Convnets for Object Detection

Computer Vision (CSCI 4220U)

Faisal Z. Qureshi

http://vclab.science.ontariotechu.ca

Lesson Plan

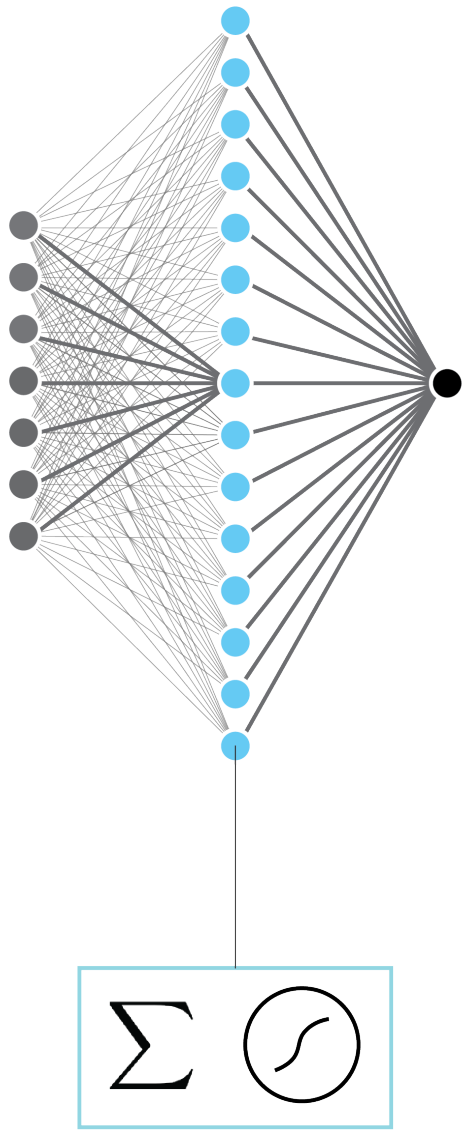
- Convolutional Networks
- Convolution
- Pooling layers
- Dilated convolutions
- Common architecture
  - GoogLeNet
  - ResNet
  - Densenet
  - Squeeze-and-Excitation network
- ConvNext

Convolutional networks for computer vision tasks

Task: We want to classify the following image

```
import cv2
import matplotlib.pyplot as plt

img = cv2.imread('./convnets/1.jpeg', 0)
plt.figure(figsize=(10,8))
plt.imshow(img, cmap='gray')
plt.xlabel('width')
plt.xticks([])
plt.ylabel('height')
plt.yticks([])
plt.title(f'Image dimensions: {img.shape[0]} x {img.shape[1]}.  #pixels: {img.shape[0] * img.shape[1]}');
```

Classical neural networks for computer vision tasks



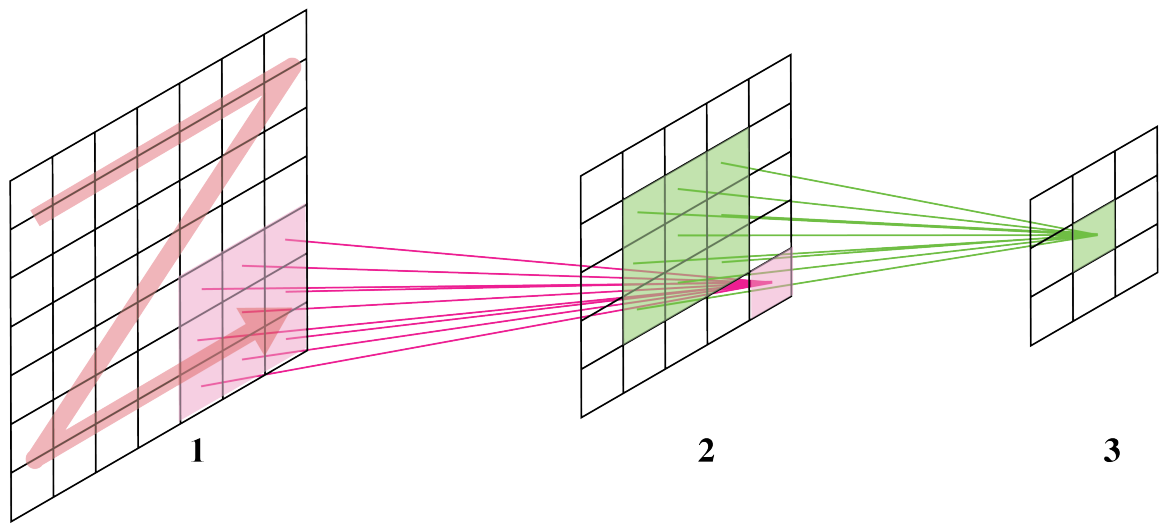
- **Q.** How many paramters per hidden layer unit?
- Computational issues
- Poor performance
  - The model is prone to overfitting
  - Model capacity issues

Convolutional neural network

- David Hubel and Torsten Wiesel studied cat visual cortex and showed that visual information goes through a series of processing steps: 1) edge detection; 2) edge combination; 3) motion perception; etc. (Hubeland Wiesel, 1959)
  - Neurons are spatially localized
  - Topographic feature maps
  - Hierarchical feature processing

Convolutional layers

- Convolutional layers achieve these properties
  - Each output unit is a linear function of a localized subset of input units
  - Same linear transformation is applied at each location
  - Local features detection is translation invariant



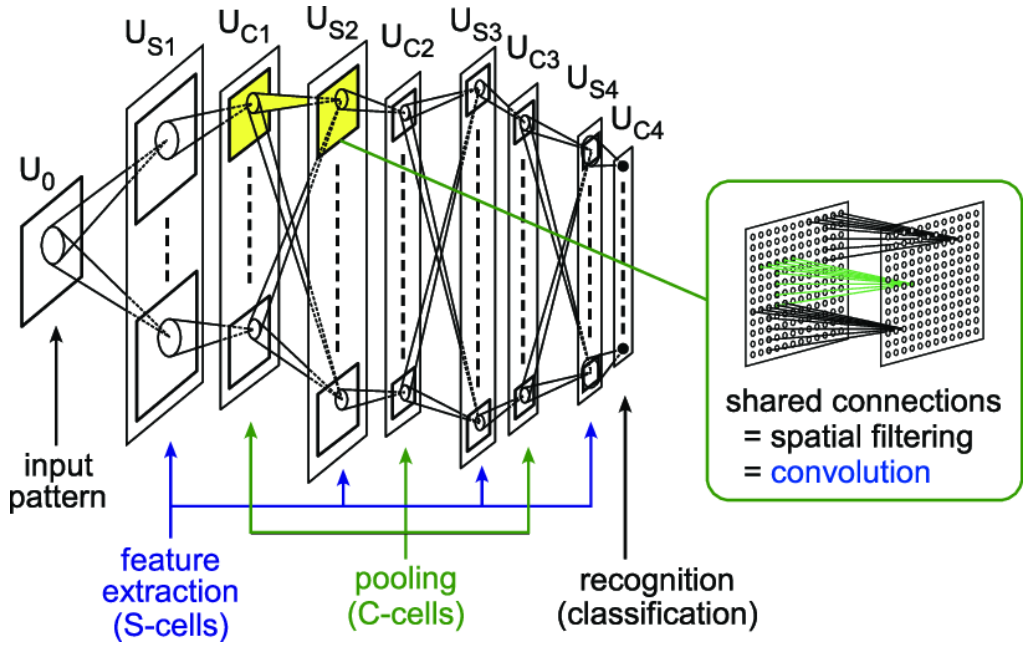
Convolutional layers

- Convolutional layers provide architectural constraints
- Number of parameters depend upon kernel sizes and not the size of the input
- Inductive bias
  - Examples:
    - Architectural constraints
    - Image augmentation
    - Regularization

Convolutional Neural Networks

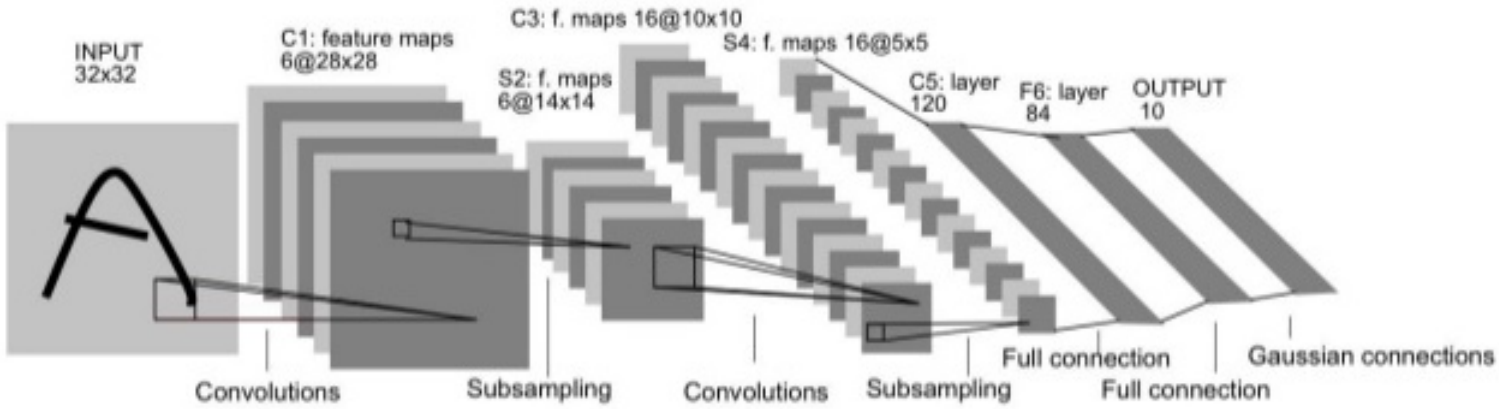
- Proposed in 1980 by Kunihiko Fukushima

  Neocognitron: A self-organizing neural network model for a mechanism of pattern recognition unaffected by shift in position, Biological Cybernetics. 1980



LeNet

- Classifying digits (LeCun 1988)



LeNet

- The first few layers are convolution layers, and the last few layers are fully connected layers

LeNet

- The first few layers are convolution layers, and the last few layers are fully connected layers

- **Q.** Why?
  - The convolutional layers are compute heavy, but have fewer parameters
  - The fully connected layer have far more parameters, but these are easy to compute
  - Convolutional layers compute *features*
  - Linear layers implement structure similar to the classical *aritifical neural networks*

AlexNet

- ImageNet Large Scale Visual Recognition Competition, 2012
  - Top-5 error of 15.3%, more than 10.8 percentage points lower than that of the runner up.
- Model depth (i.e., the number of layers) is important for performance
  - Computational expensive; requires GPUs for training

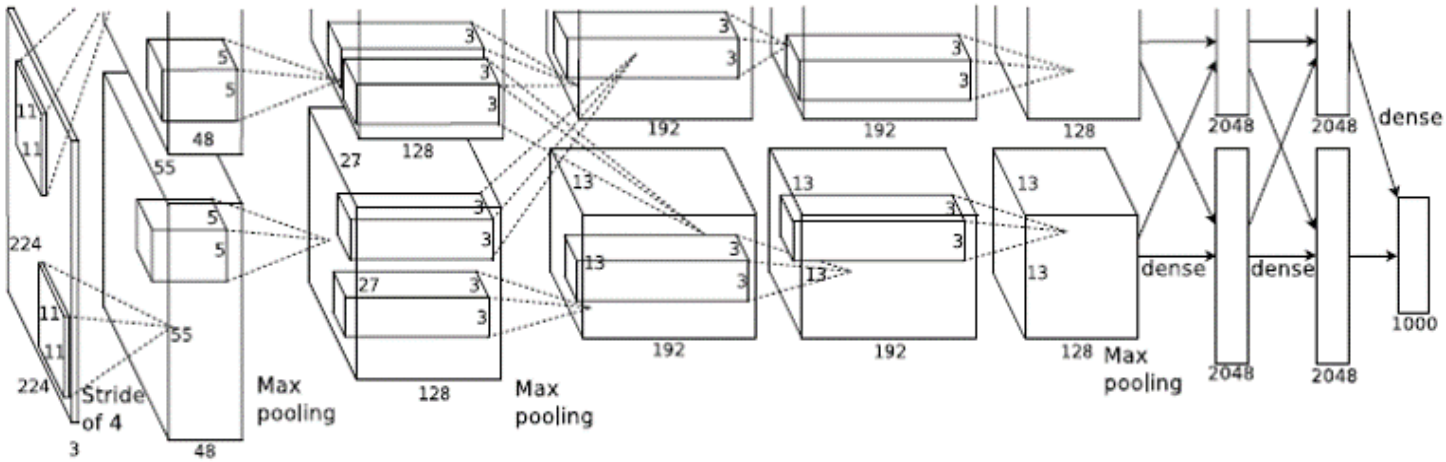


Figure from Krizhevsky, Sutskever, and Hinton, 2012

ImageNet

- https://www.image-net.org/index.php
- Currently ImageNet has 14 million images and roughly 21,000 classes.

LeNet vs. AlexNet

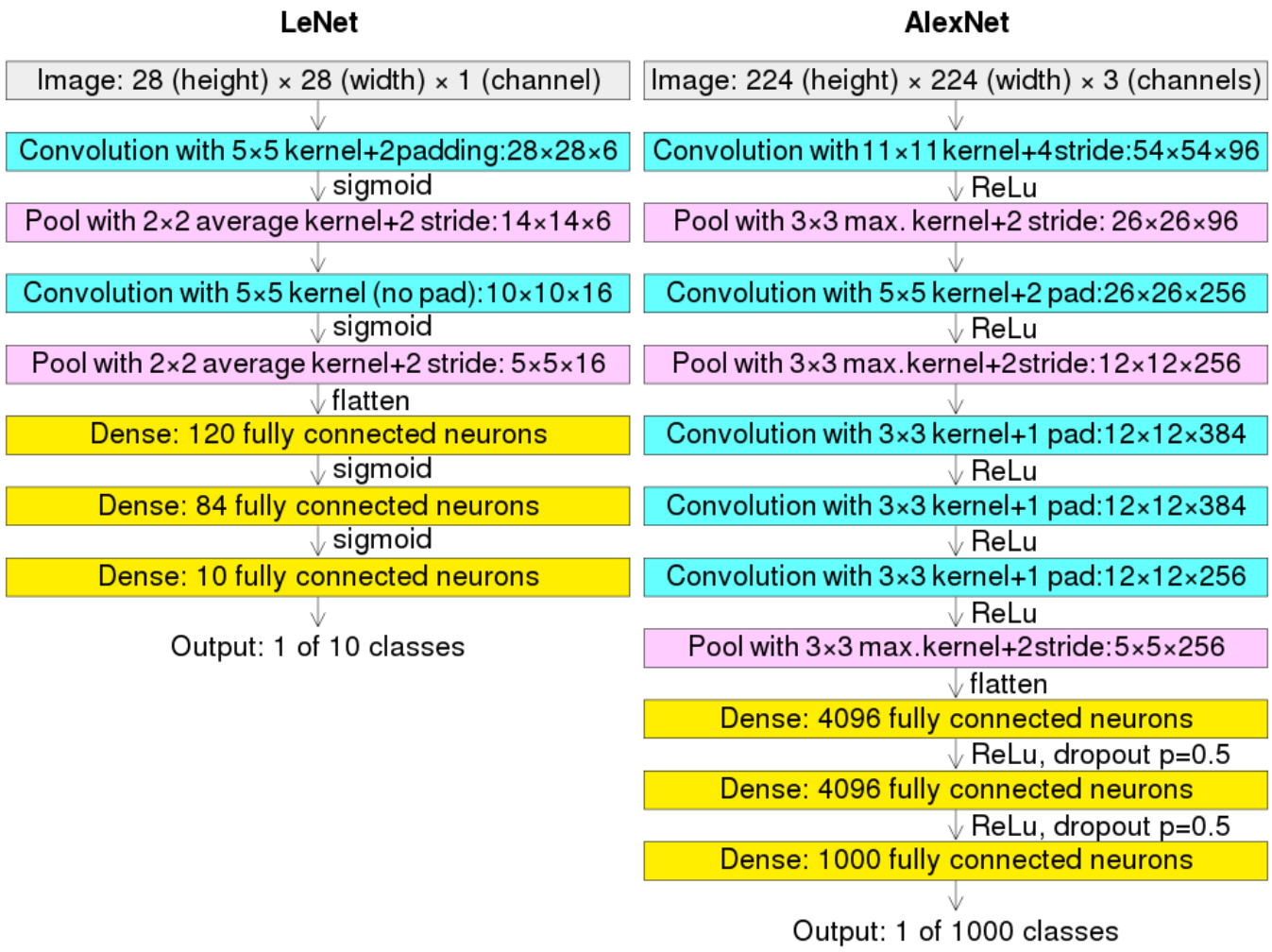
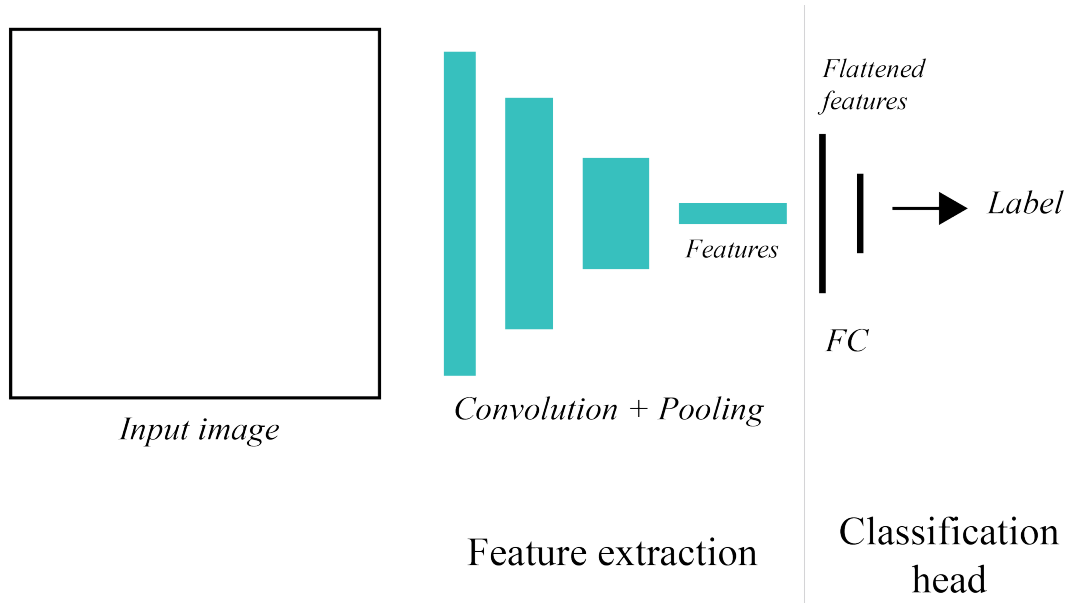


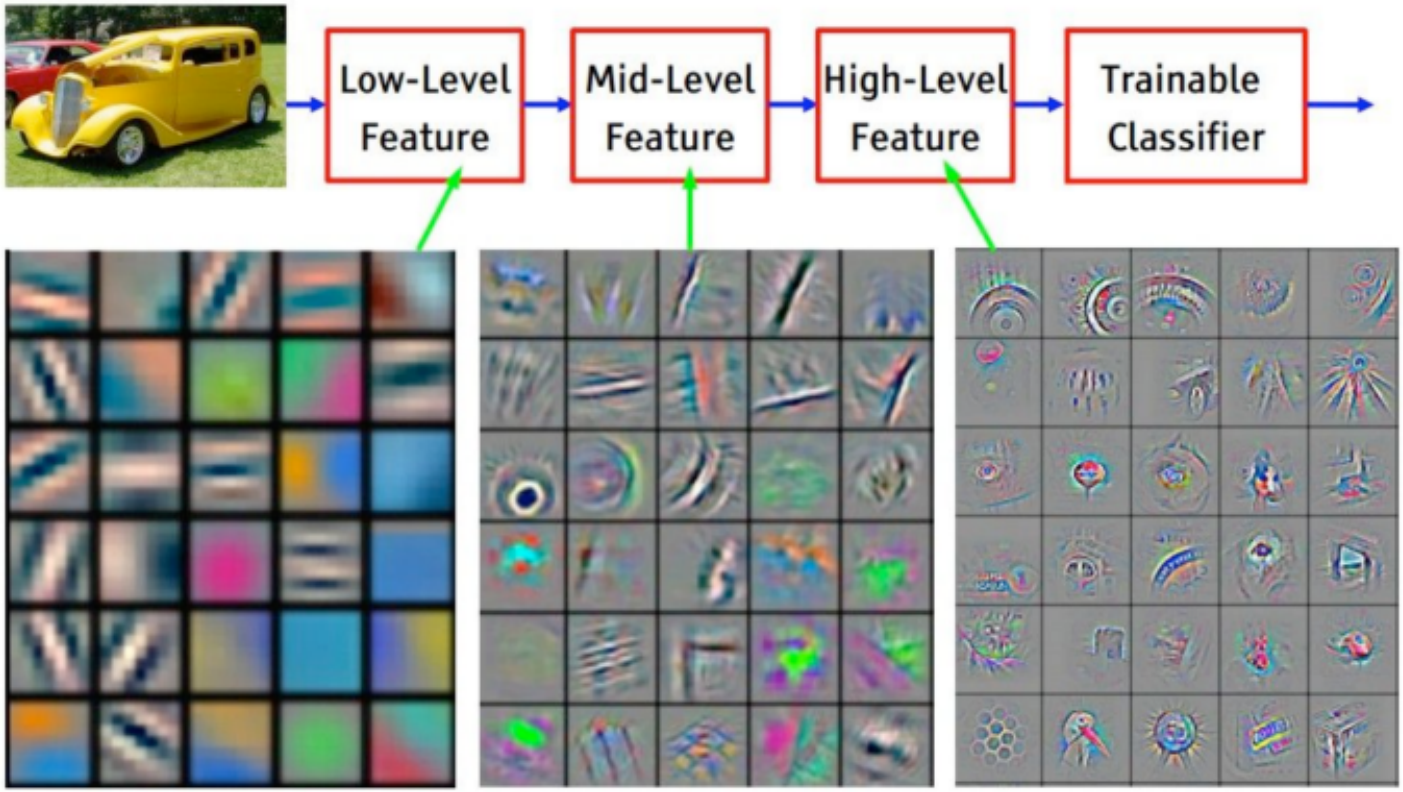
Figure from Wikipedia

General idea

- Generally speaking we can interpret convolutional deep networks as composed of two parts: 1) a (latent) feature extractor and 2) task head.
  - Feature extractor learns to construct powerful representations given an input. These representations are well-suited to the task at hand.
  - The task head uses these features to perform that task, e.g., classification, regression, etc.

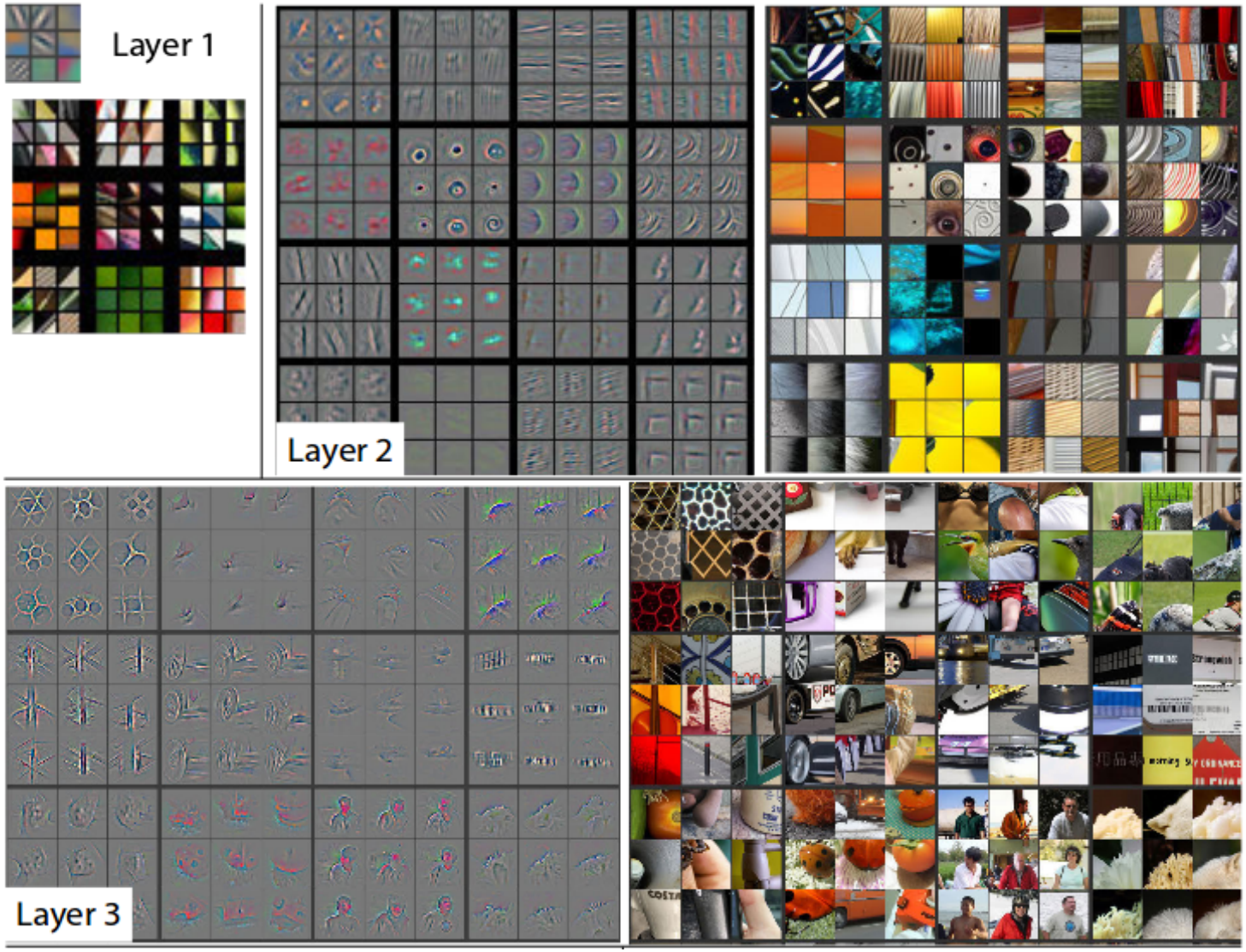


Feature Maps



Feature visualization of convolutional net trained on ImageNet from [Zeiler & Fergus 2013]

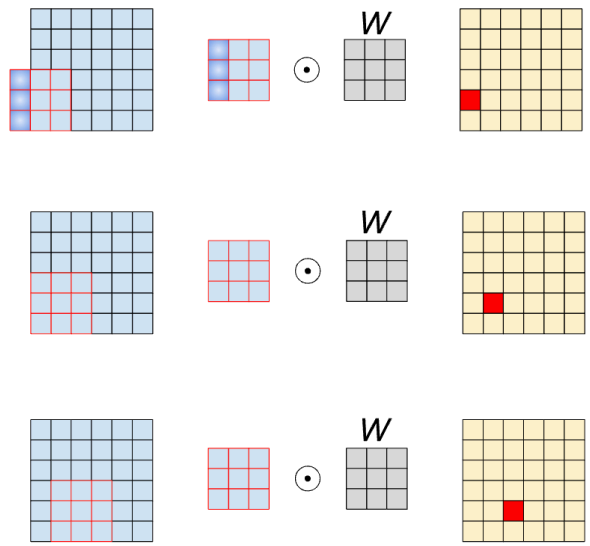
Feature Maps



Convolution

- In a nutshell: point-wise multiplication and sum

$$(\mathbf{f} * \mathbf{k})_i = \sum_{k \in [-w,w]} \mathbf{f}(i-k)\mathbf{h}(k)$$



Exercise: computing a 1D convolution (from scratch)

Compute $\mathbf{f} * \mathbf{h}$ given

$$\mathbf{f} = \begin{bmatrix} 1 & 3 & 4 & 1 & 10 & 3 & 0 & 1 \end{bmatrix}$$

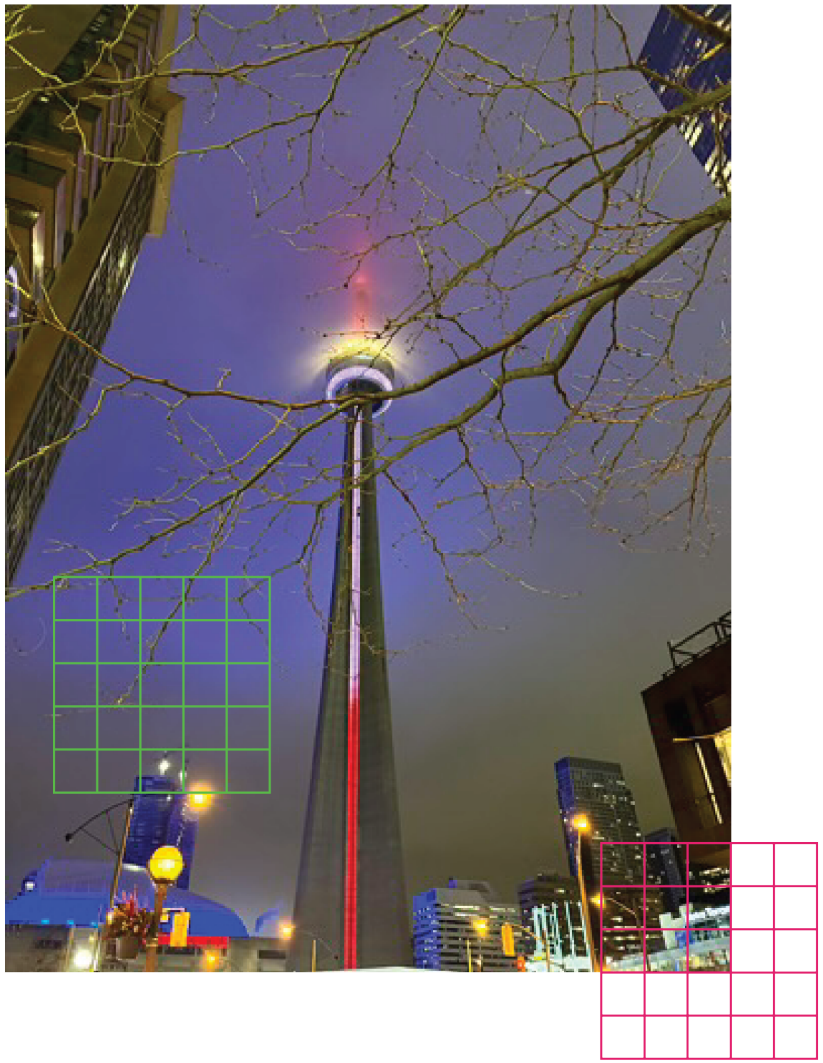
and

$$\mathbf{h} = \begin{bmatrix} 1 & 0 & -1 \end{bmatrix}$$

```
import numpy as np

f = np.array([1,3,4,1,10,3,0,1])
h = np.array([1,0,-1])
width = 1
result = np.ones(len(f)-2*width)   # Array to store computed moving averages
for i in range(len(result)):
    centre = i + width
    result[i] = np.dot(h[::-1], f[centre-width:centre+width+1]) # Note the flip
print(f'signal f:\t\t{f}')
print(f'kernel h:\t\t{h}')
print(f'convolution (f*h):\t{result}')
```

Dealing with edges

- Clipping
- Replication
- Symmetric padding
- Circular padding



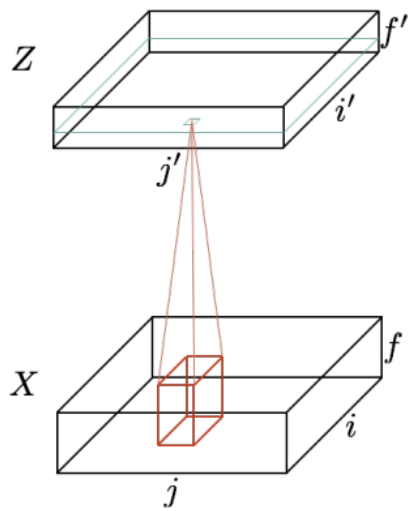
Convolutions as matrix-vector multiplication

- Exercise: Describe 1D convolution as a matrix-vector multiplication.

Convolution layer

- We can define the convolution layer used in deep networks as follows

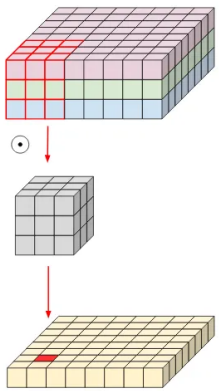
$$\mathbf{z}_{i',j',f'} = b_{f'} + \sum_{i=1}^{H_f} \sum_{j=1}^{W_f} \sum_{f=1}^{F} \mathbf{x}_{i'+i-1,j'+j-1,f} \theta_{ijff'}.$$



Convolution layer

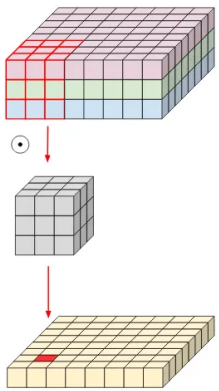
Convolution layer

Normal

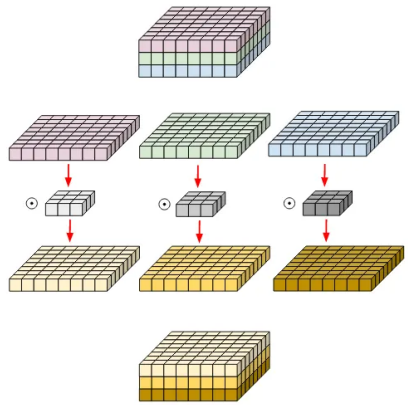


Convolution layer

Normal

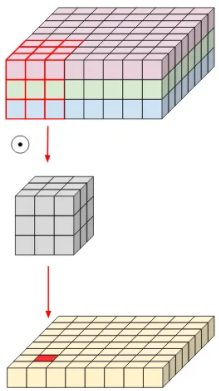


Depthwise

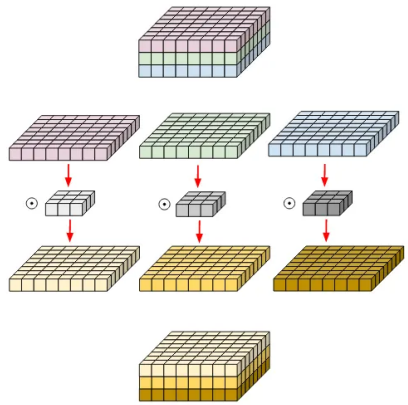


Convolution layer

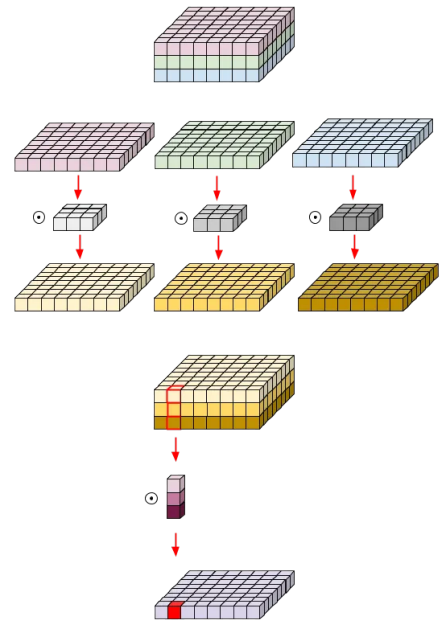
Normal



Depthwise

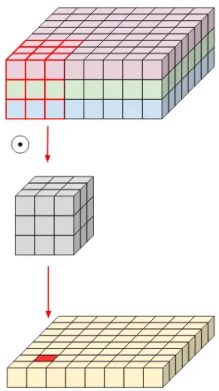


Depthwise Separable

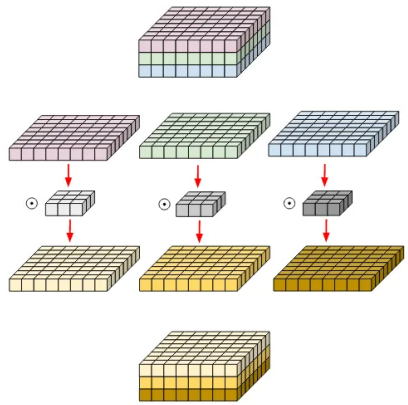


Convolution layer

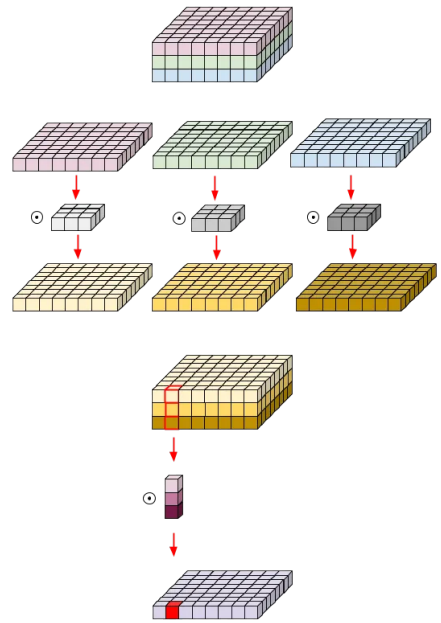
Normal



Depthwise



Depthwise Separable



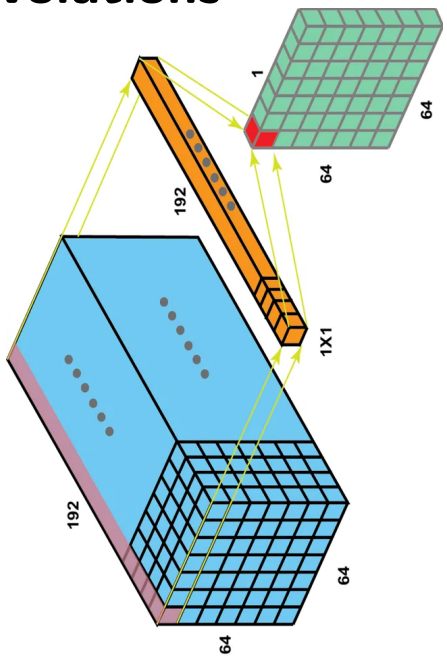
Depthwise Separable convolution

- Far fewer parameters than normal convolution
- Computational efficiency
- Prevents overfitting

Convolution layer

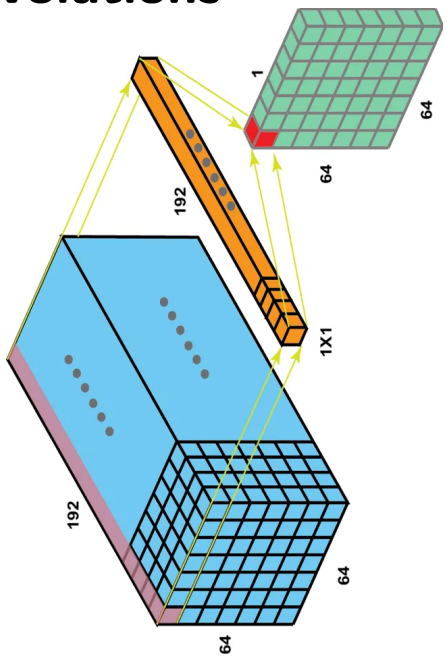
Convolution layer

1x1 convolutions



Convolution layer

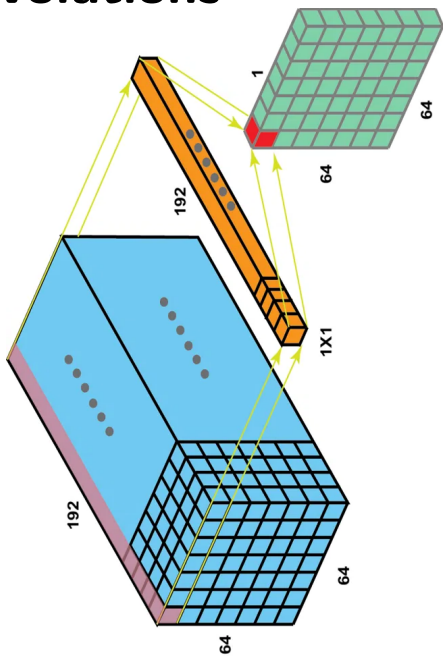
1x1 convolutions



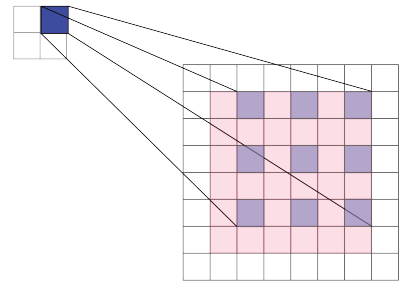
- Change feature dimensions
- Dimensionality reduction
- Reduce computational load
- Add additional non-linearity
- Implement bottleneck layer

Convolution layer

1x1 convolutions



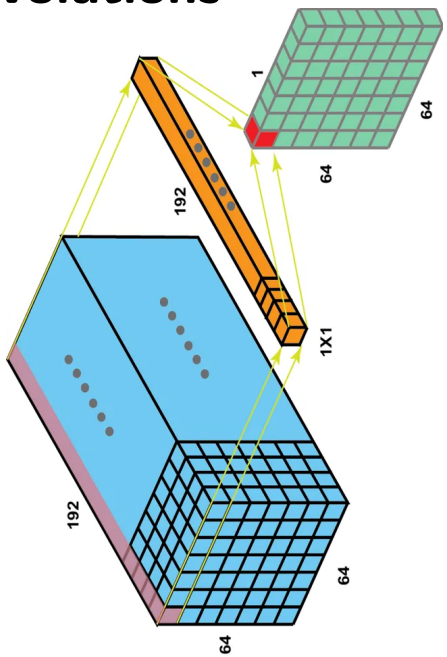
Dilated or atrous convolutions



- Change feature dimensions
- Dimensionality reduction
- Reduce computational load
- Add additional non-linearity
- Implement bottleneck layer

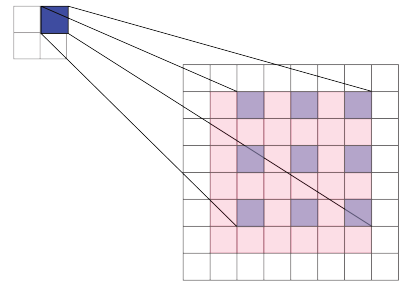
Convolution layer

1x1 convolutions



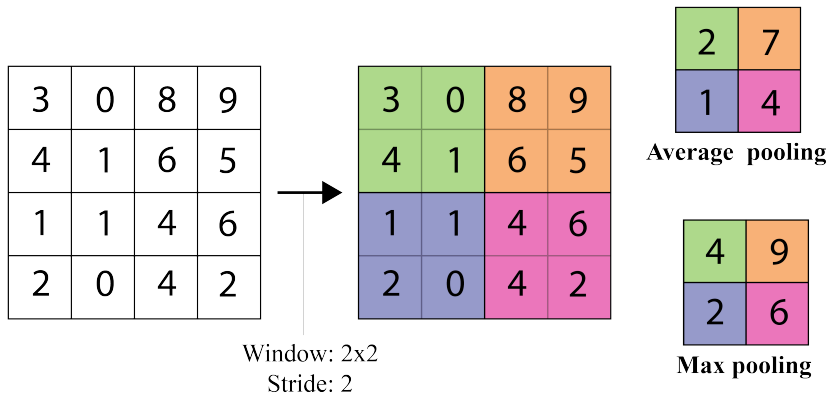
- Change feature dimensions
- Dimensionality reduction
- Reduce computational load
- Add additional non-linearity
- Implement bottleneck layer

Dilated or atrous convolutions



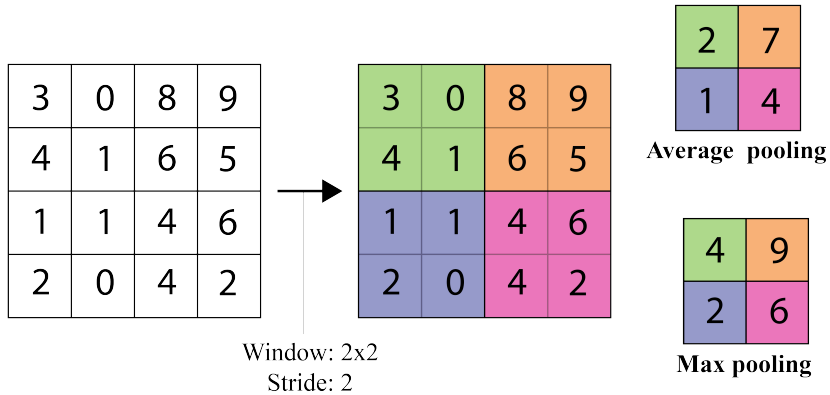
- Increased receptive field without increasing parameters
- Can capture features at multiple scales
- Reduced spatial resolution loss compared to regular convolutions with larger filters
- Increased computational cost

Pooling layers

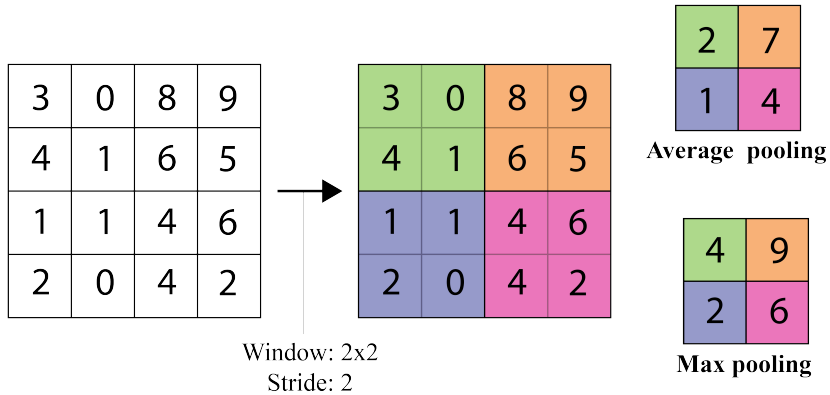


- Pooling layers are commonly used after convoluational layers
  - Decrease feature dimensions
  - Create some invariance to shifts

Pooling layers



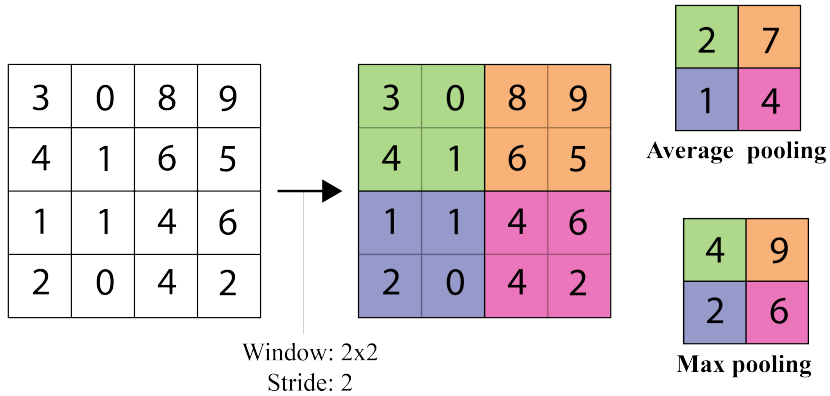
Pooling layers



- Average pooling

$$h_k(x, y, c) = \frac{1}{\mathcal{N}(x, y)} \sum_{(i,j) \in \mathcal{N}_(x,y)} h_{k-1}(i, j, c)$$

Pooling layers



- Average pooling

$$h_k(x, y, c) = \frac{1}{\mathcal{N}(x, y)} \sum_{(i,j) \in \mathcal{N}_(x,y)} h_{k-1}(i, j, c)$$

- Maxpooling

$$h_k(x, y, c) = \underset{(i,j) \in \mathcal{N}(x,y)}{\arg\max} \; h_{k-1}(i, j, c)$$

Pooling layers

- Springenberg et al. 2015 (ICLR workshops)

  maxpooling can simply be replaced by a convolutional layer with increased stride without loss in accuracy on several image recognition benchmarks

Common architectures

Run the following code to see the list of models included in `torchvision.models`

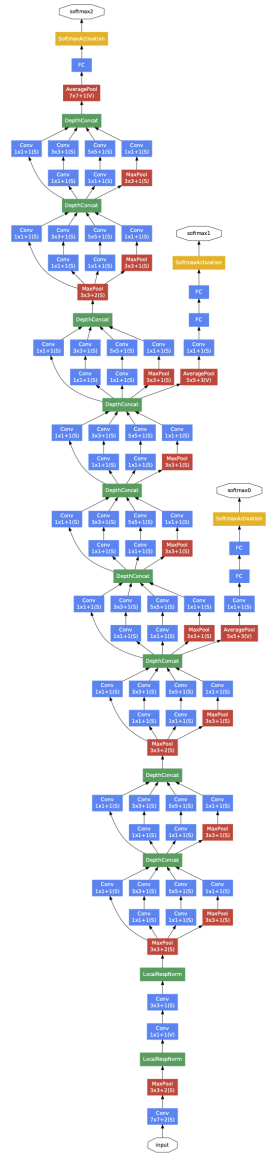
```
import torchvision
import torchvision.models as m
import pprint as pp
print(f'torchvision\nVERSION: {torchvision.__version__}')
print('MODELS:')
pp.pprint([x for x in dir(m) if x[0] != '_'])

# print out the structure of vgg16 model
vgg16 = m.vgg16()
print(vgg16)
```

GoogLeNet

- Szegedy et al. 2014

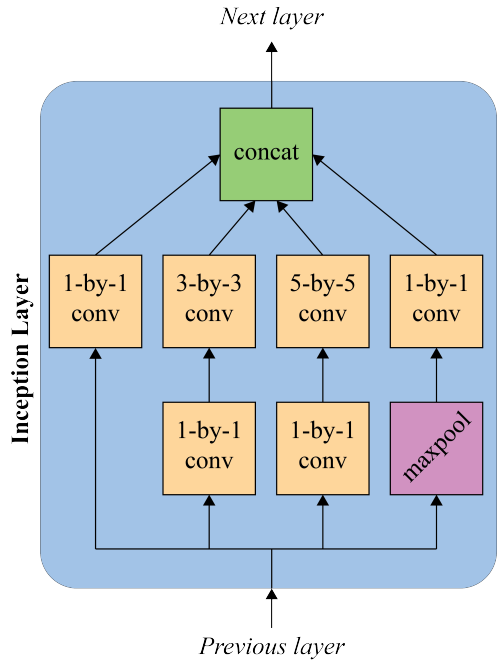
  Going Deeper with Convolutions

- Multiple feed-forward passes
- Inception module
  - An inception module aims to approximate local sparse structure in a CNN by using filters of different sizes (within the same block) whose output is concatenated and passed on to the next stage



Inception layer

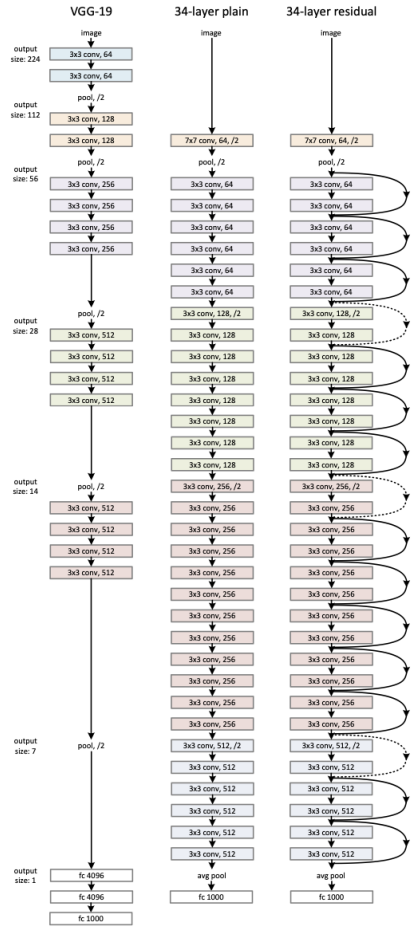
- Acts as a bottleneck layer
- 1-by-1 convolutional layers are used to reduce feature channels



Inception layer (simplified). Each conv is followed by a non-linear activation.

ResNet

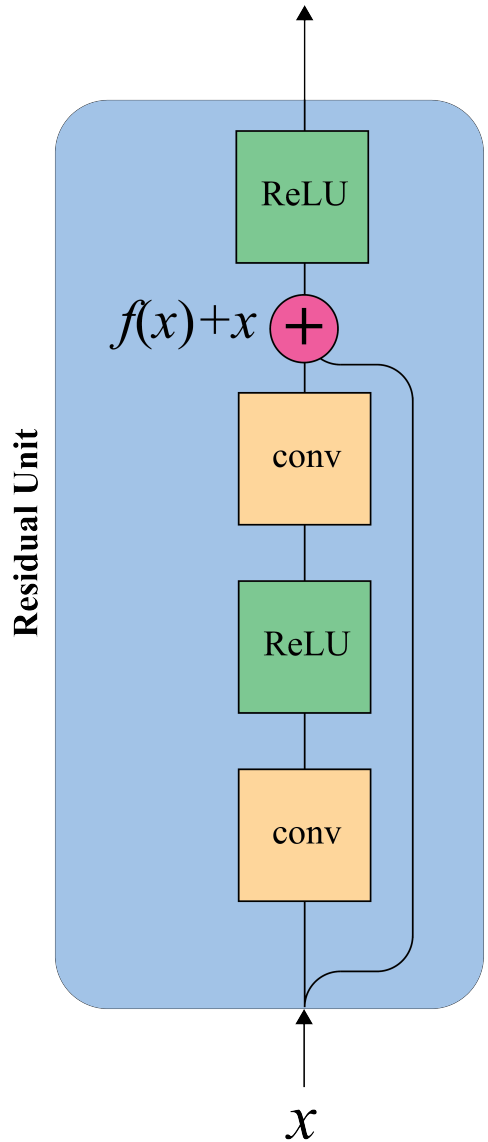
- He et al. 2016
  Deep Residual Learning for Image Recognition



Left: the VGG-19 model (19.6 billion FLOPs) as a reference. Middle: a plain network with 34 parameter layers (3.6 billion FLOPs). Right: a residual network with 34 parameter layers (3.6 billion FLOPs). The dotted shortcuts increase dimensions. Figure from He et al. 2016.

Residual unit

- Pass through connections adds the input of a layer to its output
- Deeper models are harder to train
- Learn residual function rather than direct mapping



Residual unit

- Notice the loss landscape with (right) and without (left) residual connections

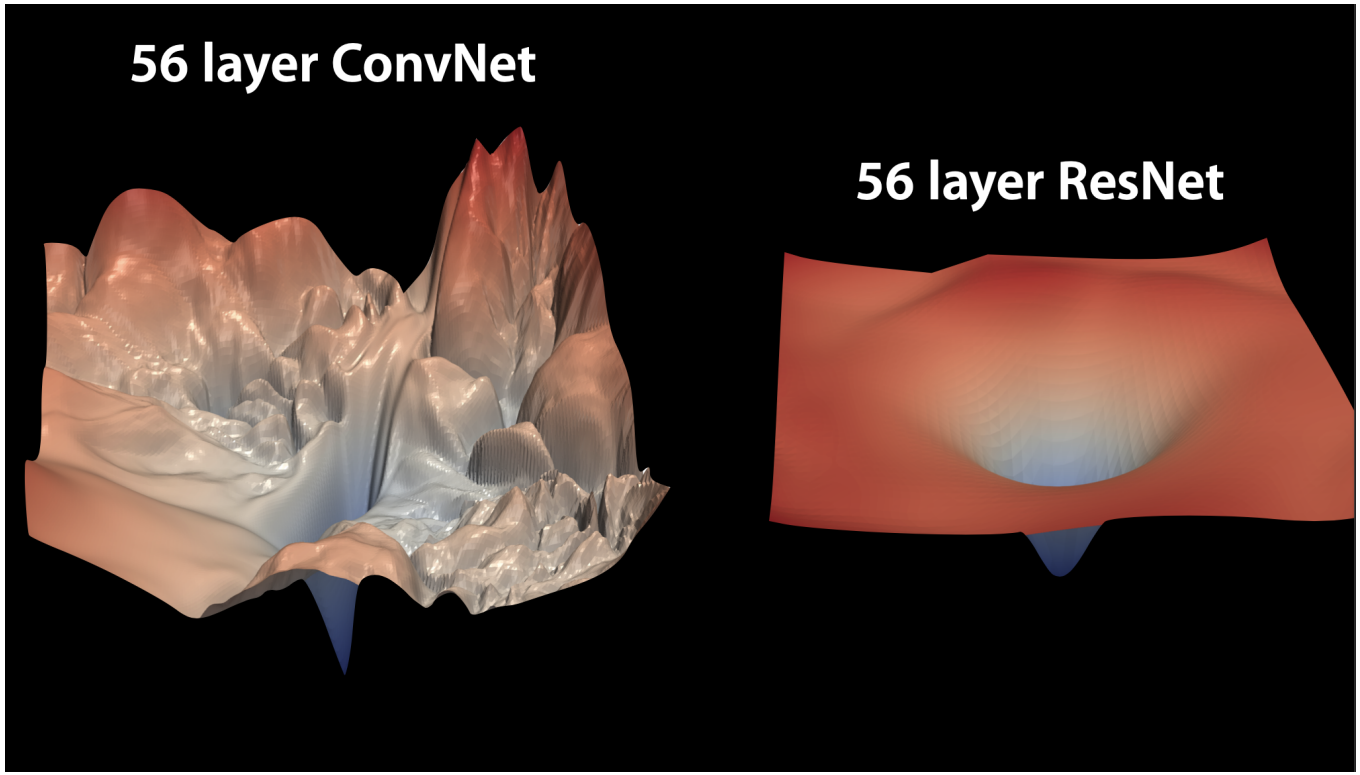


Figure taken from K. Derpanis notes on deep learning.

Densenet

- Huang et al. 2017

  Densely Connected Convolutional Networks

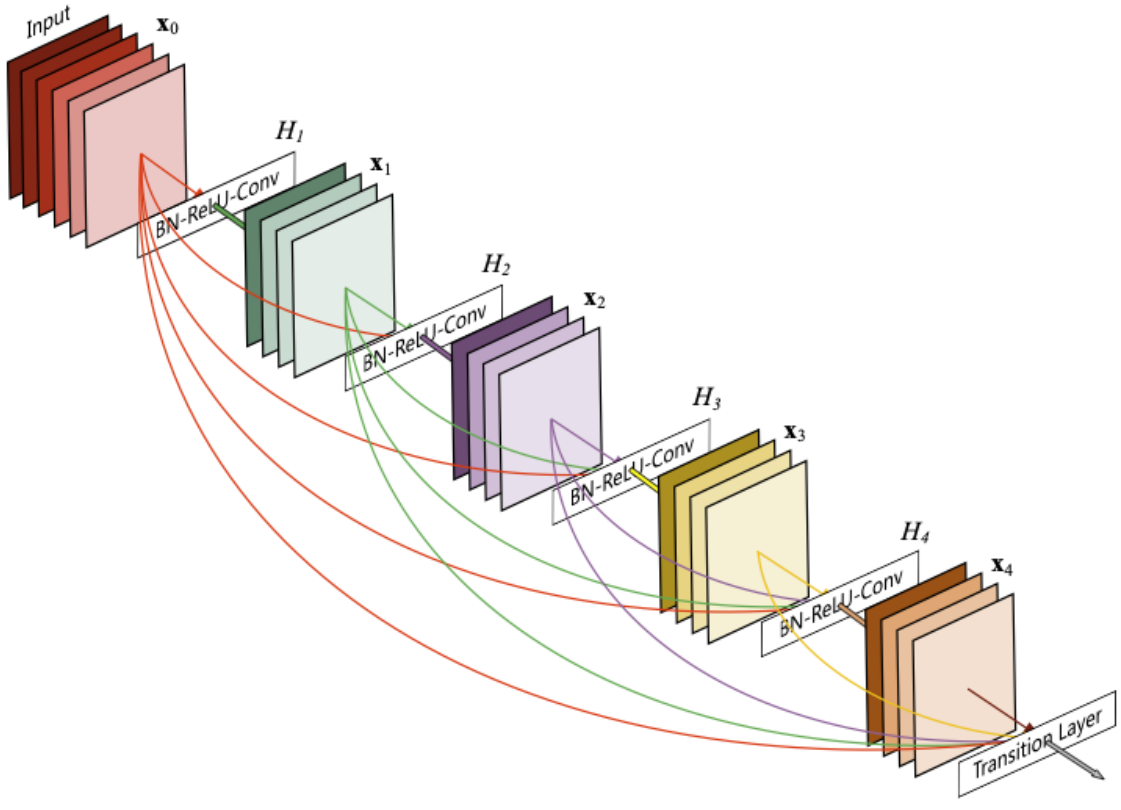
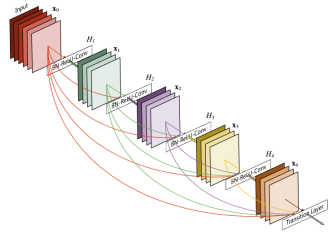


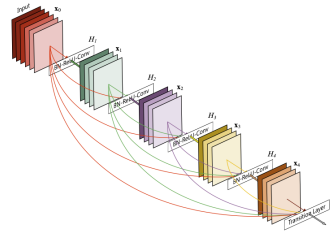
Figure from Huang et al. 2017.

Densenet



- Feature-maps learned by any of the layers can be accessed by all subsequent layers.
  - Encourages feature reuse throughout the network
  - Leads to more compact models
  - Supports diversified depth

Densenet



- Feature-maps learned by any of the layers can be accessed by all subsequent layers.
  - Encourages feature reuse throughout the network
  - Leads to more compact models
  - Supports diversified depth
- Improved training
  - Individual layers get additional supervision from loss function through shorter (more direct) connections
    - Similar to DSN (Lee et al. 2015) that attach classifiers to each hidden layer forcing intermediate layers to learn discriminative features
  - Scale to hundreds of layers without any optimization difficulties

Dense blocks and transition layers

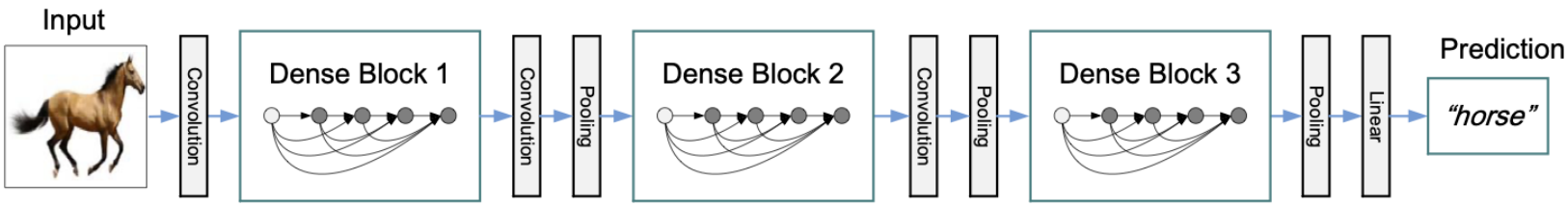


Figure 2: A deep DenseNet with three dense blocks. The layers between two adjacent blocks are referred to as transition layers and change feature-map sizes via convolution and pooling.

Figure from Huang et al. 2017

Densenet vs. Resnet

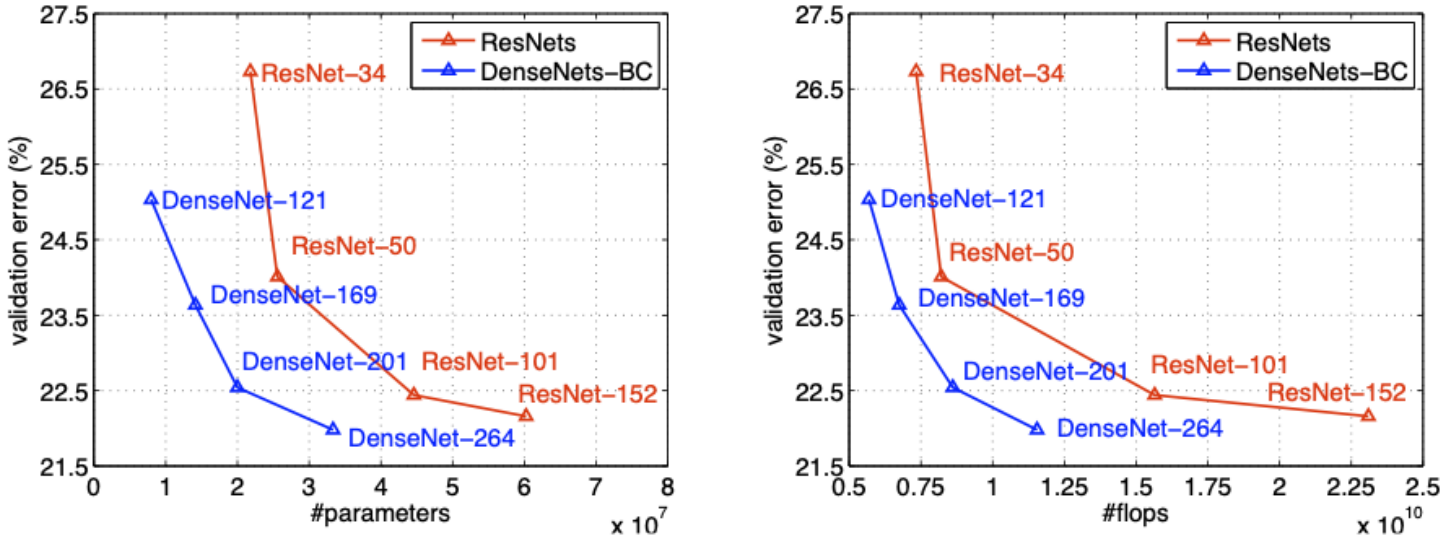


Figure 3: Comparison of the DenseNets and ResNets top-1 error rates (single-crop testing) on the ImageNet validation dataset as a function of learned parameters (*left*) and FLOPs during test-time (*right*).

Figure from Huang et al. 2017

Squeeze-and-Excitation Networks

- Hu et al. 2018
  Squeeze-and-Excitation Networks

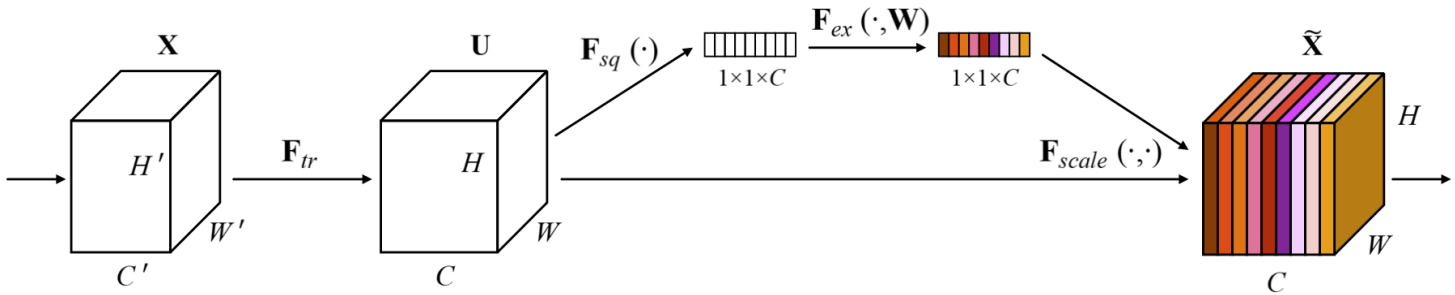
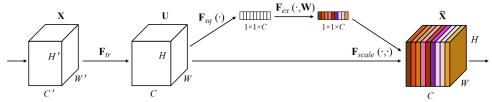


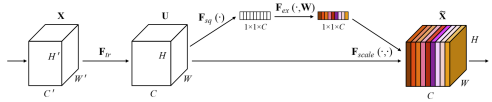
Figure from Hu et al. 2018

SE Block



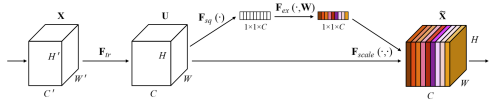
- Squeeze operator
  - Allows global information to be used when computing channel-wise weights

SE Block



- Squeeze operator
  - Allows global information to be used when computing channel-wise weights
- Excitation operator
  - Distribution across different classes is similar in early layers, suggesting that feature channels are “equally important” for different classes in early layers
  - Distribution becomes class-specific in deeper layers

SE Block



- Squeeze operator
  - Allows global information to be used when computing channel-wise weights
- Excitation operator
  - Distribution across different classes is similar in early layers, suggesting that feature channels are "equally important" for different classes in early layers
  - Distribution becomes class-specific in deeper layers
- SE blocks may be used for model prunning and network compression

SE Performance

TABLE 4
Classification error (%) on CIFAR-10.

	original	SENet
ResNet-110 [14]	6.37	5.21
ResNet-164 [14]	5.46	4.39
WRN-16-8 [67]	4.27	3.88
Shake-Shake 26 2x96d [68] + Cutout [69]	2.56	2.12

TABLE 5
Classification error (%) on CIFAR-100.

	original	SENet
ResNet-110 [14]	26.88	23.85
ResNet-164 [14]	24.33	21.31
WRN-16-8 [67]	20.43	19.14
Shake-Even 29 2x4x64d [68] + Cutout [69]	15.85	15.41

TABLE 6
Single-crop error rates (%) on Places365 validation set.

	top-1 err.	top-5 err.
Places-365-CNN [72]	41.07	11.48
ResNet-152 (ours)	41.15	11.61
SE-ResNet-152	**40.37**	**11.01**

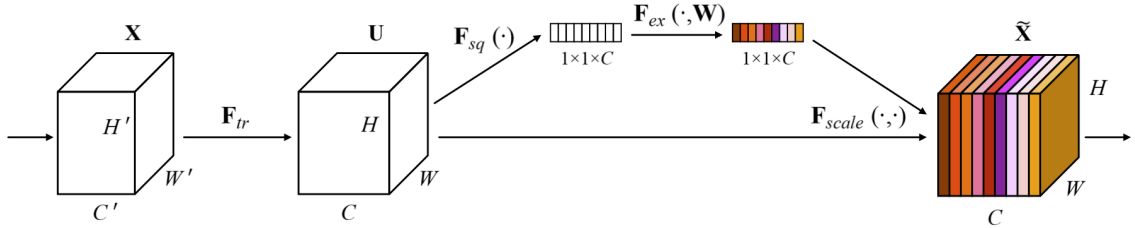
TABLE 7
Faster R-CNN object detection results (%) on COCO *minival* set.

	AP@IoU=0.5	AP
ResNet-50	57.9	38.0
SE-ResNet-50	61.0	40.4
ResNet-101	60.1	39.9
SE-ResNet-101	62.7	41.9

Taken from Hu et al. 2018

Spatial attention

- SE computes channel weights; however, we can easily extend this idea to compute spatial weights to model some notion of spatial attention
  - The model will pay more attention to f



Other notable examples

- Larsson et al., 2016

  FractalNet: Ultra-Deep Neural Networks without Residuals

- Iandola et al., 2016

  SqueezeNet: AlexNet-Level Accuracy with 50x Fewer Parameters and <0.5MB Model Size

- Howard et al., 2017

  MobileNet: Efficient Convolutional Neural Networks for Mobile Vision Applications

Other notable examples

- Xie et al., 2017

  Aggregated Residual Transformation for Deep Neural Networks

- Han et al., 2016

  Deep Pyramidal Residual Networks

- Chollet, 2017

  Xception: Deep Learning with Depthwise Separable Convolutions

Attention-based Networks

Transformers (2017)

- Vaswani et al., 2017

  Attention Is All You Need

- Transformers use attention-based computation.
- These models are popular in Natural Language Processing community.
- GPT3 language model also uses attention-based computation and it has roughly 175 billion parameters.

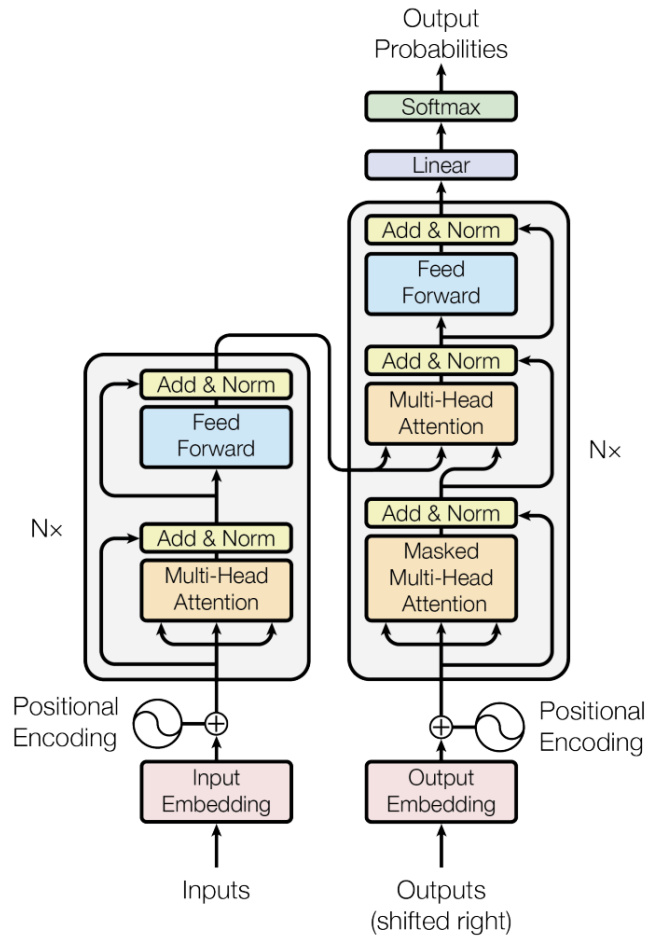


Figure 1: The Transformer - model architecture.

Attention

- Zhao et al., 2020

  Exploring Self-attention for Image Recognition

- Carion et al., 2020

  End-to-End Object Detection with Transformers

- Dosovitsky et al., 2020

  An Image is Worth 16x16 Words: Transformers for Image Recognition at Scale

- Zheng et al., 2020

  Rethinking Semantic Segmentation from a Sequence-to-Sequence Perspective with Transformers

Learning convolution kernels

- Observation
  - CNNs benefit from different kernel sizes at different layers
  - Exploring all possible combinations of kernel sizes is infeasible in practice
- Romero et al. 2022

  FlexConv: Continuous Kernel Convolutions with Differentiable Kernel Sizes

- Riad et al. 2022

  Learning Strides in Convolutional Neural Networks

MLPs

- Tolstikhin et al. 2021
  MLP-Mixer: An all-MLP Architecture for Vision

- Melas-Kyriazi 2021
  Do You Even Need Attention? A Stack of Feed-Forward Layers Does Surprising Well on ImageNet

- Touvron et al. 2021
  ResMLP: Feedforward Networks for Image Classification with Data Efficient Training

A ConvNet for 2020

- Liu et al. 2020
  A ConvNet for 2020s

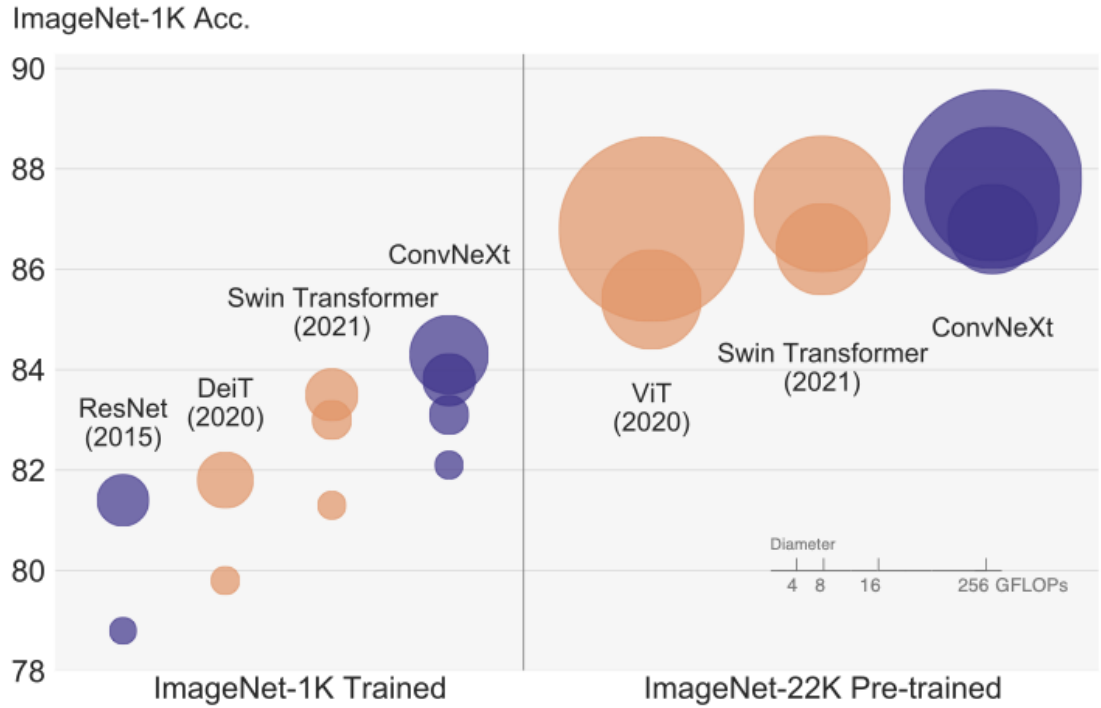


Figure 1. **ImageNet-1K classification** results for • ConvNets and ○ vision Transformers. Each bubble's area is proportional to FLOPs of a variant in a model family. ImageNet-1K/22K models here take $224^2/384^2$ images respectively. ResNet and ViT results were obtained with improved training procedures over the original papers. We demonstrate that a standard ConvNet model can achieve the same level of scalability as hierarchical vision Transformers while being much simpler in design.

Key ideas

- Change stem to Patchify

  Replace the ResNet-style stem cell with a patchify layer implemented using a 4×4*, stride* 4 *convolutional layer. The accuracy has changed from* 79.4% *to* 79.5%*.*

  The stem cell in standard ResNet contains a 7×7 *convolution layer with stride* 2*, followed by a max pool, which results in a* $4\times$ *downsampling of the input images.*

Key ideas

- ResNeXtify
  - Grouped convolutions idea from Xie et al. 2016
    - Depthwise convolution where the number of groups equal to the number of channels. Similar to MobileNet and Xception.
    - Only mixes information in the spatial domain.

The combination of depthwise conv and 1×1 *convs leads to a separation of spatial and channel mixing, a property shared by vision Transformers, where each operation either mixes information across spatial or channel dimension, but not both.*

Key ideas

- Inverted bottleneck

  One important design in every Transformer block is that it creates an inverted bottleneck, i.e., the hidden dimension of the MLP block is four times wider than the input dimension.

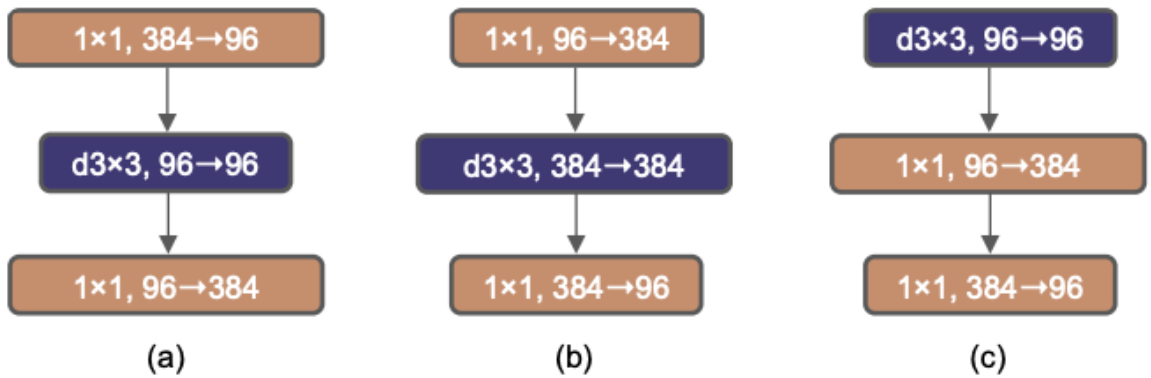


Figure 3. **Block modifications and resulted specifications. (a)** is a ResNeXt block; in **(b)** we create an inverted bottleneck block and in **(c)** the position of the spatial depthwise conv layer is moved up.

Figure from Lie et al. 2020

Key ideas

- Large kernel sizes

  One of the most distinguishing aspects of vision Transformers is their non-local self-attention, which enables each layer to have a global receptive field.

  To explore large kernels, one prerequisite is to move up the position of the depthwise conv layer.

Key ideas

- Replacing ReLU with GELU
  - Gaussian Error Linear Unit (Hendrycks and Gimpel, 2016)
- Use fewer activation functions
- Use fewer normalization layers
  - Replace batch normalization with layer normalization

ConvNext

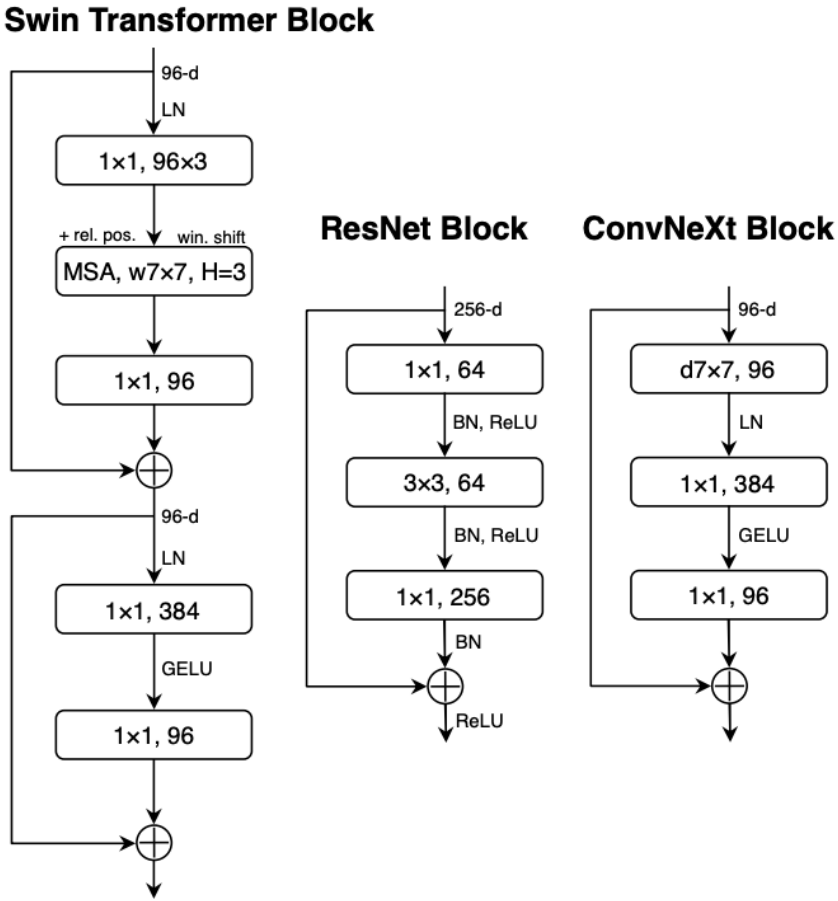


Figure 4. **Block designs** for a ResNet, a Swin Transformer, and a ConvNeXt. Swin Transformer's block is more sophisticated due to the presence of multiple specialized modules and two residual connections. For simplicity, we note the linear layers in Transformer MLP blocks also as "1×1 convs" since they are equivalent.

Figure from Lie et al. 2020

ConvNext

	output size	• ResNet-50	• ConvNeXt-T	○ Swin-T
stem	56×56	7×7, 64, stride 2 3×3 max pool, stride 2	4×4, 96, stride 4	4×4, 96, stride 4
res2	56×56	[1×1, 64 3×3, 64 1×1, 256] × 3	[d7×7, 96 1×1, 384 1×1, 96] × 3	[1×1, 96×3 MSA, w7×7, H=3, rel. pos. 1×1, 96 1×1, 384 1×1, 96] × 2
res3	28×28	[1×1, 128 3×3, 128 1×1, 512] × 4	[d7×7, 192 1×1, 768 1×1, 192] × 3	[1×1, 192×3 MSA, w7×7, H=6, rel. pos. 1×1, 192 1×1, 768 1×1, 192] × 2
res4	14×14	[1×1, 256 3×3, 256 1×1, 1024] × 6	[d7×7, 384 1×1, 1536 1×1, 384] × 9	[1×1, 384×3 MSA, w7×7, H=12, rel. pos. 1×1, 384 1×1, 1536 1×1, 384] × 6
res5	7×7	[1×1, 512 3×3, 512 1×1, 2048] × 3	[d7×7, 768 1×1, 3072 1×1, 768] × 3	[1×1, 768×3 MSA, w7×7, H=24, rel. pos. 1×1, 768 1×1, 3072 1×1, 768] × 2
FLOPs		4.1×10^9	4.5×10^9	4.5×10^9
# params.		25.6×10^6	28.6×10^6	28.3×10^6

Table 9. **Detailed architecture specifications** for ResNet-50, ConvNeXt-T and Swin-T.

Figure from Lie et al. 2020

Aside: Normalization techniques

Wu and He, 2018

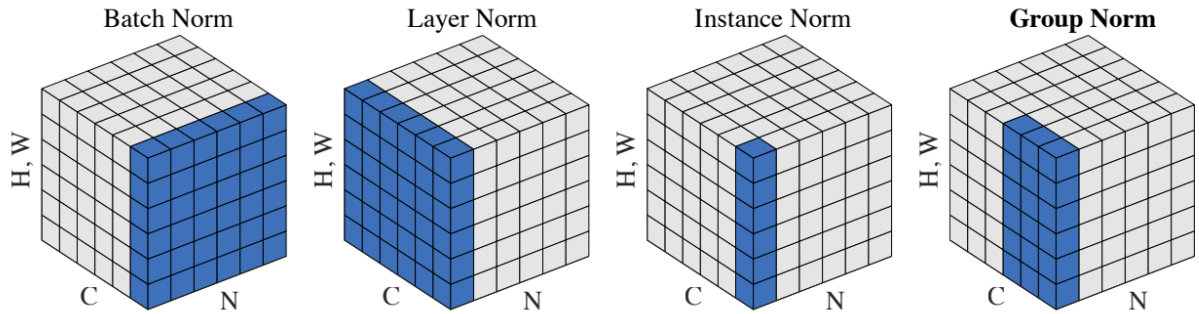
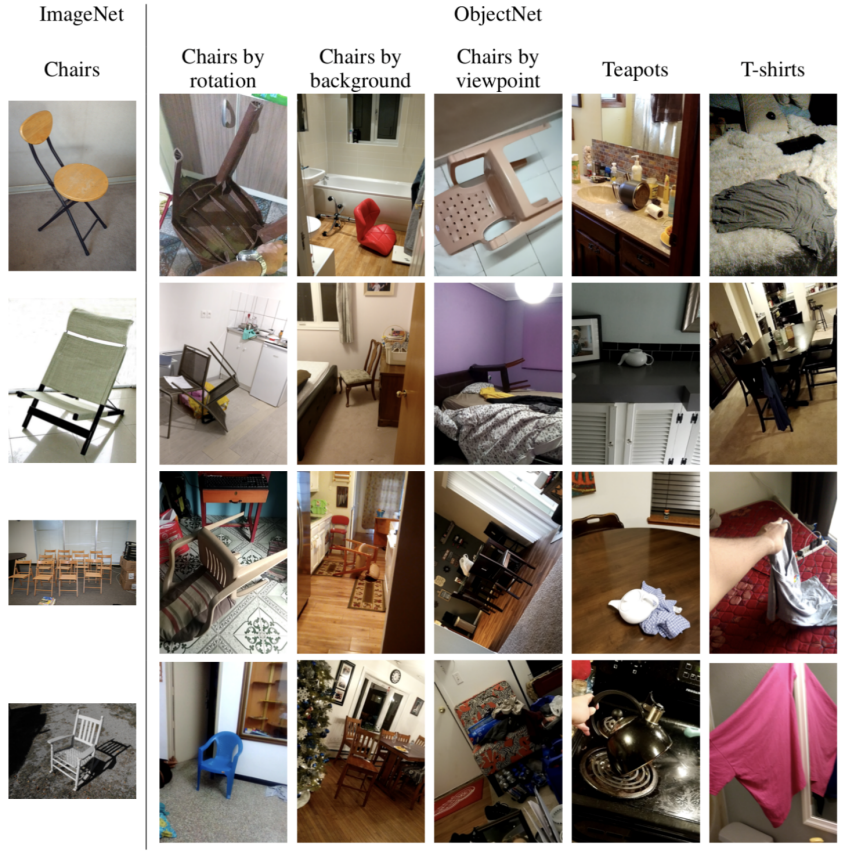


Figure 2. **Normalization methods**. Each subplot shows a feature map tensor, with N as the batch axis, C as the channel axis, and (H, W) as the spatial axes. The pixels in blue are normalized by the same mean and variance, computed by aggregating the values of these pixels.

Figure from Wu and He 2018

Is object detection solved?

- Barbu et al. 2019
  ObjectNet: A Large-Scale Bias-Controlled Dataset for Pushing the Limits of Object Recognition Models‘
- Performance on ObjectNet benchmark
  - 40 to 45% drop in performance



Afterward

- Adapted from Jeff Hawkins, Founder of Palm Computing.

  The key to object recognition is representation.

- Convolutional neural networks are particularly well-suited for computer vision tasks
- Convolutional layers "mimic" processing in visual cortex
- Exploits spatial relationship between neighbouring pixels
- Learns powerful representations that reduce the semantic gap

Practical matters: where to go from here?

- Deep learning is as much about engineering as it is about science
- Learn at least one of the many available deep learning frameworks really well
  - Become an efficient coder
- Don't be afraid to use high-level deep learning tools to quickly prototype baselines (e.g., huggingface)
  - Deep learning projects share common features
    - Data loaders
    - Measuring performance, say accuracy, precision, etc.

Copyright and License