Visual Learning
(Tutorial C2)
Convolutional Networks

Instructor - Simon Lucey

RVSS - 2024

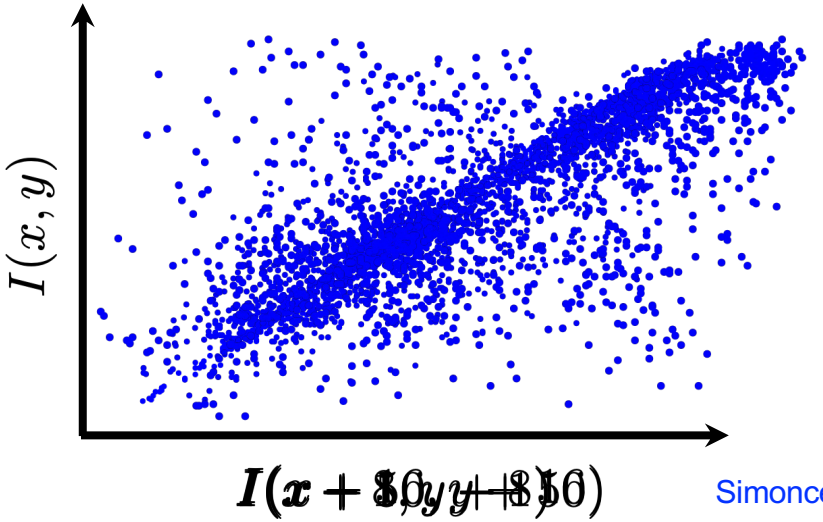
THE UNIVERSITY of ADELAIDE

AUSTRALIAN INSTITUTE FOR MACHINE LEARNING

Today

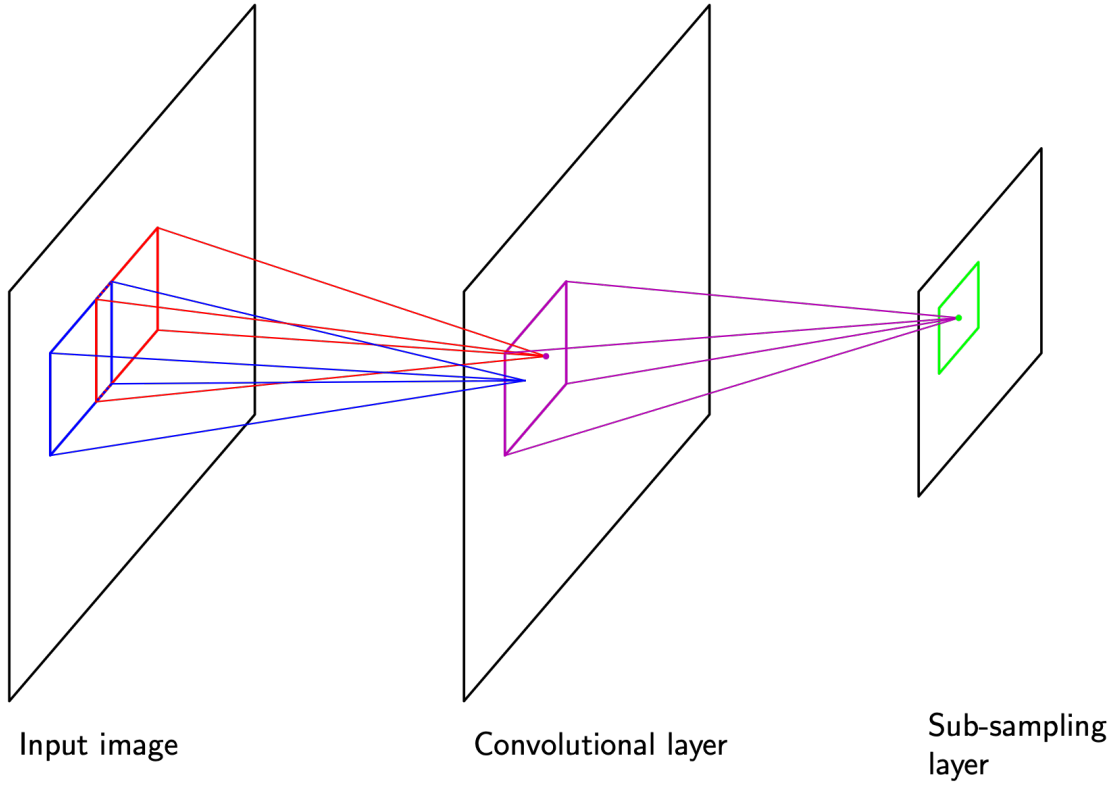
- **ConvNets and AlexNet**
- Depthwise Convolution
- Skip/Residual Connections

I



Simoncelli & Olshausen 2001

Convolutional Neural Network

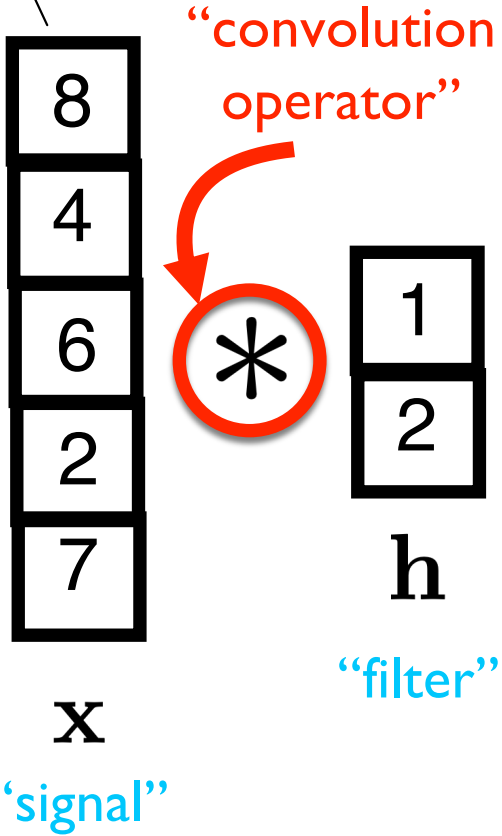


LeCun 1980

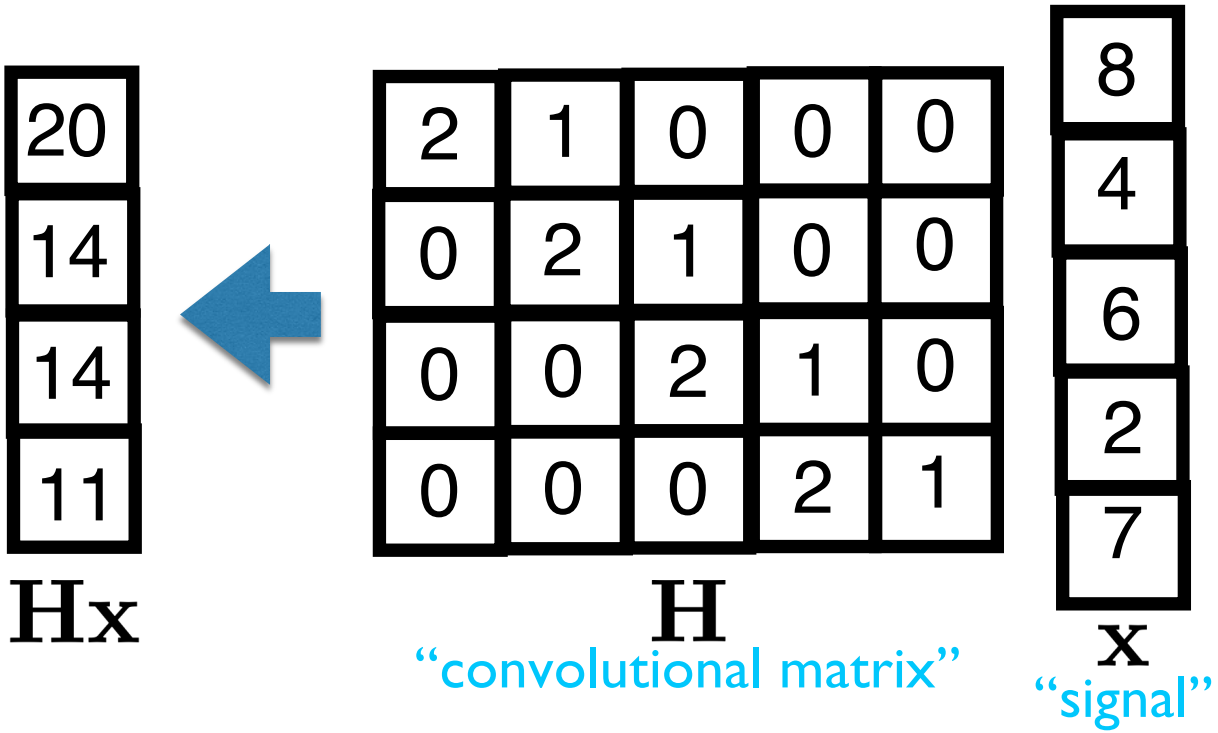
Reminder: Convolution

```
>>> from scipy.signal import \
    convolve as conv
>>> conv(x,h,'valid')
array([20, 14, 14, 11])
```

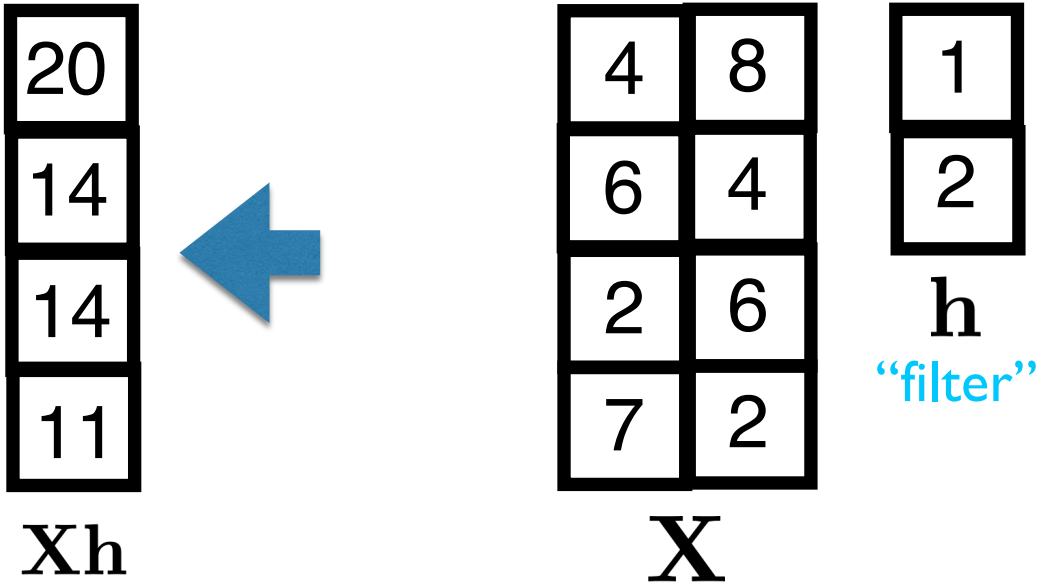
$$\frac{\partial[\mathbf{x} * \mathbf{h}]}{\mathbf{h}^T} = ???$$



Reminder: Convolution



Reminder: Convolution



$$\frac{\partial[\mathbf{x} * \mathbf{h}]}{\mathbf{h}^T} = \frac{\partial[\mathbf{X}\mathbf{h}]}{\mathbf{h}^T} = \mathbf{X}^T$$

Efficiency of Convolution

Input size: 320 by 280
Filter size: 2 by 1
Output size: 319 by 280

	2	319*280*320*280 > 8e9	2*319*280 = 178,640
	319*280*3 = 267,960	> 16e9	Same as convolution (267,960)

Vectorizing 2D-Convolution

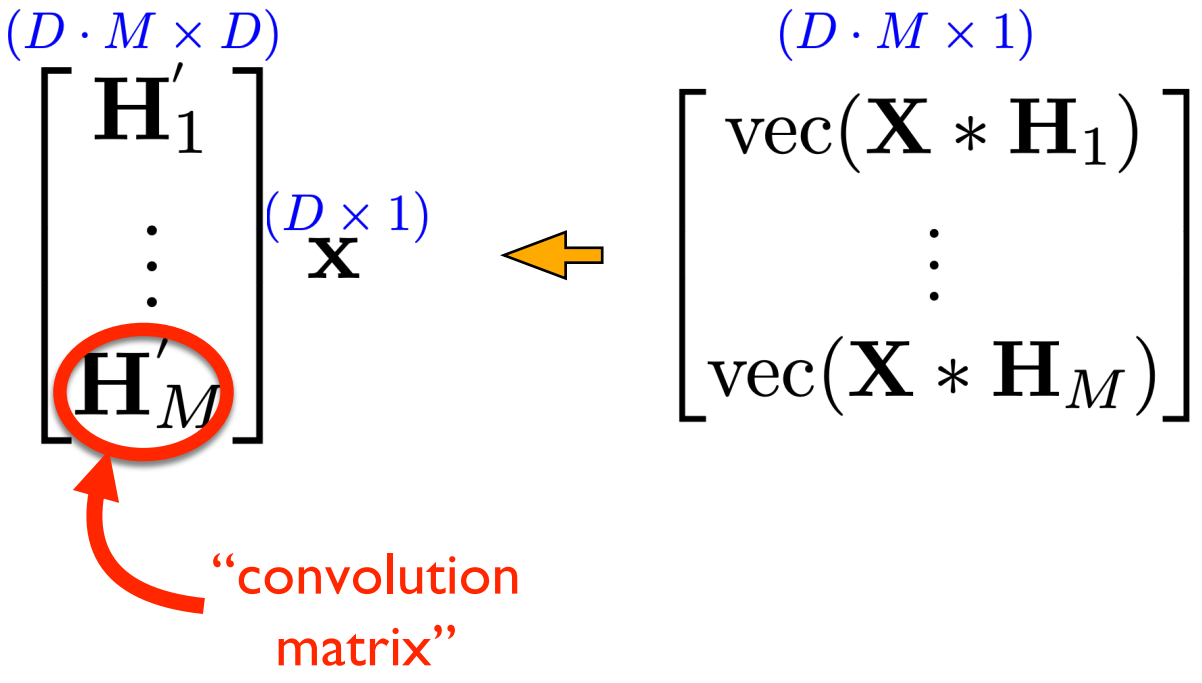
$$\text{vec}(\mathbf{Y}) = \mathbf{X}\,\text{vec}(\mathbf{K} * \mathbf{H})$$

```
def forward(self, x):
    # Flattens the image like structure into vect
    x = torch.flatten(x, start_dim=1)
```

Vectorizing 2D-Convolution

$$\mathrm{vec}(\mathbf{Y}) = \mathbf{H}' \times \mathrm{vec}(\mathbf{X})$$

Multiple Filters



Removing Redundancy - Striding

```
Stride: [4,4]
```

Every 4th element

Every 4th element

207	245	77	21	247	211	240	1
219	41	58	179	161	154	184	98
215	145	187	71	251	249	65	100
192	2	189	247	166	63	232	213
105	94	66	190	156	61	89	145
159	154	87	184	101	105	72	71
192	111	6	94	60	70	65	226
175	120	210	226	80	183	168	184
134	56	36	240	159	178	76	135
239	244	199	9	132	104	188	185
245	210	78	199	0	92	9	246
5	121	187	122	107	47	12	119
230	171	135	36	82	54	65	37
61	140	79	19	161	96	127	187
56	223	46	6	180	186	142	244
28	20	61	2	178	187	98	220

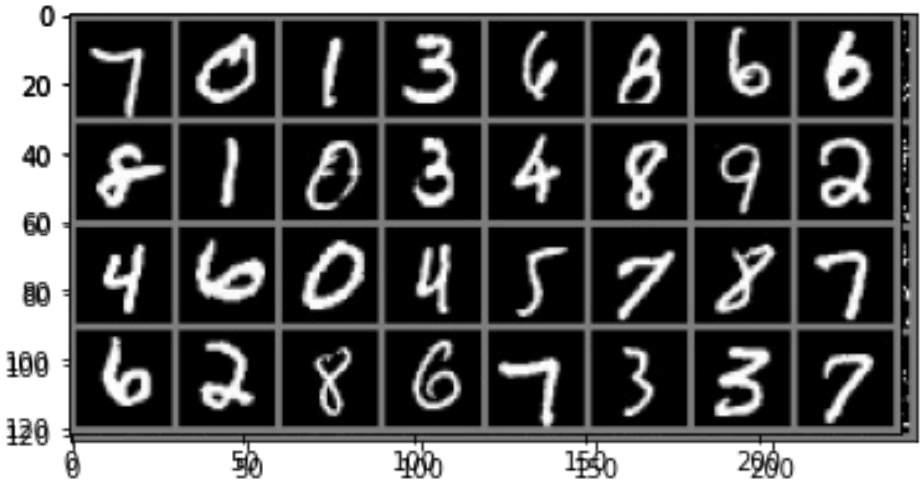
Let’s have another play!!!

https://colab.research.google.com/github/slucey-cs-cmu-edu/RVSS24/blob/main/CNN_Example.ipynb

Some things to try!!!

- What happens to performance if you use a linear function?
- What happens when you permute the pixels?

```
from numpy.random import permutation
idx_permute = torch.from_numpy(permutation(784))
transform = transforms.Compose([transforms.ToTensor(),
                                transforms.Lambda(lambda x: x.view(-1)[idx_permute].view(1, 28, 28) ),
                                transforms.Normalize((0.5,), (0.5,)),
                                ])
```



Adding Striding - CNN

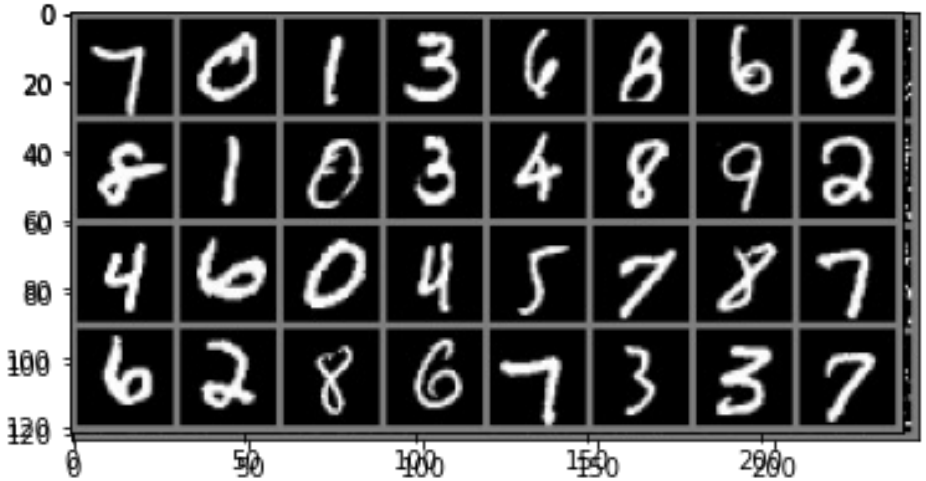
```
class ConvNet(nn.Module):
    def __init__(self):
        super(ConvNet, self).__init__()
        self.conv1 = nn.Conv2d(1, 6, 5, 2)
        self.conv2 = nn.Conv2d(6, 16, 5, 2)
        self.fc1 = nn.Linear(256, 84)
        self.fc2 = nn.Linear(84, 10)

    def forward(self, x):
        # Input goes to convolution so no need t
        x = self.conv1(x)
        x = F.relu(x)
        x = self.conv2(x)
        x = F.relu(x)
        x = torch.flatten(x, start_dim=1)
        x = self.fc1(x)
        x = F.relu(x)
        x = self.fc2(x)
```

$$\eta(\mathbf{W}^{(1)}\eta(\mathbf{W}^{(0)}\mathbf{x}+\boldsymbol{\delta}^{(0)})+\boldsymbol{\delta}^{(1)})$$

Permuting Pixels - CNN

- What happens when you permute the pixels in a CNN?

```
from numpy.random import permutation
idx_permute = torch.from_numpy(permutation(784))
transform = transforms.Compose([transforms.ToTensor(),
                                transforms.Lambda(lambda x: x.view(-1)[idx_permute].view(1, 28, 28) ),
                                transforms.Normalize((0.5,), (0.5,)),
                                ])
```



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

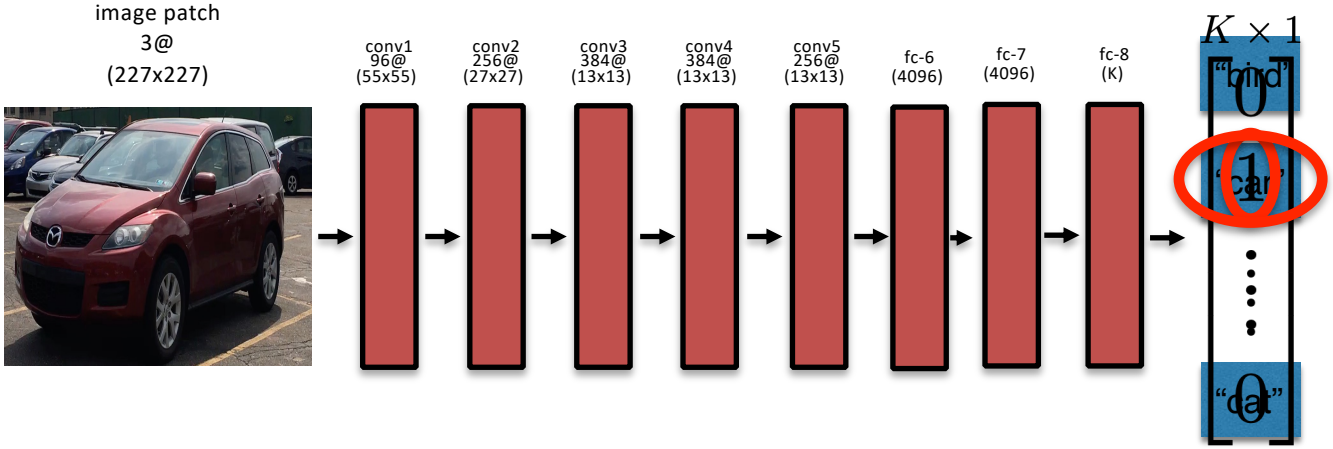
Geoffrey E. Hinton
University of Toronto
hinton@cs.utoronto.ca

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.

AlexNet

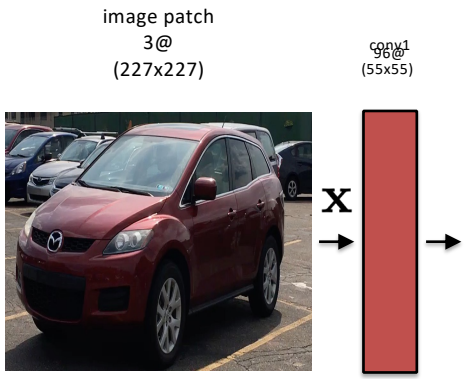
- AlexNet won the ILSVRC-2012 competition and achieved a winning top-5 test error rate of 15.3%. (Second best was 26.2 %).
- Network has 25 layers, but only 8 layers with learnable weights.
  - 5 convolutional weights.
  - 3 fully connected weights.



A. Krizhevsky, I. Sutskever, and G. E. Hinton. Imagenet classification with deep convolutional neural networks. NeurIPS, 2012.

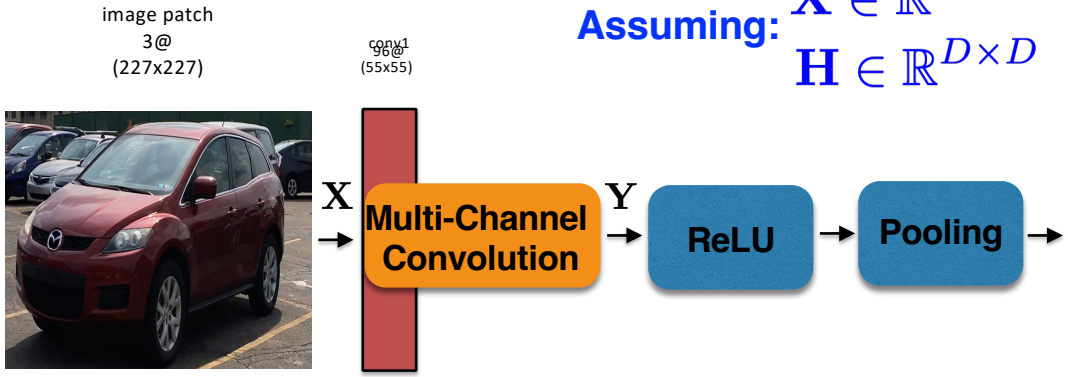
AlexNet Drawbacks

- AlexNet gave amazing performance but had some drawbacks.
- Fundamentally, it is extremely expensive to evaluate & learn.



- 55x55x96 = 290,400 hidden states
- each has 11x11x3 = 363 weights and 1 bias
- 96x364 = 34,944 parameters on the first layer of the AlexNet alone!

Reminder: Multi-Channel Convolution



Assuming: $\mathbf{X} \in \mathbb{R}^{M \times N}$, $\mathbf{H} \in \mathbb{R}^{D \times D}$

$$\mathbf{Y}^{(j)} = \sum_{k=1}^{K} \mathbf{X}^{(k)} * \mathbf{H}^{(j,k)}$$

Naive computational cost - $\mathcal{O}(M \cdot N \cdot D^2 \cdot K \cdot J)$

A. Krizhevsky, I. Sutskever, and G. E. Hinton. Imagenet classification with deep convolutional neural networks. In NIPS, 2012.

Today

- ConvNets and AlexNet
- **Depthwise Convolution**
- Skip/Residual Connections

Depthwise Convolution

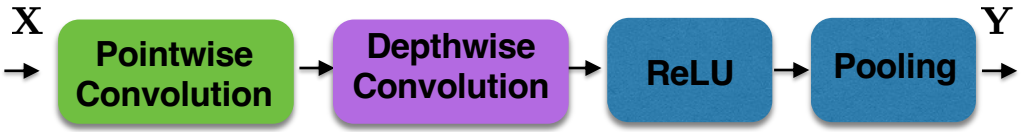
image patch
3@
(227x227)

$$\mathbf{Y}^{(j)} = \sum_{k=1}^{K} \underbrace{\mathbf{W}[j,k]}_{\text{“1x1 Pointwise Convolution”}} \cdot [\underbrace{\mathbf{X}^{(k)} * \mathbf{H}^{(k)}}_{\text{“Depthwise Convolution”}}]$$

$$\mathcal{O}(M \cdot N \cdot D^2 \cdot J \cdot K) \rightarrow \mathcal{O}(M \cdot N \cdot (D^2 + J) \cdot K)$$

Depthwise Convolution

image patch
3@
(227x227)



$$\mathbf{Y}^{(j)} = \underbrace{\mathbf{H}^{(j)}}_{\text{“1x1 Pointwise Convolution”}} * [\sum_{k=1}^{K} \underbrace{\mathbf{W}[j,k] \cdot \mathbf{X}^{(k)}}_{\text{“Depthwise Convolution”}}]$$

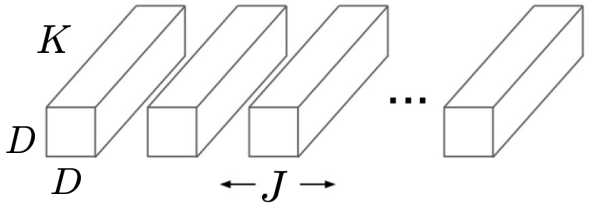
$$\mathcal{O}(M \cdot N \cdot D^2 \cdot J \cdot K) \rightarrow \mathcal{O}(M \cdot N \cdot (D^2 + J) \cdot K)$$

MobileNet

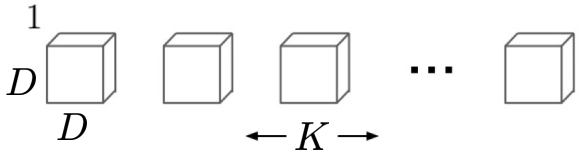
- MobileNet: (a) is divided into (b) and (c).
- Computation reduction

$$\frac{b+c}{a} = \frac{D^2 + J}{D^2 \cdot J}$$

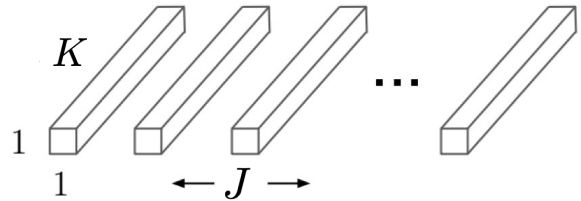
- D = 3: 8~9 times less computation.



(a) Standard convolution filters



(b) Depthwise convolution filters



(c) 1x1 pointwise convolution

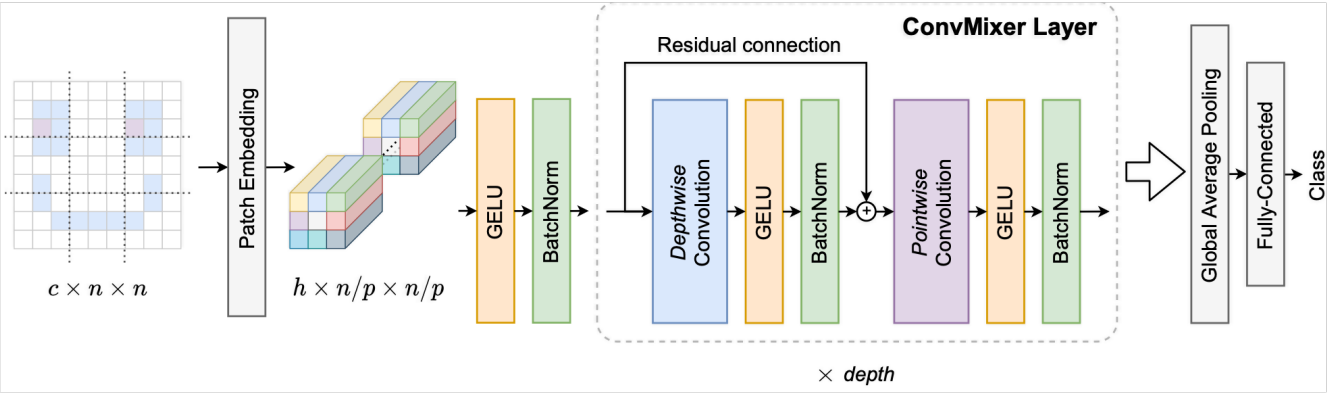
Howard et al. Mobilenets: Efficient convolutional neural networks for mobile vision applications. arXiv preprint arXiv:1704.04861, 2017.

MobileNet

- MobileNet with similar accuracy but less computation and fewer parameters than VGG16 and GoogleNet
- $\alpha = 0.5$, input 160×160: better than AlexNet while being 45 times smaller and 9.4 times less compute than AlexNet

Model	ImageNet Accuracy	Million Mult-Adds	Million Parameters
1.0 MobileNet-224	70.6%	569	4.2
GoogleNet	69.8%	1550	6.8
VGG 16	71.5%	15300	138
0.50 MobileNet-160	60.2%	76	1.32
Squeezenet	57.5%	1700	1.25
AlexNet	57.2%	720	60

ConvMixer - Patches are all you need




```
def ConvMixer(h, depth, kernel_size=9, patch_size=7, n_classes=1000):
    Seq, ActBn = nn.Sequential, lambda x: Seq(x, nn.GELU(), nn.BatchNorm2d(h))
    Residual = type('Residual', (Seq,), {'forward': lambda self, x: self[0](x) + x})
    return Seq(ActBn(nn.Conv2d(3, h, patch_size, stride=patch_size)),
        *[Seq(Residual(ActBn(nn.Conv2d(h, h, kernel_size, groups=h, padding="same"))),
          ActBn(nn.Conv2d(h, h, 1))) for i in range(depth)],
         nn.AdaptiveAvgPool2d((1,1)), nn.Flatten(), nn.Linear(h, n_classes))
```

Let's Have a Play

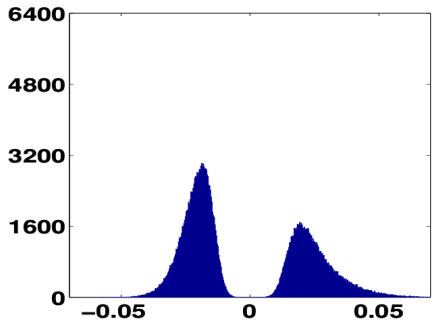
```
class ConvNet(nn.Module):
    def __init__(self):
        super(ConvNet, self).__init__()


        # Two convolution layers I am writing the first one
        # First convolutional layer takes single chennel images (batch_size specify the number of images) as input
        # We have 5x5 convolutions
        # We have 6 convolutional filter to produce output size 6*28*28 for a single training sample.
        # structure is : nn.conv2d(number of input channels, number of filters, conv kernel size, stride = 1)
        self.conv1 = nn.Conv2d(1, 6, 5, 1)
        # Note that Nparam 1*6*5*5 = 150 (+ 5 for bias per output).

        ##############################################################################
        # TODO: add another layer called self.conv2, 5x5 convolutions 16 filters in total.
        ##############################################################################
        #self.conv2 = nn.Conv2d(6, 16, 5, 1)
        self.mlp2 = nn.Conv2d(6,16, 1, 1)
        self.conv2 = nn.Conv2d(16, 16, 5, 1, groups=16)

        # Two more fully connected layers arguments (input size, output size)
```

https://colab.research.google.com/github/slucey-cs-cmu-edu/RVSS24/blob/main/CNN_Example.ipyn

Trends - Lower Precision

Reducing Precision
- Saving Memory
- Saving Computation

[Han et al. 2016]



$$\mathbb{R} \rightarrow \mathbb{I} \rightarrow \mathbb{B} \in \{-1, +1\}$$

32-bit 8-bit 1-bit

{-1,+1}	{0,1}
MUL	XNOR
ADD, SUB	Bit-Count (popcount)

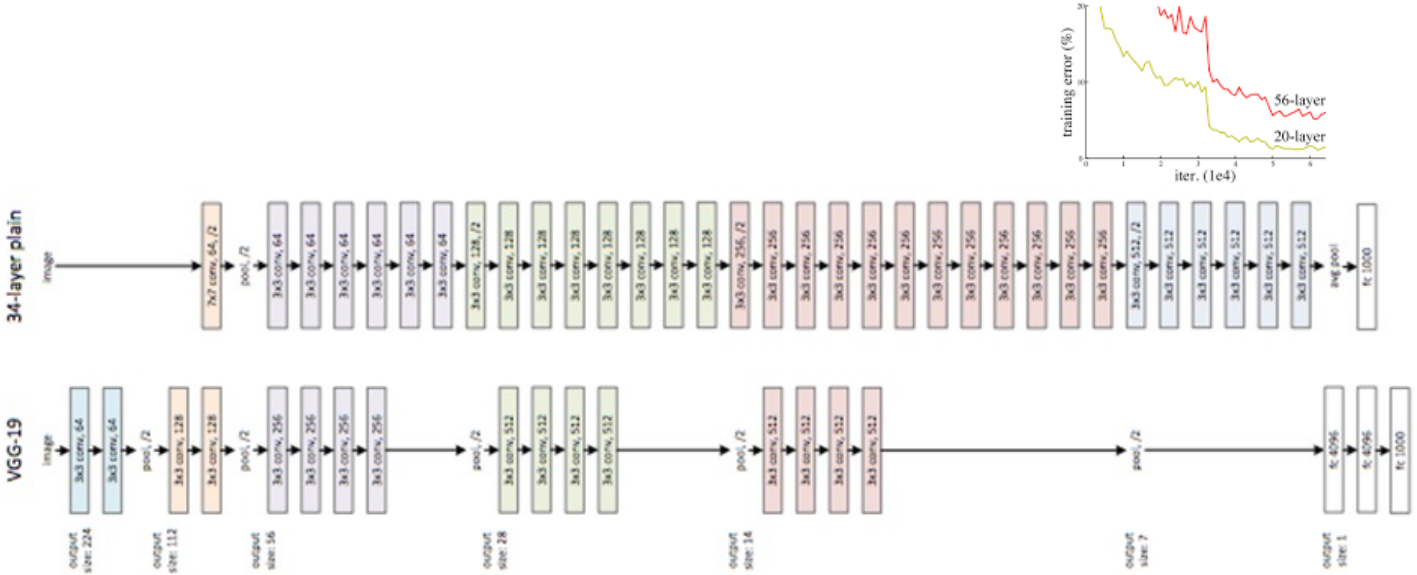
Rastegari, Mohammad, et al. "XNOR-Net: ImageNet Classification Using Binary Convolutional Neural Networks." ECCV 2016

Today

- ConvNets and AlexNet
- Depthwise Convolution
- **Skip/Residual Connections**

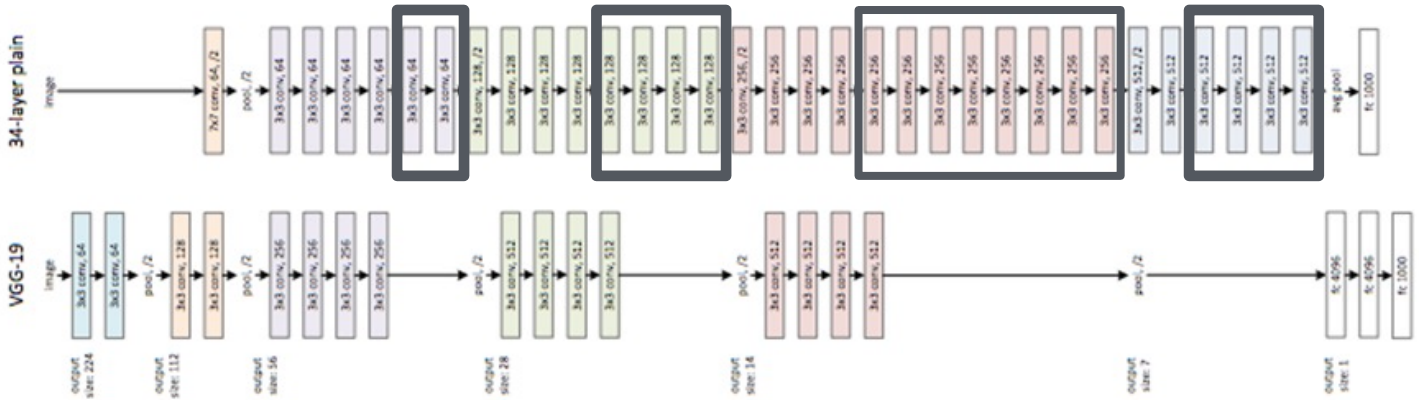
Neural Net : Deeper is Better?

- Very deep network -- training error increases.



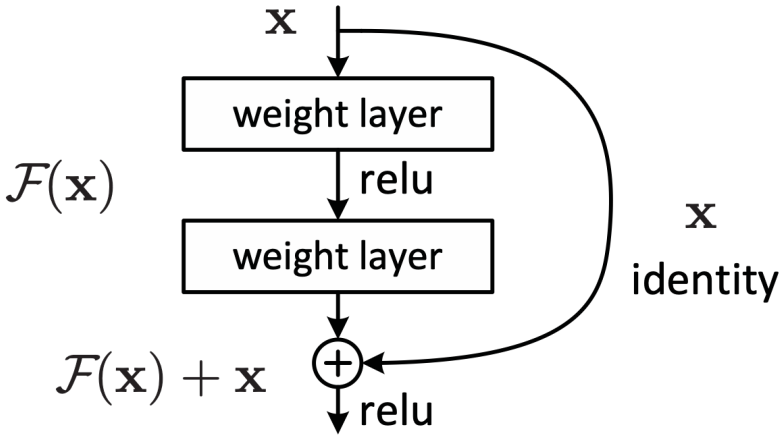
Neural Net : Deeper is Better?

- Very deep network -- training error increases.
- What if I just learn to do NOT A THING in some layer?
- 34 layer net then is just a 19 layer net with less loss!



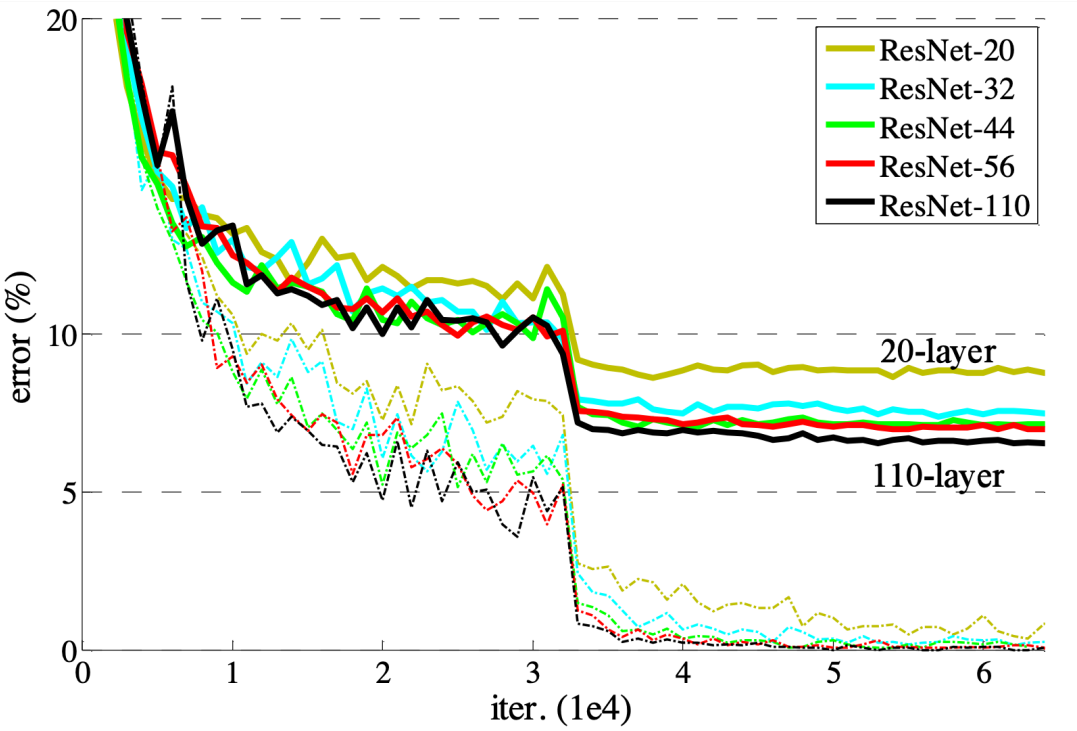
How Deep?

- Recent work has suggested that network depth is crucial for good performance (e.g. ImageNet).
- Counter intuitively, naively trained deeper networks tend to have higher train error than shallow networks.
- Innovation of residual learning has greatly helped with this.



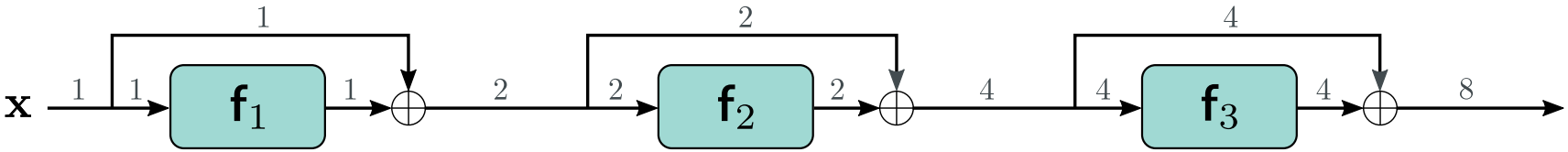
K. He, et al. "Deep residual learning for image recognition", CVPR 2016.

How Deep?



K. He, et al. "Deep residual learning for image recognition", CVPR 2016.

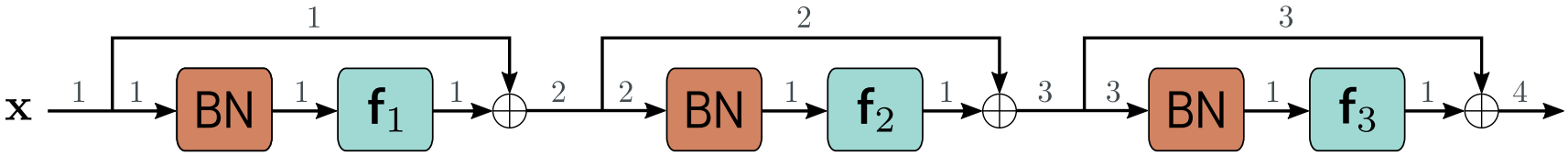
Initialization Problem



Try example:-

https://colab.research.google.com/github/udlbook/udlbook/blob/main/Notebooks/Chap11/11_3_Batch_Normalization.ipynb

Batch Norm

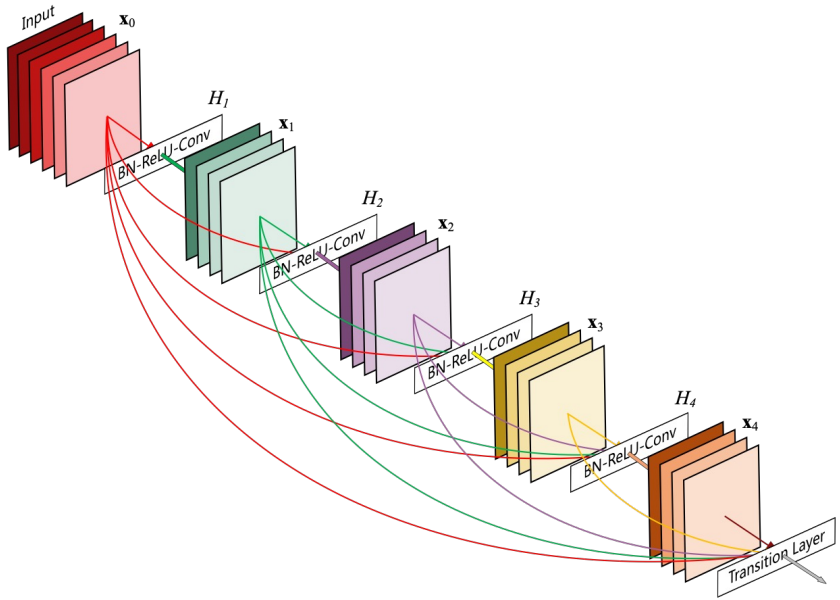


$$m_h = \frac{1}{|\mathcal{B}|}\sum_{i\in\mathcal{B}} h_i \qquad h_i \leftarrow \frac{h_i - m_h}{s_h + \epsilon}$$

$$s_h = \sqrt{\frac{1}{|\mathcal{B}|}\sum_{i\in\mathcal{B}}(h_i - m_h)^2}, \qquad h_i \leftarrow \gamma h_i + \delta$$

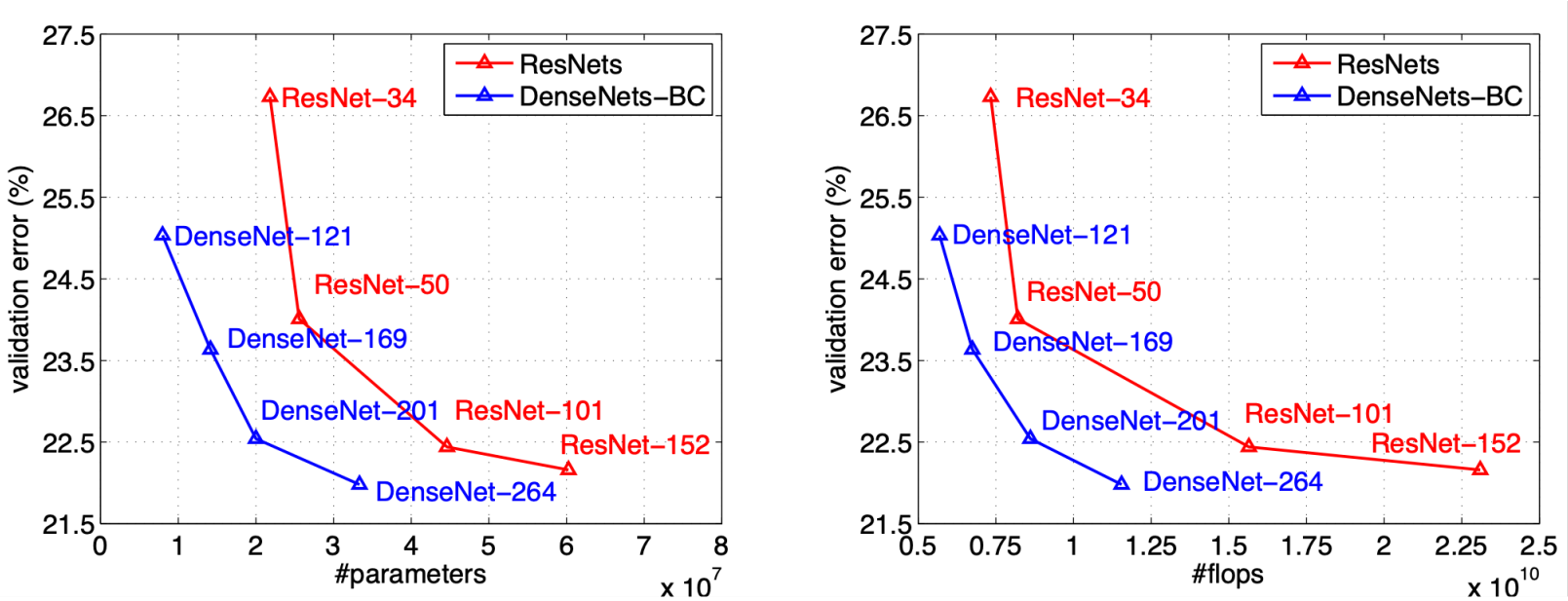
Prince “Understanding Deep Learning” 2023.

Densely Connected Networks

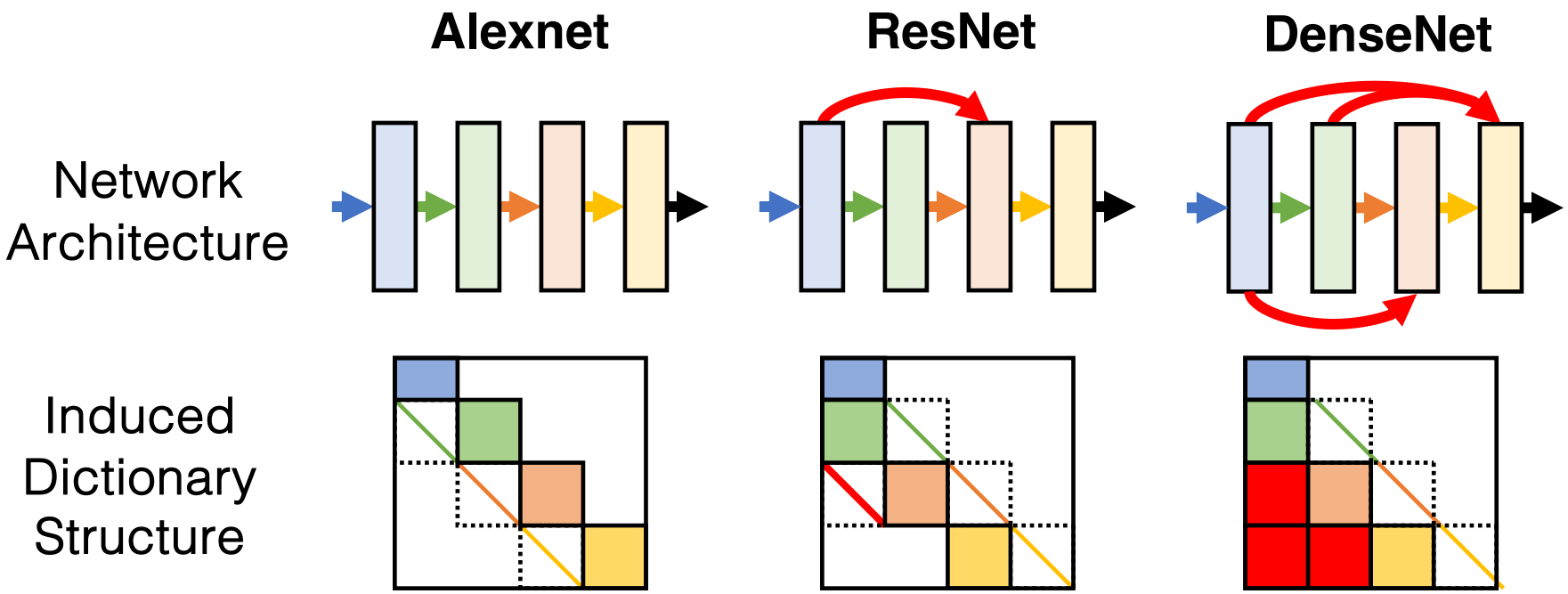


G. Huang et al. "Densely Connected Convolutional Networks", CVPR 2017.

Densely Connected Networks



G. Huang et al. "Densely Connected Convolutional Networks", CVPR 2017.

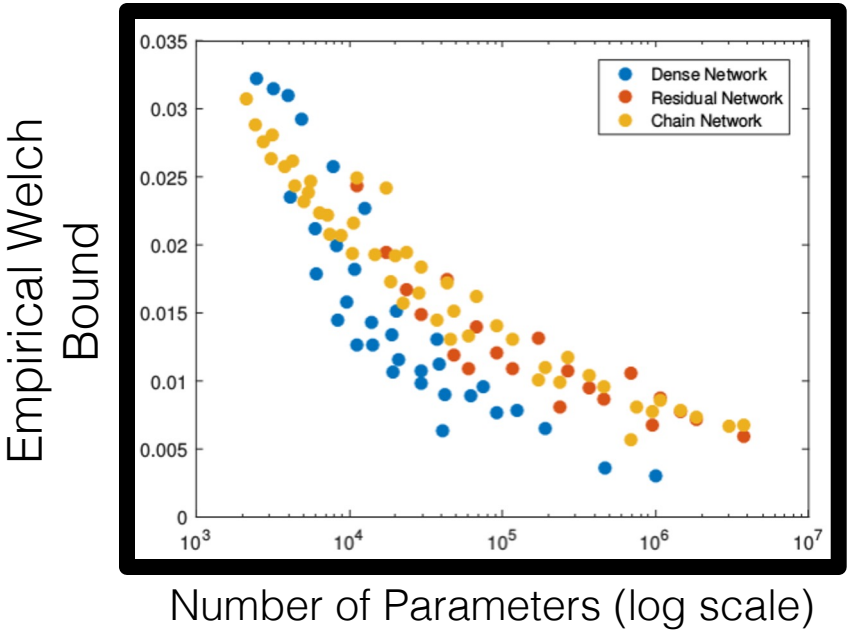


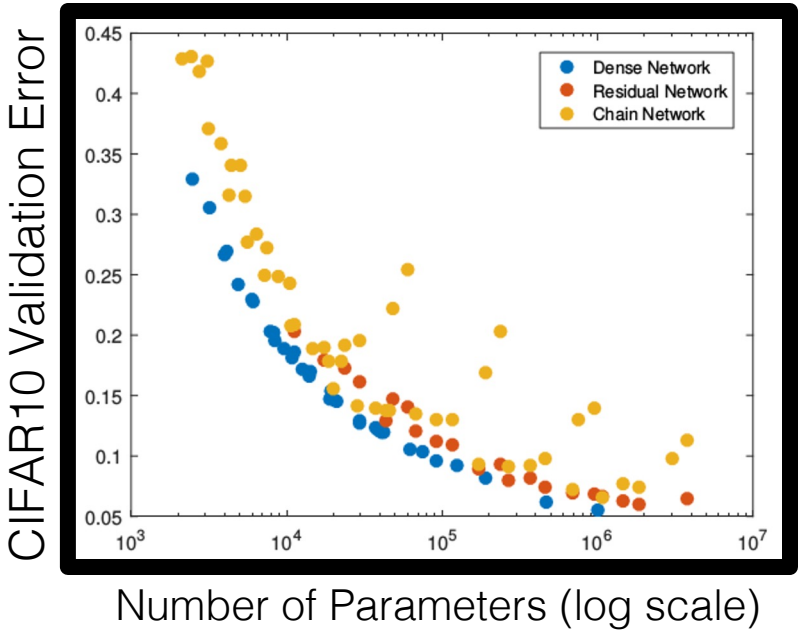
Gao Huang, Zhuang Liu, Kilian Weinberger, and Laurens van der Maaten.
"*Densely Connected Convolutional Networks.*" CVPR, 2017.

Murdock and Lucey. "Dataless Model Selection with the Deep Frame Potential". In CVPR 2020.

Experimental Results

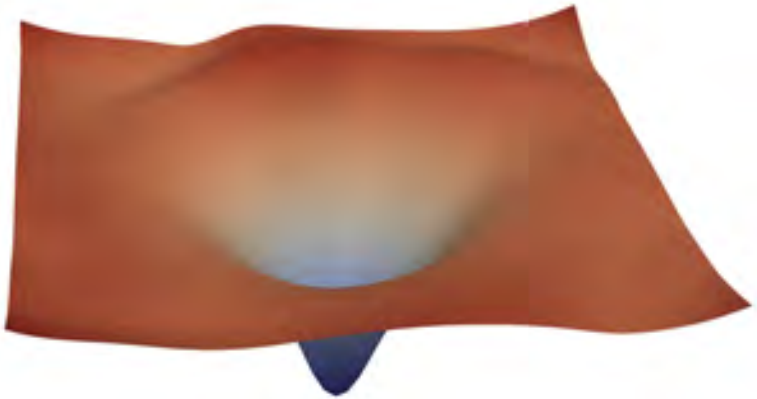
- Dense skip connections induce dictionary structures with lower Welch bounds and give better performance with fewer parameters.



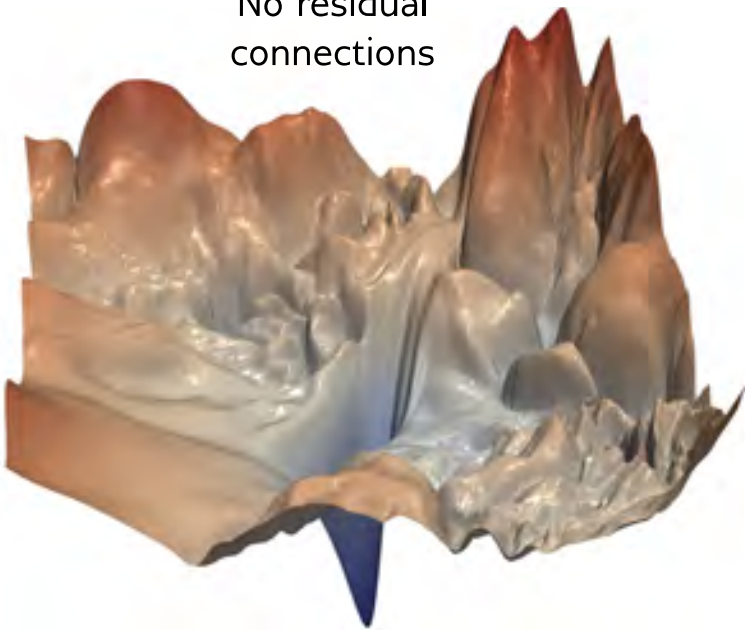


Murdock and Lucey. “Dataless Model Selection with the Deep Frame Potential”. In CVPR 2020.

a) Residual connections

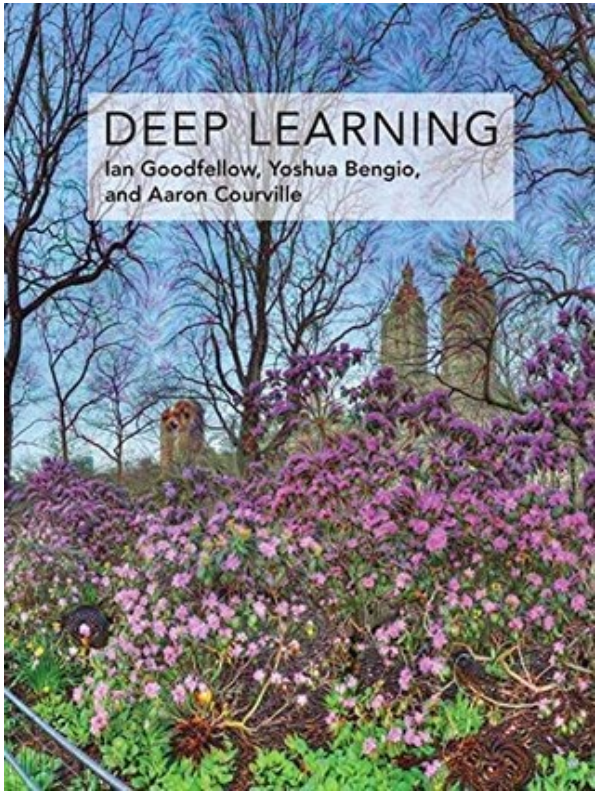


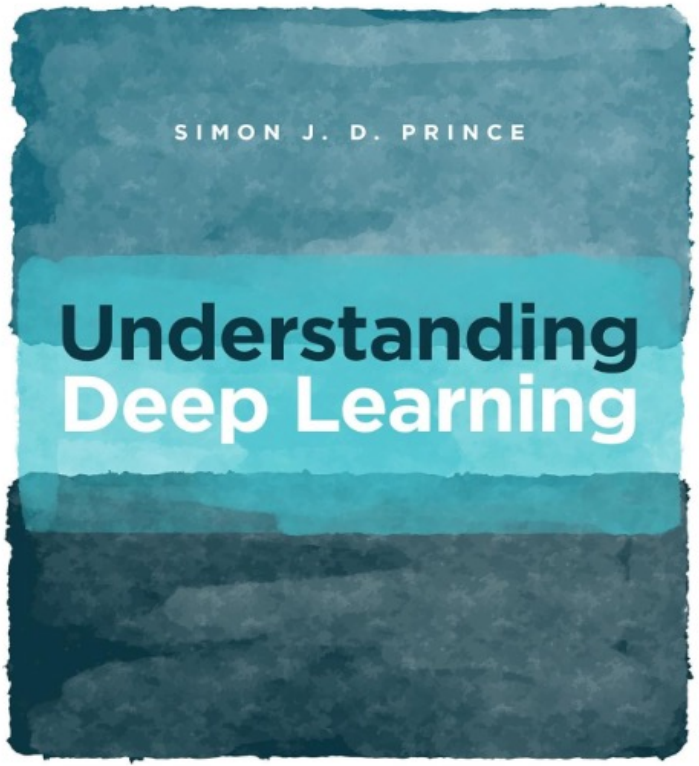
b) No residual connections



Prince “Understanding Deep Learning” 2023.

More to read…


DEEP LEARNING
Ian Goodfellow, Yoshua Bengio,
and Aaron Courville


SIMON J. D. PRINCE
Understanding
Deep Learning