Lab CudaVision
Learning Vision Systems on Graphics Cards (MA-INF 4308)

# Convolutional Neural Networks

18.05.2021

PROF. SVEN BEHNKE, ANGEL VILLAR-CORRALES

Contact: villar@ais.uni-bonn.de

# Refresher: MLP

# Multi-Layer Perceptron (MLP)

- Cascade of fully connected layers followed by non-linear activations
- MLP projects the data onto a space where it becomes linearly separable

$$\boldsymbol{x}^{(l+1)} = \sigma(\boldsymbol{W}^T \cdot \boldsymbol{x}^{(l)}) \quad \forall\ l \in \{1, 2, ..., L-1\}$$

- Output layer is interpreted as a vector of probabilities with the likelihood of the input belonging to each of the possible classes

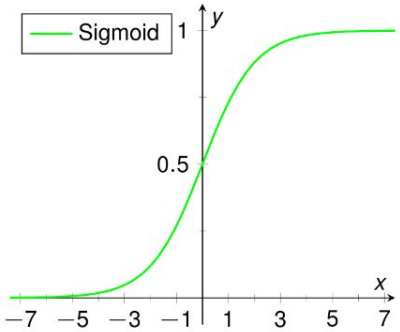
$$\boldsymbol{y} = \frac{\exp(\boldsymbol{x}^{(L)})}{\sum_{i=1}^{D} \exp(\boldsymbol{x}_i^{(L)})}$$

# Activation Functions

- Cascading linear functions □ linear function
- We need non-linear activations to learn non-linear mappings

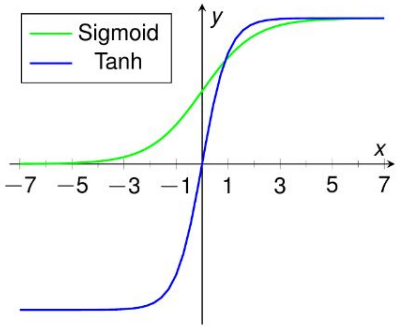
**Sigmoid**

$$f(x) = \frac{1}{1 + exp(-x)} \qquad f'(x) = f(x)(1 - f(x))$$

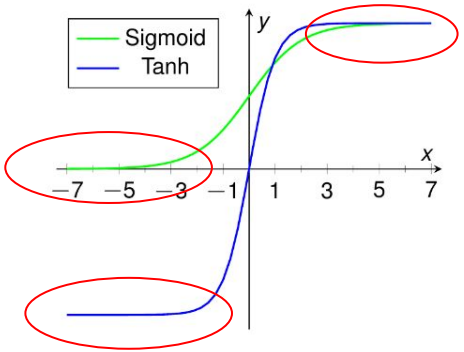



**TanH**

$$f(x) = tanh(x) \qquad f'(x) = 1 - f(x)^2$$




Watch out for vanishing gradients!



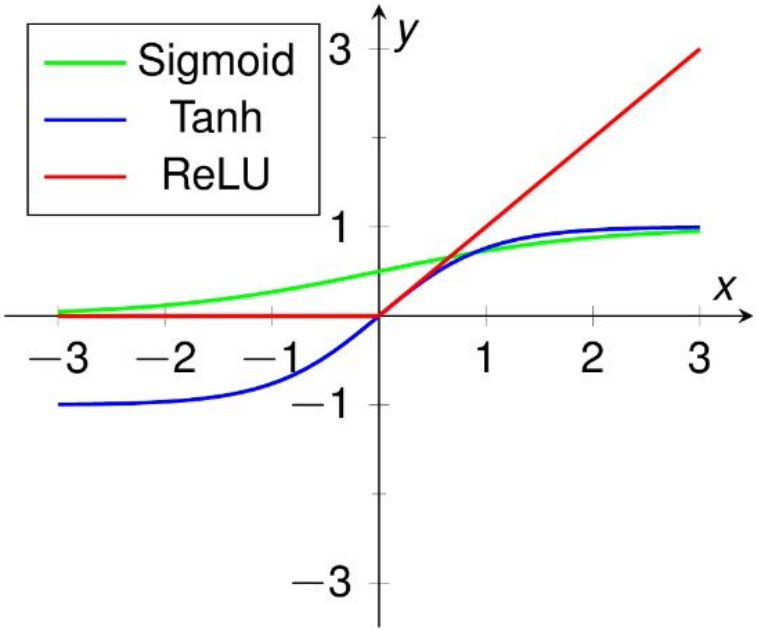
# Activation Functions

- Widely used ReLU family

Rectified Linear Unit (ReLU)

$$f(x) = \max(0, x)$$

$$f'(x) = \begin{cases} 1 & \text{if } x > 0 \\ 0 & \text{else} \end{cases}$$



# Activation Functions

- Widely used ReLU family

Rectified Linear Unit (ReLU)

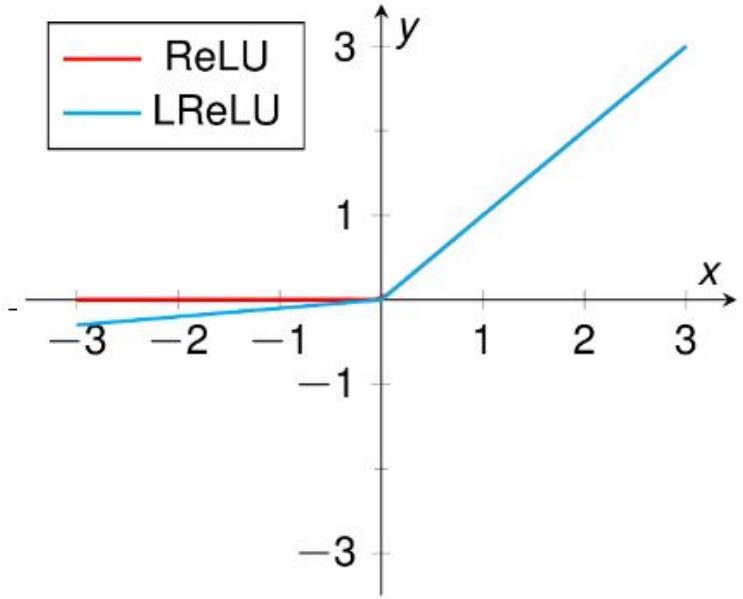
$$f(x) = \max(0, x)$$

$$f'(x) = \begin{cases} 1 & \text{if } x > 0 \\ 0 & \text{else} \end{cases}$$

Leaky ReLU / Parametric ReLU

$$f(x) = \begin{cases} x & \text{if } x > 0 \\ \alpha x & \text{else} \end{cases}$$

$$f'(x) = \begin{cases} 1 & \text{if } x > 0 \\ \alpha & \text{else} \end{cases}$$



# Activation Functions

- Widely used ReLU family

Rectified Linear Unit (ReLU)

$$f(x) = \max(0, x)$$

$$f'(x) = \begin{cases} 1 & \text{if } x > 0 \\ 0 & \text{else} \end{cases}$$

Leaky ReLU / Parametric ReLU

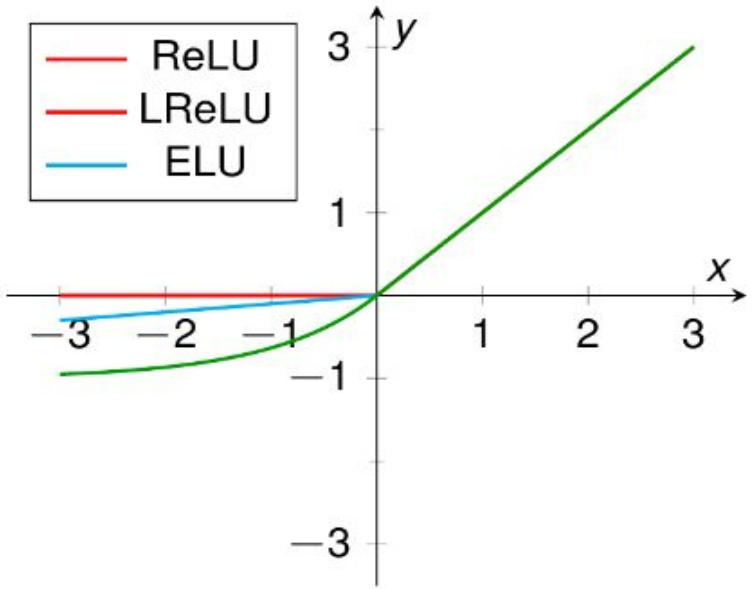
$$f(x) = \begin{cases} x & \text{if } x > 0 \\ \alpha x & \text{else} \end{cases}$$

$$f'(x) = \begin{cases} 1 & \text{if } x > 0 \\ \alpha & \text{else} \end{cases}$$

Exponential Linear Unit (ELU)

$$f(x) = \begin{cases} x & \text{if } x > 0 \\ \alpha(\exp(x) - 1) & \text{else} \end{cases}$$

$$f'(x) = \begin{cases} 1 & \text{if } x > 0 \\ \alpha \exp(x) & \text{else} \end{cases}$$



# Activation Functions

- Widely used ReLU family

Rectified Linear Unit (ReLU)

$$f(x) = \max(0, x)$$

$$f'(x) = \begin{cases} 1 & \text{if } x > 0 \\ 0 & \text{else} \end{cases}$$

Leaky ReLU / Parametric ReLU

$$f(x) = \begin{cases} x & \text{if } x > 0 \\ \alpha x & \text{else} \end{cases}$$

$$f'(x) = \begin{cases} 1 & \text{if } x > 0 \\ \alpha & \text{else} \end{cases}$$

Exponential Linear Unit (ELU)

$$f(x) = \begin{cases} x & \text{if } x > 0 \\ \alpha(\exp(x) - 1) & \text{else} \end{cases}$$

$$f'(x) = \begin{cases} 1 & \text{if } x > 0 \\ \alpha \exp(x) & \text{else} \end{cases}$$

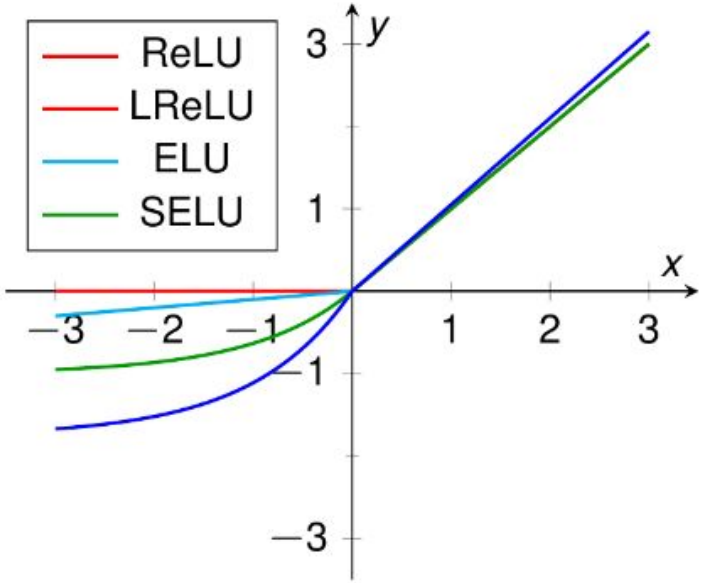
Scaled Exponential Linear Unit

$$f(x) = \lambda \begin{cases} x & \text{if } x > 0 \\ \alpha(\exp(x) - 1) & \text{else} \end{cases}$$

$$f'(x) = \lambda \begin{cases} 1 & \text{if } x > 0 \\ \alpha f(x) & \text{else} \end{cases}$$

$$\lambda_{01} = 1.0507$$

$$\alpha_{01} = 1.6733$$



# Problem with MLP

- MLPs require (in general) a huge number of parameters
- Assume the following:
  - Input image of size 3 x 224 x 224  (ImageNet)
  - Hidden layer with 10 neurons

( (3 x 224 x 224) + 1) x 10  > 1.5 million parameters

- No suitable inductive bias
  - No account for locality
  - Not hierarchical

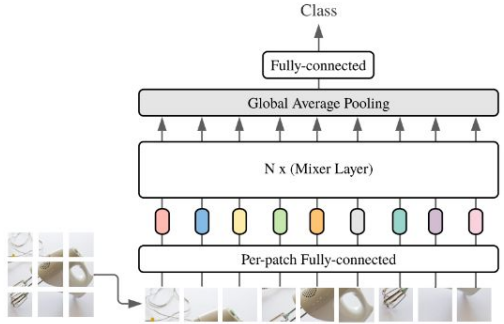
# Make MLPs Great Again

**MLP-Mixer: An all-MLP Architecture for Vision**

Ilya Tolstikhin*, Neil Houlsby*, Alexander Kolesnikov*, Lucas Beyer*,
Xiaohua Zhai, Thomas Unterthiner, Jessica Yung, Daniel Keysers,
Jakob Uszkoreit, Mario Lucic, Alexey Dosovitskiy
*equal contribution
Google Research, Brain Team
{tolstikhin, neilhoulsby, akolesnikov, lbeyer,
xzhai, unterthiner, jessicayung, keysers,
usz, lucic, adosovitskiy}@google.com

**Abstract**

Convolutional Neural Networks (CNNs) are the go-to model for computer vision. Recently, attention-based networks, such as the Vision Transformer, have also become popular. In this paper we show that while convolutions and attention are both sufficient for good performance, neither of them are necessary. We present *MLP-Mixer*, an architecture based exclusively on multi-layer perceptrons (MLPs). MLP-Mixer contains two types of layers: one with MLPs applied independently to image patches (i.e. "mixing" the per-location features), and one with MLPs applied across patches (i.e. "mixing" spatial information). When trained on large datasets, or with modern regularization schemes, MLP-Mixer attains competitive scores on image classification benchmarks, with pre-training and inference cost comparable to state-of-the-art models. We hope that these results spark further research beyond the realms of well established CNNs and Transformers.[1]




| | ImNet top-1 | ReaL top-1 | Avg 5 top-1 | VTAB-1k 19 tasks | Throughput img/sec/core | TPUv3 core-days |
|---|---|---|---|---|---|---|
| Pre-trained on ImageNet-21k (public) | | | | | | |
| HaloNet [49] | 85.8 | — | — | — | 120 | 0.10k |
| Mixer-L/16 | 84.15 | 87.86 | 93.91 | 74.95 | 105 | 0.41k |
| ViT-L/16 [14] | 85.30 | 88.62 | 94.39 | 72.72 | 32 | 0.18k |
| BiT-R152x4 [22] | 85.39 | — | 94.04 | 70.64 | 26 | 0.94k |

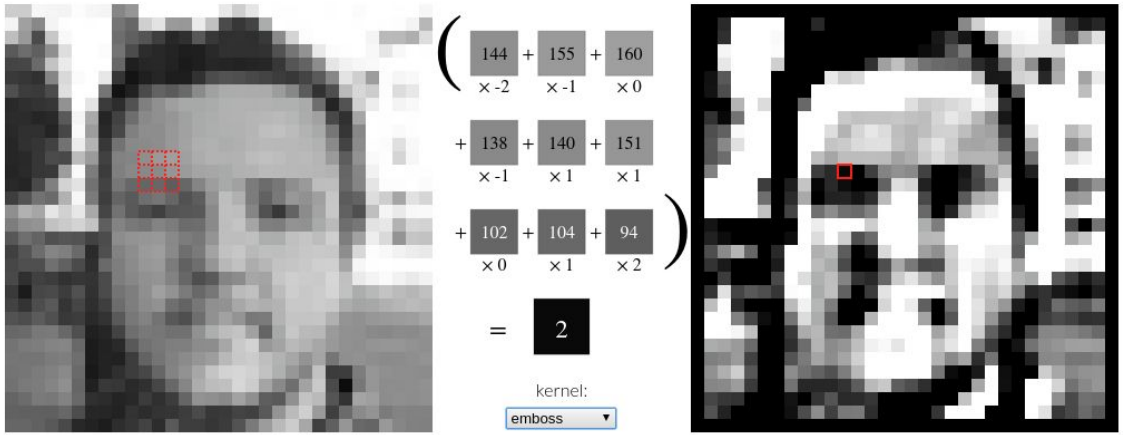
# Convolutional Neural Networks (CNNs)

# Convolutional Layers

# Convolution Operation

- Main workhorse of CNNs
- Feature extraction

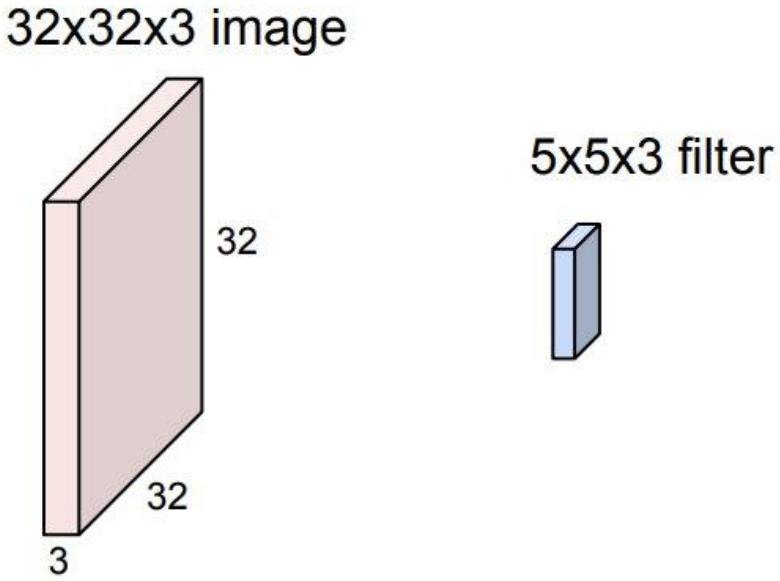
$$f_{u,v} = \sum_{n=1}^{2} \sum_{m=1}^{2} k_{n,m} \cdot i_{(n+u-1),(m+v-1)}, \quad \forall \quad u, v = 1, 2.$$




input image

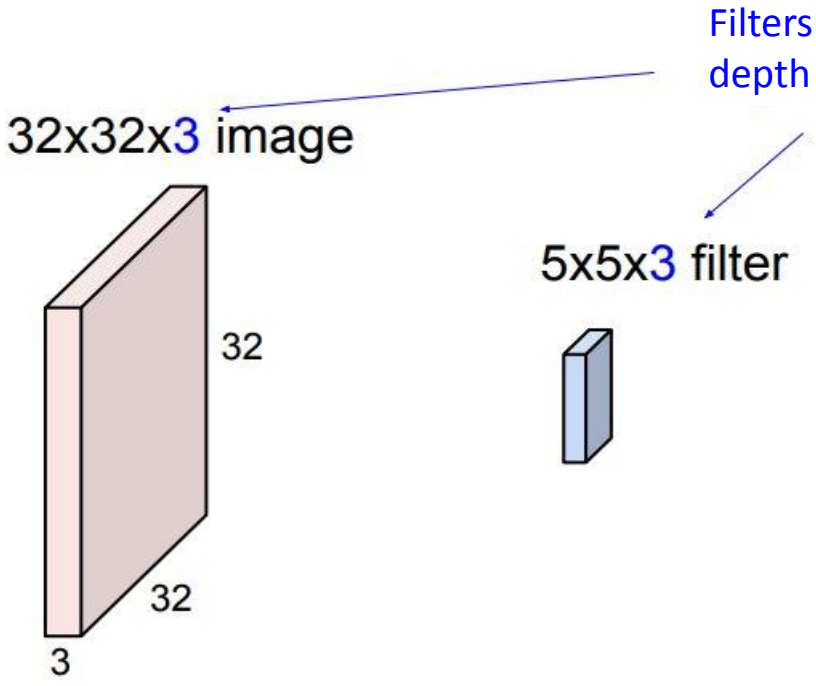
output image

# Convolutional Layer



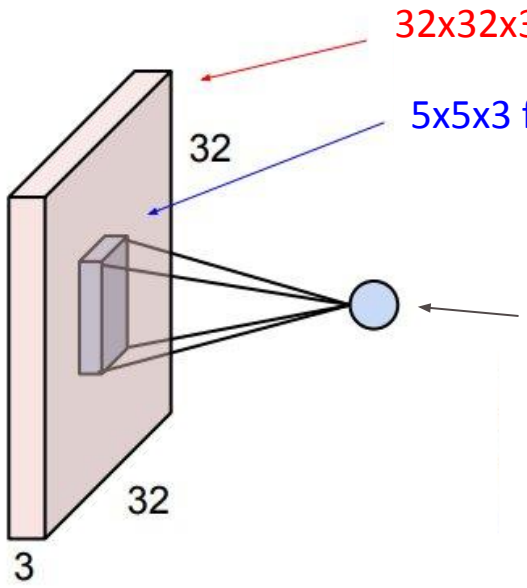

**Convolve** the filter with the image, i.e., “sliding the filter over the image spatially, computing dot products”

# Convolutional Layer




**Convolve** the filter with the image, i.e., “sliding the filter over the image spatially, computing dot products”

# Convolutional Layer




**Output (Y)**

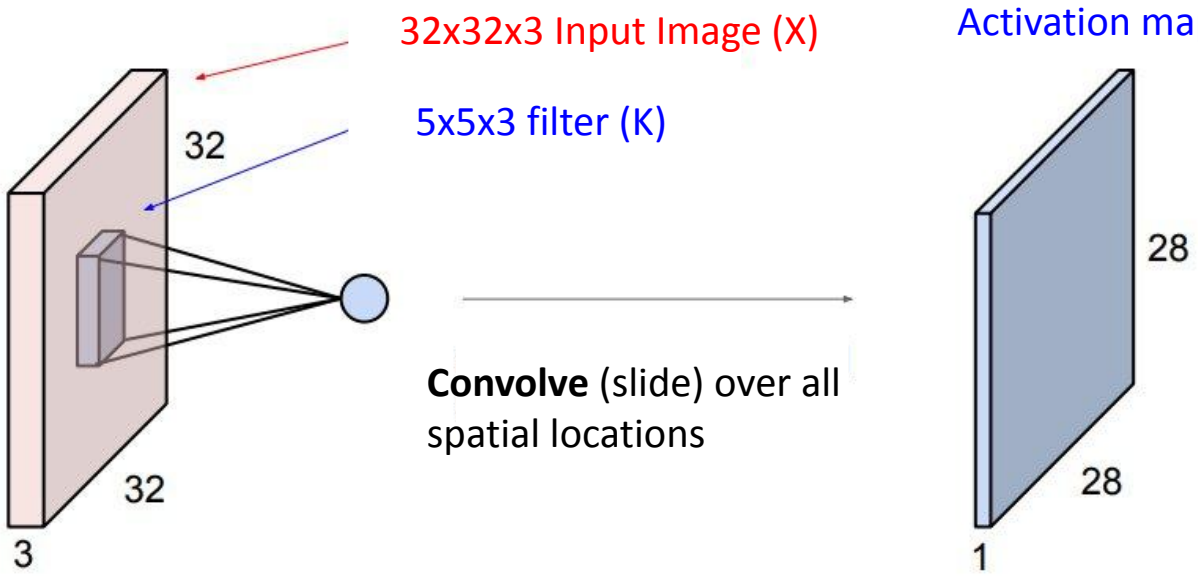
A convolution is the dot product between the filter and a (5x5x3) part of the image.

75 dimensional product + bias

$$y(u,v) = \sum_{i,j,k} K(i,j,k) \cdot X(u+i, j+v, k) + b$$

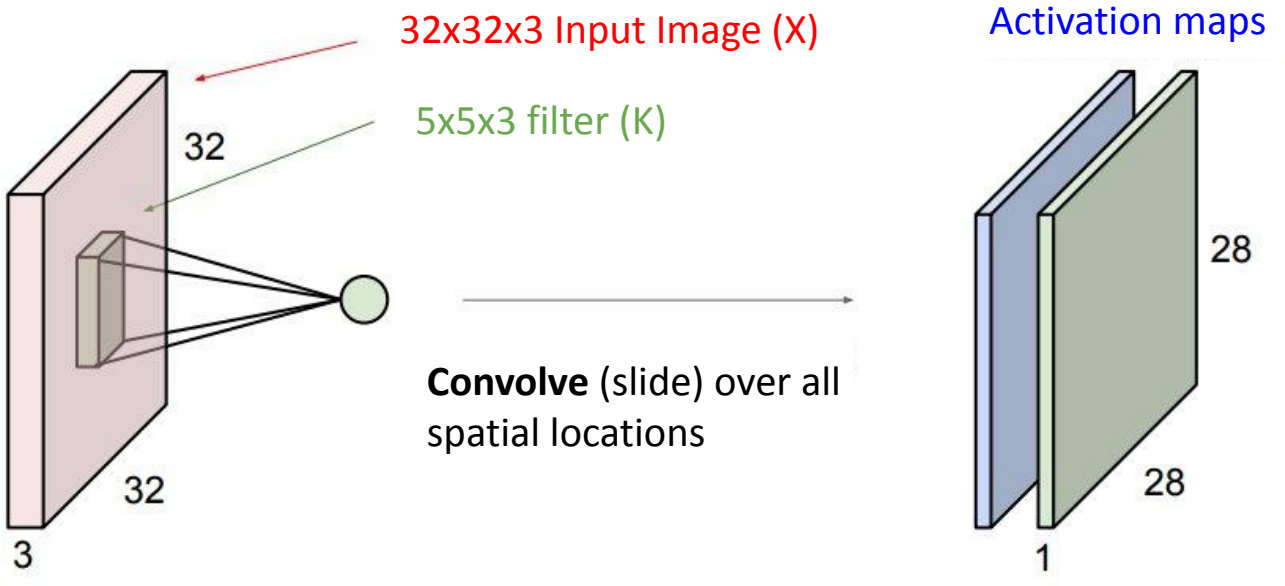
$$y(u,v) = \sum \mathbf{K}^T \odot \mathbf{X}_{u,v} + b$$

# Convolutional Layer



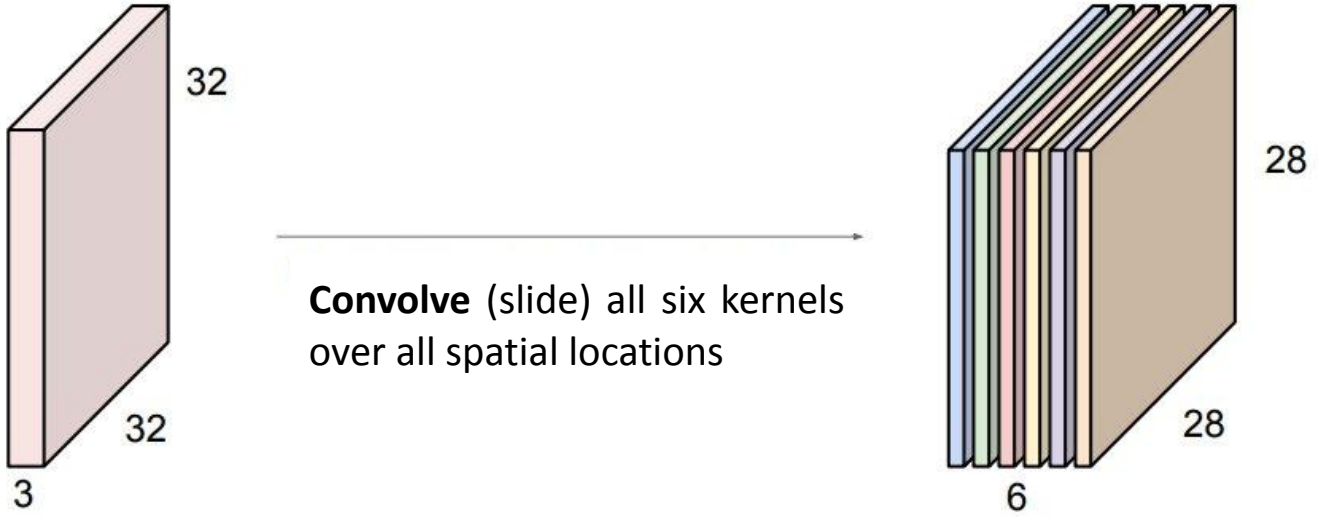
# Convolutional Layer

- Consider a second, green filter



# Convolutional Layer

- Assume we have six (5x5) filters. Then we get six separate activation maps
- We stack these maps to for a ‘new image’ of shape 28x28x6



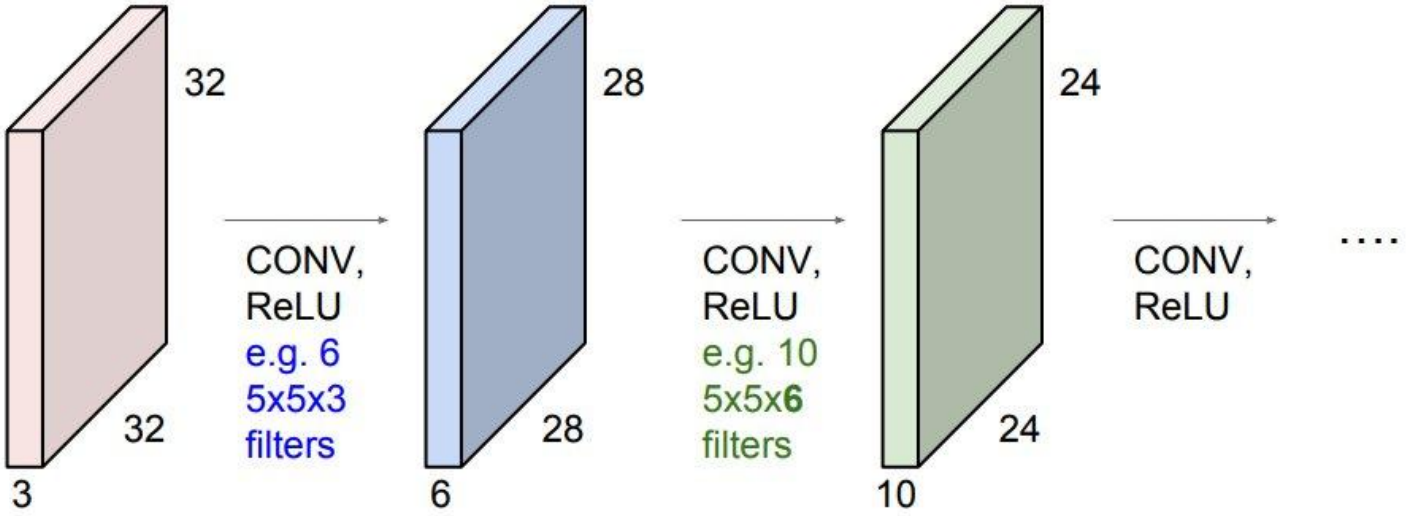
# Convolutional Layer

# Convolutional Layer

- Exploits spatial structure by only connecting pixels in a neighborhood
- Efficient implementation:
  - Multiplications and sums
  - Efficient implementation in frequency domain
- Features that are important at one location are likely important anywhere in the image
  - Use same weights all over (**weight sharing**)
- Convolution with trainable filters

# Convolutional Neural Net

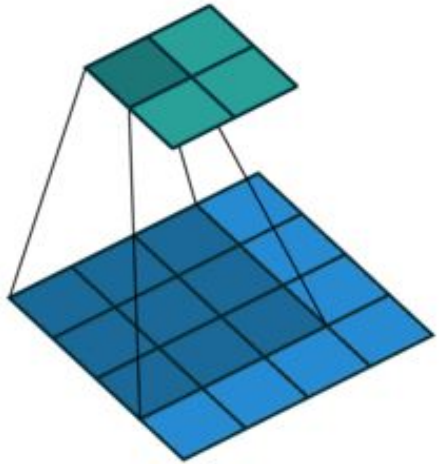
- A CNN is a sequence of Conv. Layers, interspersed with nonlinear activations and other operators (e.g., downsampling, normalization or dropout)



# Closer Look at Spatial Dimensions

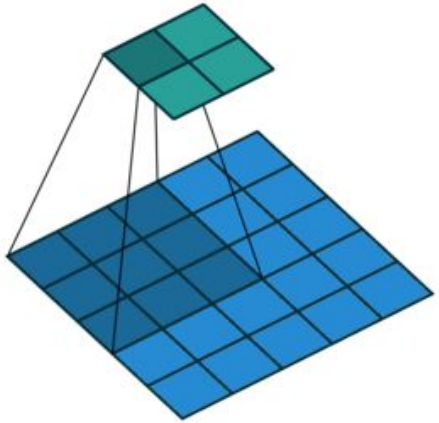
# Looking at Spatial Dimensions

- Convolutions decrease the spatial size of an image
- A (4x4) image convolved by a (3x3) filter results in an (2x2) output
- In addition to kernel size:
  - Stride
  - Padding
  - Dilation



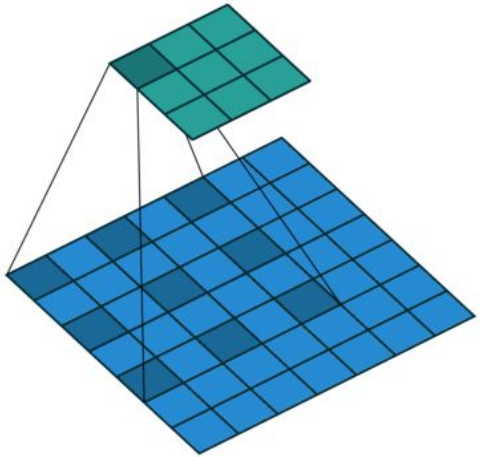
# Stride

- Instead of convolving at all positions, we skip some pixels
- Reduces the size of the output
- Strided convolution = Convolution + Downsampling



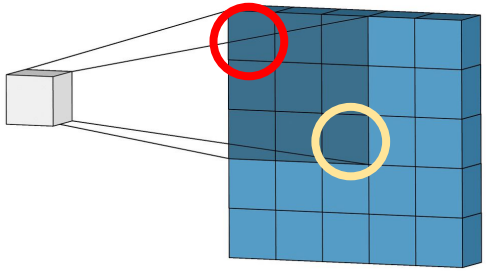
# Dilation

- Dilate convolutional kernel: skip certain pixels
- Wide receptive field with fewer number of parameters



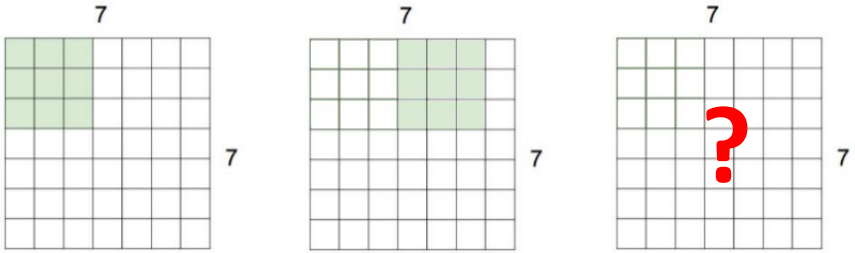
# Padding

- Convolution operation presents some difficulties



- Some pixels are seen more than others
  - Pixel in red is used only once
  - Pixel in yellow is used 9 times

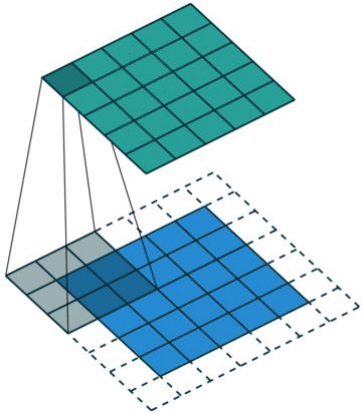
- Some sizes are not realizable
  - 7x7 input
  - 3x3 kernel with stride 3



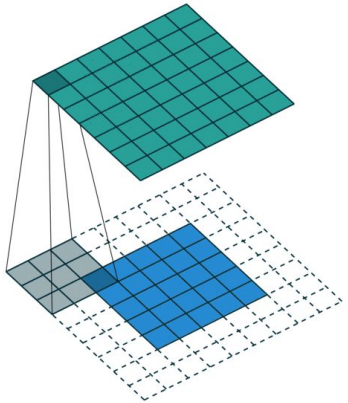
# Padding

- Pooling takes care of the aforementioned problems
- Adds new locations around the image
  - Usually filled with zeros

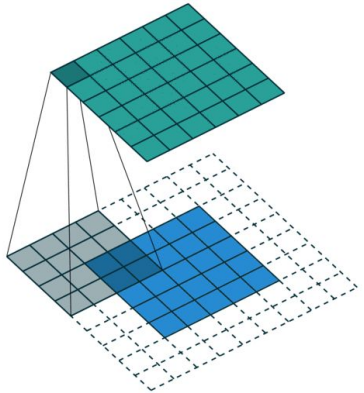
**Same Padding**



**Full Padding**



**Padding (2,2)**



# Output Size

- Conv. Layers have five hyper-parameters
  - Number of kernels
  - Kernel size
  - Amount of padding
  - Stride
  - Dilation

```
[ ]: nn.Conv2d()
Init signature:
nn.Conv2d(
    in_channels: int,
    out_channels: int,
    kernel_size: Union[int, Tuple[int, int]],
    stride: Union[int, Tuple[int, int]] = 1,
    padding: Union[int, Tuple[int, int]] = 0,
    dilation: Union[int, Tuple[int, int]] = 1,
    groups: int = 1,
    bias: bool = True,
    padding_mode: str = 'zeros',
)
```

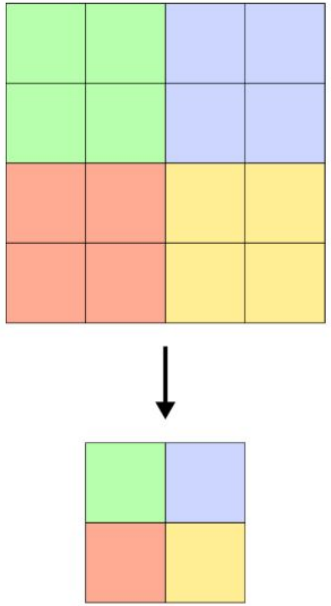
$$H_{out} = \left\lfloor \frac{H_{in} + 2 \times padding[0] - dilation[0] \times (kernel\_size[0] - 1) - 1}{stride[0]} + 1 \right\rfloor$$

$$W_{out} = \left\lfloor \frac{W_{in} + 2 \times padding[1] - dilation[1] \times (kernel\_size[1] - 1) - 1}{stride[1]} + 1 \right\rfloor$$

# Pooling

# Pooling Layer

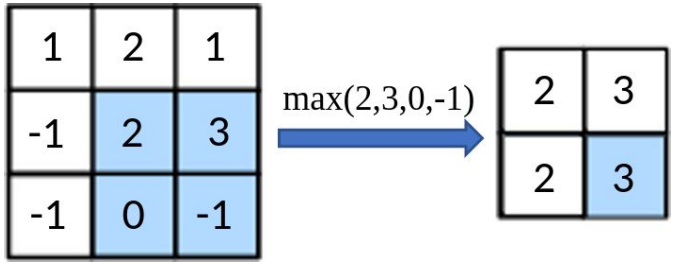
- The pooling operator reduces the spatial dimensionality of the feature maps
- Fuses information across spatial location
- Decreases the total number of parameters
- Reduces computational cost and overfitting



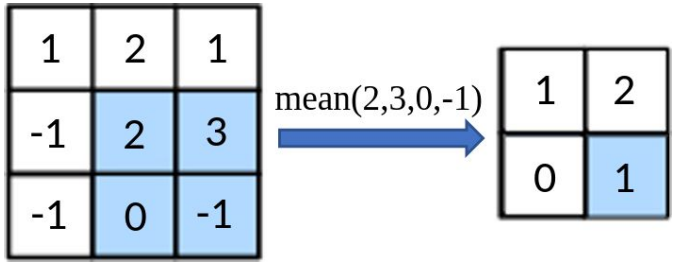
# Pooling

- Several types of pooling. Also easy to come up with your own
- Operate on each channel independently

**Max Pooling**

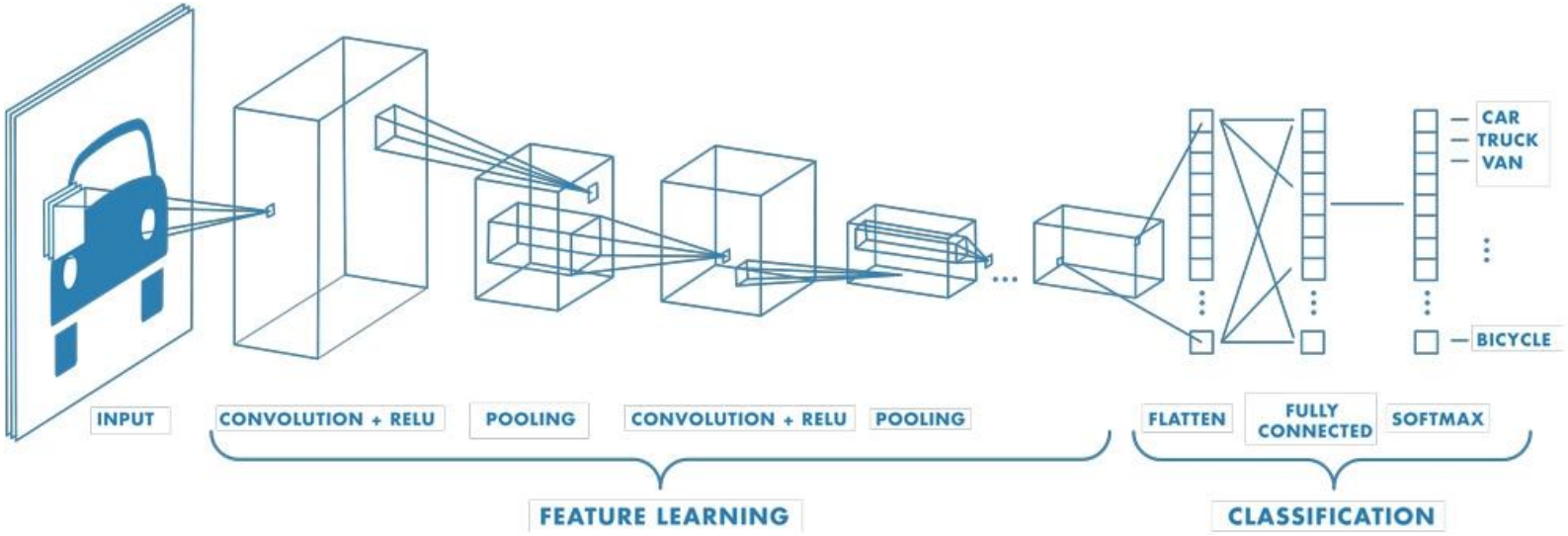



**Mean Pooling**



# Conv. Net

# Classical CNN Architecture



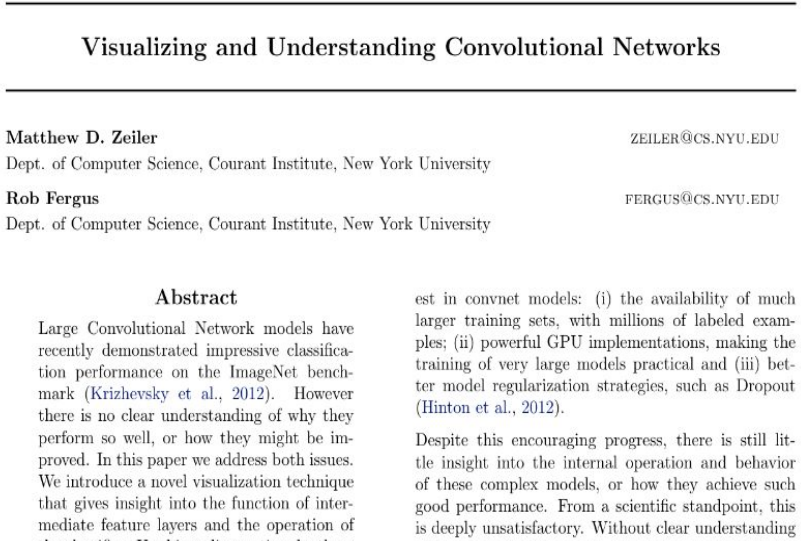
# Learning Hierarchical Features

- CNNs cascade learned convolutional kernels with pooling operators and nonlinear activations
- Kernels in second layer are applied to activation maps of first layer
- Cnn's learn features in a hierarchical manner

# Learning Hierarchical Features

*"The projections from each layer show the hierarchical nature of the features in the network. Layer 2 responds to corners and other edge/color conjunctions. Layer 3 has more complex invariances, capturing similar textures. Layer 4 shows significant variation, but is more class-specific: dog faces; bird's legs . Layer 5 shows entire objects with significant pose variation, e.g. keyboards and dogs."*

Zeiler and Fergus, *Visualizing and Understanding Convolutional Neural Networks*. 2013



Visualizing and Understanding Convolutional Networks

Matthew D. Zeiler ZEILER@CS.NYU.EDU
Dept. of Computer Science, Courant Institute, New York University

Rob Fergus FERGUS@CS.NYU.EDU
Dept. of Computer Science, Courant Institute, New York University

Abstract

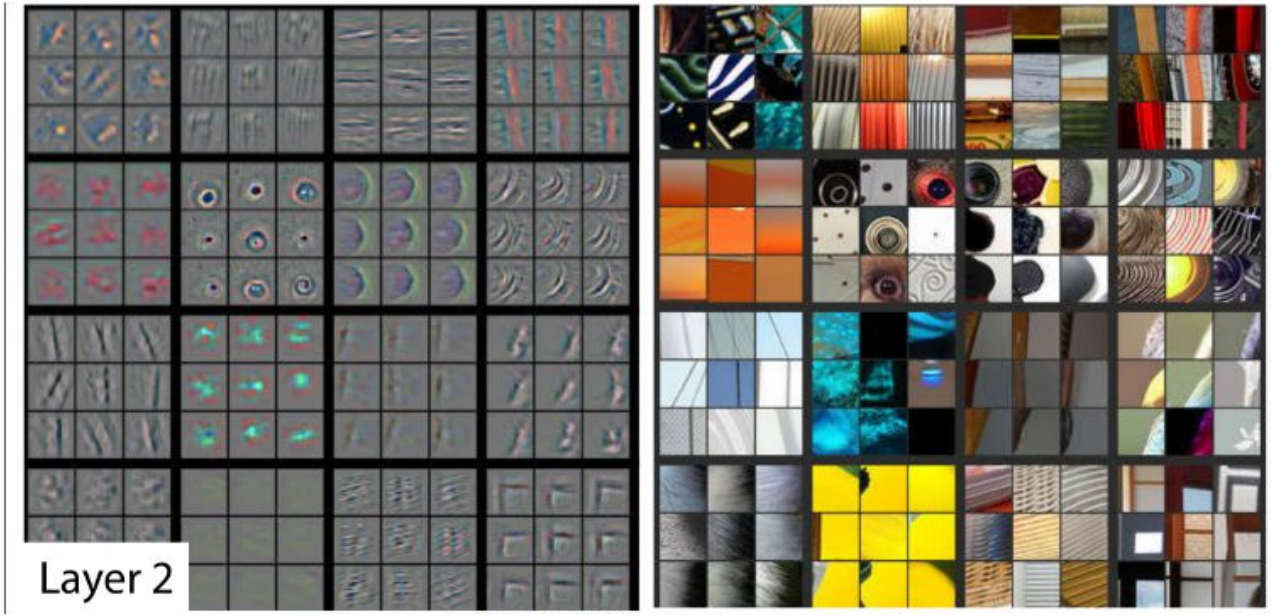
Large Convolutional Network models have recently demonstrated impressive classification performance on the ImageNet benchmark (Krizhevsky et al., 2012). However there is no clear understanding of why they perform so well, or how they might be improved. In this paper we address both issues. We introduce a novel visualization technique that gives insight into the function of intermediate feature layers and the operation of

est in convnet models: (i) the availability of much larger training sets, with millions of labeled examples; (ii) powerful GPU implementations, making the training of very large models practical and (iii) better model regularization strategies, such as Dropout (Hinton et al., 2012).

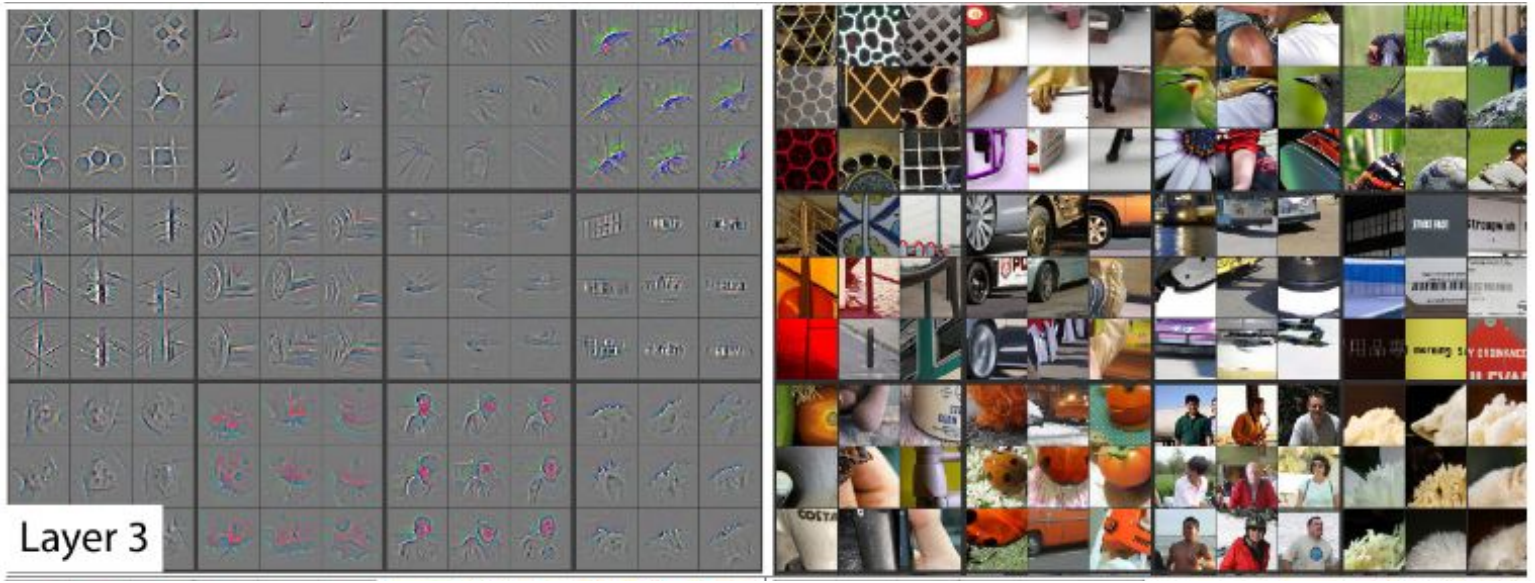
Despite this encouraging progress, there is still little insight into the internal operation and behavior of these complex models, or how they achieve such good performance. From a scientific standpoint, this is deeply unsatisfactory. Without clear understanding

# Learning Hierarchical Features

**Visualizing and Understanding Convolutional Networks**



# Learning Hierarchical Features



# Learning Hierarchical Features