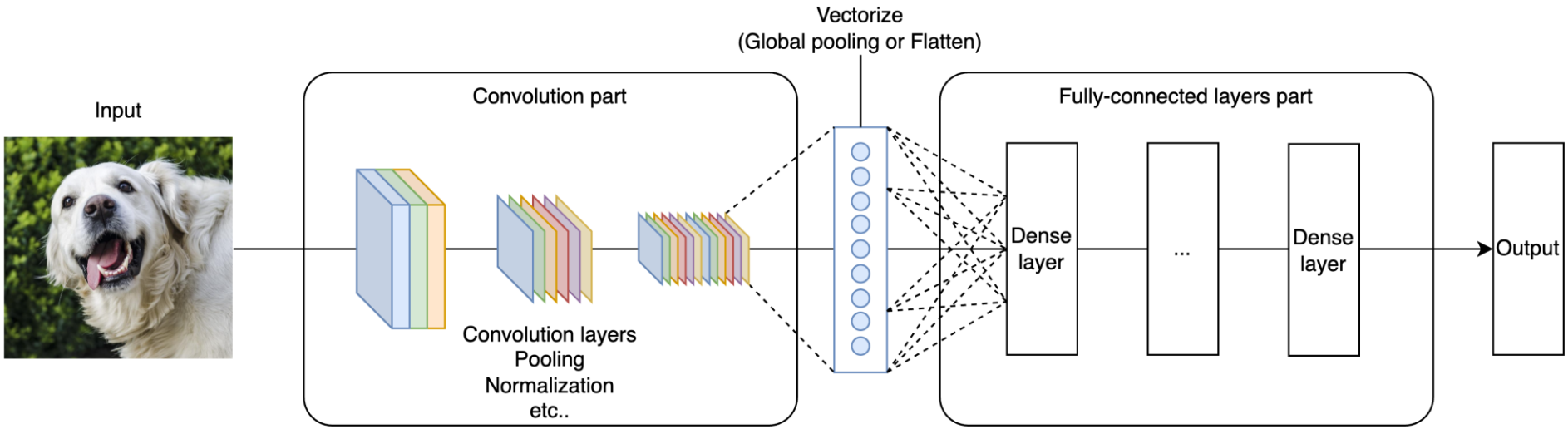
Basic idea

- A CNN is a sequence of layers, each transforming one activation volume to another via a differentiable function (similar with regular neural nets)
- Main layer types:
  - **Convolutional (CONV)**
  - **Pooling (POOL)**
  - **Fully Connected (FC)**
- ReLU is an activation function often applied after CONV/FC layers

Layers used to build CNN

Example architecture

- Typical structure of CNN



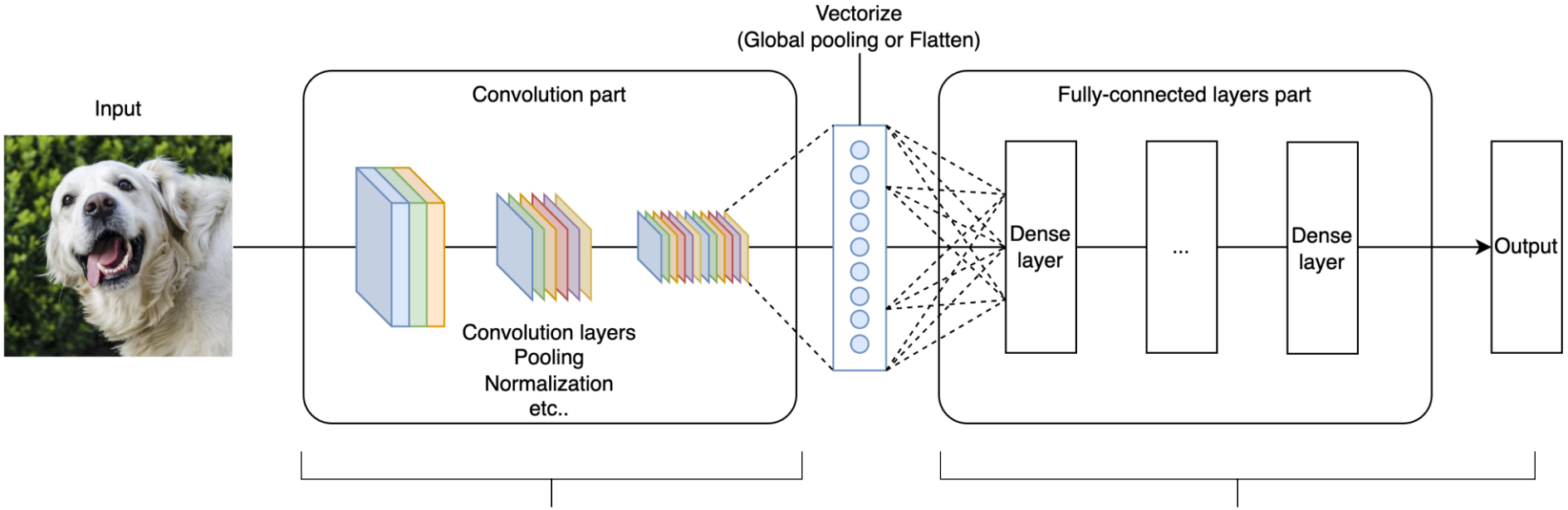
- **CONV / FC layers**
  - Contain trainable parameters (weights & biases)
  - Updated using gradient descent

- **ReLU / POOL layers**
  - Apply fixed functions
  - Contain no trainable parameters

Layers used to build CNN

Example architecture

- Typical structure of CNN



Convolution and **pooling** operators extract features while respecting 2D image structure

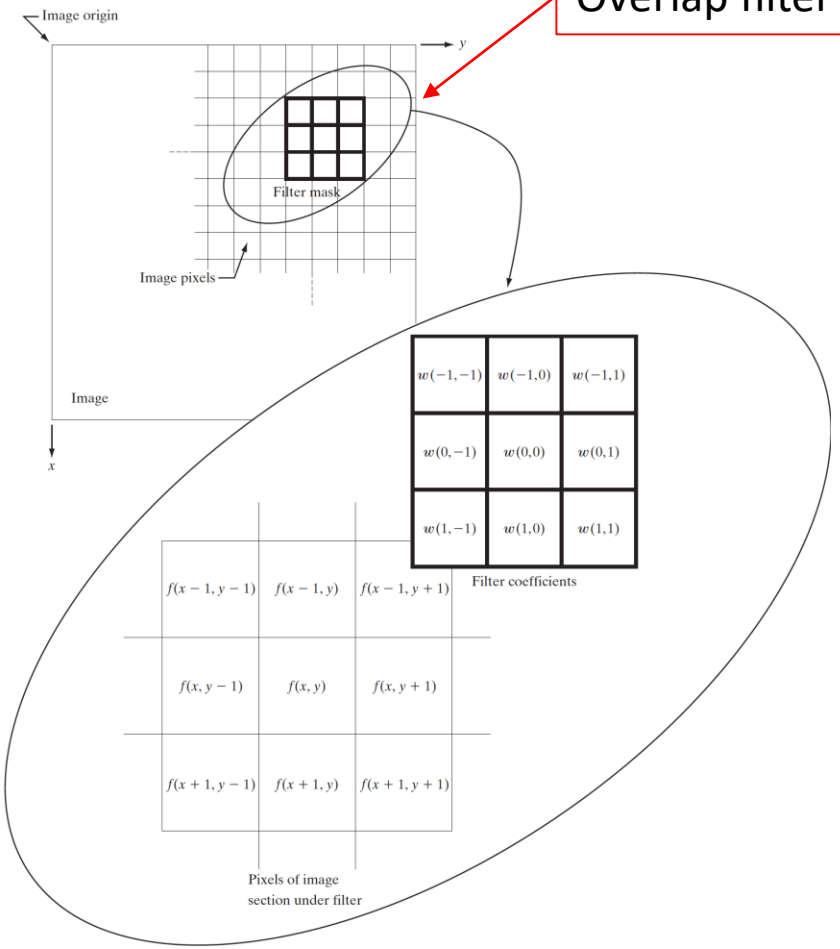
Fully-Connected layers form an MLP at the end to predict scores

Fundamentals of Convolutional Neural Network

Prerequisites - Fundamentals of Spatial filtering

The mechanics of spatial filtering

Overlap filter on the image



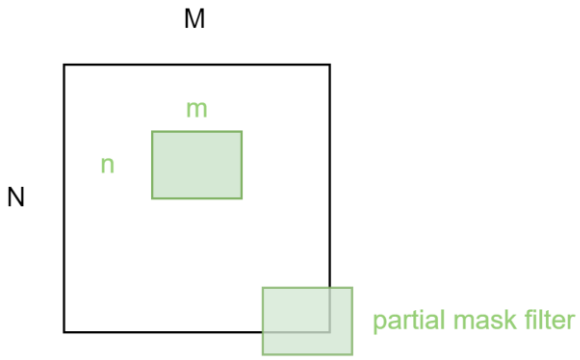
Filtered image g(x)

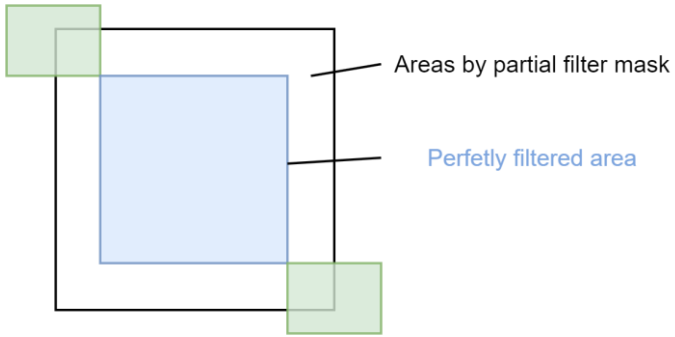
$$g(x, y) = \sum_{s=-a}^{a} \sum_{t=-b}^{b} w(s, t) f(x + s, y + t)$$

$f(x_0,y_0) \rightarrow g(x_0,y_0)$

Prerequisites - Fundamentals of Spatial filtering

The mechanics of spatial filtering





In general, images get small after filtering operation

Prerequisites - Fundamentals of Spatial filtering

Vector representation of linear filtering

w_1	w_2	w_3
w_4	w_5	w_6
w_7	w_8	w_9

3x3 filter mask

$$R = w_1 z_1 + w_2 z_2 + \ldots + w_9 z_9$$
$$= \sum_{k=1}^{9} w_k z_k$$
$$= \mathbf{w}^T \mathbf{z}$$

Where, z is corresponding pixel intensity

→ We already know that artificial neurons perform similar operation

Prerequisites - Fundamentals of Spatial filtering

Vector representation of linear filtering

w_1	w_2	w_3
w_4	w_5	w_6
w_7	w_8	w_9

3x3 filter mask

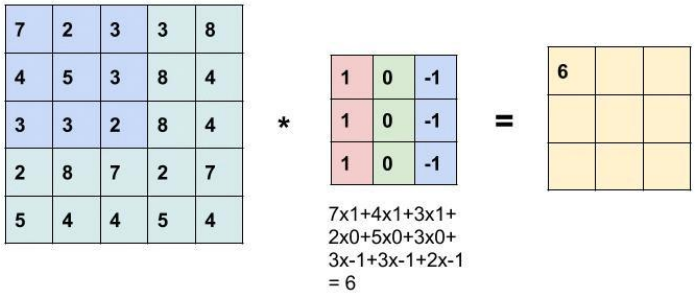
$$R = w_1 z_1 + w_2 z_2 + \ldots + w_9 z_9$$
$$= \sum_{k=1}^{9} w_k z_k$$
$$= \mathbf{w}^T \mathbf{z}$$

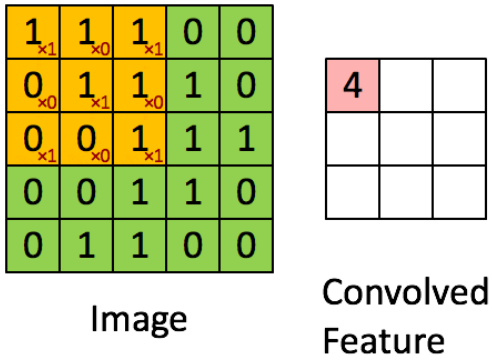
Where, z is corresponding pixel intensity

- **We also call this as Image Convolution**
- Filter slides (convolves) over the input's width & height
- We can extraction image features through this (ex, edge, texture, color etc.)
  → Frequently used in conventional computer vision / image processing

Prerequisites - Fundamentals of Spatial filtering

Vector representation of linear filtering





Convolutional layer

Convolutional layer

Convolutional Layer

- Core building block of a Convolutional Neural Network
  - Consists of a set of learning filters
  - Performs the main computational work in a CNN
- Learns filters (kernels) to detect patterns in input images
- Produces feature maps that represent the presence of learned features

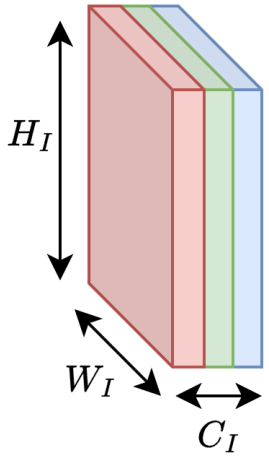
w_1	w_2	w_3
w_4	w_5	w_6
w_7	w_8	w_9

Convolutional layer

Input of convolutional layer

- Input of convolutional layer has size of

$$(Width) \times (Height) \times (Channel)$$



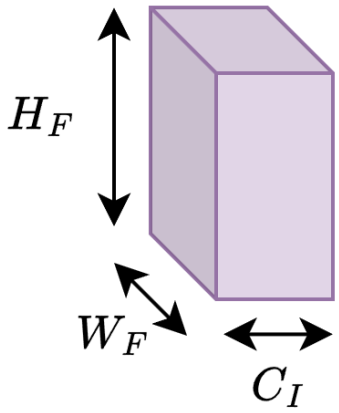
- If input of layer is color image size of 64x64
  → $W_I = 64, H_I = 64, C_I = 3$

Convolutional layer

Convolutional filter (kernel)

- Convolution filter (kernel) of convolutional layer has shape of

$$(Width) \times (Height) \times (C_I)$$

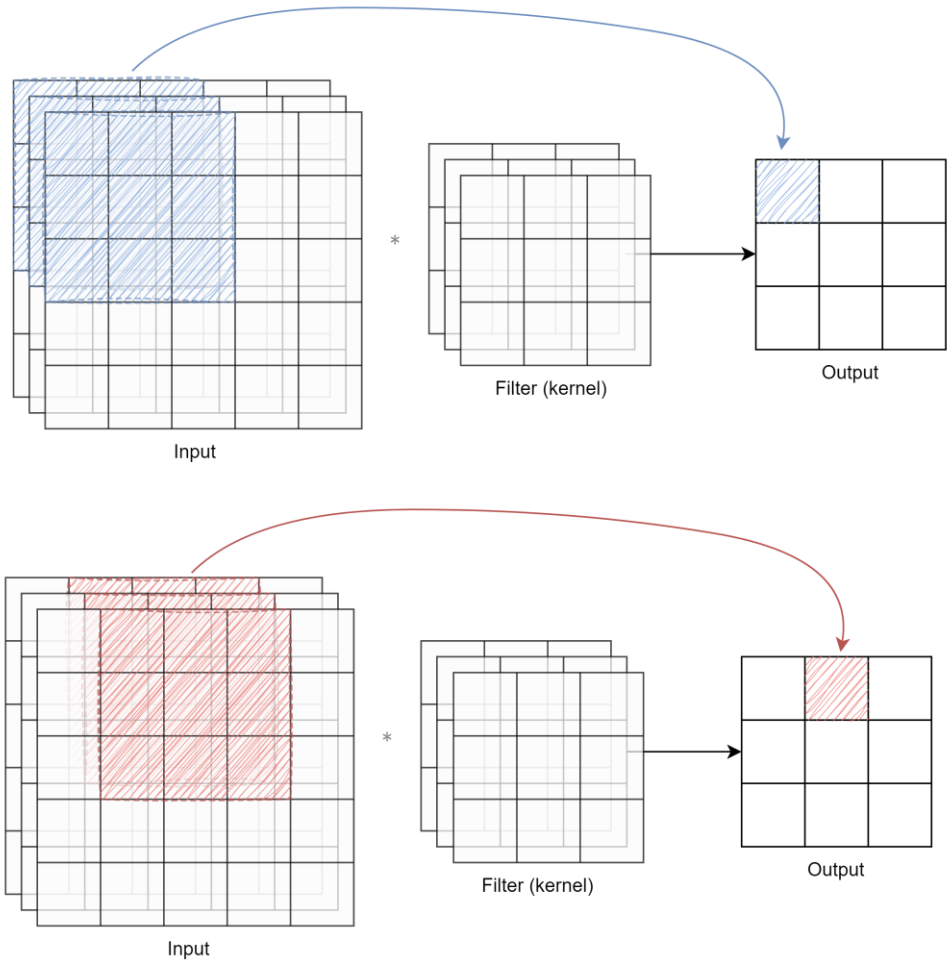


- The number of channel (i.e., depth) of convolution filter is same with the input's channel.

- Width and Height of filter are normally odd numbers (because of convolution), and should be smaller than (at least same with) input's width and height

Convolutional layer

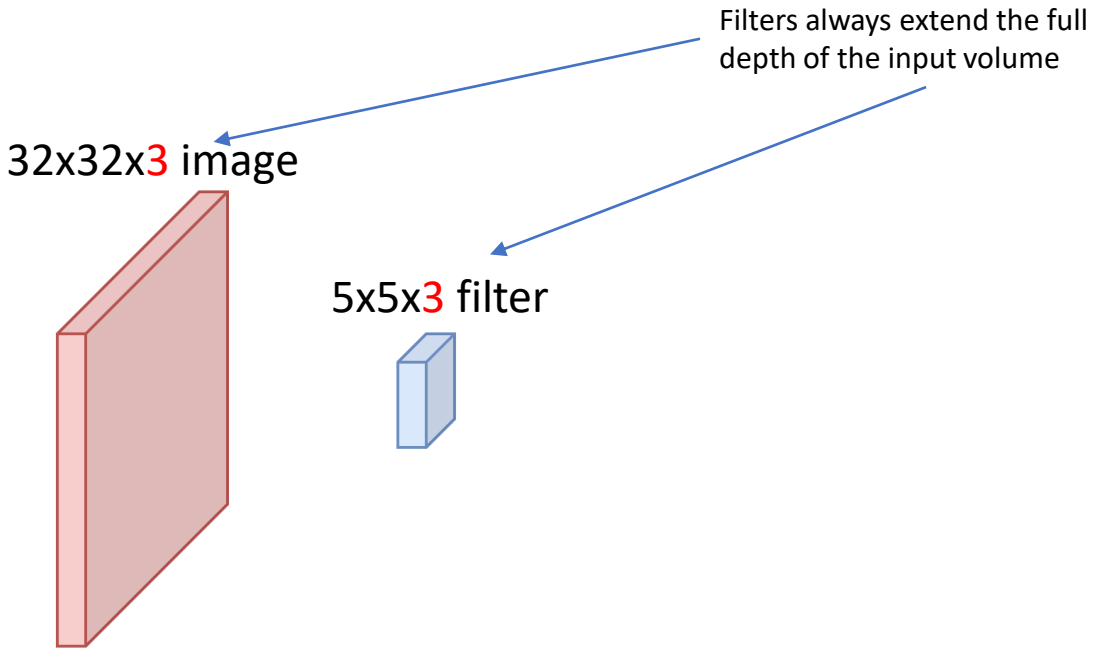
How the CONV layer works



- During forward pass:
  - Filters slide (convolve) over the input's width & height
  - Compute dot products at each spatial position
- Each filter produce a 2D feature map
- Multiple filters
  → 2D feature maps are stacked along depth
  → output 3D volume

Convolutional layer

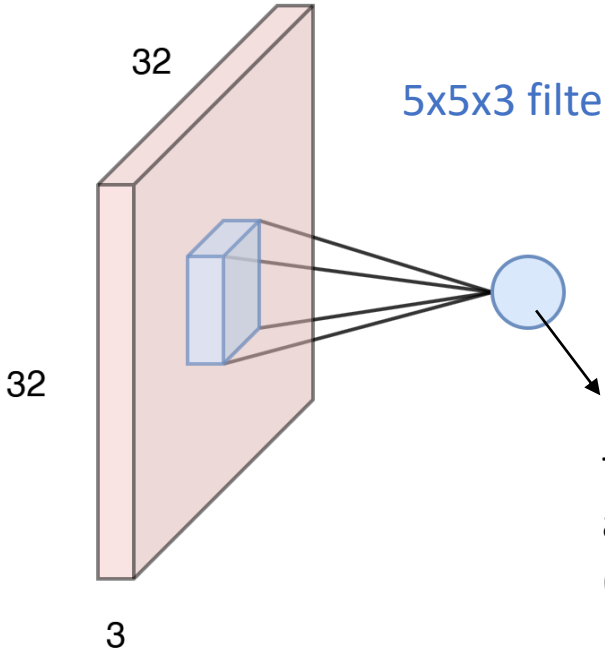
How the CONV layer works



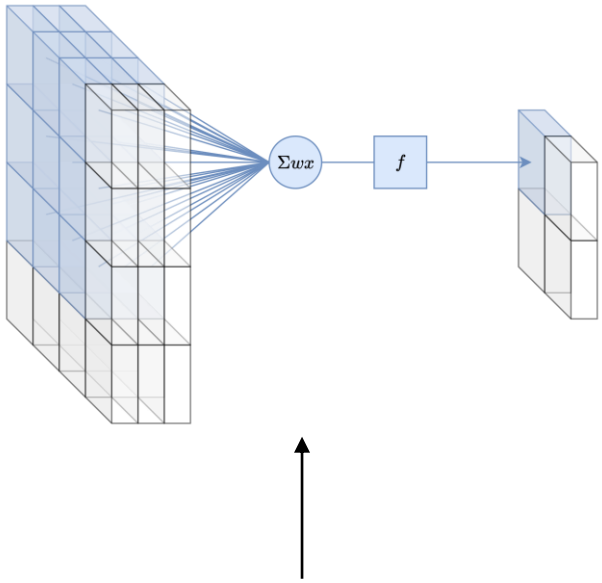
Convolutional layer

How the CONV layer works

32x32x3 image



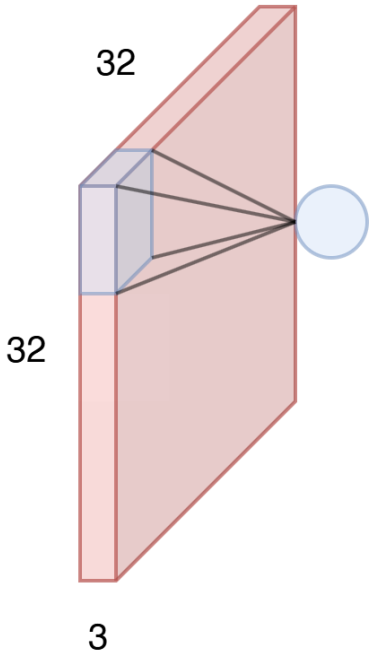
5x5x3 filter w



1 number:
the result of taking a dot product between the filter and a small 5x5x3 chunk of the image
(i.e. 5*5*3 = 75-dimensional dot product + bias)

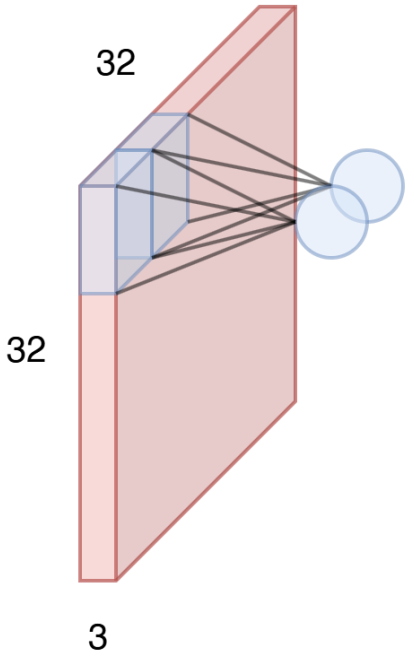
Convolutional layer

How the CONV layer works



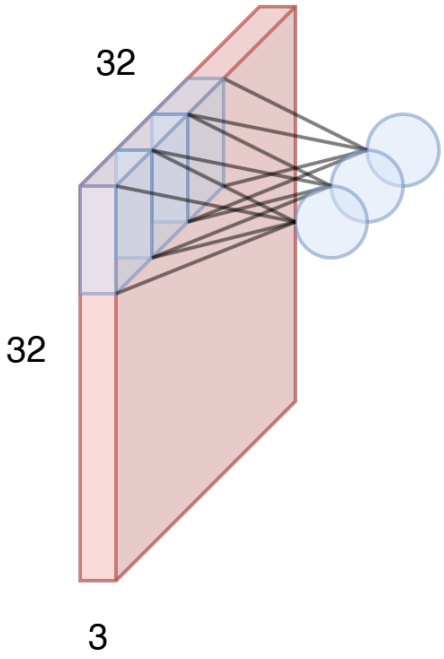
Convolutional layer

How the CONV layer works



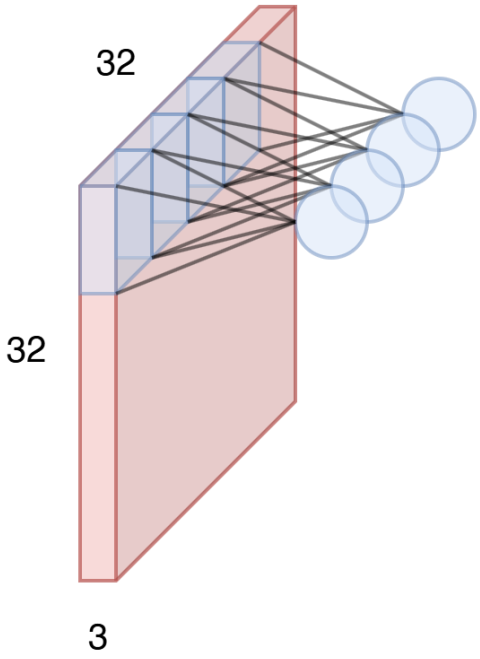
Convolutional layer

How the CONV layer works



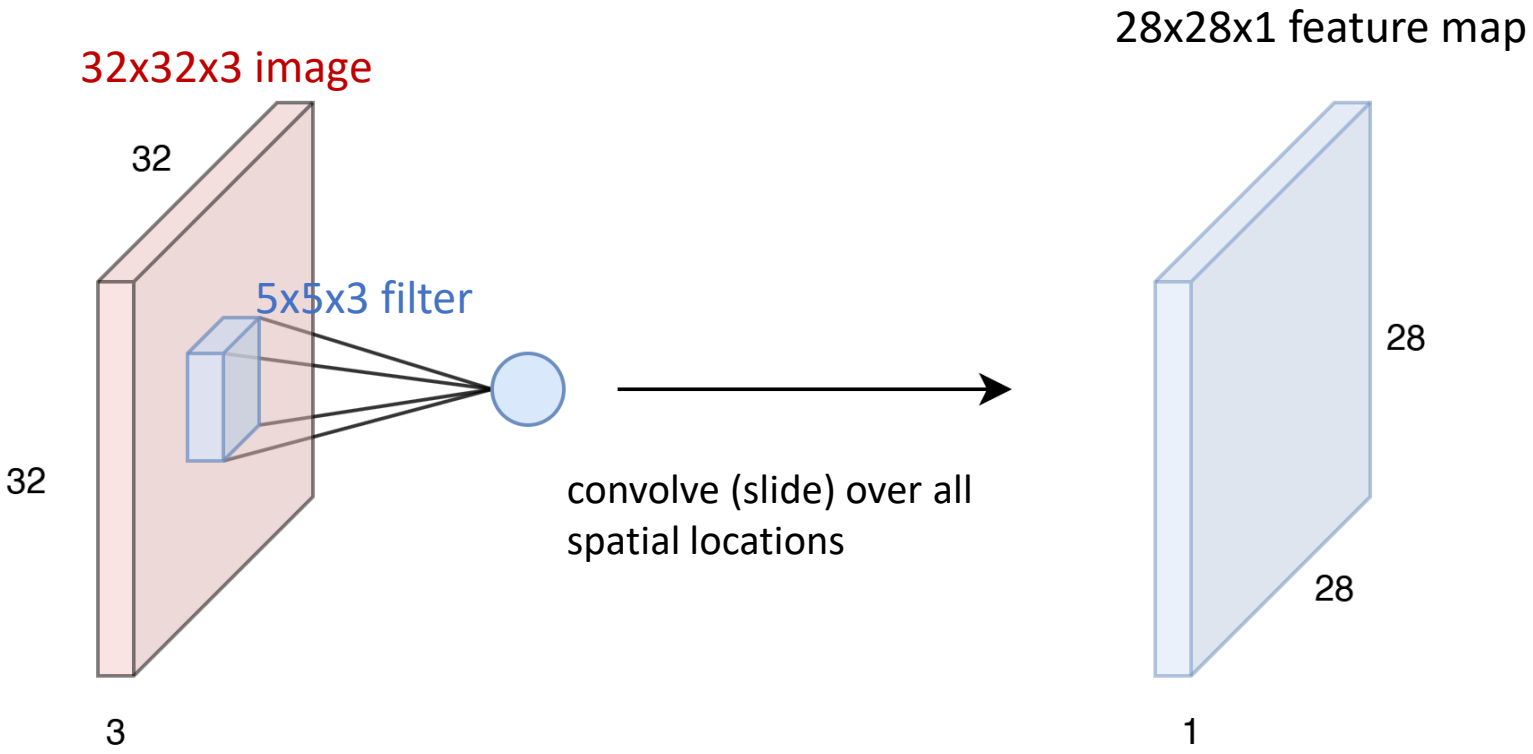
Convolutional layer

How the CONV layer works



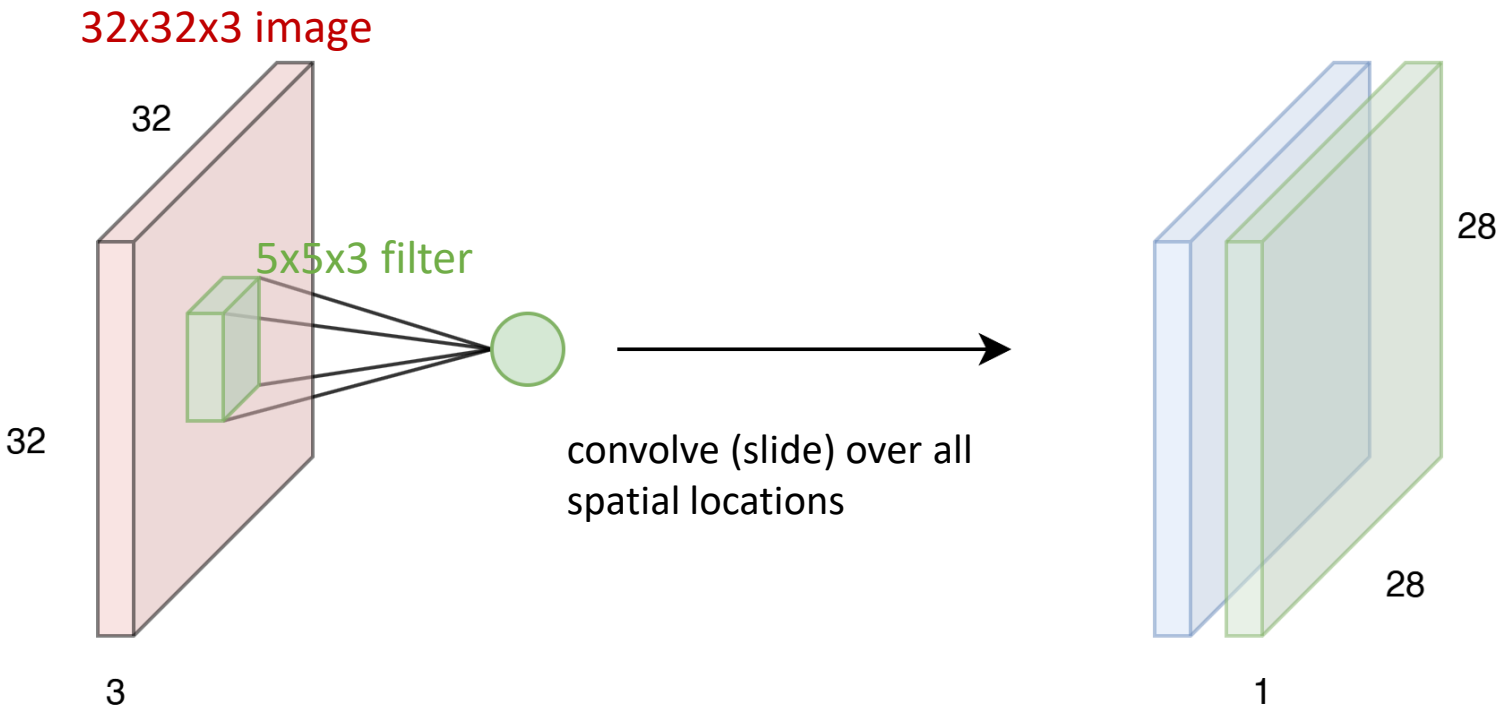
Convolutional layer

How the CONV layer works



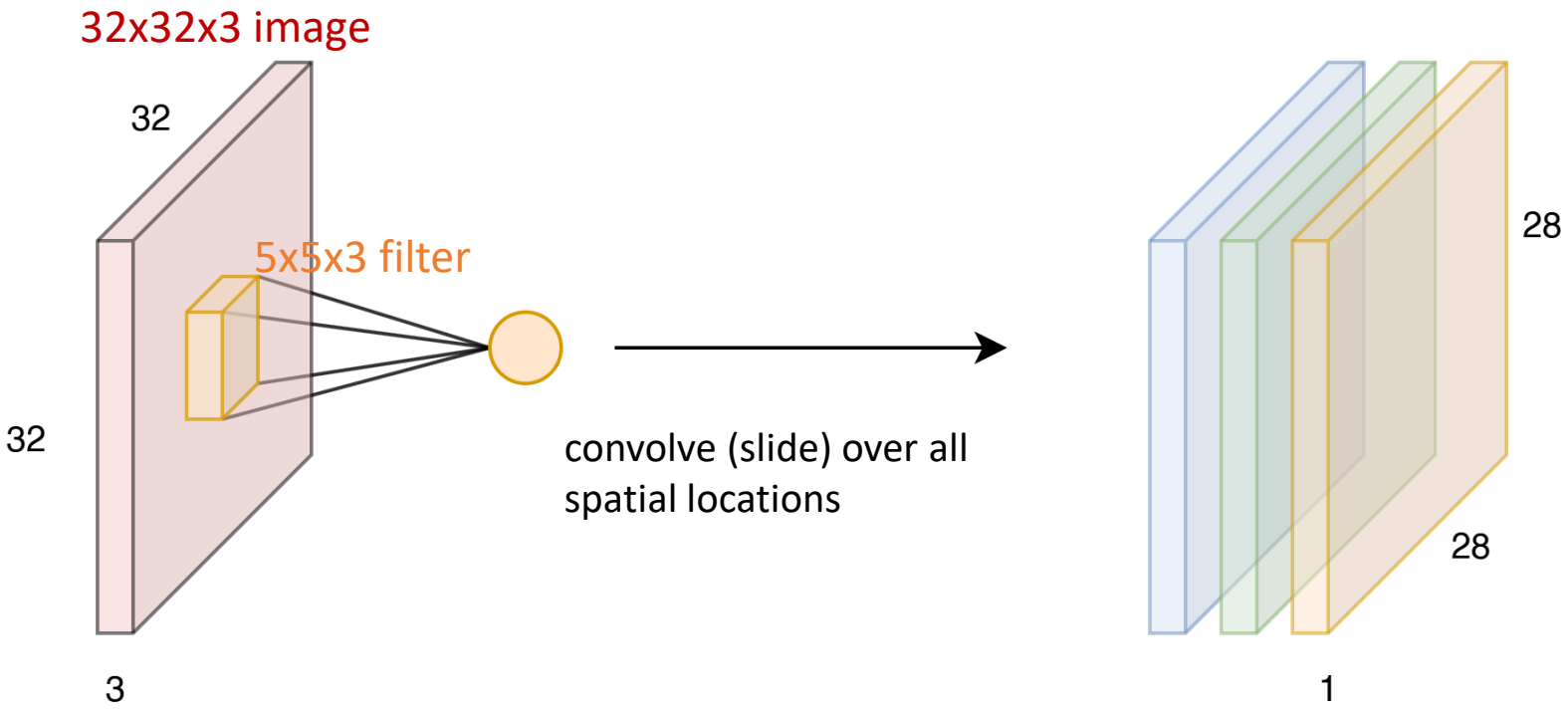
Convolutional layer

How the CONV layer works



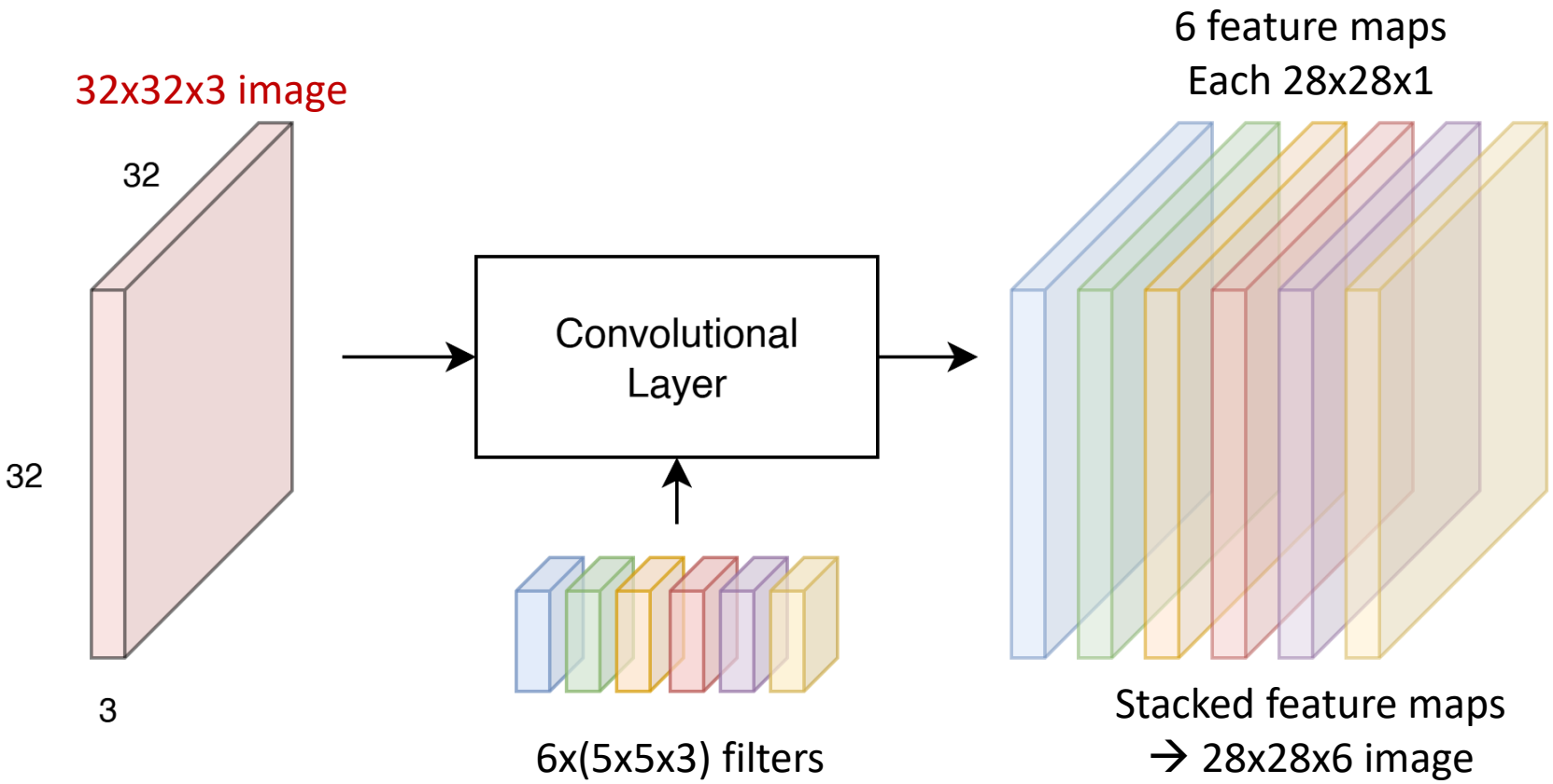
Convolutional layer

How the CONV layer works



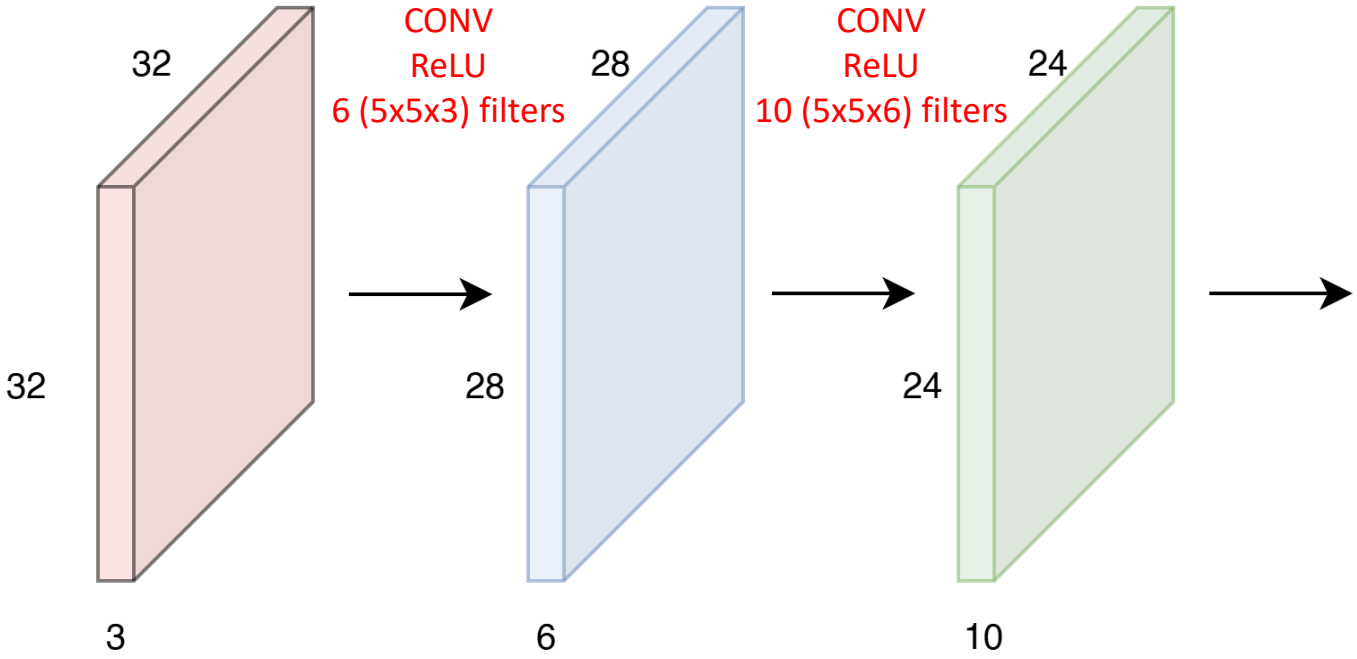
Convolutional layer

How the CONV layer works



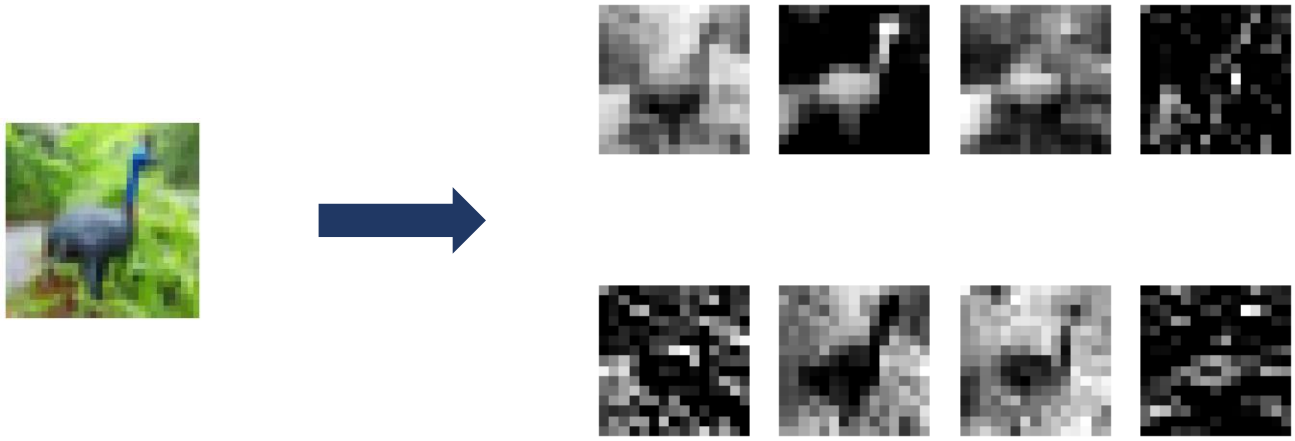
Convolutional layer

How the CONV layer works

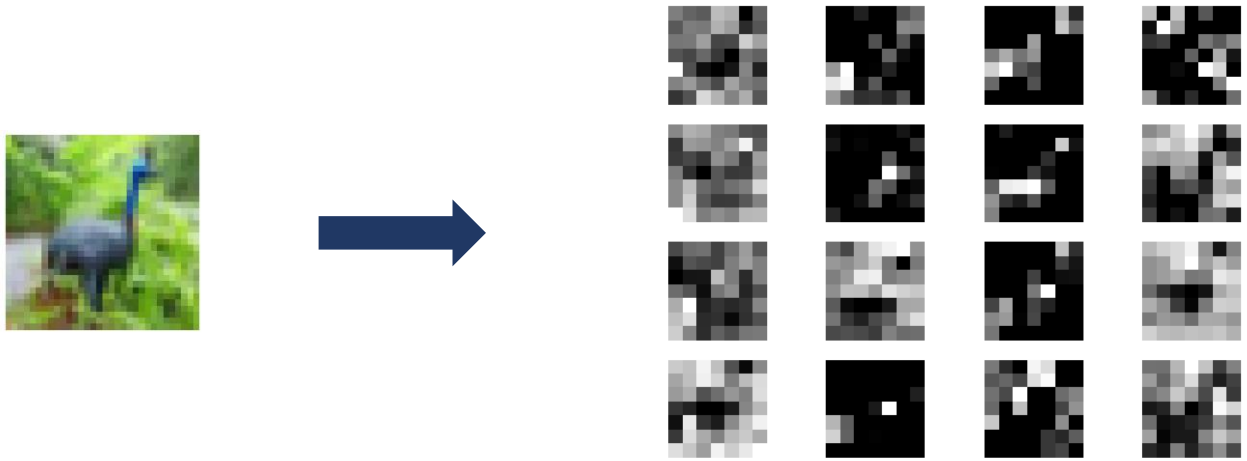


Convolutional layer

Shallow layer – low level abstraction (shape, edge, color, size)



Deeper layer – high level abstraction (complex pattern)



Stride, Padding and Pooling

Convolutional layer – Stride and Padding

Convolution in previous section

- Convolution is applied at every position in the spatial location of the feature map.

- High overlap
  - When the filter size is larger, more overlap occurs between neighboring areas.
  - Increased Computation: More overlap leads to additional computations.

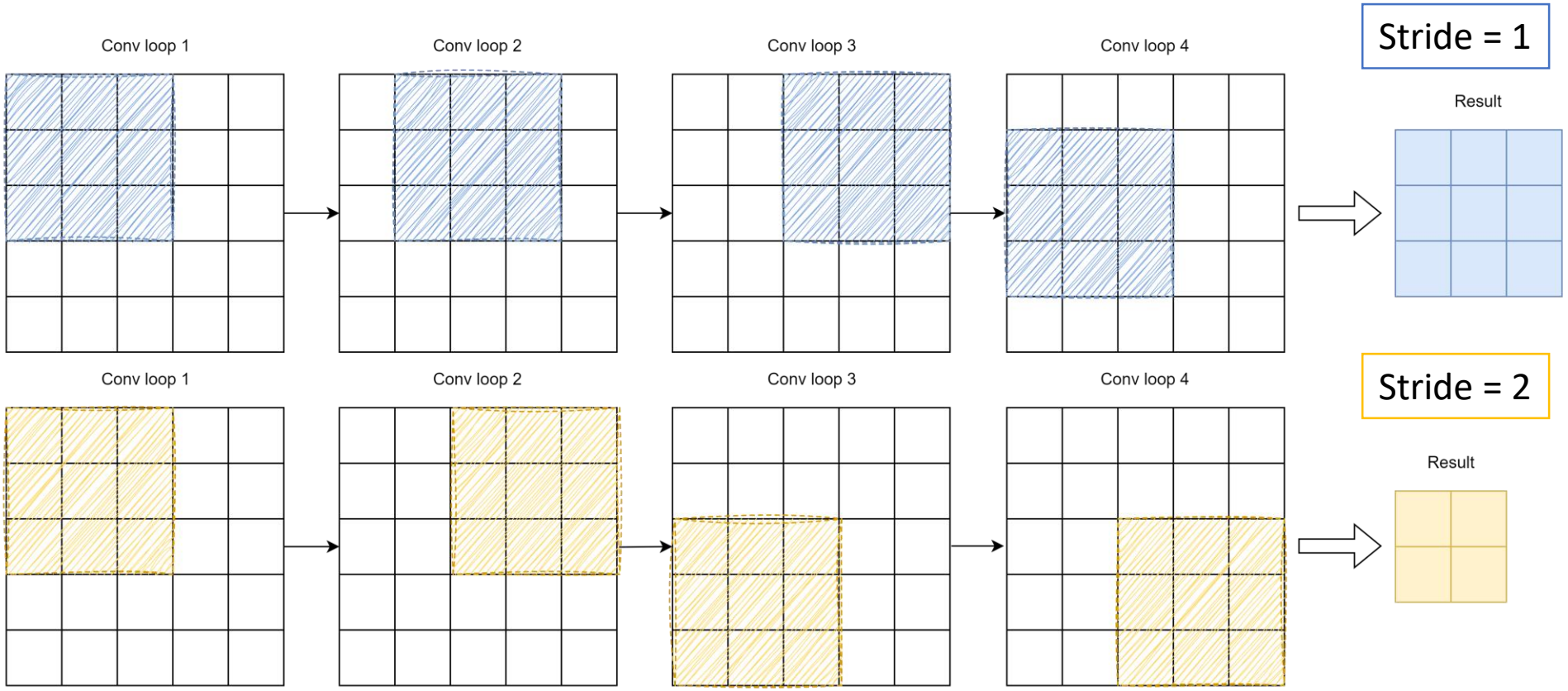
- It is not necessary

- One can reduce the level of granularity of convolution → Stride

Convolutional layer – Stride and Padding

Convolution with stride

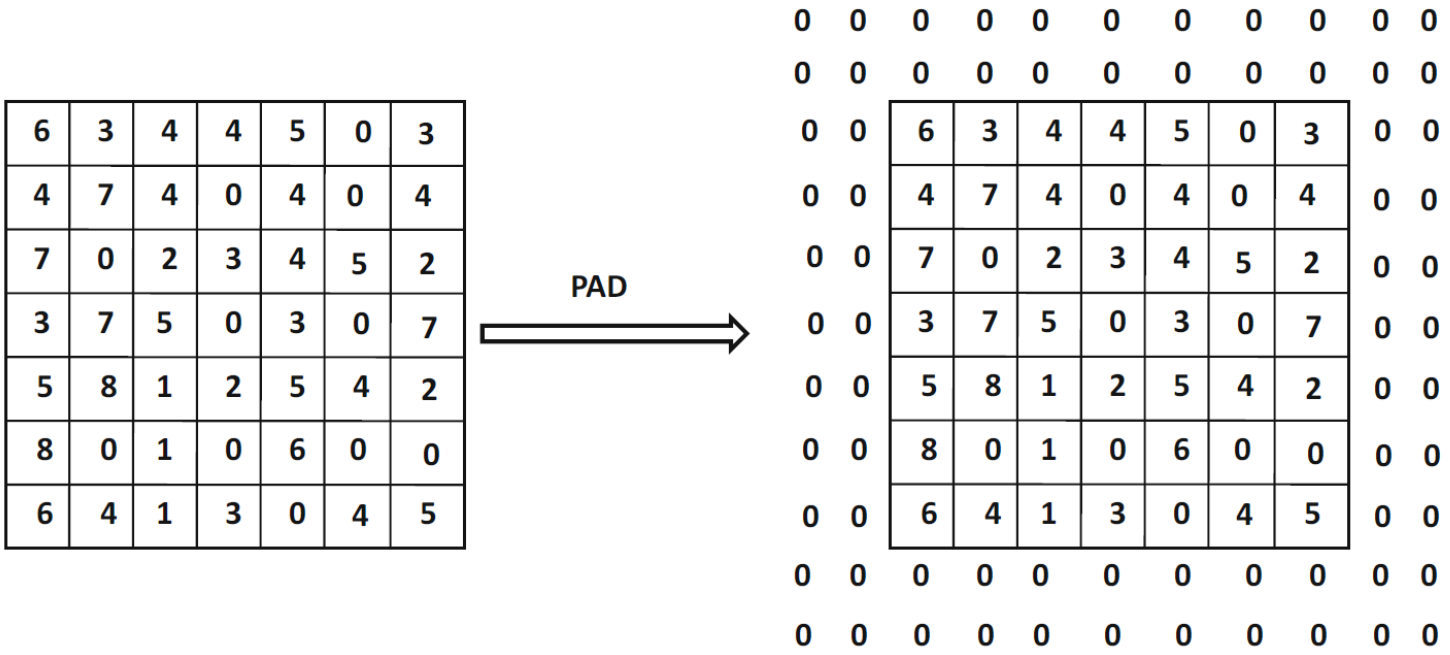
- Stride: step of sliding window → allow us to skip some positions



Convolutional layer – Stride and Padding

Padding

- Fill values (or 0s) around the image
- Prevents rapid **spatial shrinkage** through layers
  - Ex> padding = 2

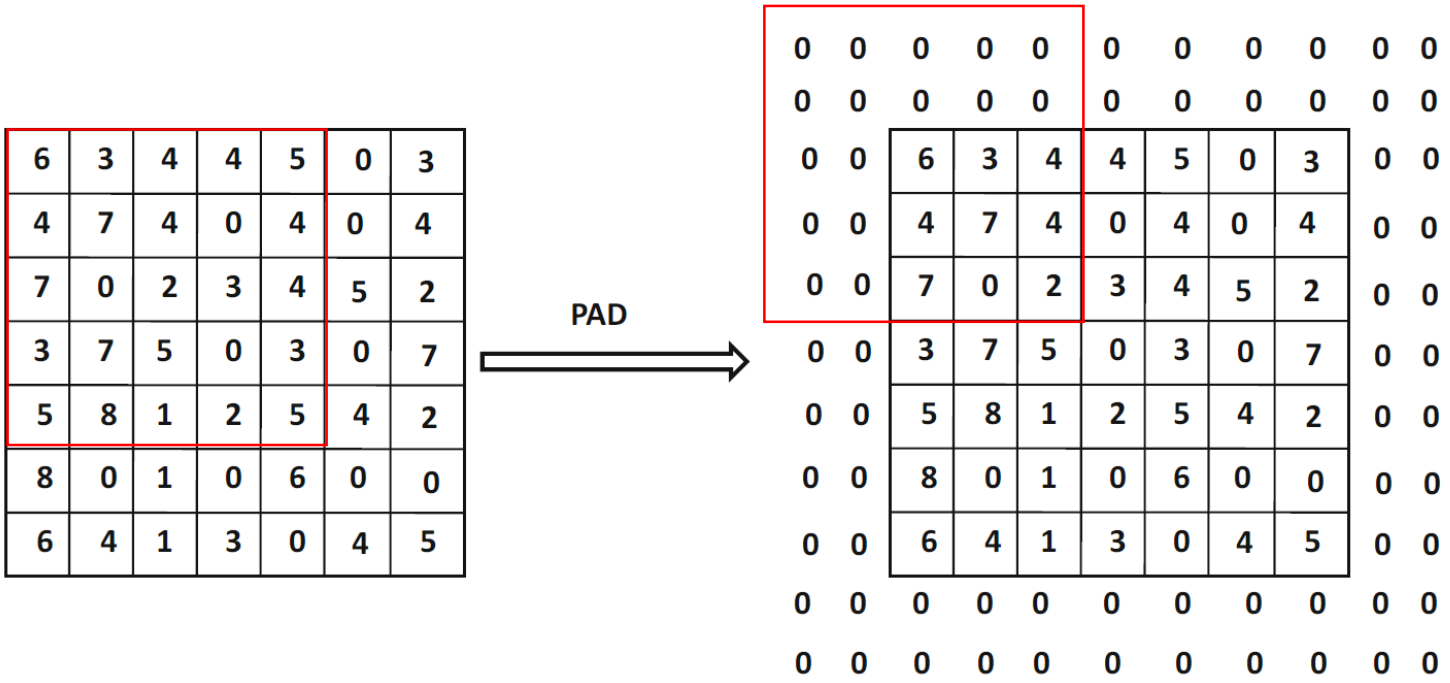


Convolutional layer – Stride and Padding

Padding

- Usually, padding is used in CNN to maintain the same spatial dimensions for the feature map as the input.

- For this, stride = 1 is used.



Pooling

Pooling layer

- A key component in convolutional neural network that performs a down sampling, reducing spatial dimensions
- A pooling function replaces the output of the net at a certain location with a summary statistic of the nearby outputs
- Representative pooling methods
  - Max pooling
  - Average pooling
  - Global pooling

Pooling

Pooling layer

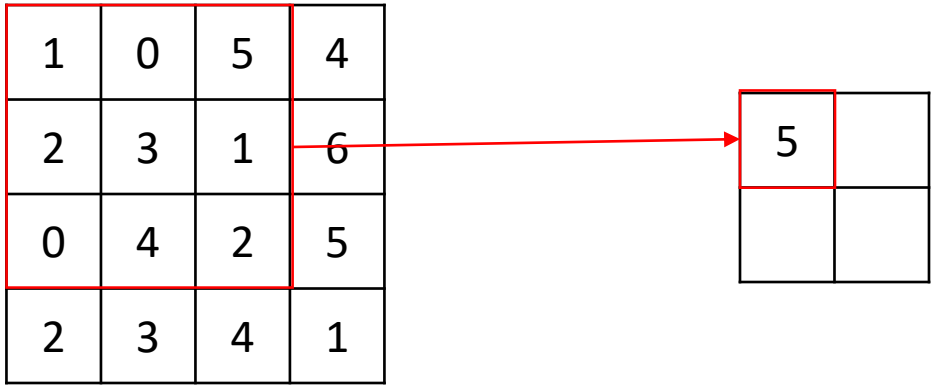
- Max pooling
  - Take maximum value in pooling mask
  - Pooling size = (3,3), stride = 1

1	0	5	4
2	3	1	6
0	4	2	5
2	3	4	1

Pooling

Pooling layer

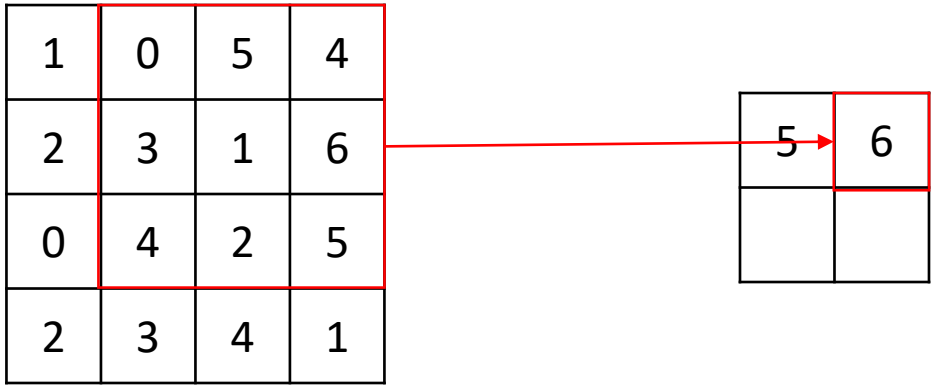
- Max pooling
  - Take maximum value in pooling mask
  - Pooling size = (3,3), stride = 1



Pooling

Pooling layer

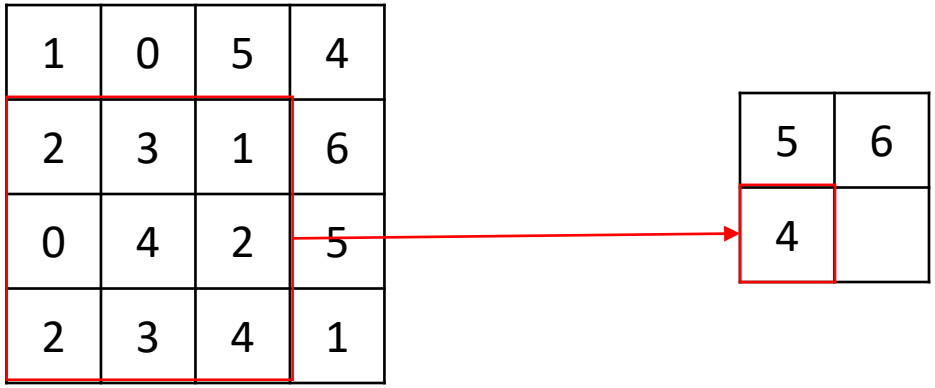
- Max pooling
  - Take maximum value in pooling mask
  - Pooling size = (3,3), stride = 1



Pooling

Pooling layer

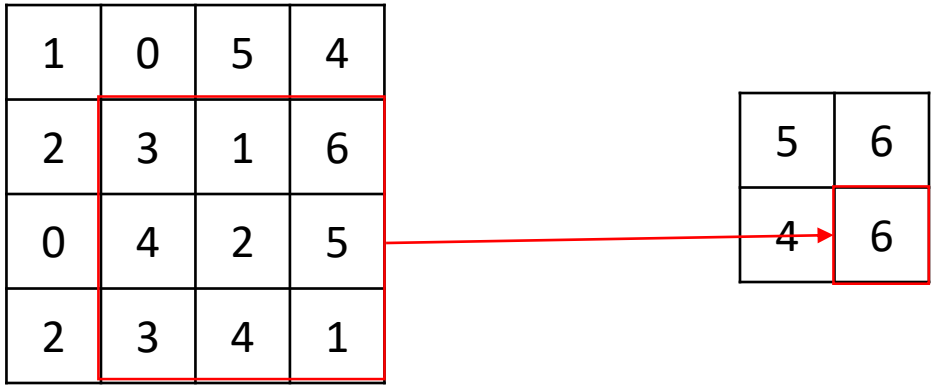
- Max pooling
  - Take maximum value in pooling mask
  - Pooling size = (3,3), stride = 1



Pooling

Pooling layer

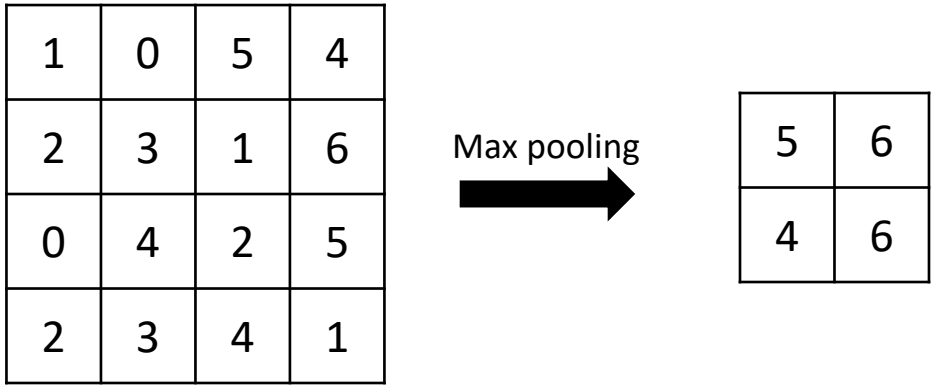
- Max pooling
  - Take maximum value in pooling mask
  - Pooling size = (3,3), stride = 1



Pooling

Pooling layer

- Max pooling
  - Take maximum value in pooling mask
  - Pooling size = (3,3), stride = 1



Pooling

Pooling layer

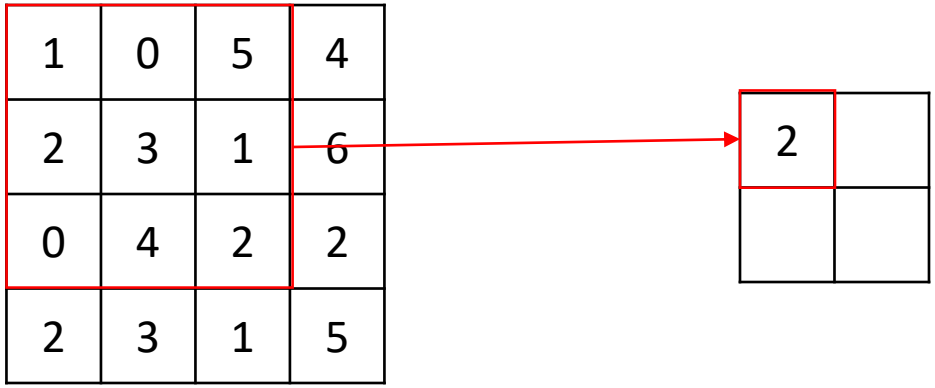
- Average pooling
  - Take average value in pooling mask
  - Pooling size = (3,3), stride = 1

1	0	5	4
2	3	1	6
0	4	2	2
2	3	1	5

Pooling

Pooling layer

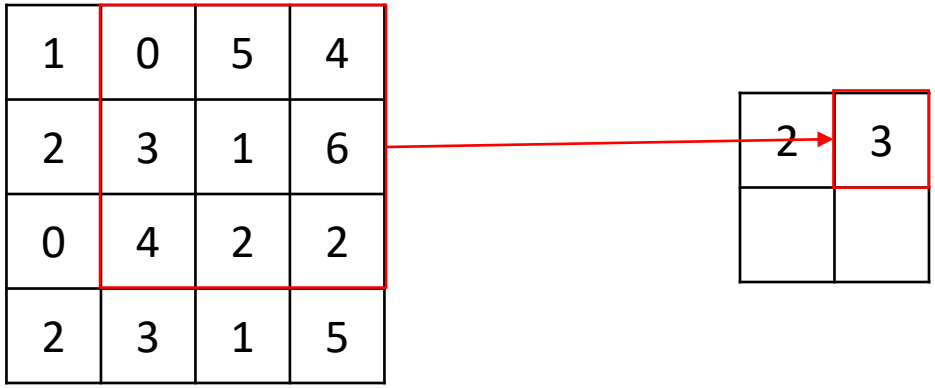
- Average pooling
  - Take average value in pooling mask
  - Pooling size = (3,3), stride = 1



Pooling

Pooling layer

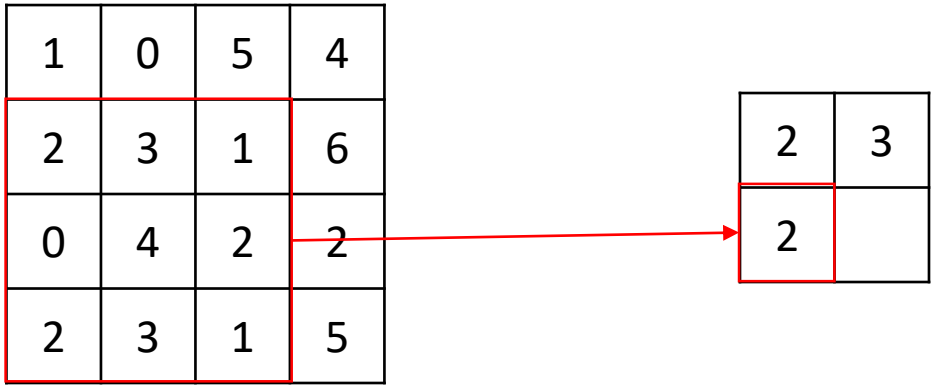
- Average pooling
  - Take average value in pooling mask
  - Pooling size = (3,3), stride = 1



Pooling

Pooling layer

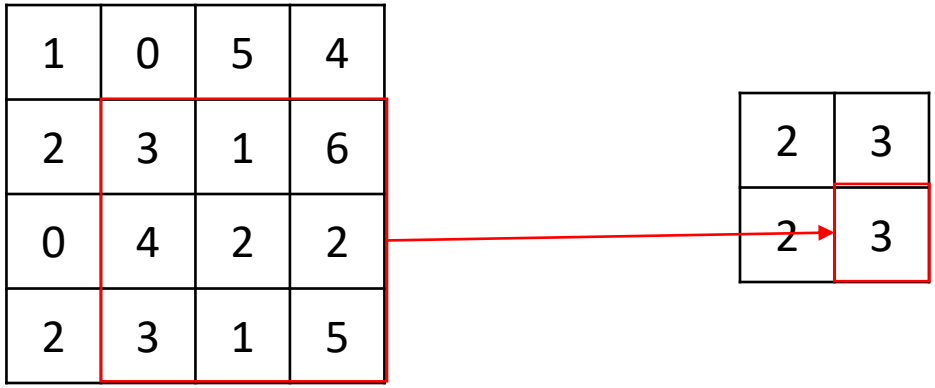
- Average pooling
  - Take average value in pooling mask
  - Pooling size = (3,3), stride = 1



Pooling

Pooling layer

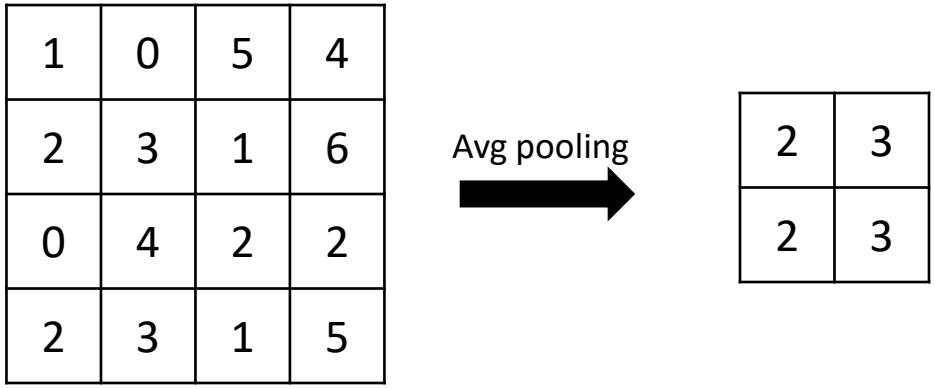
- Average pooling
  - Take average value in pooling mask
  - Pooling size = (3,3), stride = 1



Pooling

Pooling layer

- Average pooling
  - Take average value in pooling mask
  - Pooling size = (3,3), stride = 1



Pooling

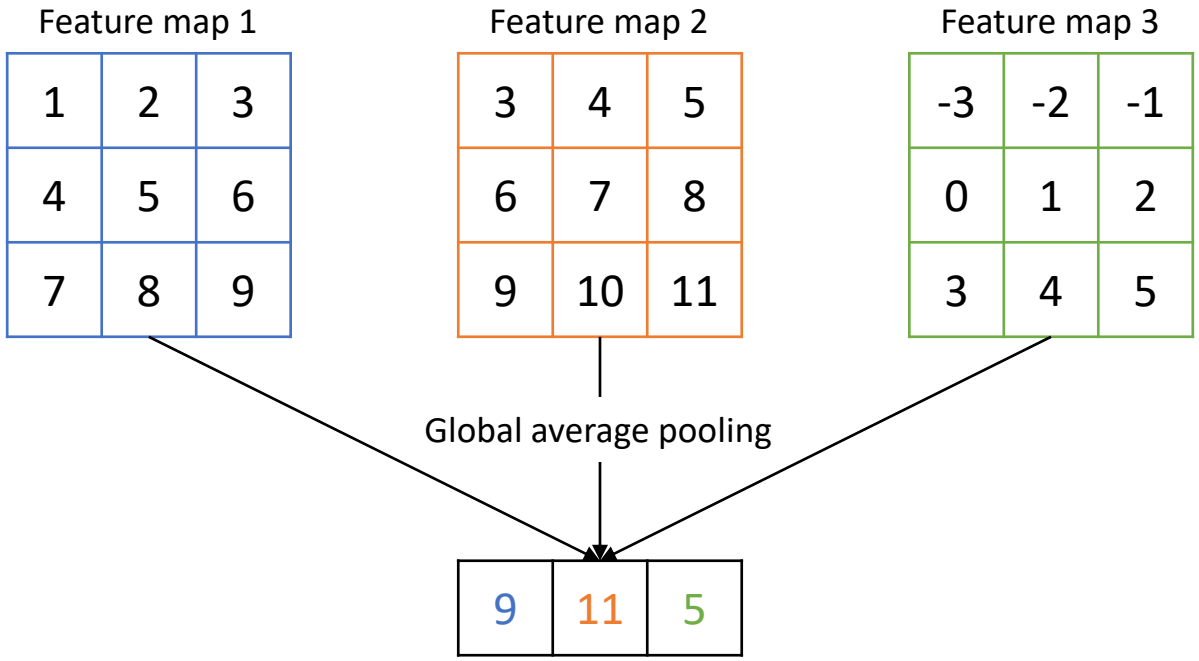
Global pooling

- Feature map to vector
  - Ultimately, we need to convert feature maps to vector (embedding) somewhere in the architecture.
  - Flatten layer not helps to compress dimension and make us to lose spatial information
  - Global poolings allow us to use global statistics of feature map

Pooling

Global pooling

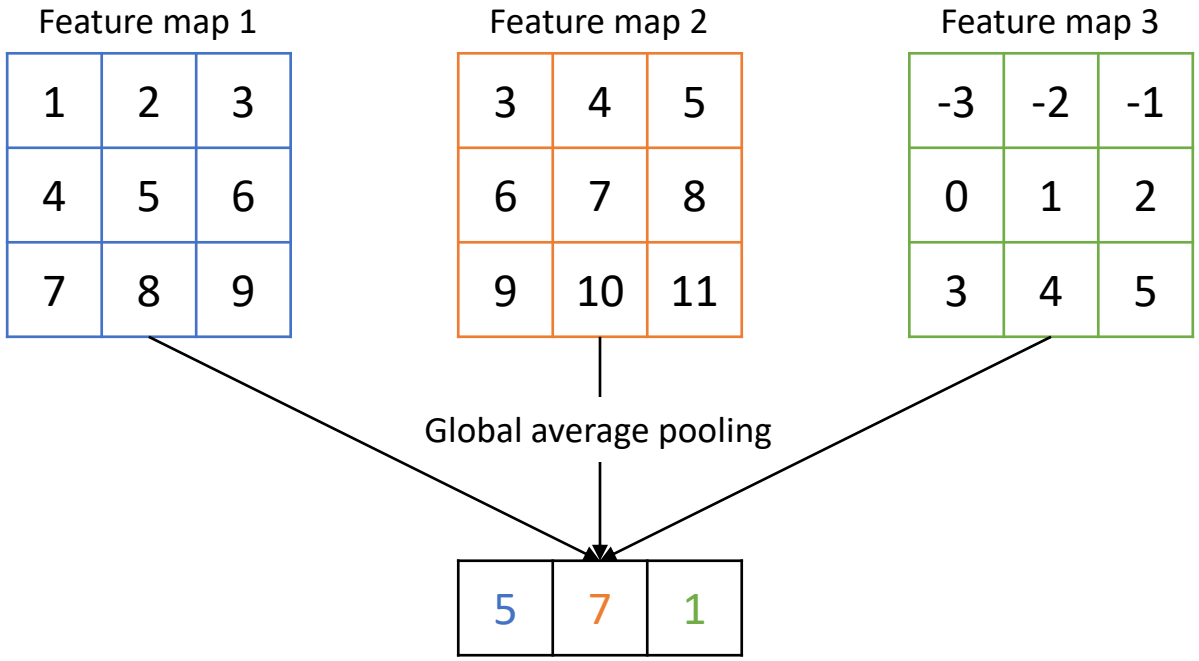
- Global max pooling
  - Take maximum value in feature map and make matrix to be scalar
  - Usually, global pooling is followed by Dense layers



Pooling

Global pooling

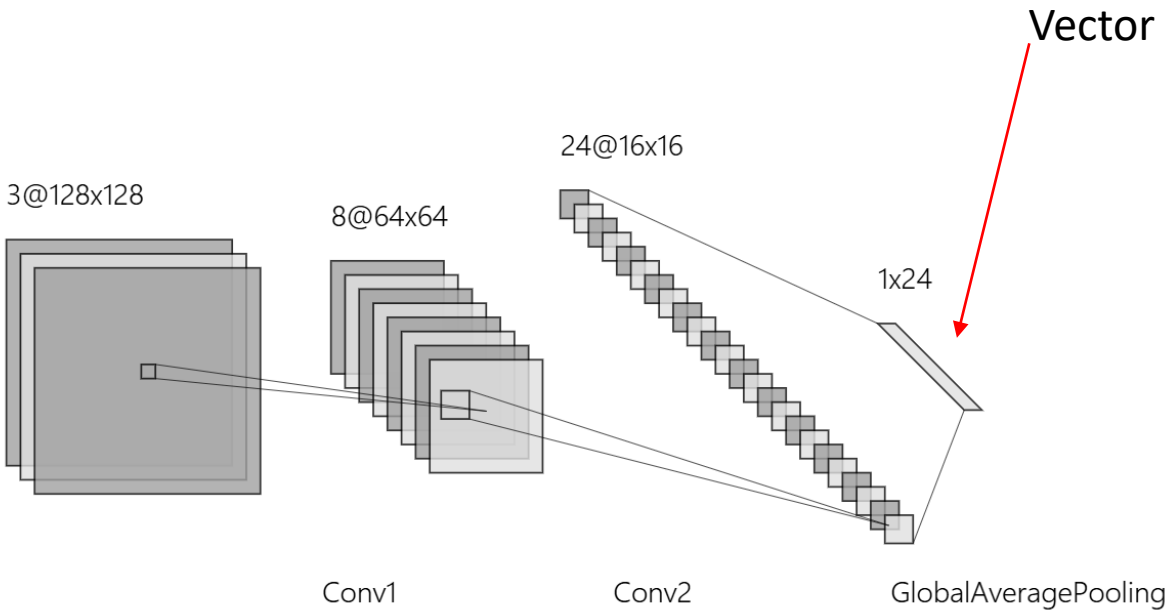
- Global average pooling
  - Take average value in feature map and make matrix to be scalar
  - Usually, GAP is followed by Dense layers



Pooling

Pooling layer

- Architecture after GAP

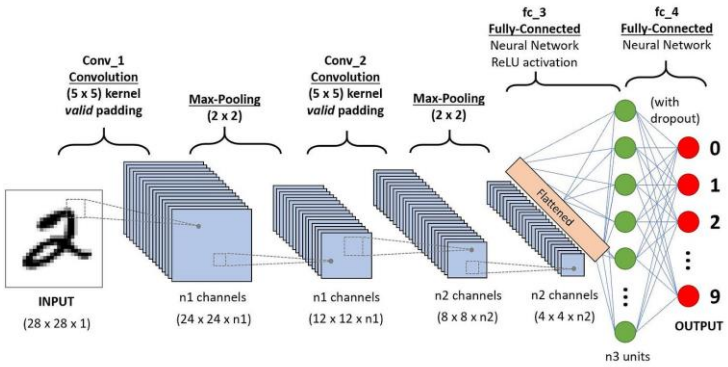


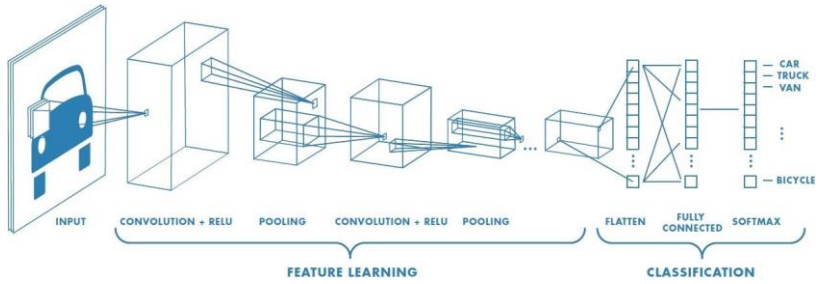
Basic backbone architectures

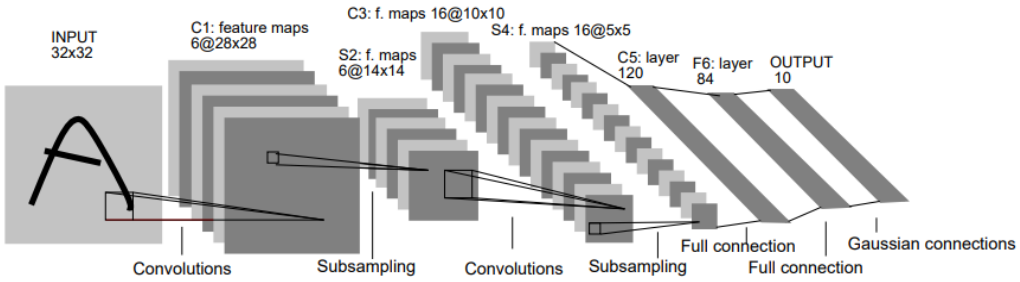
CNN Architectures

We have seen that Convolutional Networks are commonly made up of three layer types:

- Convolutional layer / Pooling layer / Fully-connected layer



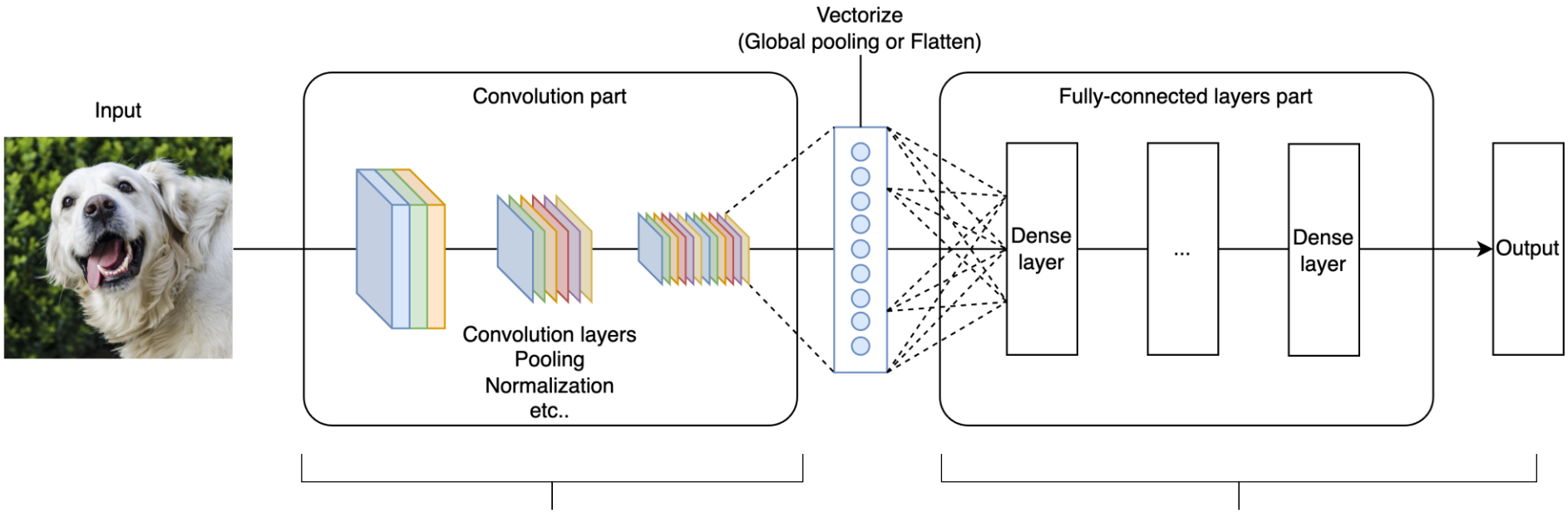




CNN Architectures

We have seen that Convolutional Networks are commonly made up of three layer types:

- Convolutional layer / Pooling layer / Fully-connected layer



Convolution and **pooling** operators extract features while respecting 2D image structure

Fully-Connected layers form an MLP at the end to predict scores

CNN Architectures

Case studies

- **AlexNet**
  - Krizhevsky, Alex, Ilya Sutskever, and Geoffrey E. Hinton. "Imagenet classification with deep convolutional neural networks." *Advances in neural information processing systems* 25 (2012).
- **VGG (Visual Geometry Group) Net**
  - Simonyan, Karen, and Andrew Zisserman. "Very deep convolutional networks for large-scale image recognition." *arXiv preprint arXiv:1409.1556* (2014).
- **GoogLeNet (Inception)**
  - Szegedy, Christian, et al. "Going deeper with convolutions." *Proceedings of the IEEE conference on computer vision and pattern recognition*. 2015.
- **ResNet**
  - He, Kaiming, et al. "Deep residual learning for image recognition." *Proceedings of the IEEE conference on computer vision and pattern recognition*. 2016.
- **DenseNet**
  - Huang, Gao, et al. "Densely connected convolutional networks." *Proceedings of the IEEE conference on computer vision and pattern recognition*. 2017.

Case study of architecture

AlexNet

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

ImageNet Classification with Deep Convolutional Neural Networks

Alex Krizhevsky
University of Toronto
kriz@cs.utoronto.ca

Ilya Sutskever
University of Toronto
ilya@cs.utoronto.ca

Geoffrey E. Hinton
University of Toronto
hinton@cs.utoronto.ca

Abstract

We trained a large, deep convolutional neural network to classify the 1.2 million high-resolution images in the ImageNet LSVRC-2010 contest into the 1000 different classes. On the test data, we achieved top-1 and top-5 error rates of 37.5% and 17.0% which is considerably better than the previous state-of-the-art. The neural network, which has 60 million parameters and 650,000 neurons, consists of five convolutional layers, some of which are followed by max-pooling layers, and three fully-connected layers with a final 1000-way softmax. To make training faster, we used non-saturating neurons and a very efficient GPU implementation of the convolution operation. To reduce overfitting in the fully-connected layers we employed a recently-developed regularization method called "dropout" that proved to be very effective. We also entered a variant of this model in the ILSVRC-2012 competition and achieved a winning top-5 test error rate of 15.3%, compared to 26.2% achieved by the second-best entry.

Case study of architecture

AlexNet

Summary

- Submitted to ImageNet Large Scale Visual Recognition Challenge (ILSVRC) 2012
- Achieved top-5 error: 16% vs 26% of the runner-up → huge performance leap
- Architecture:
  - Introduced stacked CONV layers before POOL (previously rare)
  - Marked the start of modern deep CNN era in Computer Vision

AlexNet

Introduction

- Current approaches to object recognition make essential use of machine learning methods.
- To improve their performance, we can collect larger datasets, learn more powerful models, and use better techniques for preventing overfitting.
- Until recently, datasets of labeled images were relatively small (order of tens of thousand, Ex> CIFAR-10/100).
    - It has only recently become possible to collect labeled datasets with millions of images.
    - Ex> ImageNet – 15million labeled high-resolution images in over 22,000 categories

AlexNet

Introduction

- To learn about thousands of objects from millions of images, we need a model with a large learning capacity.
- **Convolutional neural networks**
  - Capacity can be controlled by varying their depth and breadth
  - Strong and mostly correct assumptions about the nature of images (namely, stationarity of statistics and locality of pixel dependencies).
- Compared to standard feedforward neural networks with similarly-sized layers
  - CNNs have much fewer connections and parameters and so they are easier to train

AlexNet

Introduction

- Despite the attractive qualities of CNNs,
  - They have still been prohibitively expensive to apply in large scale to high-resolution images.
- Luckily,
  - Current GPUs are powerful enough to facilitate the training of interestingly-large CNNs
  - Recent datasets such as ImageNet contain enough labeled examples to train such models without severe overfitting.

AlexNet

Introduction

- Contribution of this paper
  - Trained one of the largest CNN on the subsets of ImageNet used in ILSVRC(ImageNet Large-Scale Visual Recognition Challenge)-2010 and ILSVRC-2012
  - By far, the best result ever reported on these dataset
  - Our network contains a number of new and unusual features which improve its performance and reduce its training time
  - Several effective techniques for preventing overfitting

AlexNet

Dataset

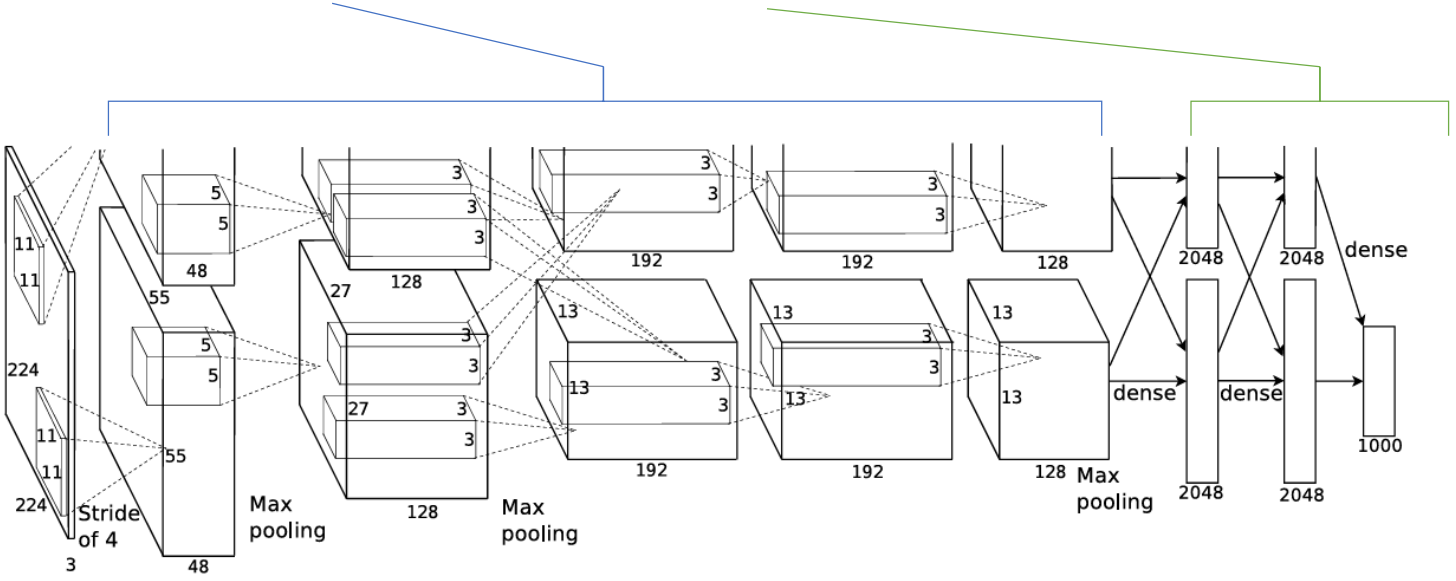
- ImageNet
  - Over 15million labeled high-resolution images, belonging to roughly 22,000 categories
  - ILSVRC dataset – subset of ImageNet
    - 1000 categories
    - Training images: 1.2 million
    - Validation images: 50,000
    - Testing images: 150,000
  - Preprocessing
    - 256x256 down-sampling

AlexNet

Architecture

- 8 layers
  - 5 convolutional layer + 3 fully-connected layer

Input
224x224



Output
1000x1

Figure 2: An illustration of the architecture of our CNN, explicitly showing the delineation of responsibilities between the two GPUs. One GPU runs the layer-parts at the top of the figure while the other runs the layer-parts at the bottom. The GPUs communicate only at certain layers. The network's input is 150,528-dimensional, and the number of neurons in the network's remaining layers is given by 253,440–186,624–64,896–64,896–43,264–4096–4096–1000.

AlexNet

Architecture

- **Novel or unusual features 1 - ReLU Nonlinearity**
  - Standard way to activation
    - $\tanh(x)$
    - $(1 + e^{-x})^{-1}$ (Sigmoid)
  - Rectified Linear Unit(ReLU)
    - $\max(0, x)$
    - Much faster (several times)
      → in large model training on large dataset

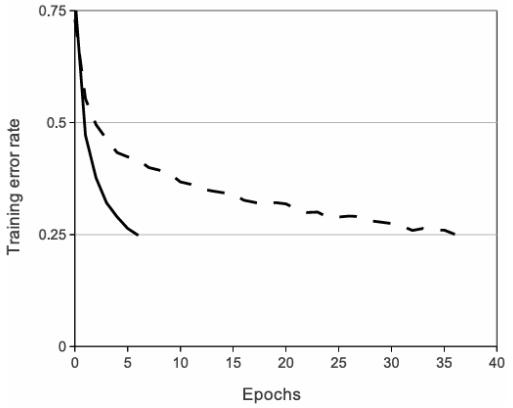


Figure 1: A four-layer convolutional neural network with ReLUs (**solid line**) reaches a 25% training error rate on CIFAR-10 six times faster than an equivalent network with tanh neurons (**dashed line**). The learning rates for each network were chosen independently to make training as fast as possible. No regularization of any kind was employed. The magnitude of the effect demonstrated here varies with network architecture, but networks with ReLUs consistently learn several times faster than equivalents with saturating neurons.

AlexNet

Architecture

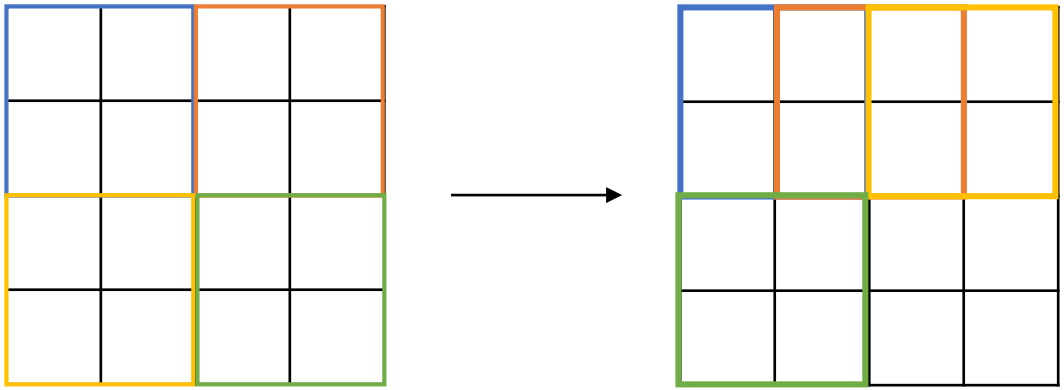
- **Novel or unusual features 2 – Local Response Normalization**
  - Not used in recent network..
  - Normalize Relu output

$$b_{x,y}^{i} = a_{x,y}^{i} / \left(k + \alpha \sum_{j=\max(0,i-n/2)}^{\min(N-1,i+n/2)} (a_{x,y}^{j})^2 \right)^{\beta}$$

AlexNet

Architecture

- **Novel or unusual features 3 – Overlapping Pooling**
  - Traditionally, the neighborhoods summarized by adjacent pooling units do not overlap
  - Pooling size 3x3, stride 2 is used.



Before
(Non-overlapping pooling)

This work
(Overlapping pooling)

AlexNet

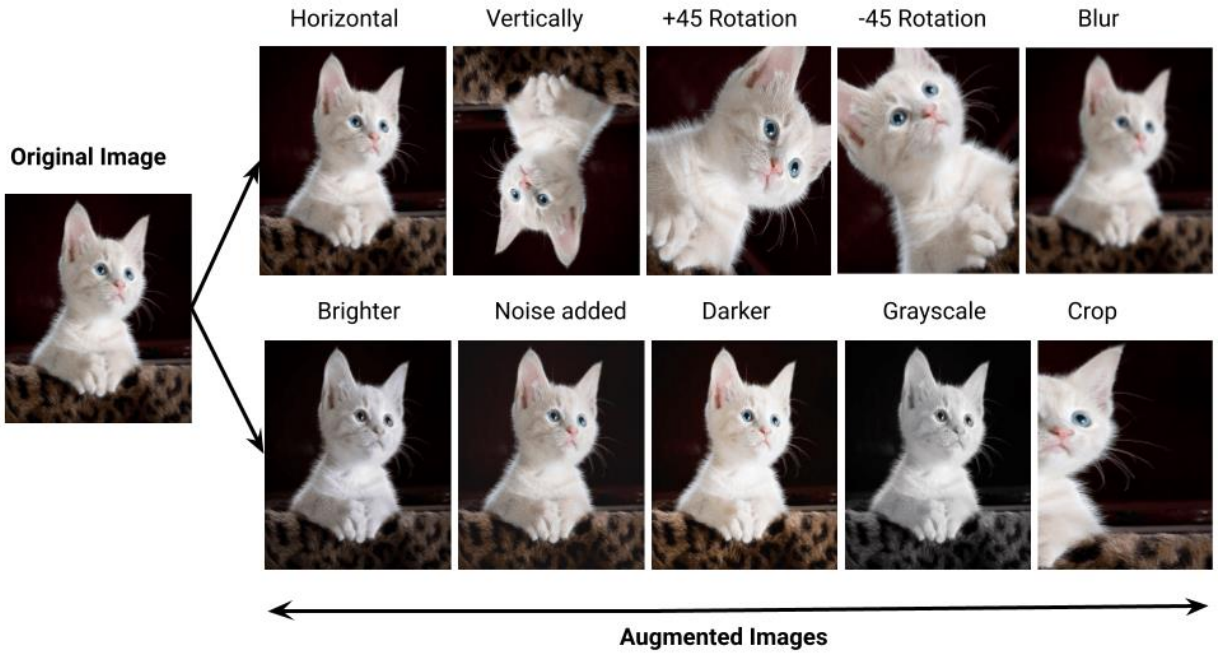
Reducing overfitting

- Our neural network architecture has 60 million parameters.
- Although the 1000 classes of ILSVRC make each training example impose 10 bits of constraint on the mapping from image to label, this turns out to be insufficient to learn so many parameters without considerable overfitting.

AlexNet

Reducing overfitting

- **Data augmentation**
  - Artificially enlarge the dataset using label-preserving transformations



AlexNet

Reducing overfitting

- **Data augmentation**
  - Cropping
  - Reflection (Flip)
  - Rotation
  - Add noise
  - Random erasing
  - Color jittering
  - etc

AlexNet

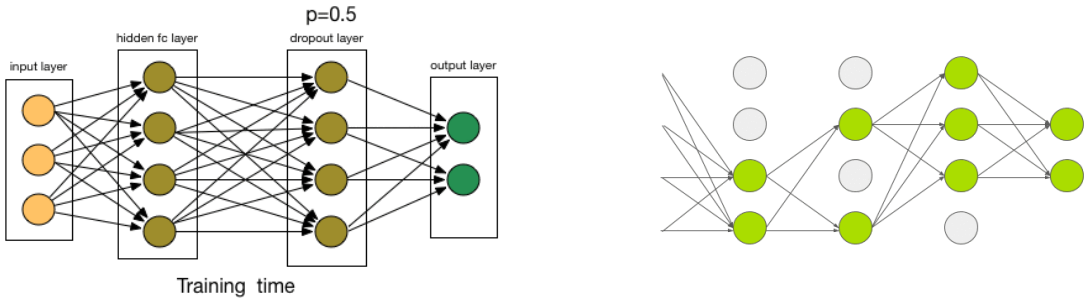
Reducing overfitting

- **Data augmentation (In this study)**
  - Random crop 256x256 → 224x224
  - Random flip
  - Random color alteration
  - This scheme reduces the top-1 error rate by over 1%

AlexNet

Reducing overfitting

- **Dropout**
  - To prevent overfitting during model training, randomly ignoring some neuron's output (dropped out).
  - This is deactivated during validation.
  - This study used drop probability 0.5 on first two fully-connected layer



AlexNet

Details of learning

- Batch size 128
- SGD with Momentum 0.9
- Weight decay (L2-norm) 0.0005

$$
\begin{aligned}
v_{i+1} &:= 0.9 \cdot v_i - 0.0005 \cdot \epsilon \cdot w_i - \epsilon \cdot \left\langle \frac{\partial L}{\partial w}\Big|_{w_i} \right\rangle_{D_i} \\
w_{i+1} &:= w_i + v_{i+1}
\end{aligned}
$$

- Weight initialization: zero-mean Gaussian, std 0.01
- Learning rate: initial 0.01, learning rate decay 1/3 per epoch
- 90 epoch (5~6 days)
- Gpu: NVIDIA GTX 580 3GB

AlexNet

Results

Model	Top-1 (val)	Top-5 (val)	Top-5 (test)
SIFT + FVs [7]	—	—	*26.2%*
1 CNN	40.7%	18.2%	—
5 CNNs	38.1%	16.4%	**16.4%**
1 CNN*	39.0%	16.6%	—
7 CNNs*	36.7%	15.4%	**15.3%**

Table 2: Comparison of error rates on ILSVRC-2012 validation and test sets. In *italics* are best results achieved by others. Models with an asterisk* were "pre-trained" to classify the entire ImageNet 2011 Fall release. See Section 6 for details.

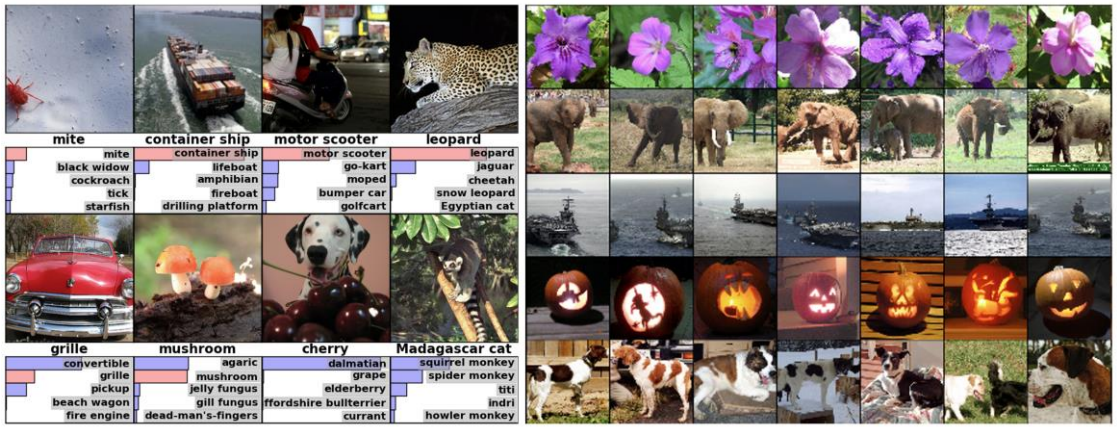


Figure 4: **(Left)** Eight ILSVRC-2010 test images and the five labels considered most probable by our model. The correct label is written under each image, and the probability assigned to the correct label is also shown with a red bar (if it happens to be in the top 5). **(Right)** Five ILSVRC-2010 test images in the first column. The remaining columns show the six training images that produce feature vectors in the last hidden layer with the smallest Euclidean distance from the feature vector for the test image.

Case study of architecture

VGG (Visual Geometry Group) Net

Simonyan, Karen, and Andrew Zisserman. "Very deep convolutional networks for large-scale image recognition." *arXiv preprint arXiv:1409.1556* (2014).

Published as a conference paper at ICLR 2015

VERY DEEP CONVOLUTIONAL NETWORKS FOR LARGE-SCALE IMAGE RECOGNITION

Karen Simonyan* **& Andrew Zisserman+**
Visual Geometry Group, Department of Engineering Science, University of Oxford
{karen,az}@robots.ox.ac.uk

ABSTRACT

In this work we investigate the effect of the convolutional network depth on its accuracy in the large-scale image recognition setting. Our main contribution is a thorough evaluation of networks of increasing depth using an architecture with very small (3×3) convolution filters, which shows that a significant improvement on the prior-art configurations can be achieved by pushing the depth to 16–19 weight layers. These findings were the basis of our ImageNet Challenge 2014 submission, where our team secured the first and the second places in the localisation and classification tracks respectively. We also show that our representations generalise well to other datasets, where they achieve state-of-the-art results. We have made our two best-performing ConvNet models publicly available to facilitate further research on the use of deep visual representations in computer vision.

Case study of architecture

VGG (Visual Geometry Group) Net

Summary

- Runner-up at ImageNet Large Scale Visual Recognition Challenge (ILSVRC) 2014

- Key contributions:
  - Showed network depth is critical for performance
  - Final model: 16 CONV/FC layers
  - Very homogeneous design: only 3×3 convolutions and 2×2 pooling throughout
  - Pretrained model available (e.g., in Caffe)

- Limitations:
  - High computational cost
  - ~140M parameters (mostly in first FC layer)
  - Later shown: FC layers can be removed with no performance drop → drastically reduces parameters

VGG net

Introduction

- ConvNets achieved breakthrough success in large-scale image/video recognition

- Enabled by:

    - Large datasets (ImageNet — Jia Deng et al., 2009)
    - High-performance computing (Jeff Dean et al., 2012 — GPUs, distributed clusters)

- ImageNet Large Scale Visual Recognition Challenge (ILSVRC)

    - Major benchmark for deep visual recognition

- Progression of winners:

    - 2011: Shallow, high-dimensional features (Florent Perronnin et al., 2010)
    - 2012: Deep ConvNets (AlexNet — Krizhevsky et al.)
    - 2013: Smaller receptive field in first CONV layer, Smaller stride (Zeiler & Fergus, Sermanet et al.)

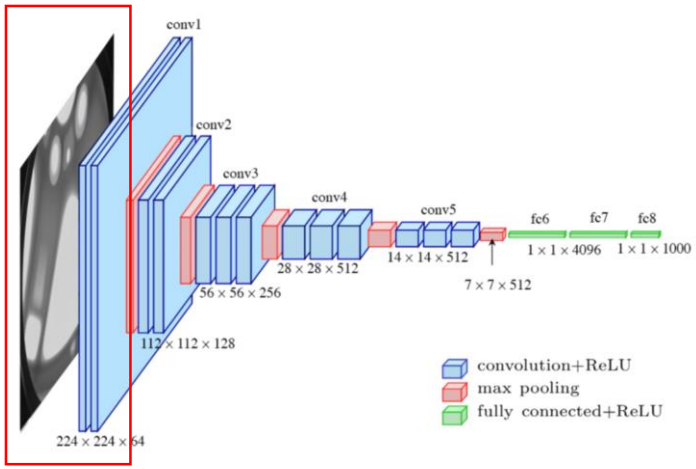
VGG net

Introduction

- In this paper, we address another important aspect of ConvNet architecture design– **its depth**.
  - To this end, we fix other parameters of the architecture, and steadily increase the depth of the network by adding more convolutional layers, which is feasible due to the use of very small (3 × 3) convolution filters in all layers.
- Led to:
  - Higher accuracy on ILSVRC classification & localization
  - Strong performance on other datasets (even as simple feature extractor + linear SVM)
  - Released pretrained models to support research

VGG net

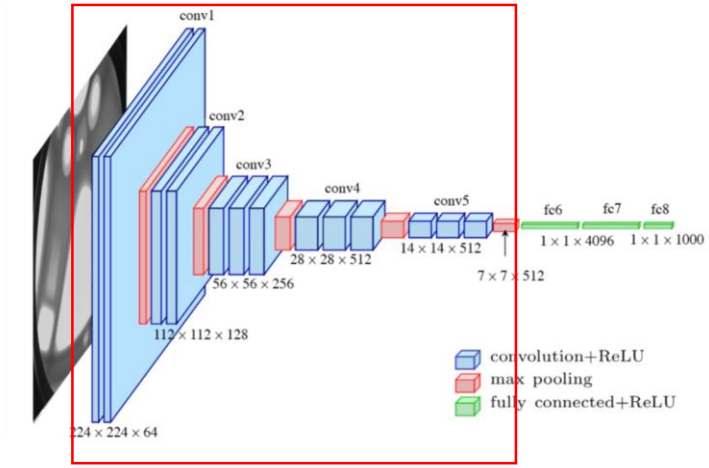
Architecture



- **Input**: 224 × 224 RGB image
- **Preprocessing**:
  - Subtract mean RGB value (computed from training set)
  - This normalized image goes into the ConvNet

VGG net

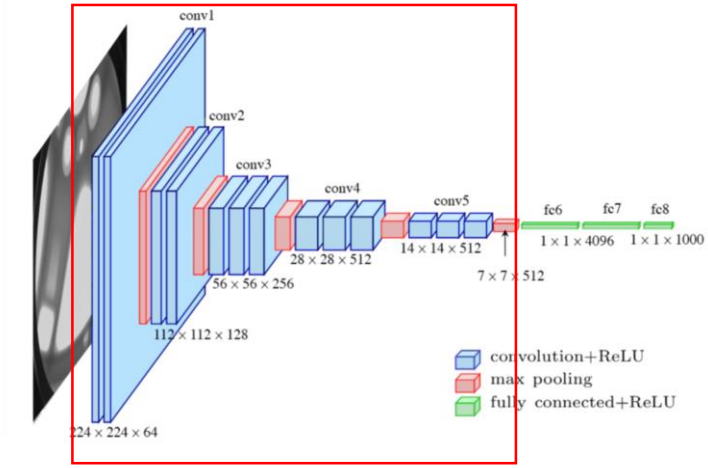
Architecture



- **Convolutional Layers**
  - Use very small receptive fields: 3×3 filters
  - Sometimes also use 1×1 filters
    - Acts as linear channel-wise transformation + nonlinearity
  - Stride = 1
  - Padding = 1 (for 3×3) → preserves spatial resolution
  - Not every conv layer is followed by pooling

VGG net

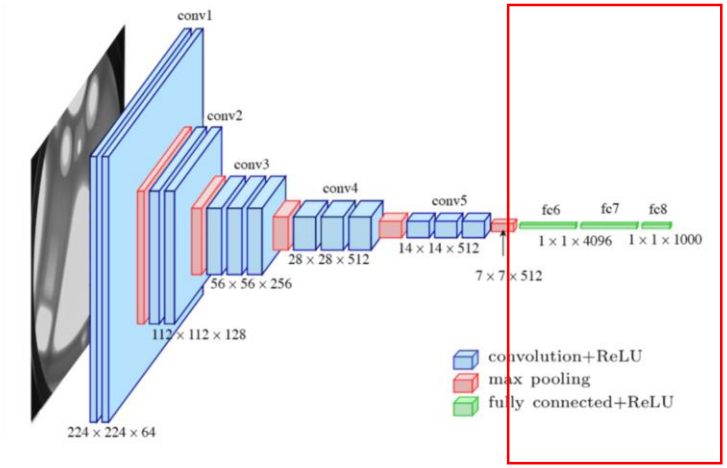
Architecture



- **Pooling Layers**
  - Five max-pooling layers inserted after some conv layers
  - Max-pooling settings:
    - 2×2 window
    - Stride = 2
  - Reduces spatial size, keeps depth

VGG net

Architecture



- **Fully connected layer**
  - After the conv+pool stack:
    - FC1: 4096 channels
    - FC2: 4096 channels
    - FC3: 1000 channels (for 1000-way ILSVRC classes)
  - Followed by Softmax for classification

VGG net

VGGNet configuration overview

- Five configurations (A–E) tested
- All follow the same design pattern
  - Stacked 3×3 CONV layers
  - 2×2 MaxPooling
  - 3 FC layers at the end
- **Main difference: depth**
  - A: 11 weight layers (8 CONV + 3 FC)
  - …
  - E: 19 weight layers (16 CONV + 3 FC)

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					

Table 2: **Number of parameters** (in millions).

Network	A,A-LRN	B	C	D	E
Number of parameters	133	133	134	138	144

VGG net

Training

- Optimization
  - Mini-batch SGD (Batch size: 256, Momentum: 0.9)
  - Loss: Multinomial logistic regression

- Regularization
  - Weight decay (L2): 5×10^{-4}, Dropout: 0.5 on first two FC layers

- Learning Rate Schedule
  - Start: 10^{-2} → ÷10 on plateaus → total 3 drops
  - Stop after 370k iterations (~74 epochs)

- Initialization
  - Train shallow Config A from scratch
  - Deeper nets: copy first 4 CONV + last 3 FC from A
  - Others random: Normal(0, 10^{-2}) weights, 0 biases

- Data Augmentation
  - Random 224×224 crops from rescaled images, Horizontal flips, RGB color jitter

VGG net

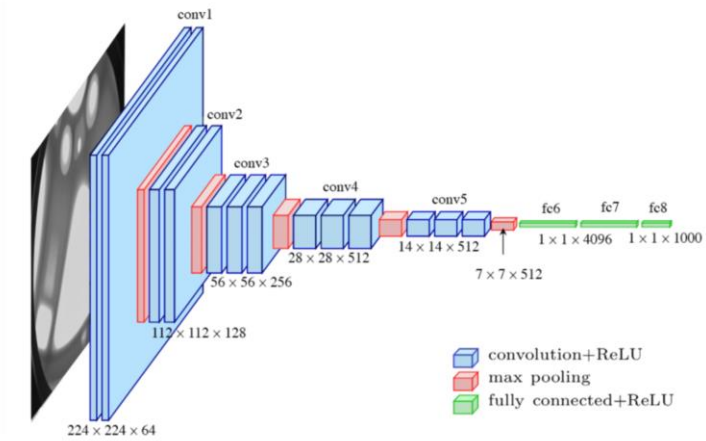
Result

Table 4: **ConvNet performance at multiple test scales.**

ConvNet config. (Table 1)	smallest image side		top-1 val. error (%)	top-5 val. error (%)
	train (S)	test (Q)		
B	256	224,256,288	28.2	9.6
C	256	224,256,288	27.7	9.2
	384	352,384,416	27.8	9.2
	[256; 512]	256,384,512	26.3	8.2
D	256	224,256,288	26.6	8.6
	384	352,384,416	26.5	8.6
	[256; 512]	256,384,512	**24.8**	**7.5**
E	256	224,256,288	26.9	8.7
	384	352,384,416	26.7	8.6
	[256; 512]	256,384,512	**24.8**	**7.5**

Case study of architecture

VGG (Visual Geometry Group) Net



- Proposed by the Oxford research team in 2014, the VGG model explores the impact of network depth on performance.

- Simple, straightforward architecture
- Two main versions: VGG16 (16 layers) and VGG19 (19 layers)
- Utilizes only 3x3 filters and max-pooling

- Designed to analyze the effect of deeper networks on accuracy and performance in image recognition.

Case study of architecture

GoogLeNet (Inception)

Going Deeper with Convolutions

Christian Szegedy[1], Wei Liu[2], Yangqing Jia[1], Pierre Sermanet[1], Scott Reed[3],
Dragomir Anguelov[1], Dumitru Erhan[1], Vincent Vanhoucke[1], Andrew Rabinovich[4]
[1]Google Inc. [2]University of North Carolina, Chapel Hill
[3]University of Michigan, Ann Arbor [4]Magic Leap Inc.
[1]{szegedy,jiayq,sermanet,dragomir,dumitru,vanhoucke}@google.com
[2]wliu@cs.unc.edu, [3]reedscott@umich.edu, [4]arabinovich@magicleap.com

Abstract

We propose a deep convolutional neural network architecture codenamed Inception that achieves the new state of the art for classification and detection in the ImageNet Large-Scale Visual Recognition Challenge 2014 (ILSVRC14). The main hallmark of this architecture is the improved utilization of the computing resources inside the network. By a carefully crafted design, we increased the depth and width of the network while keeping the computational budget constant. To optimize quality, the architectural decisions were based on the Hebbian principle and the intuition of multi-scale processing. One particular incarnation used in our submission for ILSVRC14 is called GoogLeNet, a 22 layers deep network, the quality of which is assessed in the context of classification and detection.

ger and bigger deep networks, but from the synergy of deep architectures and classical computer vision, like the R-CNN algorithm by Girshick et al [6].

Another notable factor is that with the ongoing traction of mobile and embedded computing, the efficiency of our algorithms – especially their power and memory use – gains importance. It is noteworthy that the considerations leading to the design of the deep architecture presented in this paper included this factor rather than having a sheer fixation on accuracy numbers. For most of the experiments, the models were designed to keep a computational budget of 1.5 billion multiply-adds at inference time, so that the they do not end up to be a purely academic curiosity, but could be put to real world use, even on large datasets, at a reasonable cost.

In this paper, we will focus on an efficient deep neural network architecture for computer vision, codenamed In-

- 2014 ILSVRC Winner (VGG finishing in second place).

GoogLeNet (Inception)

Introduction

- In the last three years, object classification and detection capabilities have dramatically improved
    - Due to advances in deep learning and convolutional networks
- Encouragingly, most progress came from new ideas, algorithms, and architectures — not just bigger datasets, hardware, or models
- Our GoogLeNet submission to ILSVRC 2014:
    - Used 12 time fewer parameters than the winning architecture of AlexNet (Krizhevsky et al. 2012)
    - Yet achieved **significantly higher accuracy**
- With the rise of mobile and embedded computing, Efficiency (power, memory) became as important as accuracy

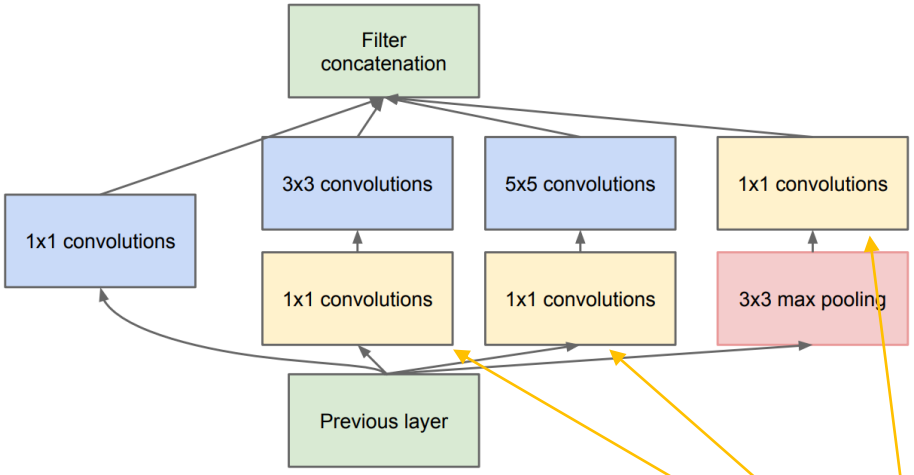
GoogLeNet (Inception)

Introduction

- In this paper,
  - we will focus on an efficient deep neural network architecture for computer vision, codenamed **Inception**
  - Significantly outperformed previous state-of-the-art on ILSVRC 2014
    - Both in classification and detection tasks

GoogLeNet (Inception)

Architecture details



(b) Inception module with dimensionality reduction

1x1 convolutions' dual role

- Dimensionality reduction before applying large filters
- Non-linear transformation

GoogLeNet (Inception)

Architecture details

- Design intuition
  - Process visual information at multiple scales
  - Aggregate these features so the next stage can
  - abstract from all scales simultaneously
- Key advantage
  - Allows increasing the number of units at each stage
  - Without uncontrolled blow-up in computational complexity
- How it's achieved
  - Ubiquitous use of dimensionality reduction
  - Applied before expensive 3×3 and 5×5 convolutions

GoogLeNet (Inception)

Architecture details

- Benefits
  - Enables wider stages and deeper networks
  - Can also build slightly inferior but much cheaper variants
  - With careful manual design, can be 3–10× faster than similarly performing non-Inception networks

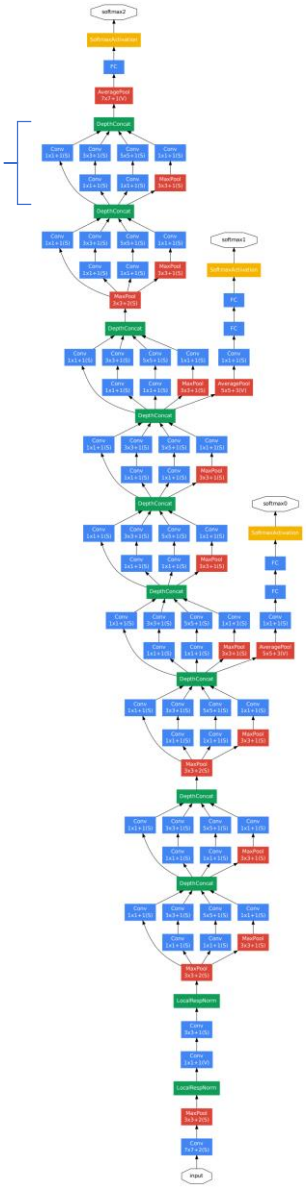
GoogLeNet (Inception)

GoogLeNet (2014) architecture

Inception block

x

Repeated



GoogLeNet (Inception)

Performance

Augmentation

- Sampling various size of patches
- Effective recipe (post-competition):
  - Random patches: 8%–100% area, 3:4–4:3 aspect ratio
  - Photometric distortions (Andrew Howard) → combat overfitting

Optimization

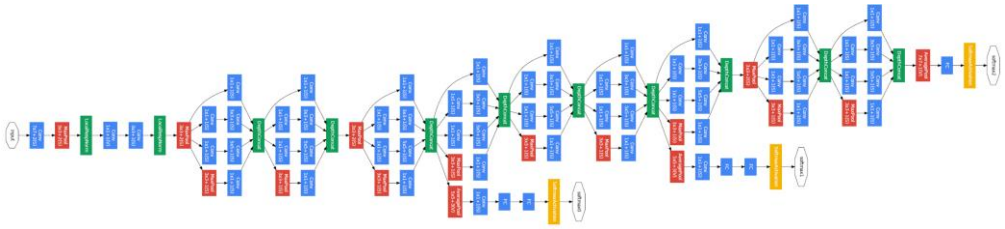
- SGD + 0.9 momentum [17]
- LR ↓ 4% every 8 epochs (fixed schedule)
- Polyak averaging [13] for final inference model

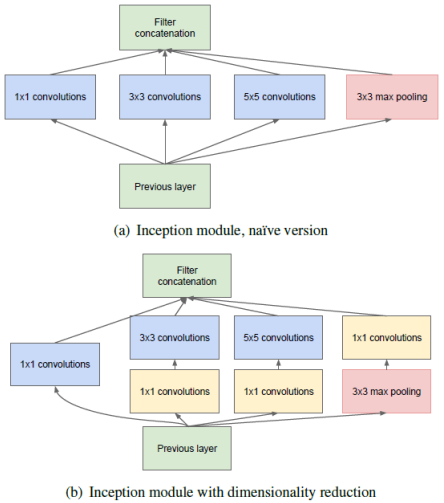
Team	Year	Place	Error (top-5)	Uses external data
SuperVision	2012	1st	16.4%	no
SuperVision	2012	1st	15.3%	Imagenet 22k
Clarifai	2013	1st	11.7%	no
Clarifai	2013	1st	11.2%	Imagenet 22k
MSRA	2014	3rd	7.35%	no
VGG	2014	2nd	7.32%	no
GoogLeNet	2014	1st	6.67%	no

Table 2: Classification performance.

Case study of architecture

GoogLeNet (Inception)





- Uses 1x1 convolutions to reduce the number of feature maps and decrease computation.
- **Inception Module:** Employs multiple filter sizes within the same layer to capture features at various scales.
- Includes an auxiliary classifier (highlighted in yellow in diagrams) to help mitigate gradient vanishing during training.

Case study of architecture

ResNet (Residual network)

Deep Residual Learning for Image Recognition

Kaiming He Xiangyu Zhang Shaoqing Ren Jian Sun
Microsoft Research
{kahe, v-xiangz, v-shren, jiansun}@microsoft.com

Abstract

Deeper neural networks are more difficult to train. We present a residual learning framework to ease the training of networks that are substantially deeper than those used previously. We explicitly reformulate the layers as learning residual functions with reference to the layer inputs, instead of learning unreferenced functions. We provide comprehensive empirical evidence showing that these residual networks are easier to optimize, and can gain accuracy from considerably increased depth. On the ImageNet dataset we evaluate residual nets with a depth of up to 152 layers—8× deeper than VGG nets [40] but still having lower complexity. An ensemble of these residual nets achieves 3.57% error on the ImageNet test set. This result won the 1st place on the ILSVRC 2015 classification task. We also present analysis on CIFAR-10 with 100 and 1000 layers.

The depth of representations is of central importance for many visual recognition tasks. Solely due to our extremely deep representations, we obtain a 28% relative improvement on the COCO object detection dataset. Deep residual nets are foundations of our submissions to ILSVRC & COCO 2015 competitions[1], where we also won the 1st places on the tasks of ImageNet detection, ImageNet localization, COCO detection, and COCO segmentation.

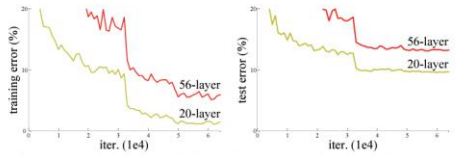


Figure 1. Training error (left) and test error (right) on CIFAR-10 with 20-layer and 56-layer "plain" networks. The deeper network has higher training error, and thus test error. Similar phenomena on ImageNet is presented in Fig. 4.

greatly benefited from very deep models.

Driven by the significance of depth, a question arises: *Is learning better networks as easy as stacking more layers?* An obstacle to answering this question was the notorious problem of vanishing/exploding gradients [14, 1, 8], which hamper convergence from the beginning. This problem, however, has been largely addressed by normalized initialization [23, 8, 36, 12] and intermediate normalization layers [16], which enable networks with tens of layers to start converging for stochastic gradient descent (SGD) with backpropagation [22].

When deeper networks are able to start converging, a *degradation* problem has been exposed: with the network

ResNet

Introduction

Deep convolutional networks

- Breakthroughs in image classification
- Naturally integrate low-, mid-, and high-level features
- Feature richness increases with network depth
- Leading results on ImageNet exploit “very deep” models (16–30 layers)
- Many visual recognition tasks benefit from very deep models

ResNet

Introduction

New Challenge: Degradation Problem

- With increasing depth:
  - Accuracy saturates → then degrades rapidly

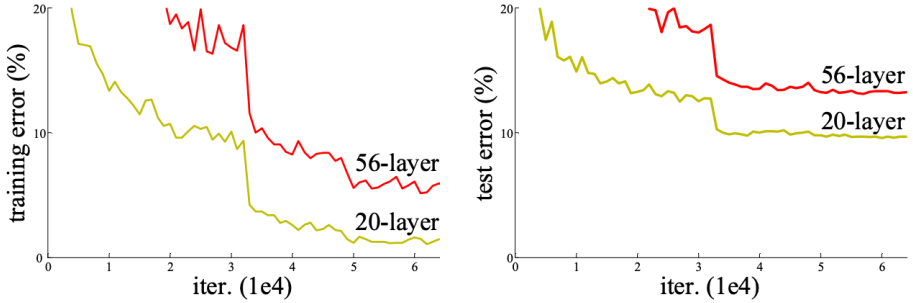


Figure 1. Training error (left) and test error (right) on CIFAR-10 with 20-layer and 56-layer "plain" networks. The deeper network has higher training error, and thus test error. Similar phenomena on ImageNet is presented in Fig. 4.

- Not due to overfitting → deeper models show higher training error
- Indicates optimization difficulty, not capacity limits

ResNet

Residual learning
In this paper,

- We address the degradation problem by **introducing a deep residual learning** framework

- Key Idea
  - Instead of fitting a direct mapping, stacked layers fit a residual mapping

- Formulation
  - Desired mapping: H(x)
  - Learn residual: F(x) := H(x) − x
  - Recast original mapping as: H(x) = F(x) + x

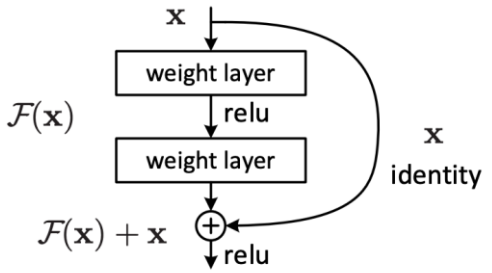


Figure 2. Residual learning: a building block.

- Hypothesis
  - It is easier to optimize residuals than raw mappings
  - If identity mapping is optimal:
    - Simply push residual F(x) → 0
    - Easier than forcing nonlinear layers to learn identity

ResNet

Experiments

layer name	output size	18-layer	34-layer	50-layer	101-layer	152-layer
conv1	112×112	7×7, 64, stride 2				
		3×3 max pool, stride 2				
conv2_x	56×56	[3×3, 64; 3×3, 64] ×2	[3×3, 64; 3×3, 64] ×3	[1×1, 64; 3×3, 64; 1×1, 256] ×3	[1×1, 64; 3×3, 64; 1×1, 256] ×3	[1×1, 64; 3×3, 64; 1×1, 256] ×3
conv3_x	28×28	[3×3, 128; 3×3, 128] ×2	[3×3, 128; 3×3, 128] ×4	[1×1, 128; 3×3, 128; 1×1, 512] ×4	[1×1, 128; 3×3, 128; 1×1, 512] ×4	[1×1, 128; 3×3, 128; 1×1, 512] ×8
conv4_x	14×14	[3×3, 256; 3×3, 256] ×2	[3×3, 256; 3×3, 256] ×6	[1×1, 256; 3×3, 256; 1×1, 1024] ×6	[1×1, 256; 3×3, 256; 1×1, 1024] ×23	[1×1, 256; 3×3, 256; 1×1, 1024] ×36
conv5_x	7×7	[3×3, 512; 3×3, 512] ×2	[3×3, 512; 3×3, 512] ×3	[1×1, 512; 3×3, 512; 1×1, 2048] ×3	[1×1, 512; 3×3, 512; 1×1, 2048] ×3	[1×1, 512; 3×3, 512; 1×1, 2048] ×3
	1×1	average pool, 1000-d fc, softmax				
FLOPs		1.8×10^9	3.6×10^9	3.8×10^9	7.6×10^9	11.3×10^9

Initial convolution & pooling

Residual blocks

Average pool & Dense

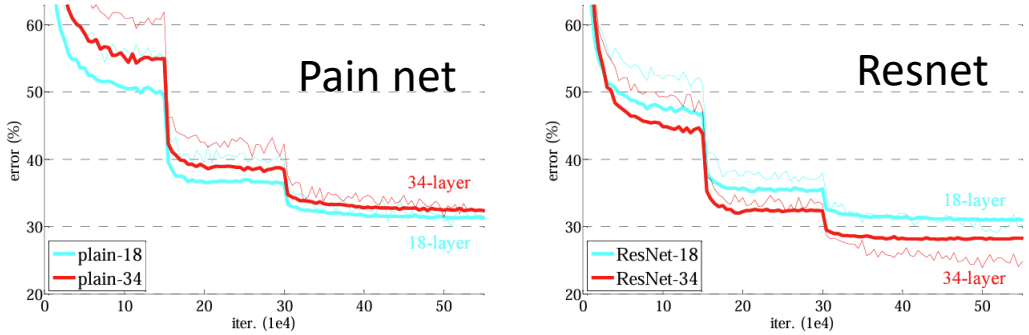


Figure 4. Training on **ImageNet**. Thin curves denote training error, and bold curves denote validation error of the center crops. Left: plain networks of 18 and 34 layers. Right: ResNets of 18 and 34 layers. In this plot, the residual networks have no extra parameter compared to their plain counterparts.

(Same number of parameters)

ResNet

Experiments

method	top-1 err.	top-5 err.
VGG [40] (ILSVRC'14)	-	8.43†
GoogLeNet [43] (ILSVRC'14)	-	7.89
VGG [40] (v5)	24.4	7.1
PReLU-net [12]	21.59	5.71
BN-inception [16]	21.99	5.81
ResNet-34 B	21.84	5.71
ResNet-34 C	21.53	5.60
ResNet-50	20.74	5.25
ResNet-101	19.87	4.60
ResNet-152	**19.38**	**4.49**

Table 4. Error rates (%) of **single-model** results on the ImageNet validation set (except † reported on the test set).

method	top-5 err. (**test**)
VGG [40] (ILSVRC'14)	7.32
GoogLeNet [43] (ILSVRC'14)	6.66
VGG [40] (v5)	6.8
PReLU-net [12]	4.94
BN-inception [16]	4.82
ResNet (ILSVRC'15)	**3.57**

Table 5. Error rates (%) of **ensembles**. The top-5 error is on the test set of ImageNet and reported by the test server.

Combined six models of different depth

Case study of architecture

ResNet

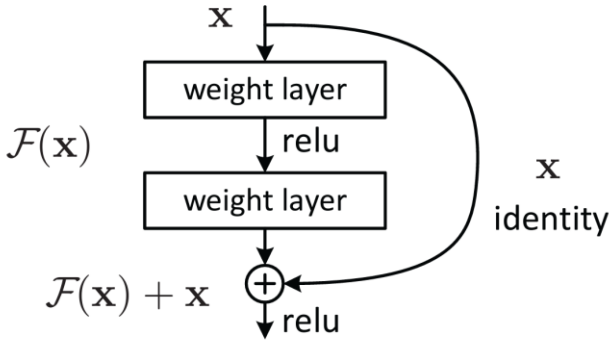
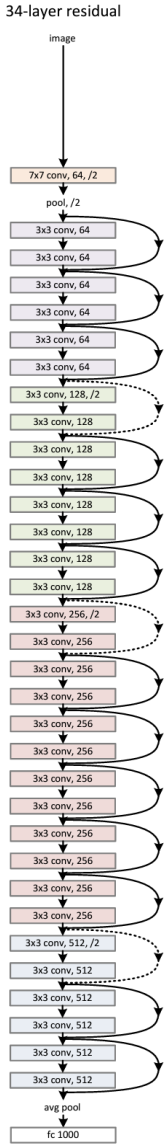


Figure 2. Residual learning: a building block.



- ResNet won both the 2015 ILSVRC (ImageNet Large Scale Visual Recognition Challenge) and the COCO competition, marking the first time an AI model exceeded human performance on certain recognition tasks.

- Introduced the concept of residual blocks to address issues of degradation problem, gradient vanishing and exploding of deep networks, enabling deeper networks.

    - Enabled the creation of significantly deeper models, such as 34, 50, 101, and 152-layer versions.

Case study of architecture

DenseNet

- 2017 CVPR

Densely Connected Convolutional Networks

Gao Huang*
Cornell University
gh349@cornell.edu

Zhuang Liu*
Tsinghua University
liuzhuang13@mails.tsinghua.edu.cn

Laurens van der Maaten
Facebook AI Research
lvdmaaten@fb.com

Kilian Q. Weinberger
Cornell University
kqw4@cornell.edu

Abstract

Recent work has shown that convolutional networks can be substantially deeper, more accurate, and efficient to train if they contain shorter connections between layers close to the input and those close to the output. In this paper, we embrace this observation and introduce the Dense Convolutional Network (DenseNet), which connects each layer to every other layer in a feed-forward fashion. Whereas traditional convolutional networks with L layers have L connections—one between each layer and its subsequent layer—our network has $\frac{L(L+1)}{2}$ direct connections. For each layer, the feature-maps of all preceding layers are used as inputs, and its own feature-maps are used as inputs into all subsequent layers. DenseNets have several compelling advantages: they alleviate the vanishing-gradient problem, strengthen feature propagation, encourage feature reuse, and substantially reduce the number of parameters. We evaluate our proposed architecture on four highly competitive object recognition benchmark tasks (CIFAR-10, CIFAR-100, SVHN, and ImageNet). DenseNets obtain significant improvements over the state-of-the-art on most of them, whilst requiring less computation to achieve high performance. Code and pre-trained models are available at https://github.com/liuzhuang13/DenseNet.

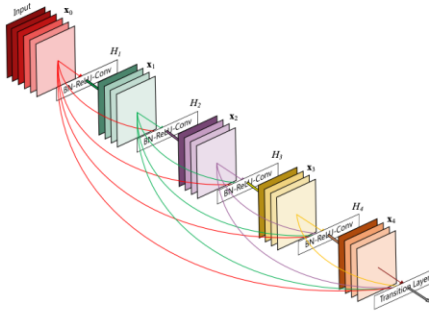


Figure 1: A 5-layer dense block with a growth rate of $k = 4$. Each layer takes all preceding feature-maps as input.

Networks [33] and Residual Networks (ResNets) [11] have surpassed the 100-layer barrier.

As CNNs become increasingly deep, a new research problem emerges: as information about the input or gradient passes through many layers, it can vanish and "wash out" by the time it reaches the end (or beginning) of the network. Many recent publications address this or related problems. ResNets [11] and Highway Networks [33] by-

Case study of architecture

DenseNet (Summary)

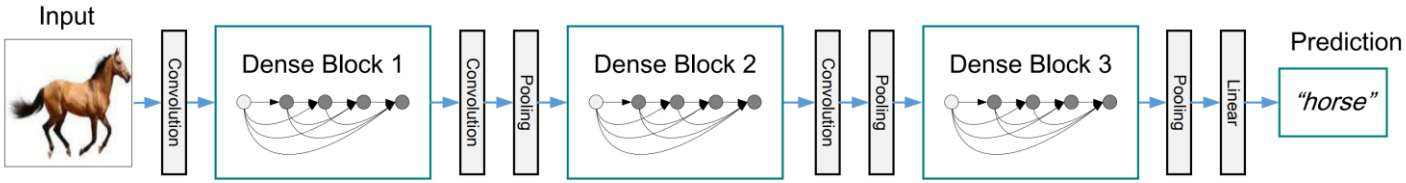


Figure 2: A deep DenseNet with three dense blocks. The layers between two adjacent blocks are referred to as transition layers and change feature-map sizes via convolution and pooling.

- DenseNet (Densely Connected Convolutional Networks) was introduced and achieved state-of-the-art results on benchmarks such as ImageNet
- Introduced the concept of **dense connectivity**, where each layer receives inputs from all previous layers and passes its feature maps to all subsequent layers.
- This design improves **feature reuse**, reduces the number of parameters, and strengthens gradient flow, addressing vanishing gradient problems.
- Enabled the construction of **efficient yet deep models** (e.g., DenseNet-121, DenseNet-169, DenseNet-201, DenseNet-264) with fewer parameters than comparable architectures.

Case study of architecture

DenseNet (Summary)

Model	top-1	top-5
DenseNet-121 ($k=32$)	25.02 (23.61)	7.71 (6.66)
DenseNet-169 ($k=32$)	23.80 (22.08)	6.85 (5.92)
DenseNet-201 ($k=32$)	22.58 (21.46)	6.34 (5.54)
DenseNet-161 ($k=48$)	22.33 (20.85)	6.15 (5.30)

Table 3: The top-1 and top-5 error rates on the ImageNet validation set, with single-crop (10-crop) testing.

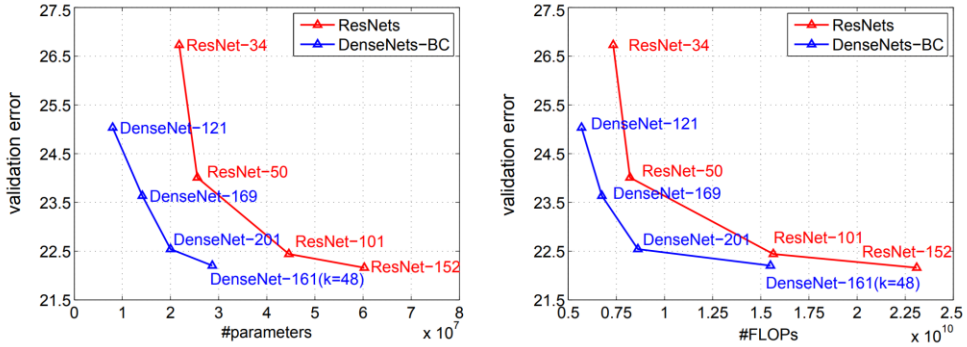


Figure 3: Comparison of the DenseNets and ResNets top-1 error rates (single-crop testing) on the ImageNet validation dataset as a function of learned parameters (*left*) and FLOPs during test-time (*right*).

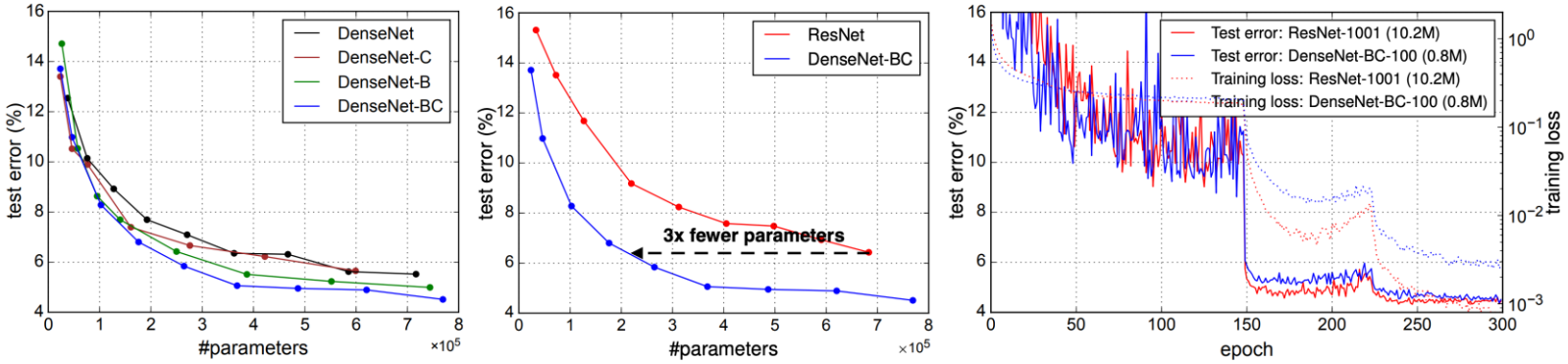


Figure 4: *Left:* Comparison of the parameter efficiency on C10+ between DenseNet variations. *Middle:* Comparison of the parameter efficiency between DenseNet-BC and (pre-activation) ResNets. DenseNet-BC requires about 1/3 of the parameters as ResNet to achieve comparable accuracy. *Right:* Training and testing curves of the 1001-layer pre-activation ResNet [12] with more than 10M parameters and a 100-layer DenseNet with only 0.8M parameters.