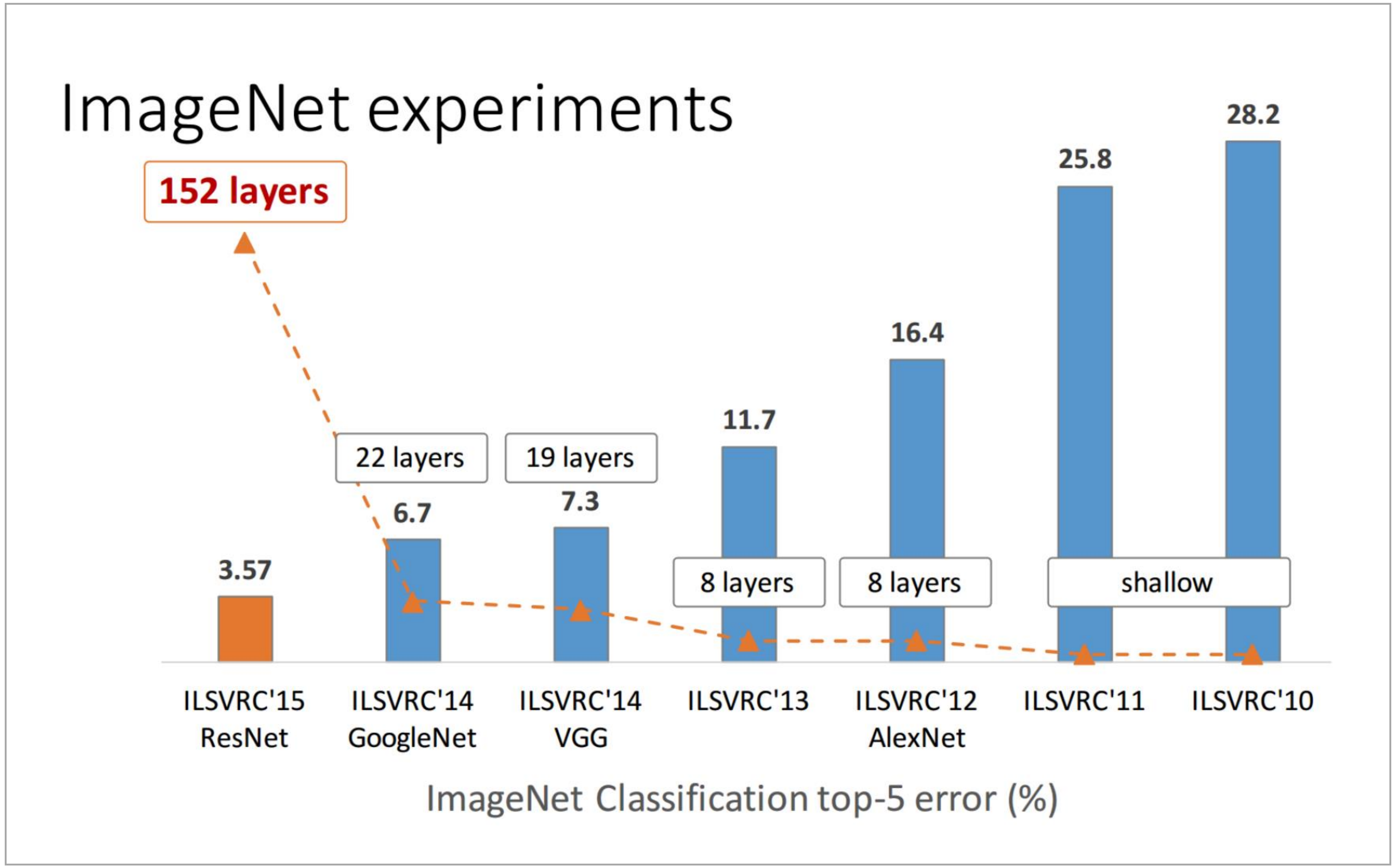
Modern CNNs

Industrial AI Lab.
Prof. Seungchul Lee

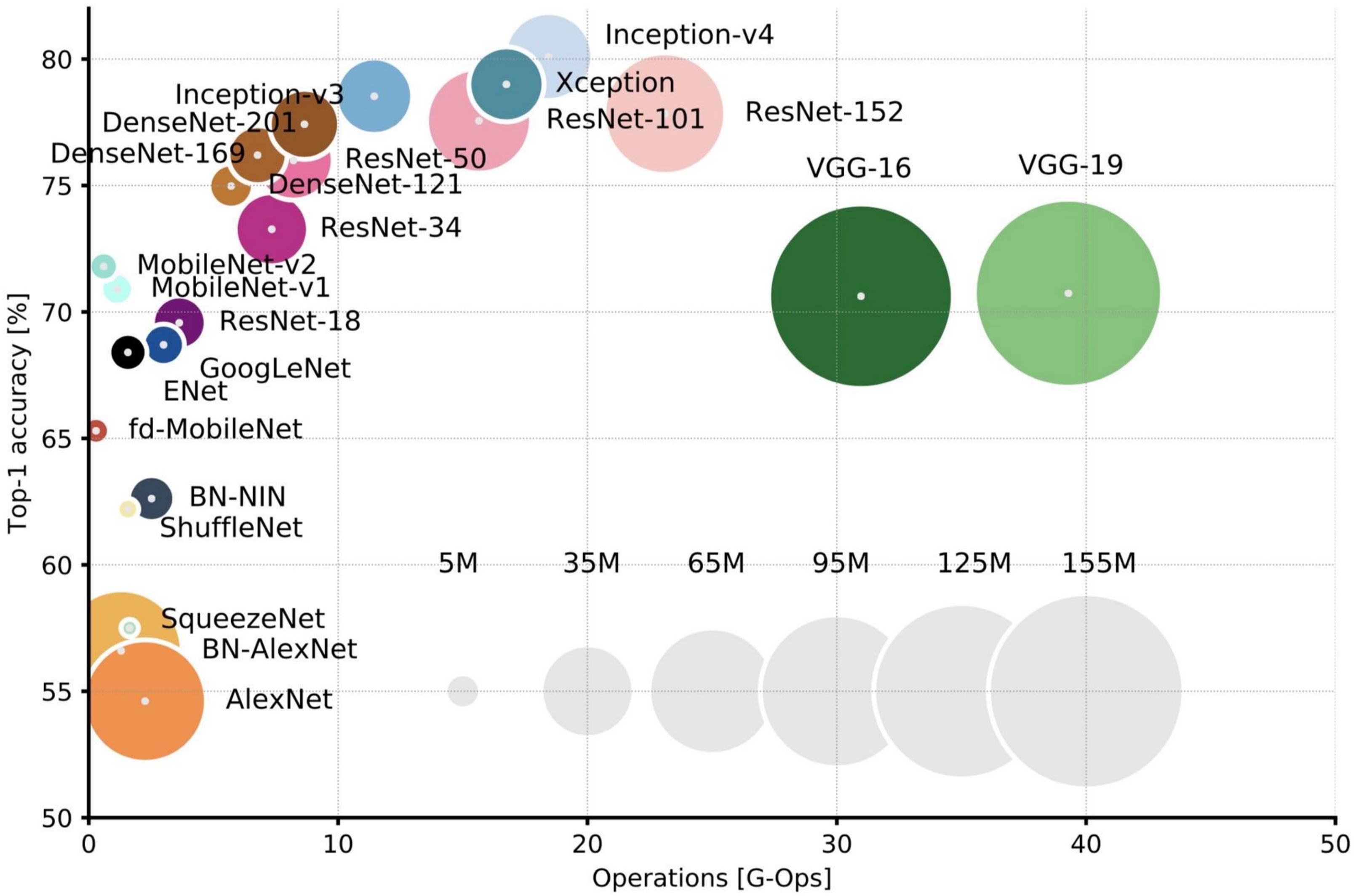
ImageNet

- Human performance = 5.1 %



from Kaiming He slides "Deep residual learning for image recognition," ICML, 2016.

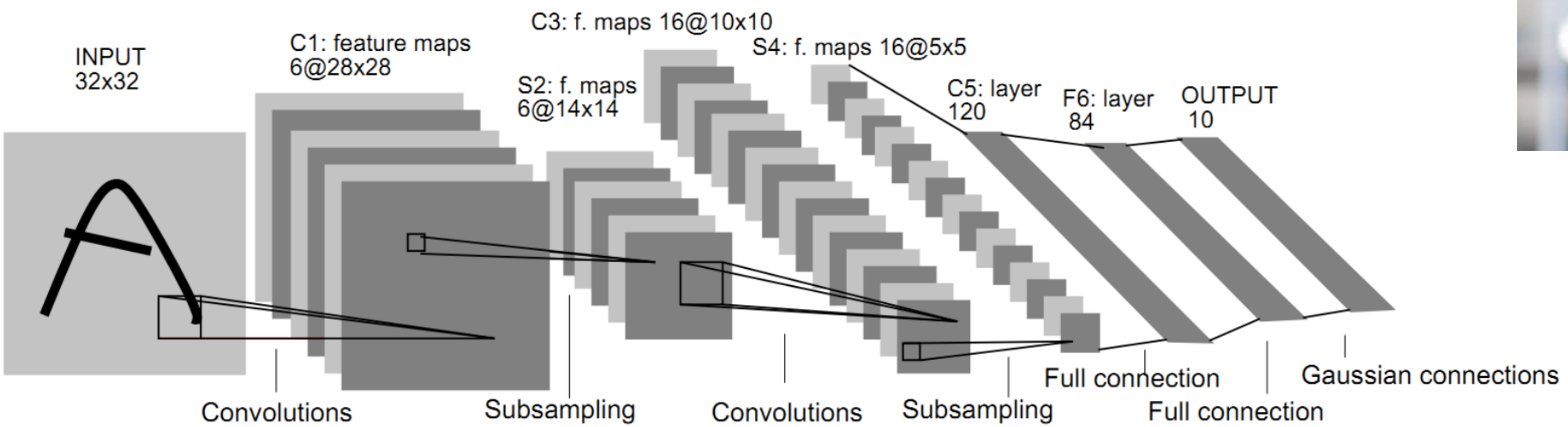
ImageNet



Inception-v4
Xception
Inception-v3
DenseNet-201
ResNet-101
ResNet-152
DenseNet-169
ResNet-50
VGG-16
VGG-19
DenseNet-121
ResNet-34
MobileNet-v2
MobileNet-v1
ResNet-18
GoogLeNet
ENet
fd-MobileNet
BN-NIN
ShuffleNet
5M
35M
65M
95M
125M
155M
SqueezeNet
BN-AlexNet
AlexNet
Top-1 accuracy [%]
80
75
70
65
60
55
50
0
10
20
30
40
50
Operations [G-Ops]

LeNet

- CNN = Convolutional Neural Networks = ConvNet
- LeCun, Y., Bottou, L., Bengio, Y., and Haffner, P. (1998). Gradient-based learning applied to document recognition.
- All are still the basic components of modern ConvNets!

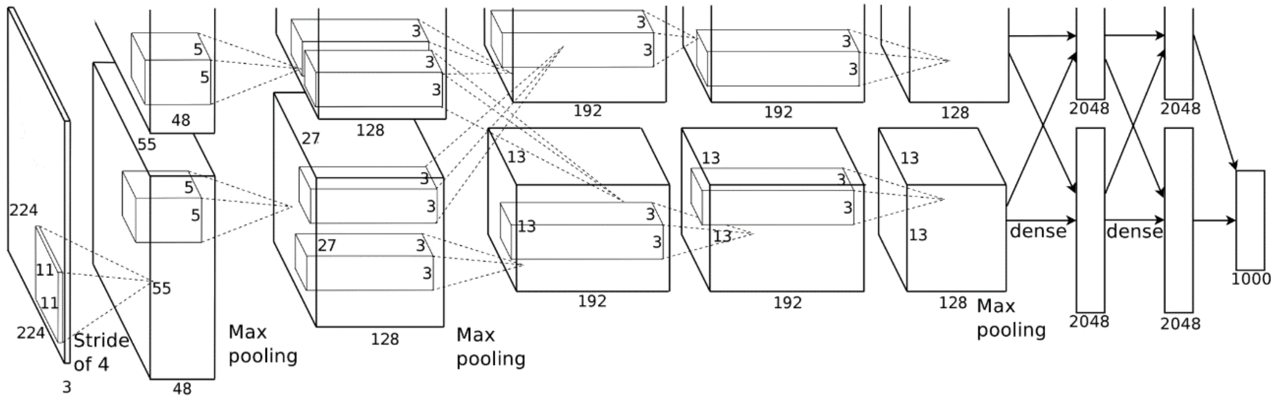


Yann LeCun

AlexNet

- Simplified version of Krizhevsky, Alex, Sutskever, and Hinton. "Imagenet classification with deep convolutional neural networks." NIPS 2012

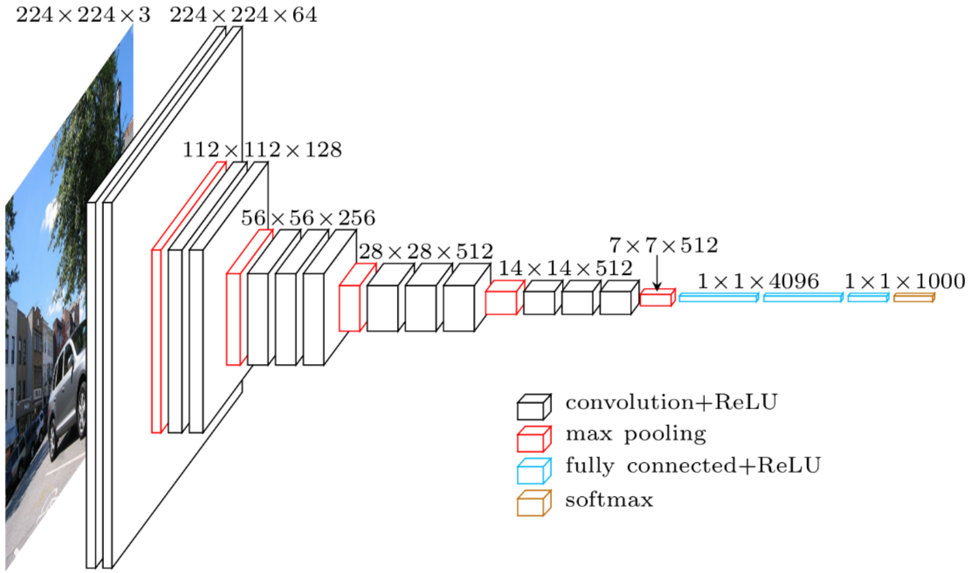
- LeNet-style backbone, plus:
  - ReLU [Nair & Hinton 2010]
    - "RevoLUtion of deep learning"*
    - Accelerate training
  - Dropout [Hinton et al 2012]
    - In-network ensembling
    - Reduce overfitting
  - Data augmentation
    - Label-preserving transformation
    - Reduce overfitting

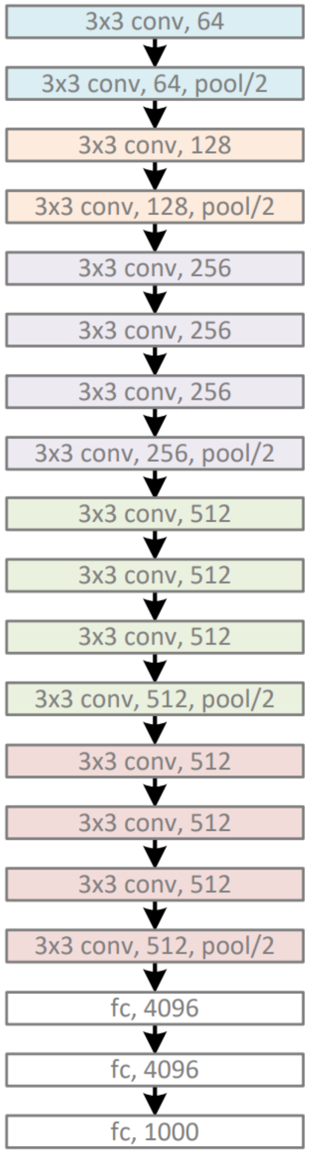


VGG-16/19

- Simonyan, Karen, and Zisserman. "Very deep convolutional networks for large-scale image recognition." (2014)

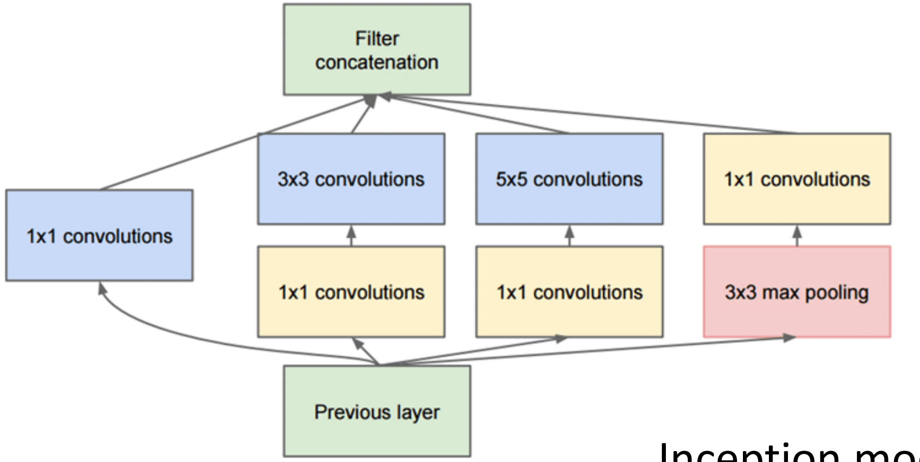
- Simply “Very Deep”!
  - Modularized design
    - 3x3 Conv as the module
    - Stack the same module
    - Same computation for each module
  - Stage-wise training
    - VGG-11 → VGG-13 → VGG-16
    - We need a better initialization…



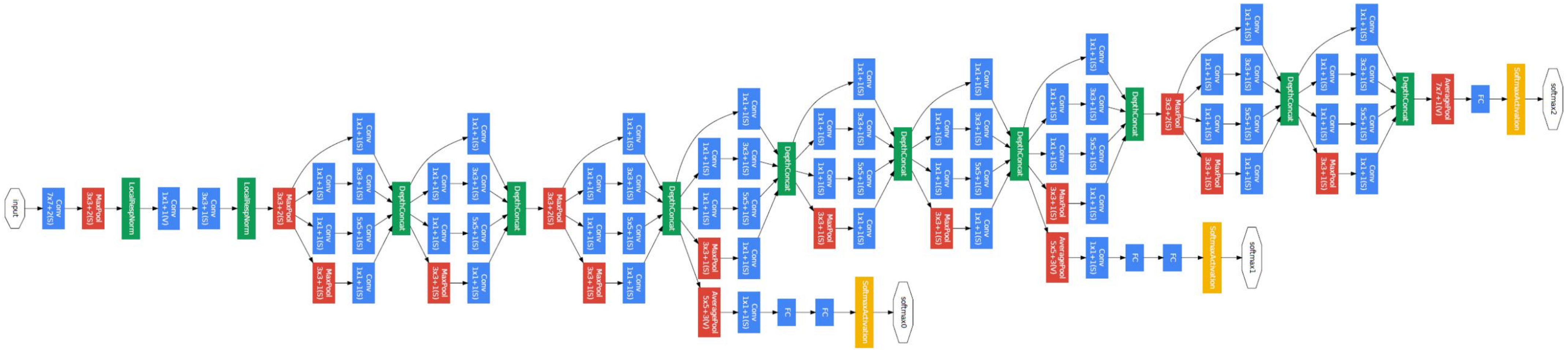


GoogleNet/Inception

- Multiple branches
  - e.g., 1x1, 3x3, 5x5, pool
- Shortcuts
  - stand-alone 1x1, merged by concatenation
- Bottleneck
  - Reduce dim by 1x1 before expensive 3x3/5x5 conv



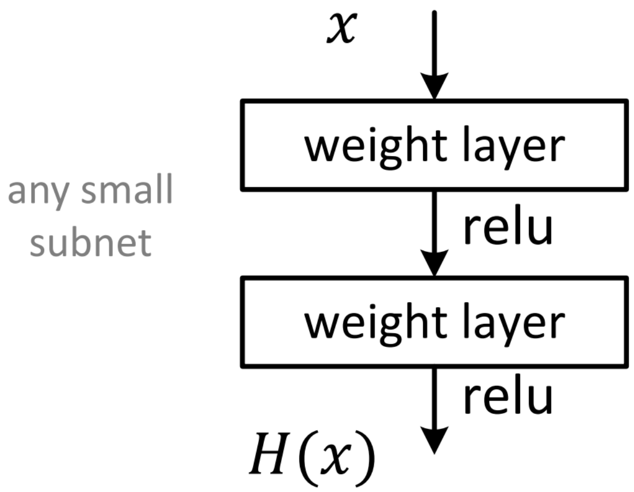
Inception module



ResNet (Deep Residual Learning)

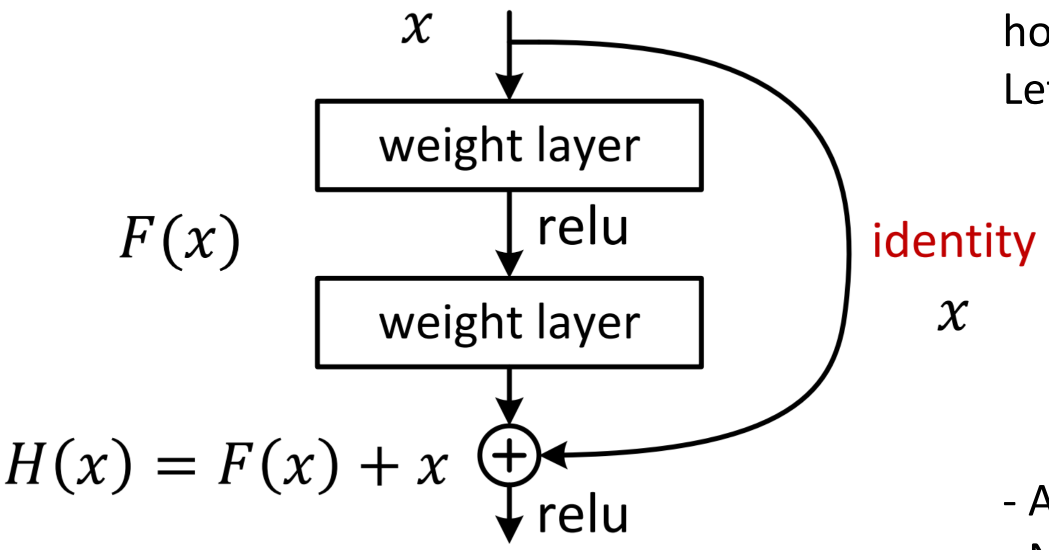
- He, Kaiming, et al. “Deep residual learning for image recognition.” CVPR. 2016.
- Plane net

$H(x)$ is any desired mapping,
hope the small subnet fit $H(x)$



ResNet (Deep Residual Learning)

- He, Kaiming, et al. "Deep residual learning for image recognition." CVPR. 2016.
- Residual net
- Skip connection

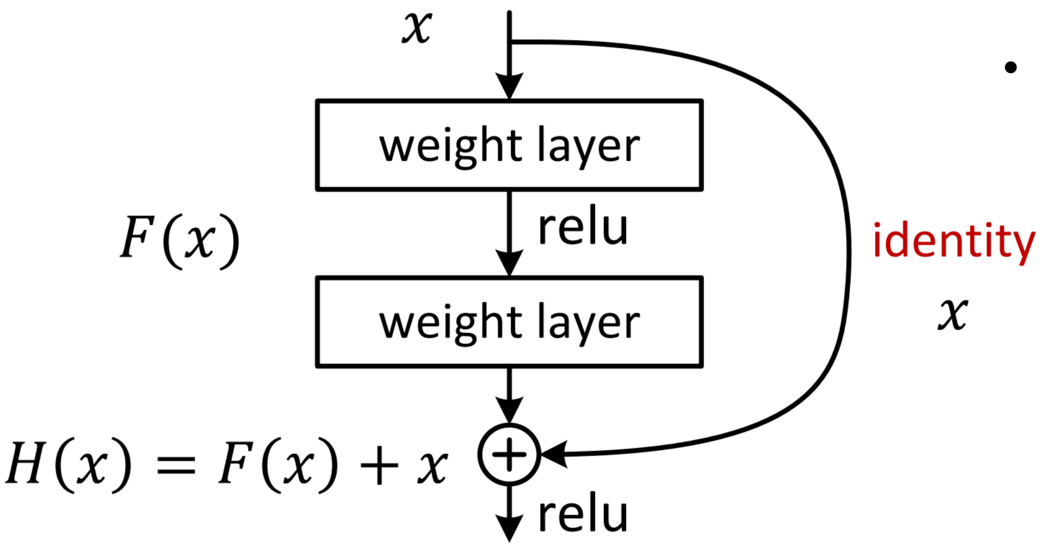


$H(x)$ is any desired mapping,
~~hope the small subnet fit $H(x)$~~
hope the small subnet fit $F(x)$
Let $H(x) = F(x) + x$

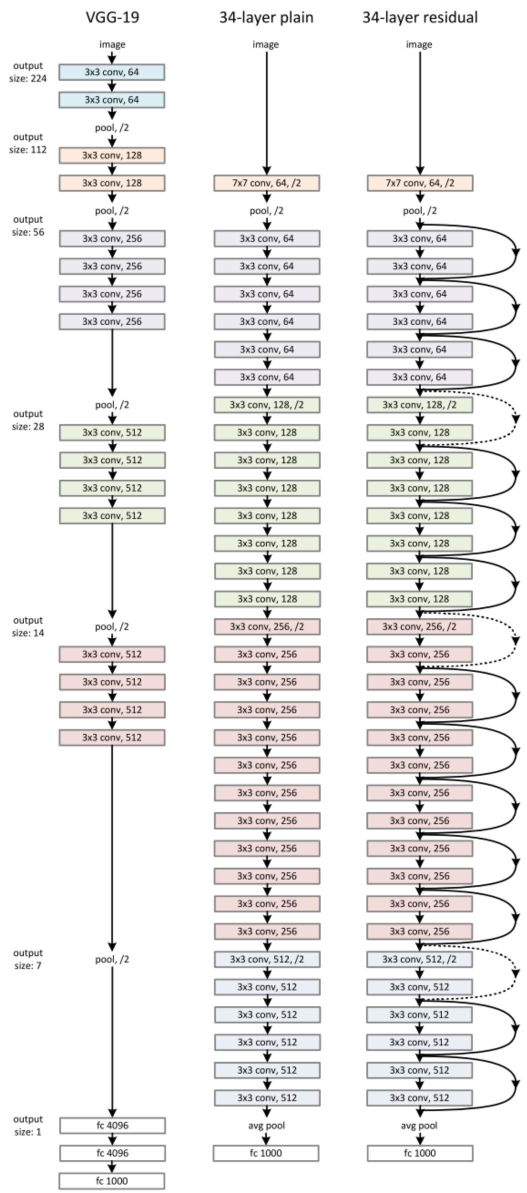
- A direct connection between 2 non-consecutive layers
- No gradient vanishing

ResNet (Deep Residual Learning)

- Parameters are optimized to learn a residual, that is the difference between the value before the block and the one needed after.
- $F(x)$ is a residual mapping w.r.t. identity



- If identity were optimal, easy to set weights as 0
- If optimal mapping is closer to identity, easier to find small fluctuations



Skip Connection

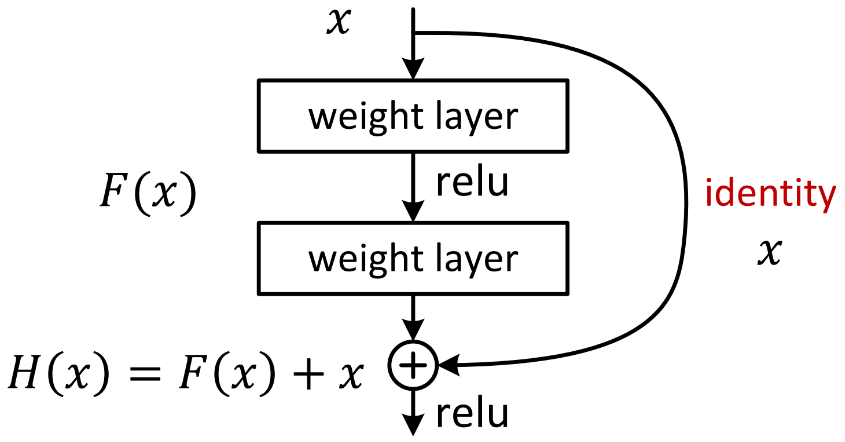
- A skip connection is a connection that bypasses at least one layer.

- Here, it is often used to transfer local information by concatenating or summing feature maps from the downsampling path with feature maps from the upsampling path.
  - Will see it at FCN later

  - Merging features from various resolution levels helps combining context information with spatial information.

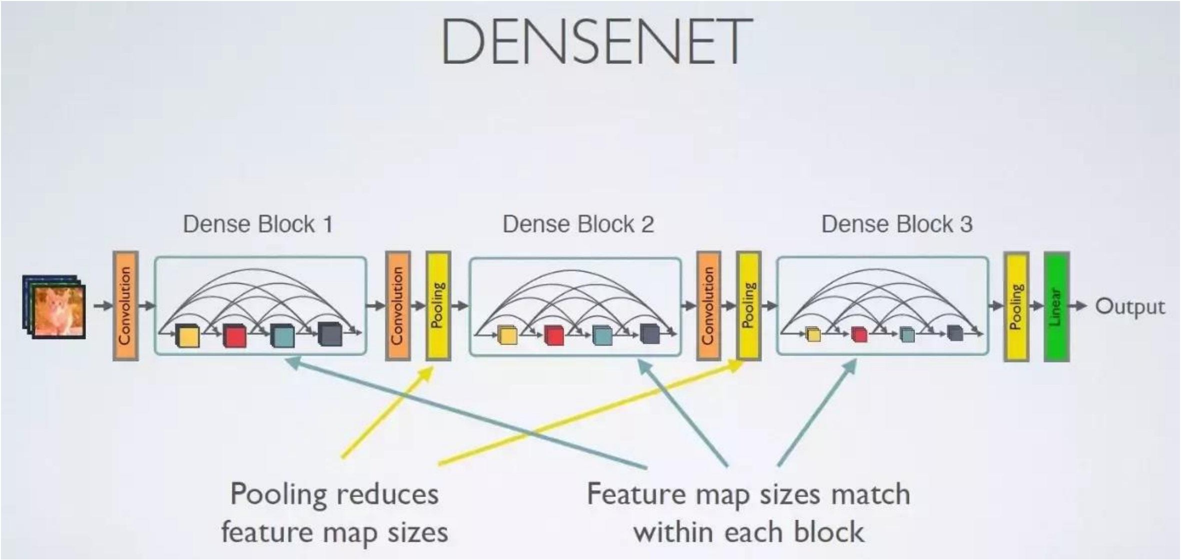
Residual Net

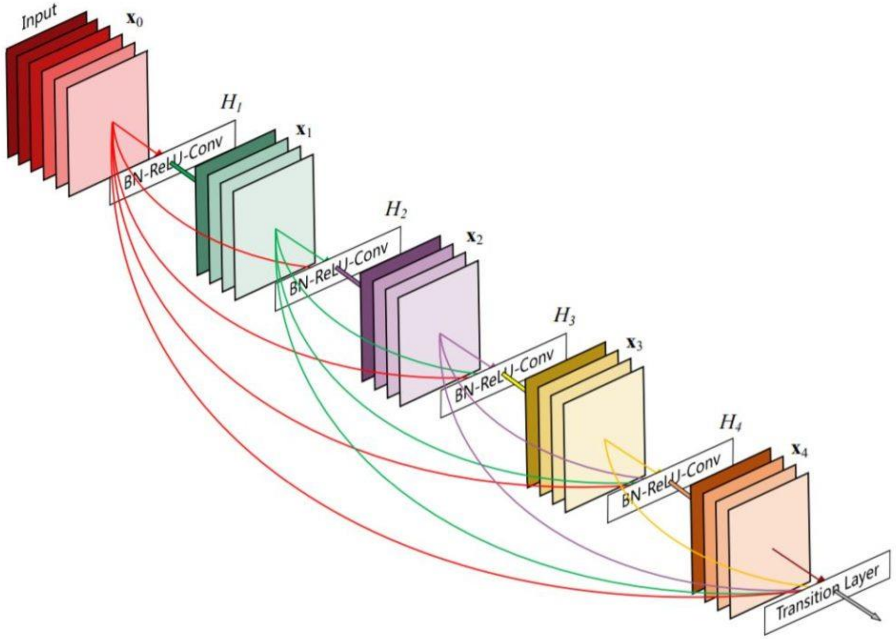
```python
def residual_net(x):
    conv1 = tf.layers.conv2d(inputs = x,
                             filters = 32,
                             kernel_size = [3, 3],
                             padding = "SAME",
                             activation = tf.nn.relu)
    conv2 = tf.layers.conv2d(inputs = conv1,
                             filters = 32,
                             kernel_size = [3, 3],
                             padding = "SAME",
                             activation = tf.nn.relu)

    maxp2 = tf.layers.max_pooling2d(inputs = x + conv2,
                                    pool_size = [2, 2],
                                    strides = 2)

    flat = tf.layers.flatten(maxp2)
    hidden = tf.layers.dense(inputs = flat,
                             units = n_hidden,
                             activation = tf.nn.relu)
    output = tf.layers.dense(inputs = hidden,
                             units = n_output)

    return output
```



DensNets

- Densely Connected Convolutional Networks

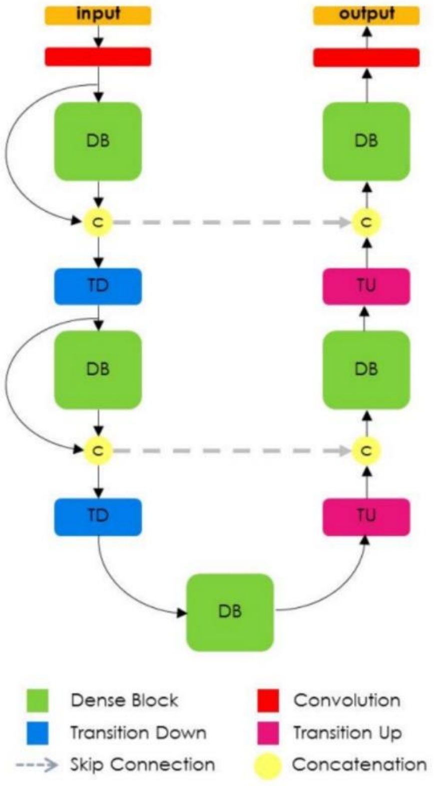


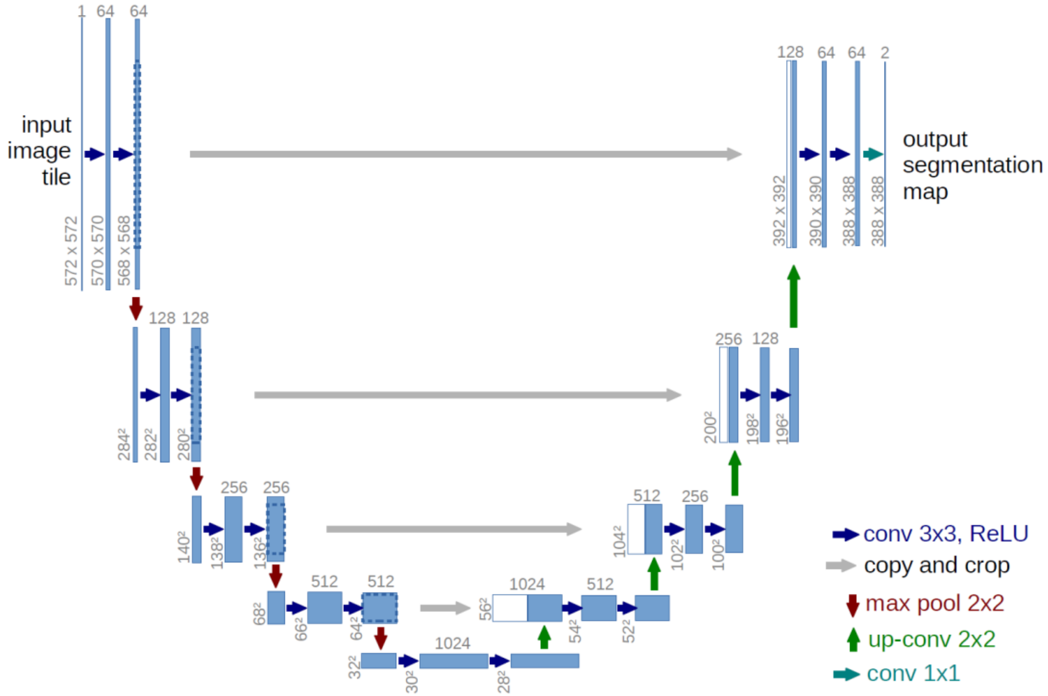


Huang, Gao, et al., "Densely connected convolutional networks"
Proceedings of the IEEE conference on computer vision and pattern recognition. Vol. 1. No. 2. 2017.

U-Net

- The U-Net owes its name to its symmetric shape
  - better segmentation in medical imaging





Ronneberger, Olaf; Fischer, Philipp; Brox, Thomas (2015),
"U-Net: Convolutional Networks for Biomedical Image Segmentation." arXiv:1505.04597

Pre-trained Models

- Training a model on ImageNet from scratch takes days or weeks.
- Many models trained on ImageNet and their weights are publicly available!
- Transfer learning
  - Use pre-trained weights, remove last layers to compute representations of images
  - The network is used as a generic feature extractor
  - Train a classification model from these features on a new classification task
  - Pre- trained models can extract more general image features that can help identify edges, textures, shapes, and object composition
  - Better than handcrafted feature extraction on natural images

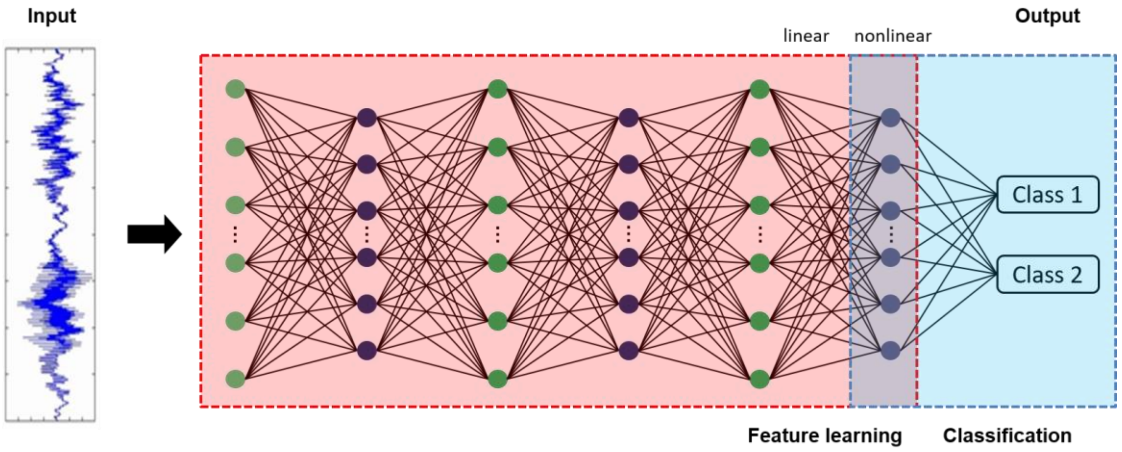


Image Classification with VGG16

- Target data
  - 5 classes

```
Dict = ['Hat','Cube','Card','Torch','screw']
```

- Target data to VGG16
  - Poor performance

```
1. mosquito_net: 4.66%
2. toilet_tissue: 4.00%
3. envelope: 2.29%
4. carton: 2.20%
5. photocopier: 1.86%
```

Label : Cube

Transfer Learning

- We assume that these model parameters contain the knowledge learned from the source data set and that this knowledge will be equally applicable to the target data set.
- We will train the output layer from scratch, while the parameters of all remaining layers are fine tuned based on the parameters of the source model.
- Or initialize all weights from pre-trained model, then train them with target data

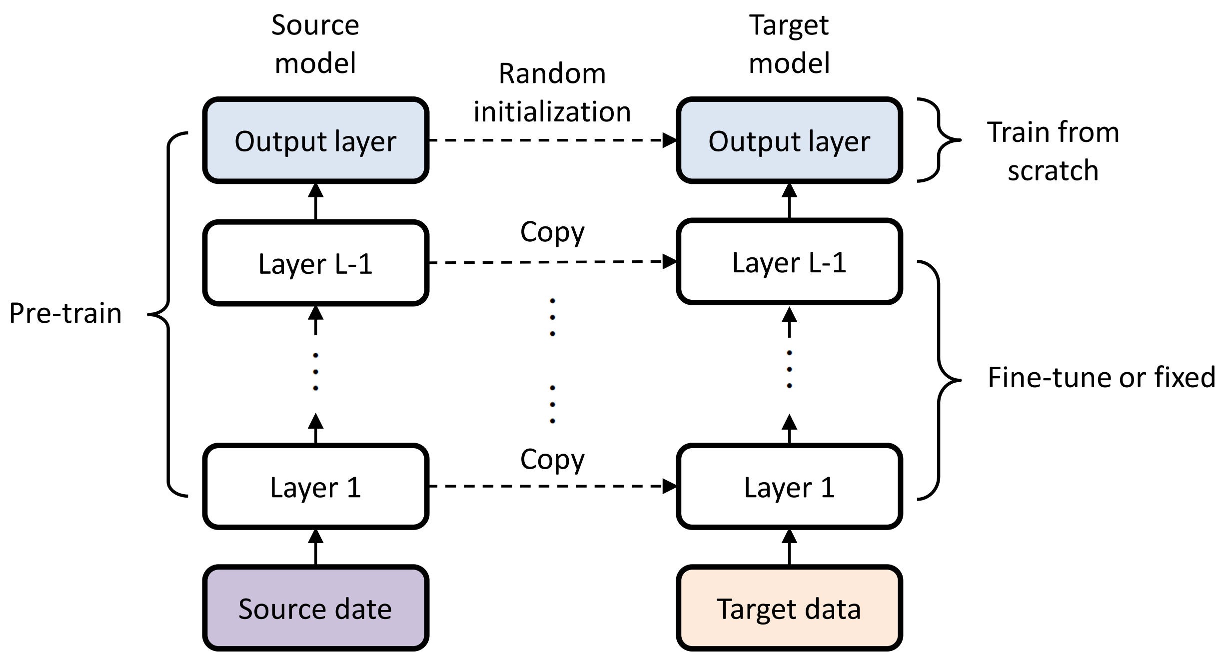
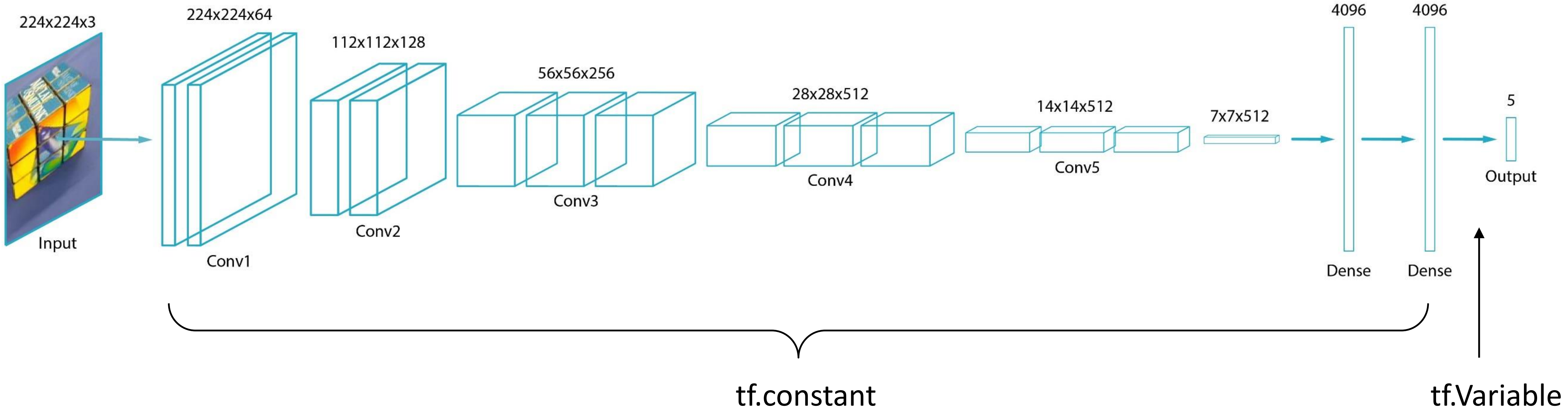


Image from http://d2l.ai/

Transfer Learning Structure



Transfer Learning Implementation

```
vgg16_weights = model.get_weights()

weights = {
    'conv1_1' : tf.constant(vgg16_weights[0]),
    'conv1_2' : tf.constant(vgg16_weights[2]),

    'conv2_1' : tf.constant(vgg16_weights[4]),
    'conv2_2' : tf.constant(vgg16_weights[6]),

    'conv3_1' : tf.constant(vgg16_weights[8]),
    'conv3_2' : tf.constant(vgg16_weights[10]),
    'conv3_3' : tf.constant(vgg16_weights[12]),

    'conv4_1' : tf.constant(vgg16_weights[14]),
    'conv4_2' : tf.constant(vgg16_weights[16]),
    'conv4_3' : tf.constant(vgg16_weights[18]),

    'conv5_1' : tf.constant(vgg16_weights[20]),
    'conv5_2' : tf.constant(vgg16_weights[22]),
    'conv5_3' : tf.constant(vgg16_weights[24]),

    'fc1' : tf.constant(vgg16_weights[26]),
    'fc2' : tf.constant(vgg16_weights[28]),
    # train from scratch
    'out' : tf.Variable(tf.random_normal([4096, 5], stddev = 0.1))
}
```

```
biases = {
    'conv1_1' : tf.constant(vgg16_weights[1]),
    'conv1_2' : tf.constant(vgg16_weights[3]),

    'conv2_1' : tf.constant(vgg16_weights[5]),
    'conv2_2' : tf.constant(vgg16_weights[7]),

    'conv3_1' : tf.constant(vgg16_weights[9]),
    'conv3_2' : tf.constant(vgg16_weights[11]),
    'conv3_3' : tf.constant(vgg16_weights[13]),

    'conv4_1' : tf.constant(vgg16_weights[15]),
    'conv4_2' : tf.constant(vgg16_weights[17]),
    'conv4_3' : tf.constant(vgg16_weights[19]),

    'conv5_1' : tf.constant(vgg16_weights[21]),
    'conv5_2' : tf.constant(vgg16_weights[23]),
    'conv5_3' : tf.constant(vgg16_weights[25]),

    'fc1' : tf.constant(vgg16_weights[27]),
    'fc2' : tf.constant(vgg16_weights[29]),
    # train from scratch
    'out' : tf.Variable(tf.random_normal([5], stddev = 0.1))
}
```

Testing

```
Dict = ['Hat','Cube','Card','Torch','screw']
```

```
Prediction : Cube
Probability : [0.   0.99 0.01 0.   0.  ]
```