From Feature Engineering to Network Engineering

ShatterLine Labs

Auro Tripathy

May 22, 2018

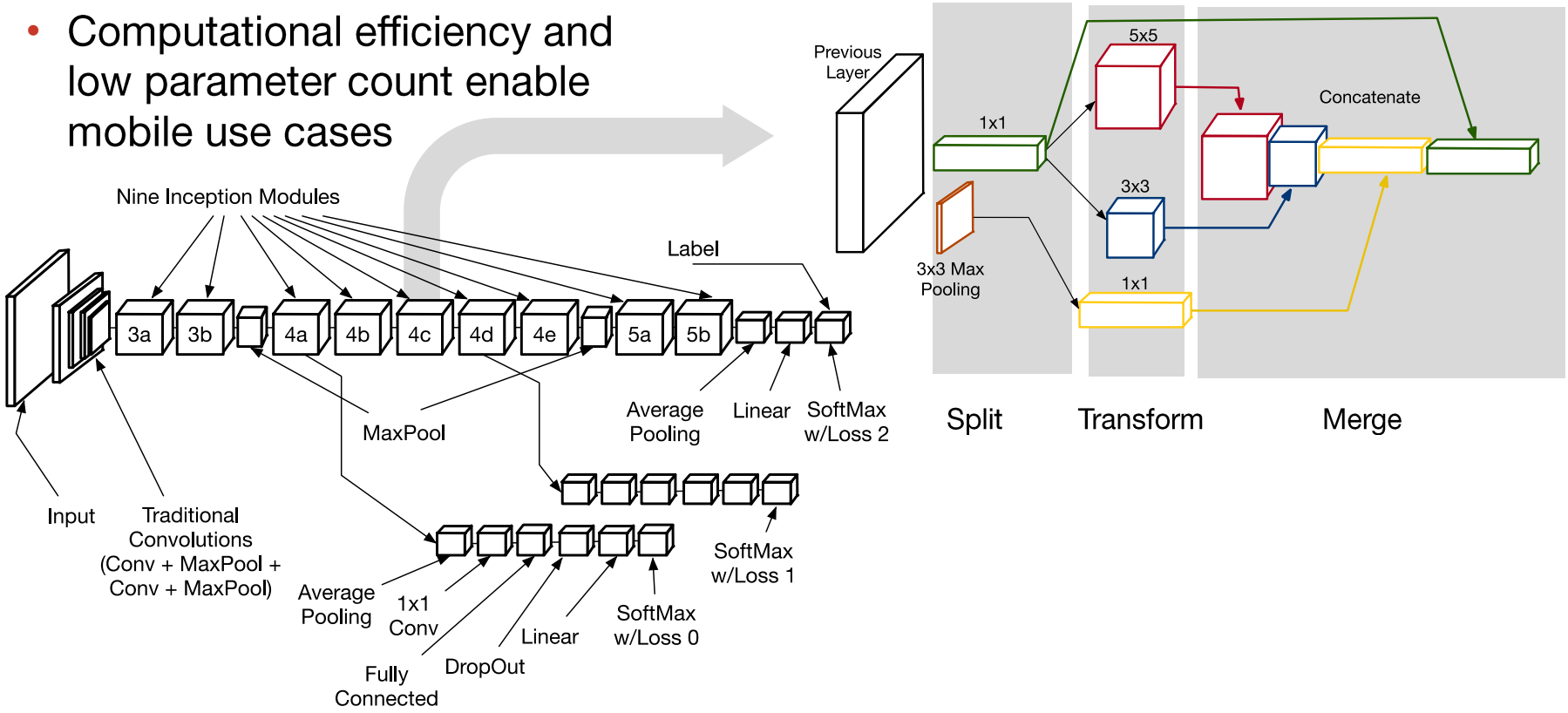
Outline

- Review Four Network Topologies
  - Split-transform-merge connections for reduced parameters
  - Residual connections for higher accuracy through depth
  - Encode/Decode networks for segmentation
  - Siamese networks for learning similarity
- Canonical Code for each Topology
- Goal - Equip you with the Basic Building Blocks for your own Designs

Split-Transform-Merge Topologies to Reduce Compute

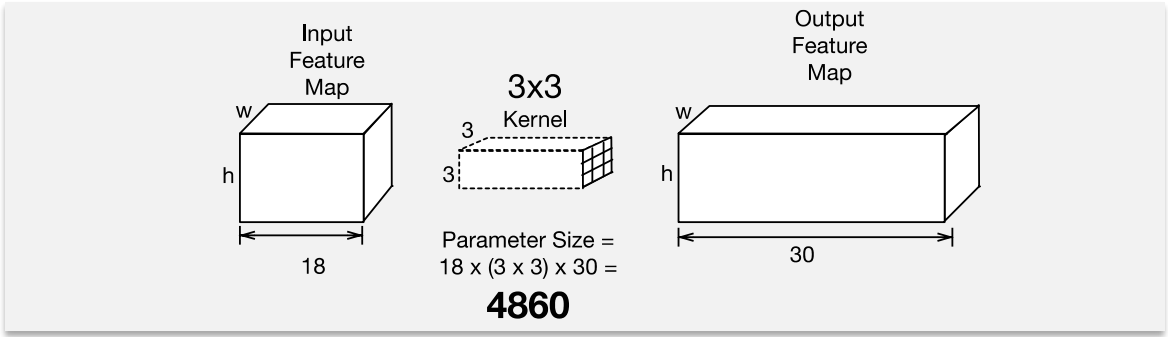
Split-Transform-Merge in the Inception Module in GoogleNet

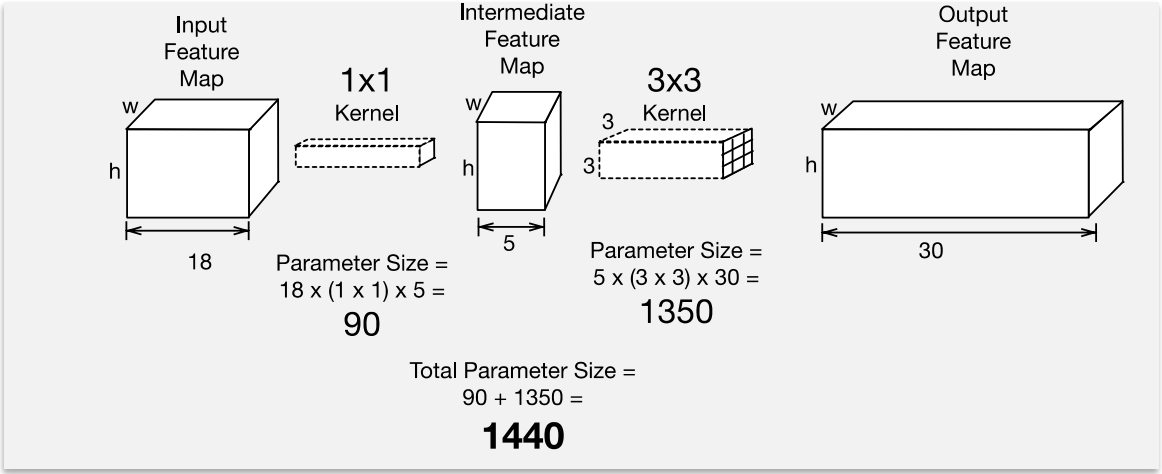
- Computational efficiency and low parameter count enable mobile use cases



Replacing a 3x3 Conv with a 1x1 Conv followed by a 3x3 Conv Results in Far Fewer Parameters

Same Size Input Feature Maps





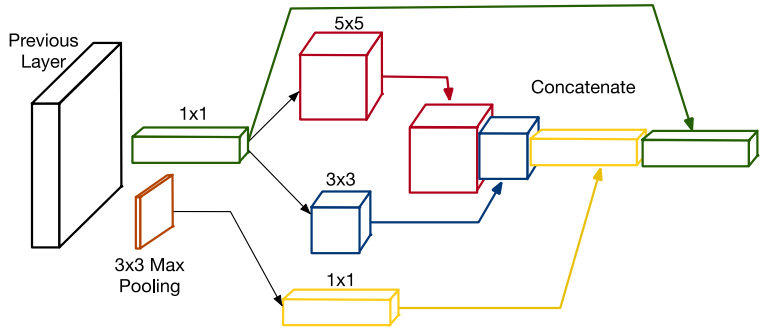
Same Size Output Feature Maps

Parameters reduce from 4860 to 1440

Expressing in Keras

1
```
conv_1x1 = Conv2D(192, (1, 1), padding='same', activation='relu')(previous_layer)
```

2
```
conv_1x1_before_5x5 = Conv2D(16, (1, 1),
          padding='same',
          activation='relu')(previous_layer)
Conv_5x5 = Conv2D(48, (5,5),
          padding='same',
          activation='relu')(conv_1x1_before_5x5)
```

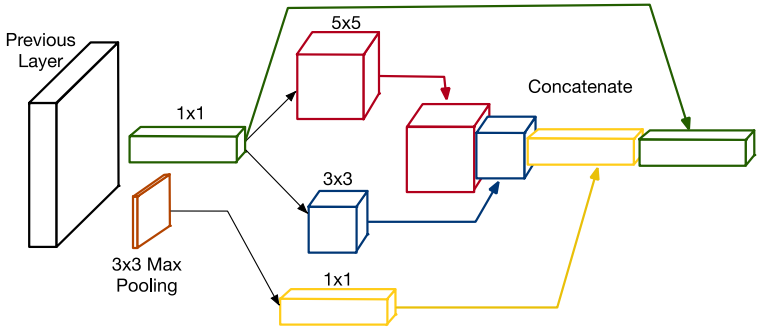


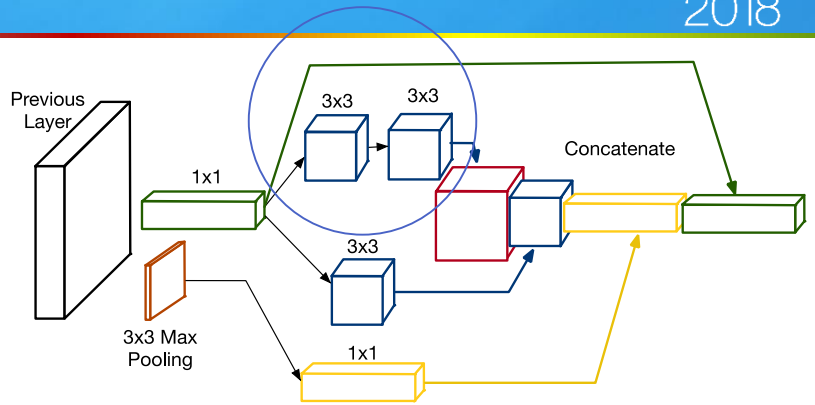
3
```
conv_1x1_before_3x3 = Conv2D(96, (1,1),
          padding='same',
          activation='relu')(previous_layer)
conv_3x3 = Conv2D(208, (3, 3) ,
          padding='same',
          activation='relu'(conv_1x1_before_3x3)
```

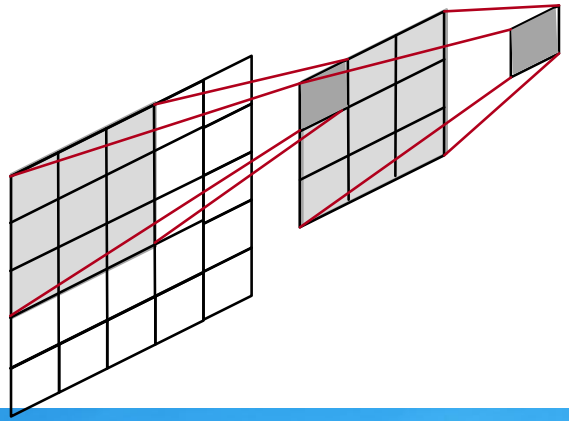
5
```
output = concatenate([conv_1x1,
          conv_3x3,
          conv_5x5,
          conv_1x1_after_pool],
          concat_axis=1)
```

4
```
pool = MaxPooling2D(pool_size=(3,3),strides=(1,1), padding='same')(previous_layer)
conv_1x1_after_pool = Conv2D(64, (1, 1), padding='same', activation='relu')(pool)
```

Layered Factorization into Smaller, Less Expensive Convolutions

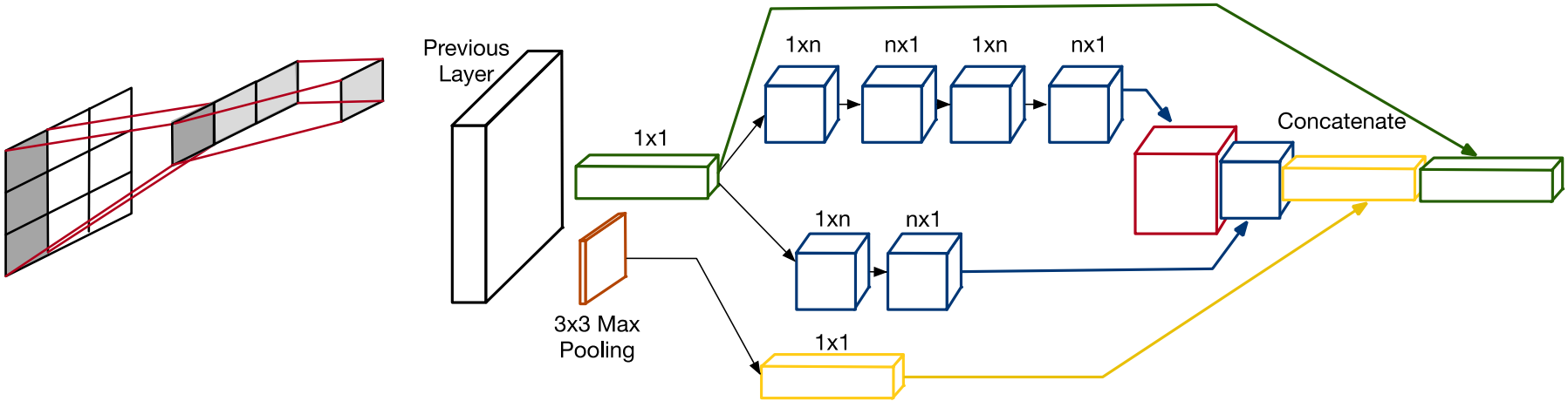






- Replace 5x5 convolution with two layers of 3x3 convolutions
- Reduces parameter count from 25 to 18
- “Note that we have factorized the traditional 7x7 convolution into three 3x3 convolutions…”

Spatial Factorization into Separable Kernels



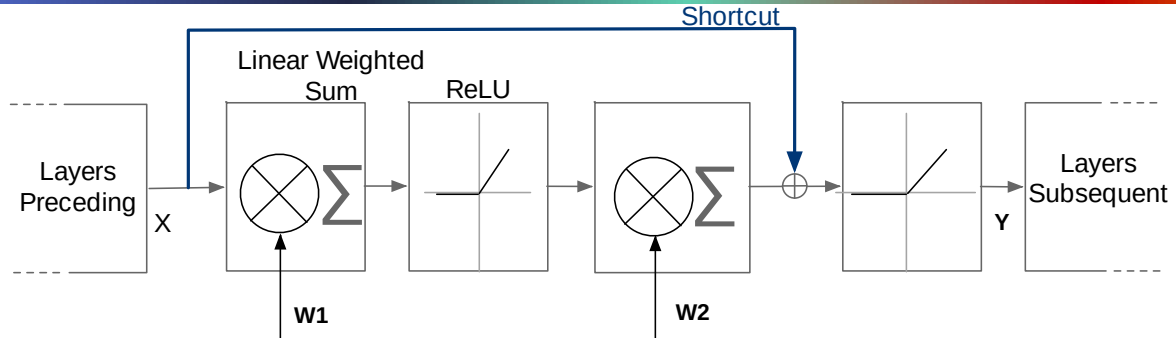
- Factored into two convolutions - a 3x1 convolution followed by a 1x3 convolution
  - Equivalent to sliding a two layer network with the same receptive field as in a 3X3 convolution
  - 33% cheaper (6 versus 9 parameters)
- In general, can be extended to replacing n x n convolutions

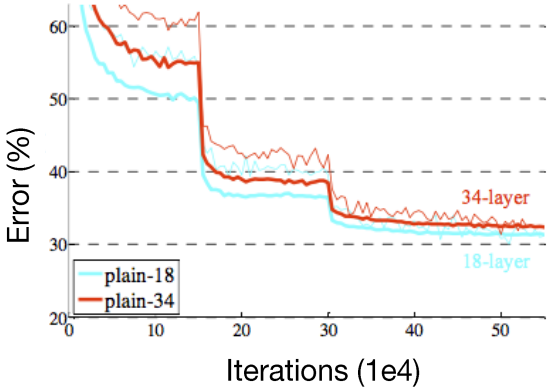
Resource for Inception Networks

- Paper
  - Going deeper with convolutions
    - Christian Szegedy et. Al.
- Code
  - https://github.com/keras-team/keras/tree/master/keras/applications

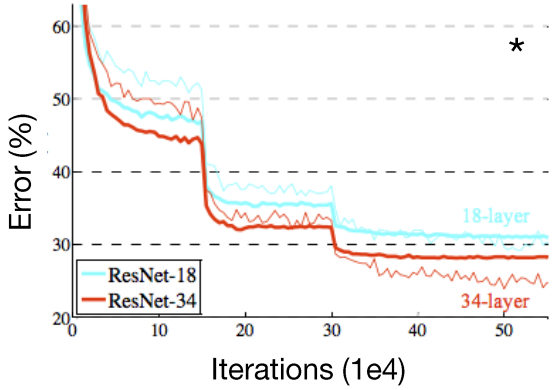
Residual Topologies for Simplicity and Uniformity

Shortcut Connections Lead to Accuracy Gains with Increased Depth





NO shortcut connection, purely stacked layers



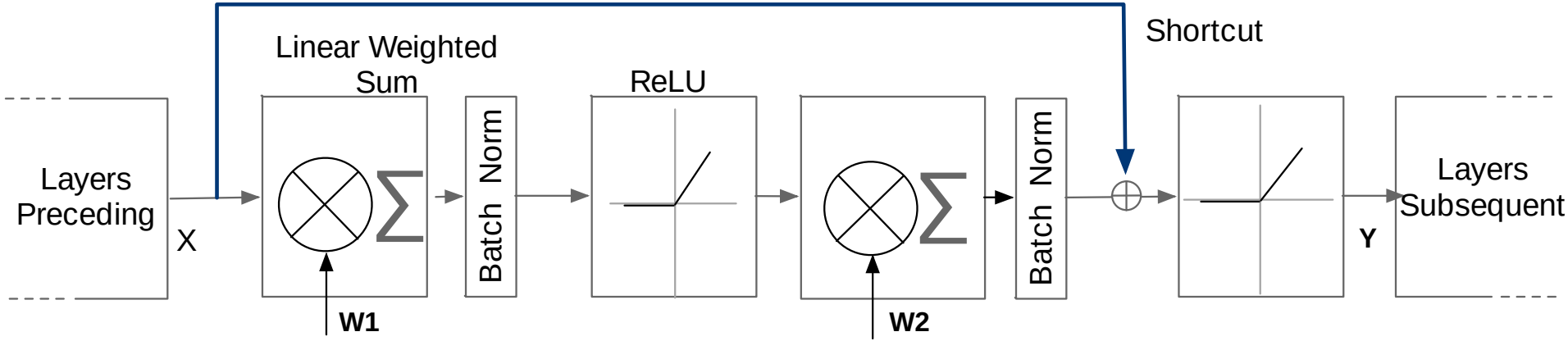
Shortcut connection added to each pair of 3x3 filters

Benefit of the Shortcut Connection

- 34-layer ResNet exhibits considerably lower training error
- More generalizable to the validation data.

*Deep Residual Learning for Image Recognition Kaiming He et. al.

Expressing the Residual Block in Keras

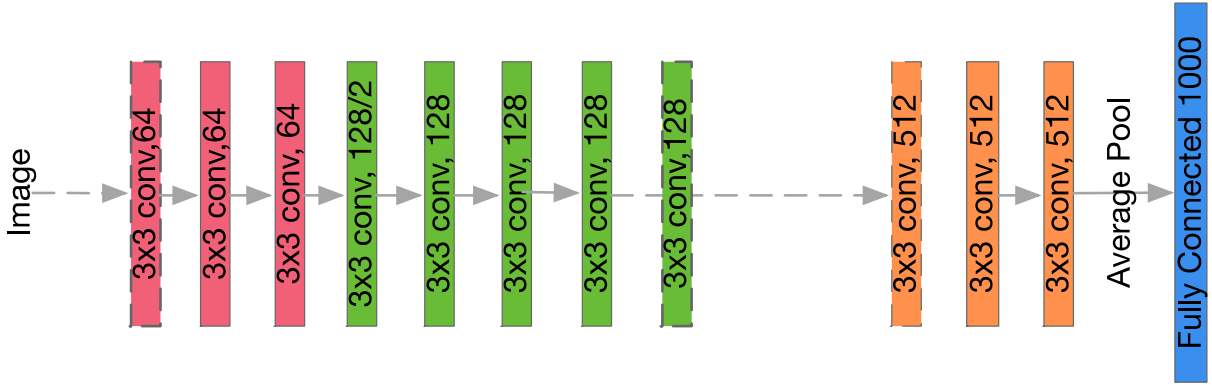



```
shortcut = x
y = Conv2D(nb_channels_in, kernel_size=(3, 3), strides=_strides, padding='same')(x)
y = BatchNormalization()(y)
y = Activation('relu')(y)
y = Conv2D(nb_channels_out, kernel_size=(3, 3), strides=(1, 1), padding='same')(y)
y = BatchNormalization()(y)
y = add([shortcut, y])
y = Activation('relu')(y)
```

High-level View of Residual Networks (ResNets)

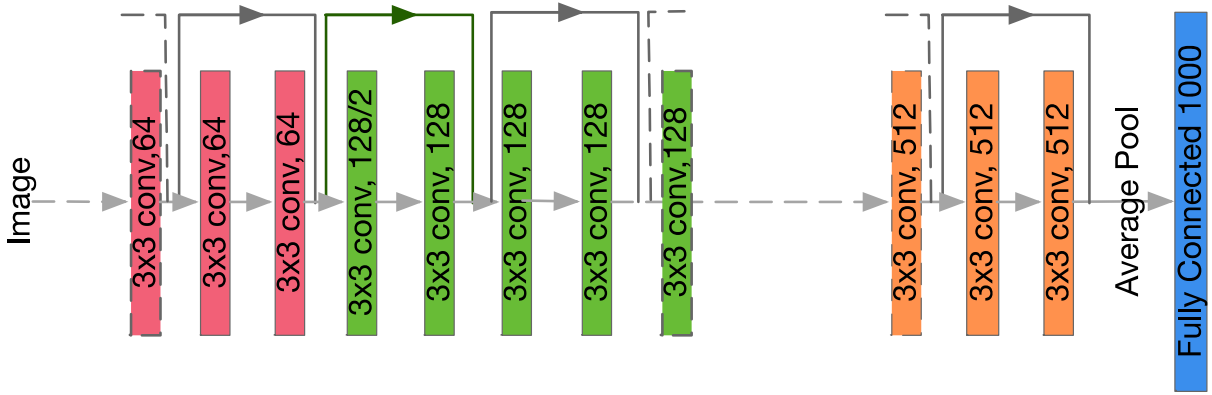
Pure Stacked Network:

Higher Error with Increased Depth



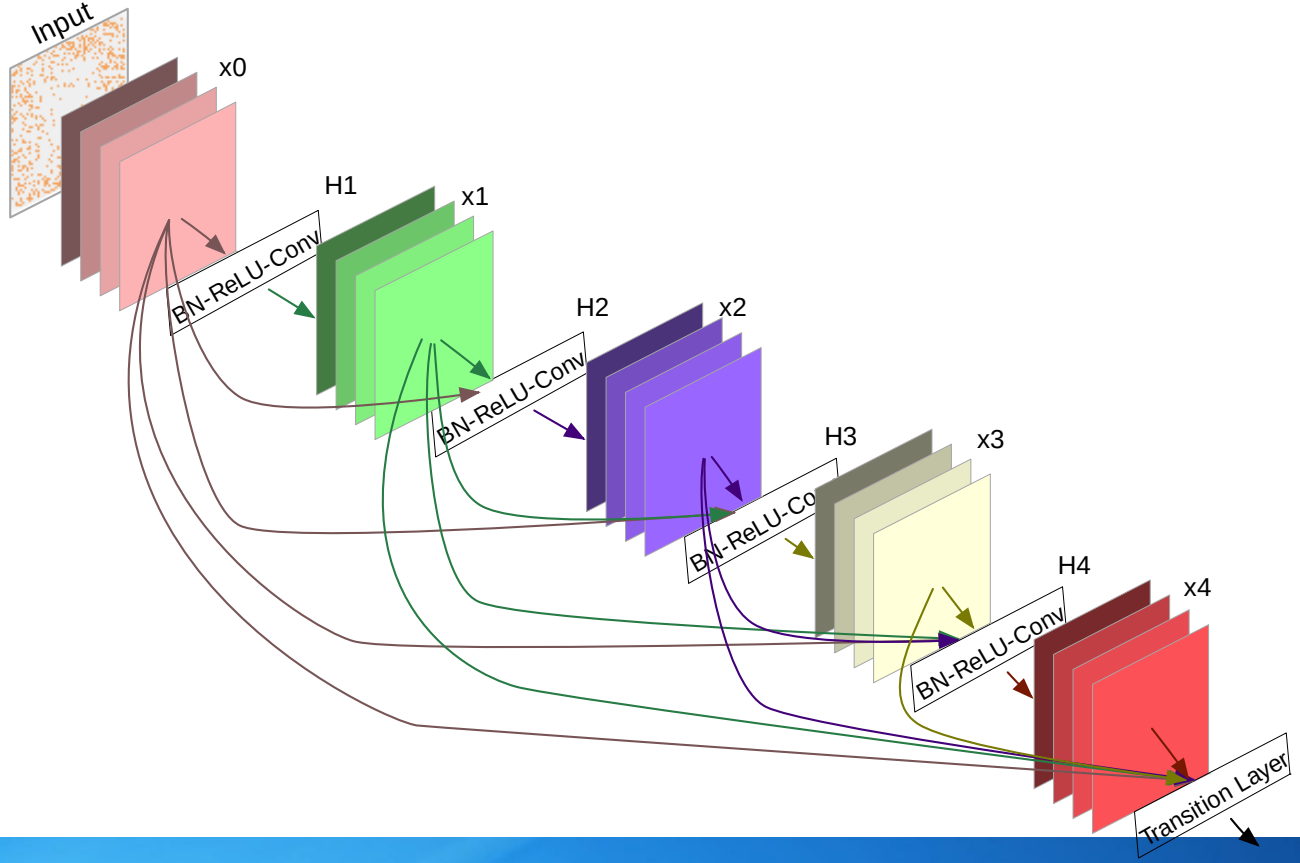
ResNet - Stacked + Shortcut Connections:

Lower Error with Increased Depth



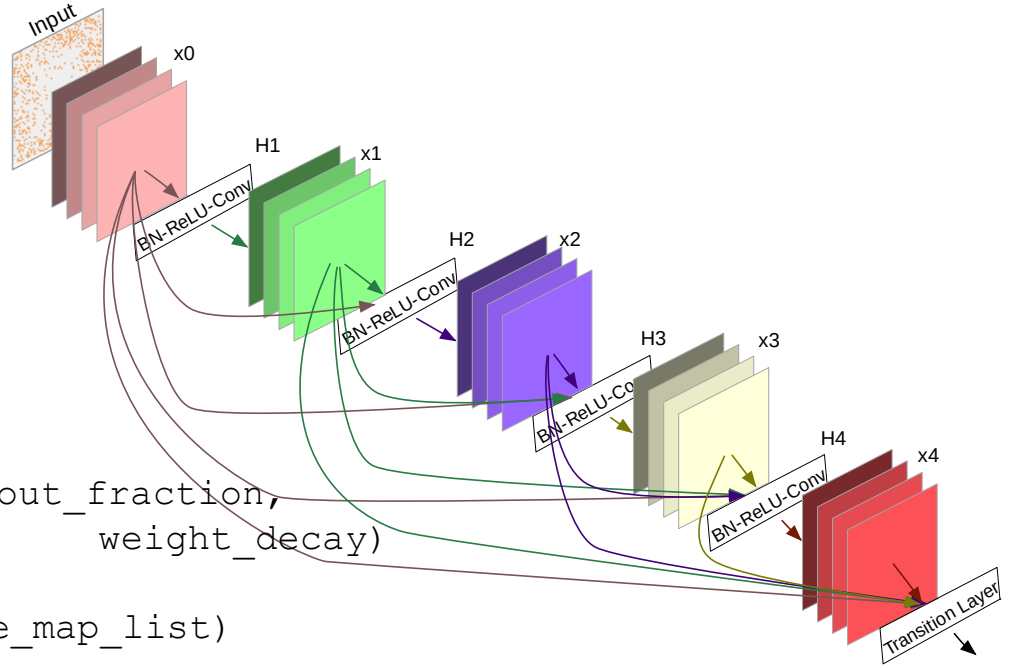
DenseNet:
Connects Each Layer to Every Other Forward Layer

- Each Layer takes all preceding feature-maps as input
- Like ResNet, however skip connections amplified
- Unlike ResNet, features are concatenated, not summed.
- Requires one-third the parameters of ResNet



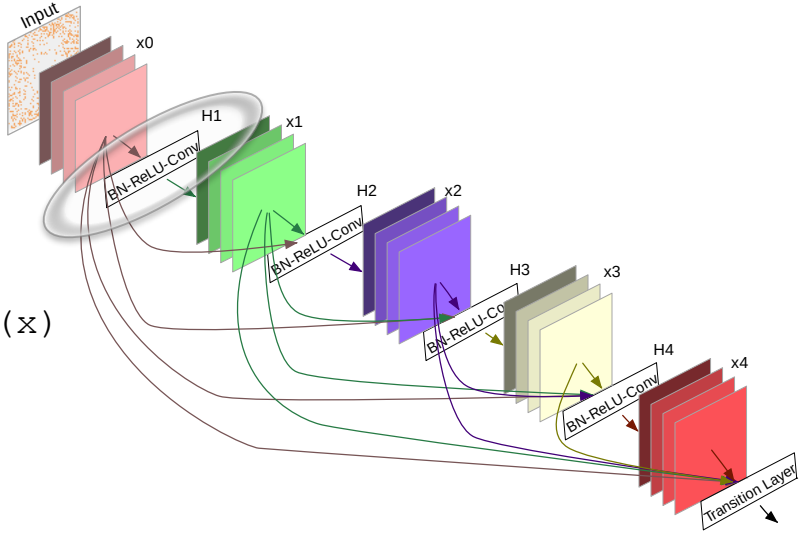
Expressing a Five-Layer Dense Block with a Growth-Rate of k=4

```
def dense_block(x,
                nb_filters,
                growth_rate,
                nb_conv_layers,
                dropout_fraction=None,
                weight_decay=1E-4):
    feature_map_list = [x]
    for i in range(nb_conv_layers):
        x = composite_bn_relu_conv(x,
                           growth_rate,
                                dropout_fraction,
                                    weight_decay)

        feature_map_list.append(x)
        x = Concatenate(axis=-1(feature_map_list)
        nb_filters += growth_rate
    return nb_filters, x
```

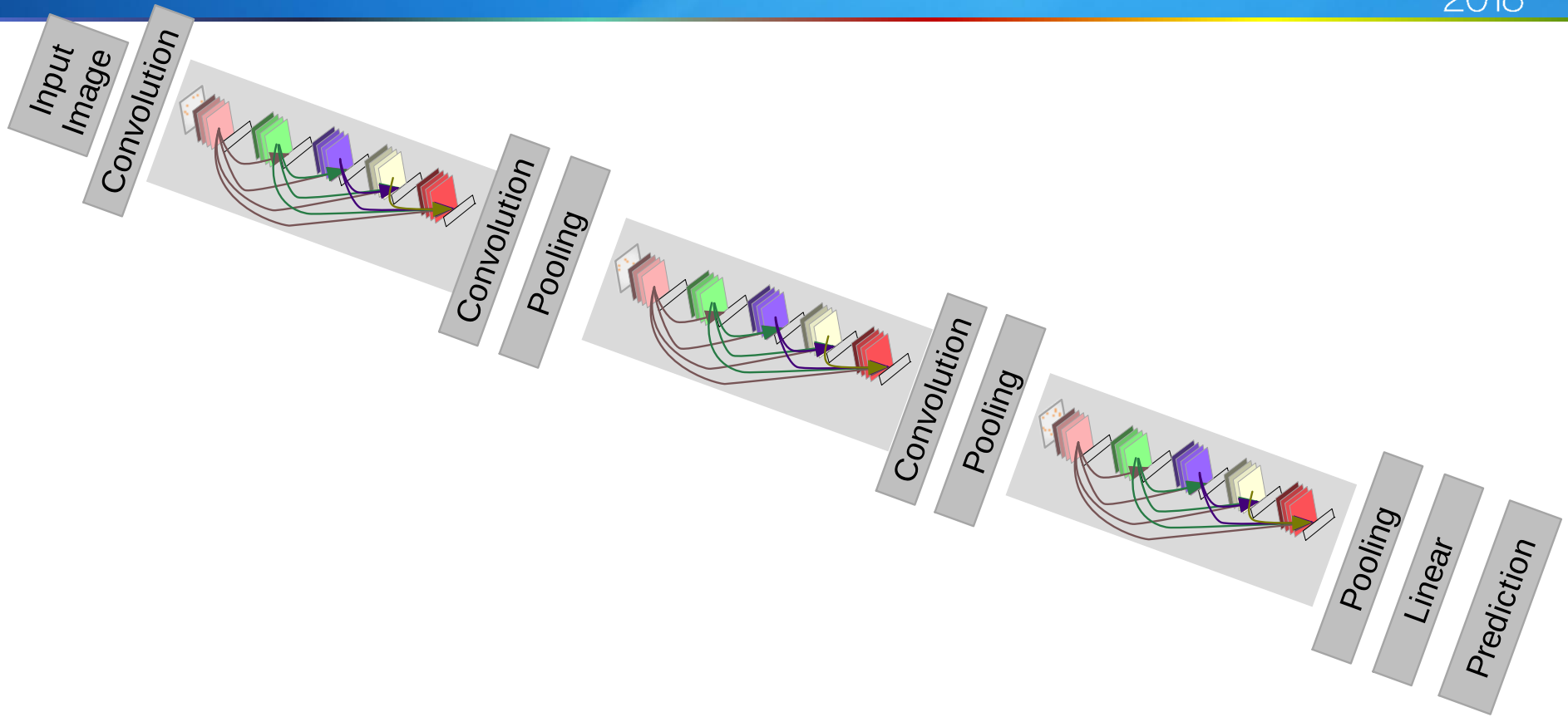


Expressing the Composite Function
Batch Norm, followed by ReLU, followed by 3X3 Conv

```
def composite_bn_relu_conv(x,
                           nb_filters,
                           dropout_fraction=None,
                           weight_decay=1E-4):
    x = BatchNormalization(axis=1,
              beta_regularizer=l2(weight_decay),
              gamma_regularizer=l2(weight_decay))(x)
    x = Activation('relu')(x)
    x = Convolution2D(nb_filters,
                      kernel_size=(3,3),
                      padding='same',
                      use_bias=False,
                      kernel_initializer='he_normal',
                      kernel_regularizer=l2(weight_decay))(x)
    if dropout_fraction:
        x = Dropout(dropout_fraction)(x)
    return x
```



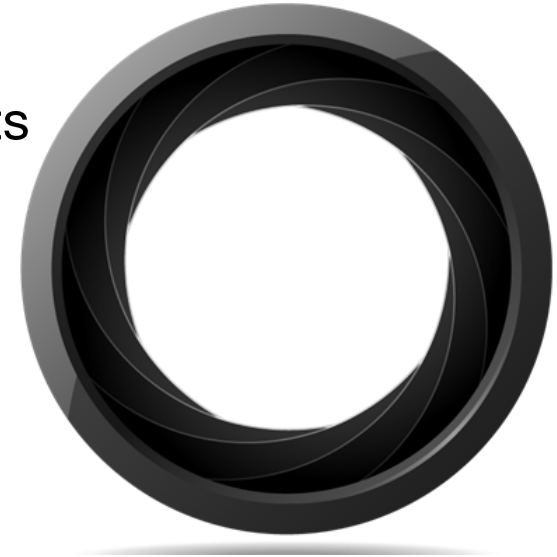
Putting Together DenseNet



Resources for Resnet and DenseNet

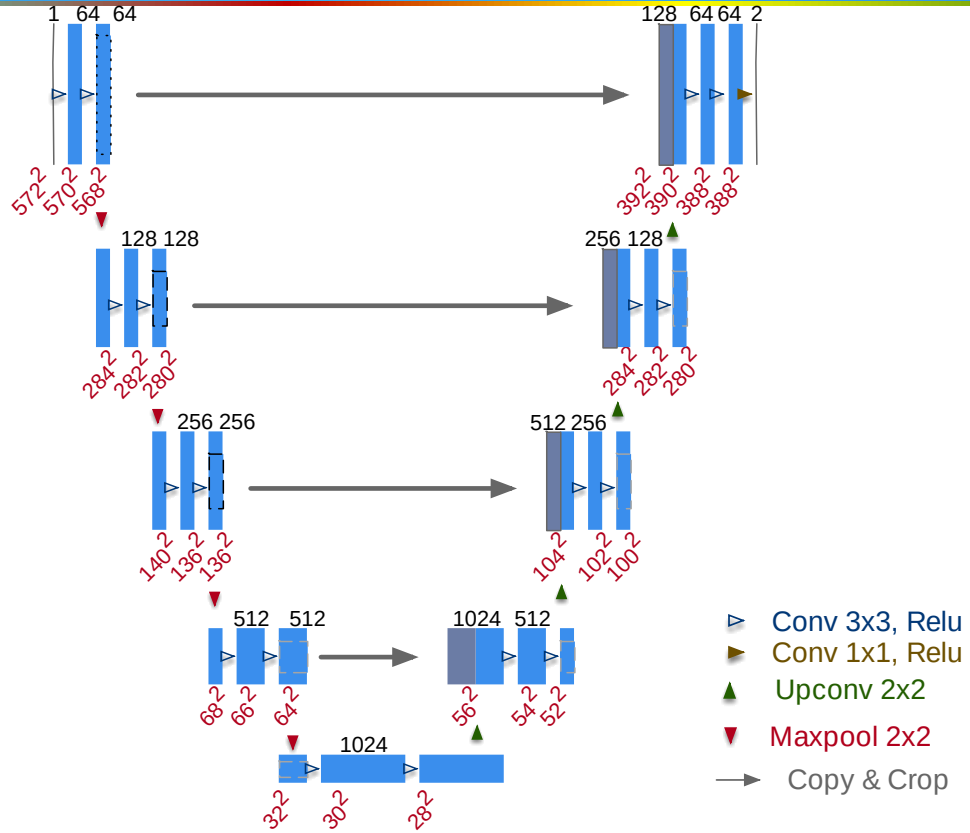
- Paper
  - Deep Residual Learning for Image Recognition
    - Kaiming He, Xiangyu Zhang, Shaoqing Ren, Jian Sun
  - Densely Connected Convolutional Networks
    - Gao Huang, Zhuang Liu, Laurens van der Maaten
- Code
  - https://github.com/keras-team/keras/blob/master/keras/applications/resnet50.py
  - https://github.com/flyyufelix/DenseNet-Keras

Encode/Decode Topologies for Segmenting Objects



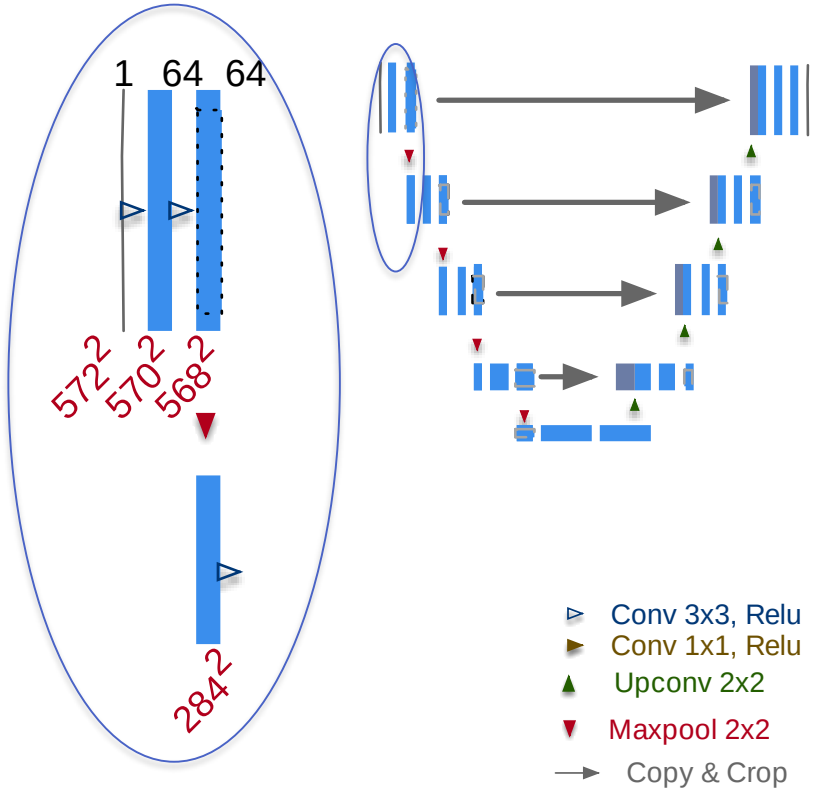
UNET: A Contracting Path Followed by Expanding Path for Segmenting Images

- Left side performs down-sampling (a.k.a. max pooling operations) lowering resolution
- Right side performs up-sampling operations increasing resolution ending up with the same resolution
- To segment, at each level the down-sampled features are combined with up-sampled output followed by a convolution.



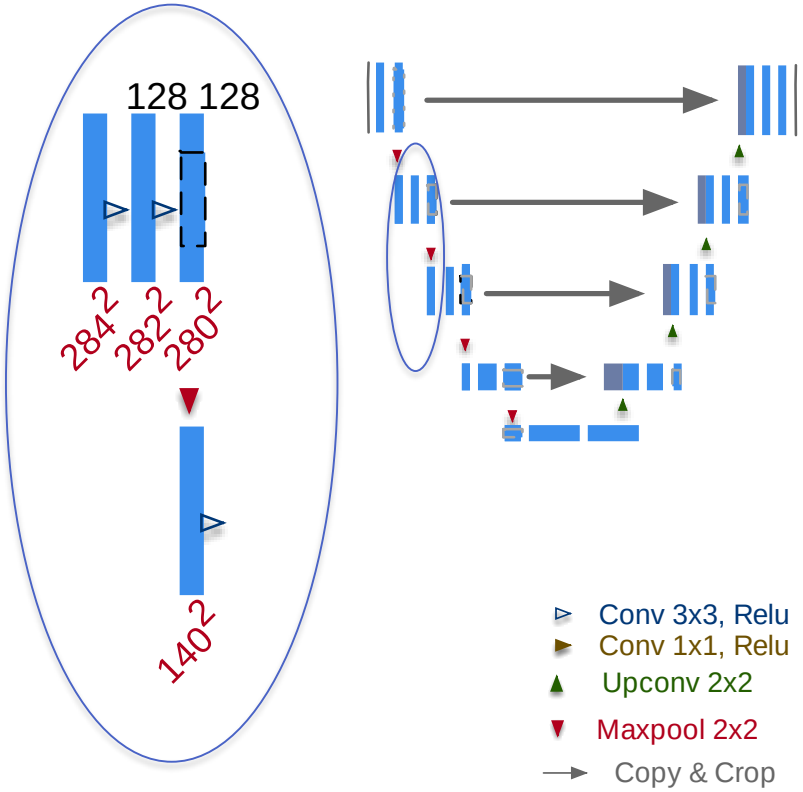
Left Side Convolutions and Pooling - One

```
conv1_L = Conv2D(64, (3, 3),
        activation='relu',
        padding='same')(inputs)

conv1_L = Conv2D(64, (3, 3),
        activation='relu',
        padding='same')(conv1_L)

pool1 = MaxPooling2D(
        pool_size=(2, 2))(conv1_L)
```



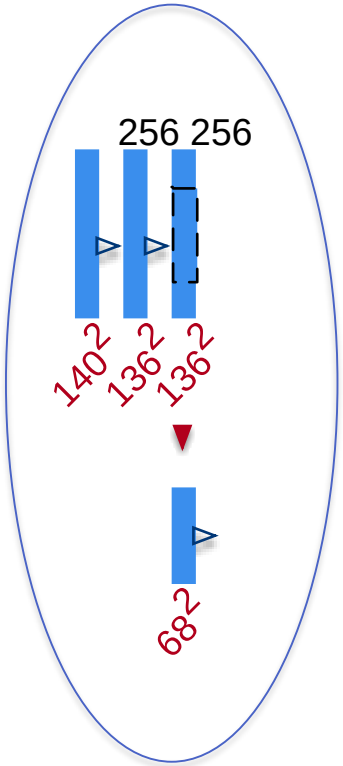
Left Side Convolutions and Pooling - Two

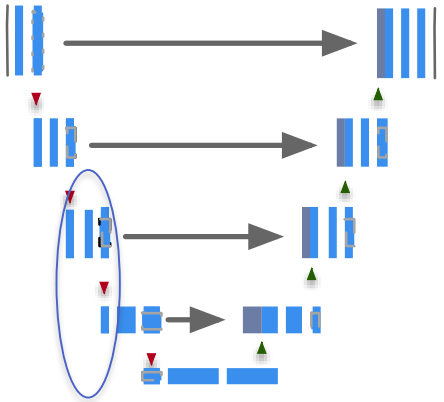
```
conv2_L = Conv2D(128, (3, 3),
        activation='relu',
        padding='same')(pool1)

conv2_L = Conv2D(128, (3, 3),
        activation='relu',
        padding='same')(conv2_L)

pool2 = MaxPooling2D(
        pool_size=(2, 2))(conv2_L)
```



Left Side Convolutions and Pooling - Three

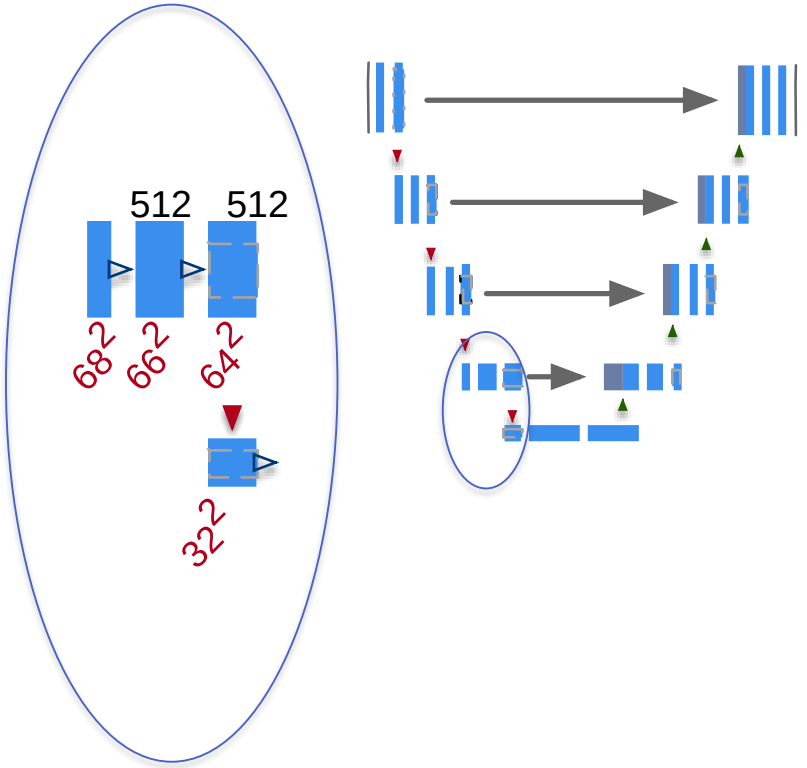
```
conv3_L = Conv2D(256, (3, 3),
        activation='relu',
        padding='same')(pool2)

conv3_L = Conv2D(256, (3, 3),
        activation='relu',
        padding='same')(conv3_L)

pool3 = MaxPooling2D(
        pool_size=(2, 2))(conv3_L)
```





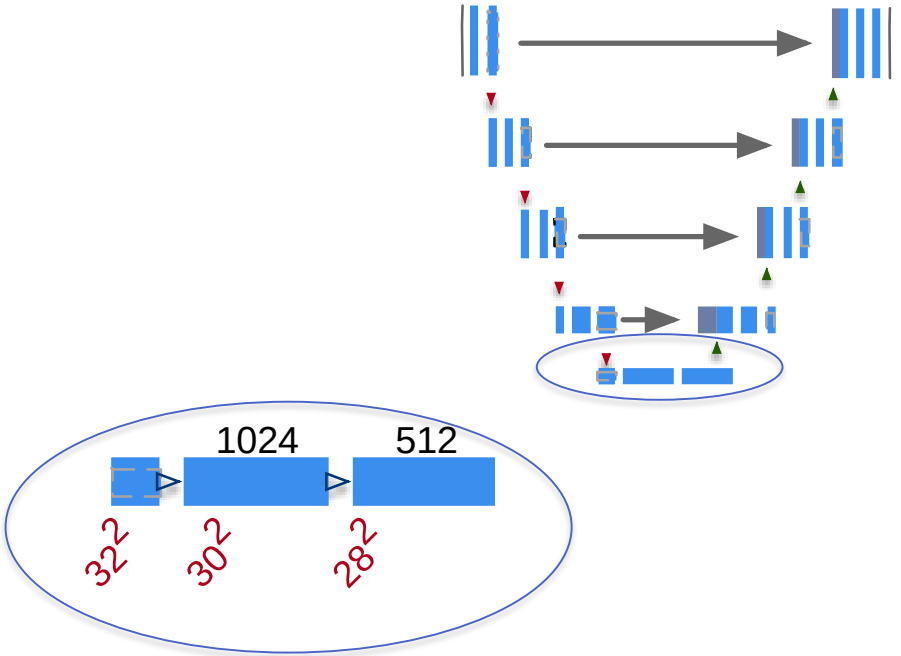
Left Side Convolutions and Pooling - Four

```
conv4_L = Conv2D(512, (3, 3),
        activation='relu',
        padding='same')(pool3)

conv4_L = Conv2D(512, (3, 3),
        activation='relu',
        padding='same')(conv4_L)

pool4 = MaxPooling2D(
        pool_size=(2, 2))(conv4_L)
```



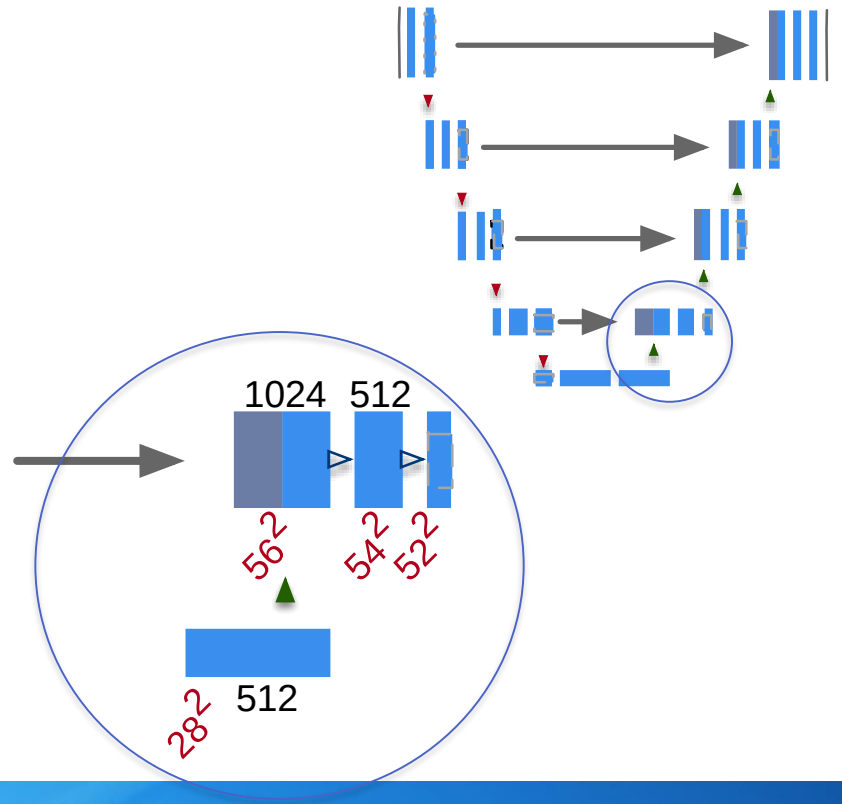
Bottom Convolutions

```
Conv5_B = Conv2D(1024, (3, 3),
        activation='relu',
        padding='same')(pool4)

Conv5_B = Conv2D(512, (3, 3),
        activation='relu',
        padding='same')(conv5_B)
```



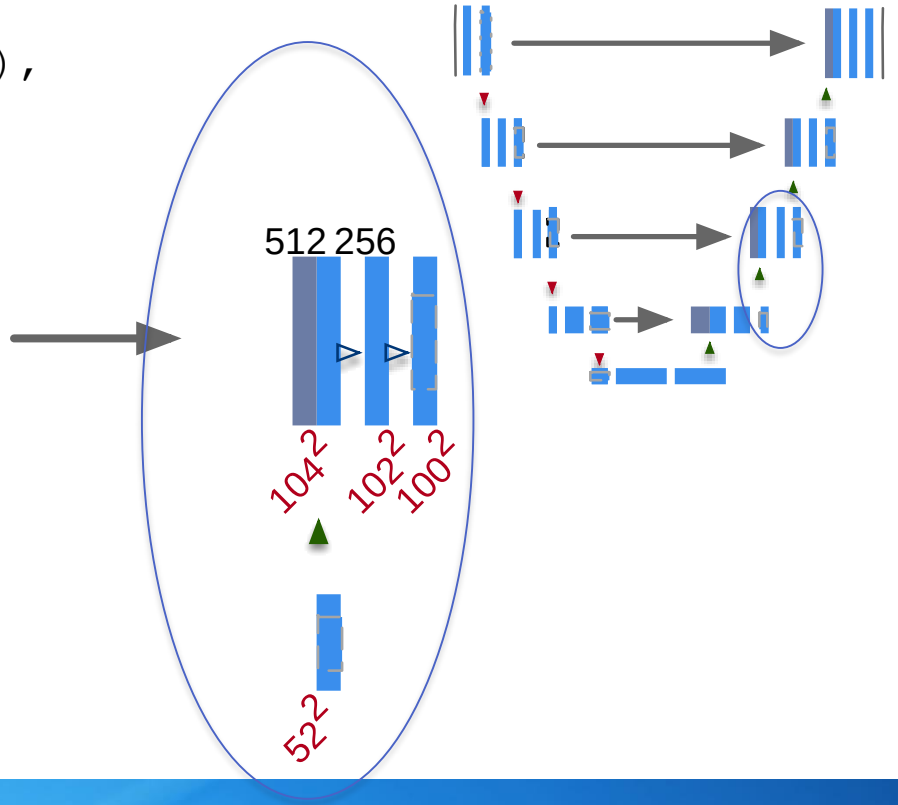
Concatenation, Up Sampling, and Convolutions - One

```
up1 = concatenate(
        [Conv2DTranspose(512, (2, 2),
        strides=(2, 2),
        padding='same')(conv5_B),
conv4_L],
        axis=3)

conv3_R = Conv2D(256, (3, 3),
        activation='relu',
        padding='same')(up1)

conv3_R = Conv2D(256, (3, 3),
        activation='relu',
        padding='same')(conv3_R)
```



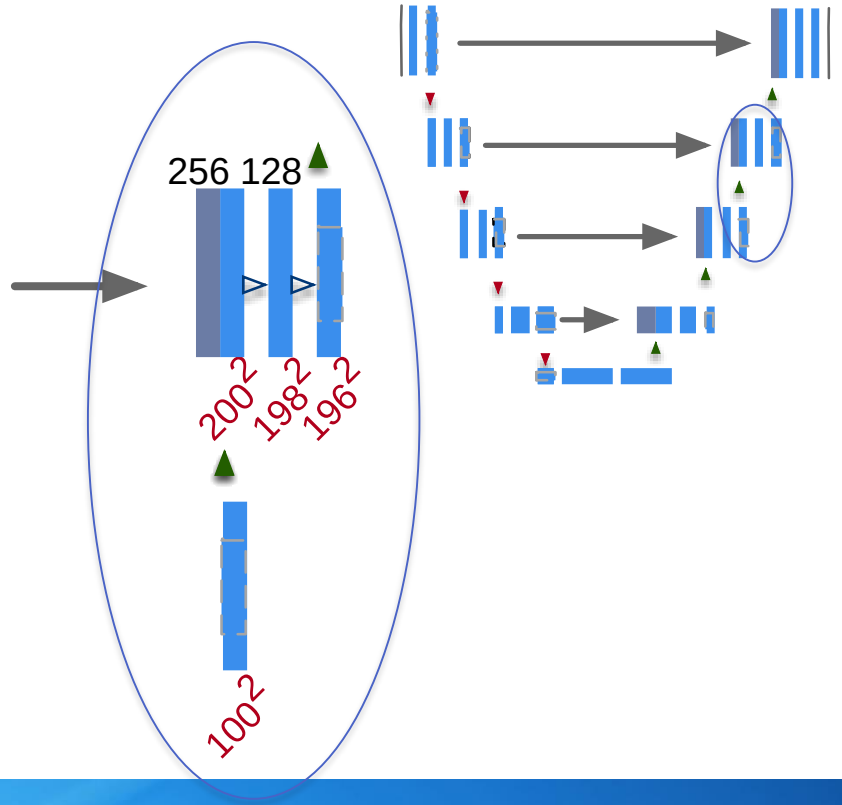
Concatenation, Up Sampling, and Convolutions - Two

```
up2 = concatenate(
        [Conv2DTranspose(512, (2, 2),
        strides=(2, 2),
        padding='same')(conv3_L),
conv3_L],
        axis=3)

conv2_R = Conv2D(256, (3, 3),
        activation='relu',
        padding='same')(up2)

conv2_R = Conv2D(256, (3, 3),
        activation='relu',
        padding='same')(conv2_R)
```



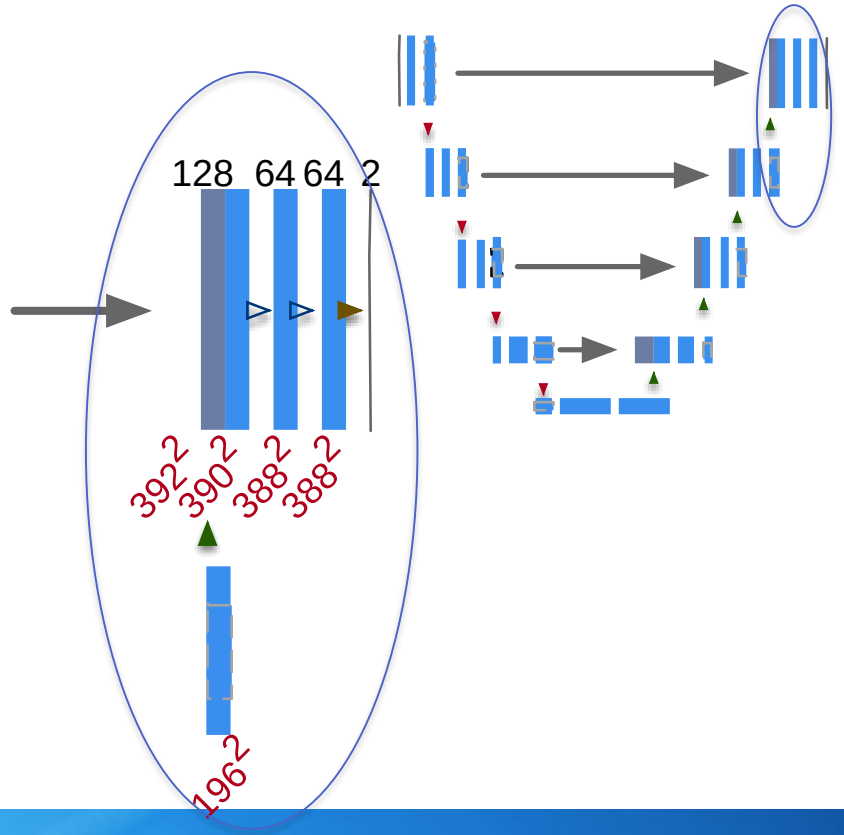
Concatenation, Up Sampling, and Convolutions - Three

```
up3 = concatenate(
        [Conv2DTranspose(128, (2, 2),
        strides=(2, 2),
        padding='same')(conv2_R),
conv2_L],
        axis=3)

conv1_R = Conv2D(128, (3, 3),
        activation='relu',
        padding='same')(up3)

conv1_R = Conv2D(128, (3, 3),
        activation='relu',
        padding='same')(conv1_R)
```



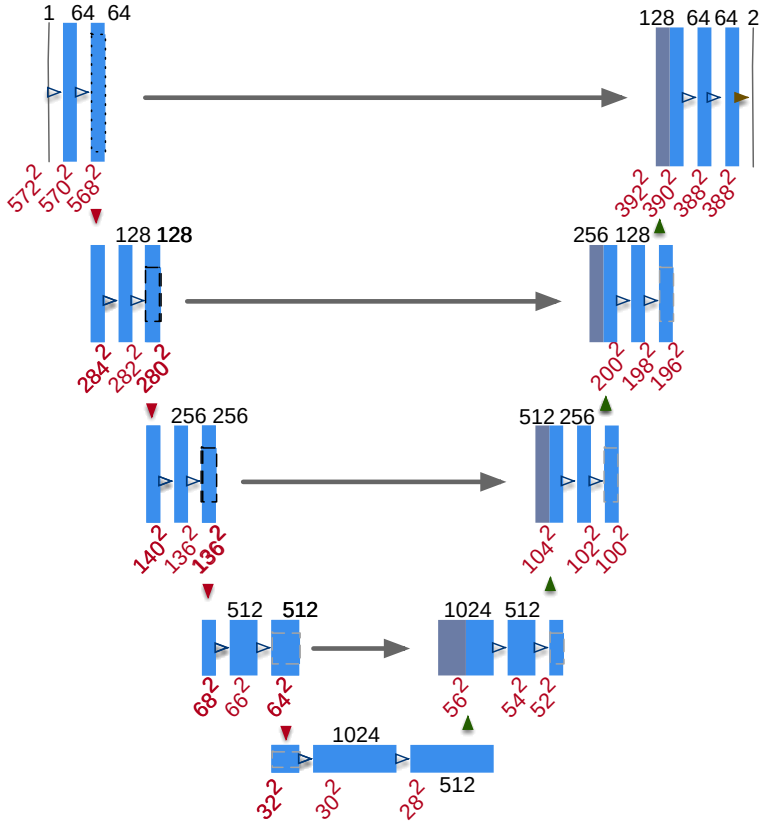
Concatenation, Up Sampling, and Convolutions - Four

```
up4 = concatenate(
        [Conv2DTranspose(64, (2, 2),
        strides=(2, 2),
        padding='same')(conv1_R),
conv1_L],
        axis=3)

conv_out = Conv2D(64, (3, 3),
        activation='relu',
        padding='same')(up3)

conv_out = Conv2D(64, (3, 3),
        activation='relu',
        padding='same')(conv_out)
```



Final Segmentation

```
conv_out_final = Conv2D(1, (1, 1),
        activation='sigmoid')(conv_out)

model = Model(inputs=[inputs],
        outputs=[conv_out_final])

model.compile(optimizer=Adam(lr=1e-5),
        loss=dice_coef_loss,
        metrics=[dice_coef])
```



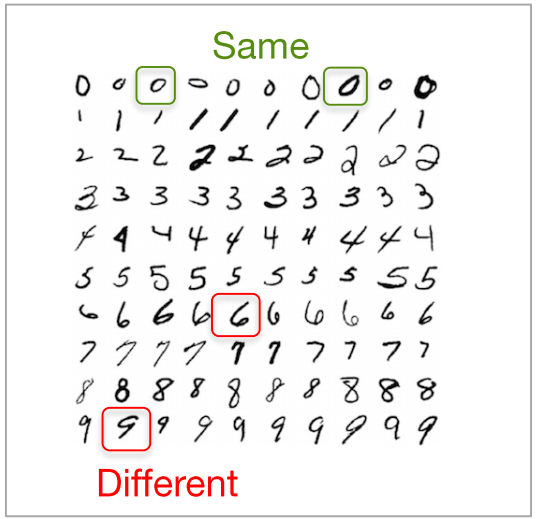
Resources for Unet

- Paper
  - U-Net: Convolutional Networks for Biomedical Image Segmentation
    - Olaf Ronneberger, Philipp Fischer, and Thomas Brox
- Code
  - https://github.com/zhixuhao/unet

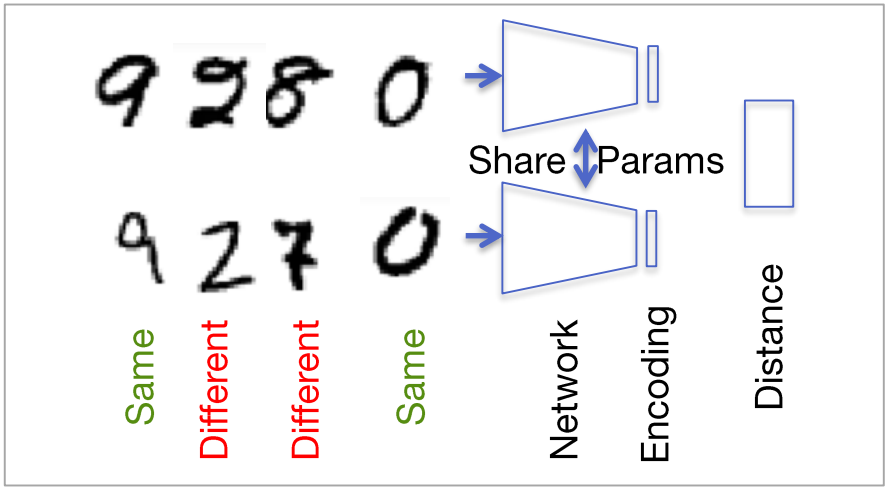
Siamese CNN for Learning Similarities

Problem Statement
Decide Whether Pairs of Images are Similar/Different

Problem



Solution



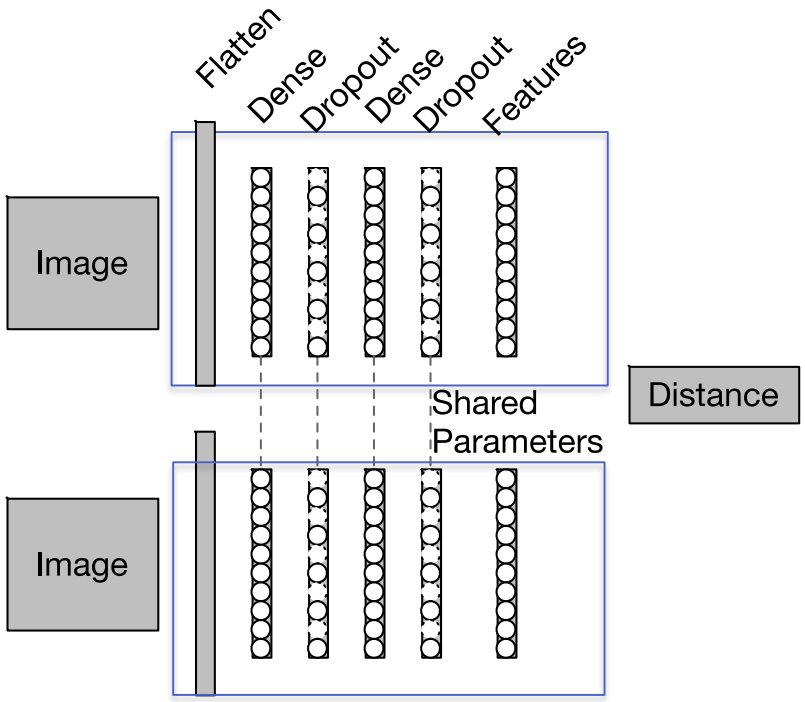
1. Generate image pairs
    1. For same digits
    2. For different digits

2. Learn parameters such that :
    2. For same digits, the distance is small
    3. For different digits, the difference is large
3. Can be done by applying gradient descent on
    2. Triplet Loss, or
    3. Contrastive Loss
4. Uses-cases include (1) Signature verification (2) Face Verification

Siamese Base Network – Identical Branches

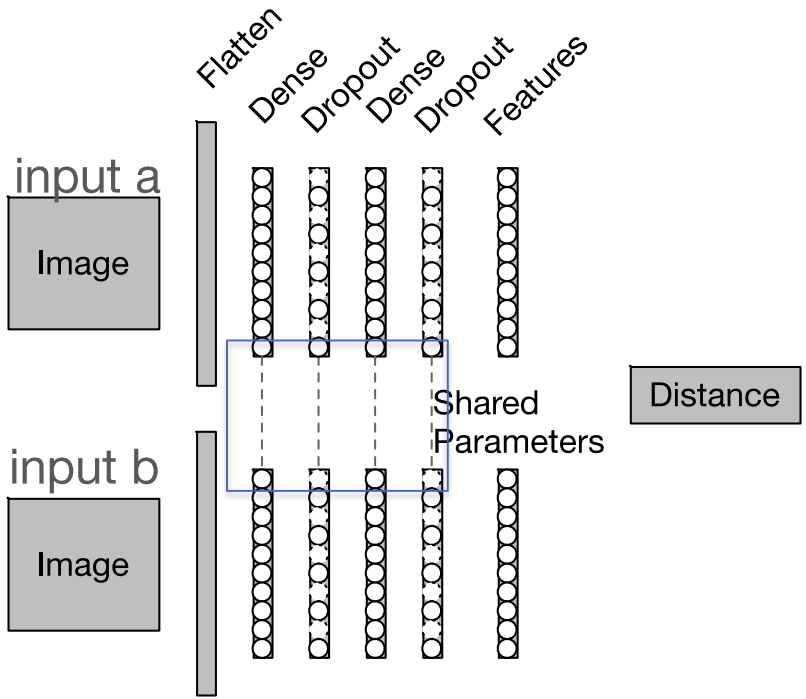
Code for each branch

```
input = Input(shape=input_shape)
x = Flatten()(input)
x = Dense(128, activation='relu')(x)
x = Dropout(0.1)(x)
x = Dense(128, activation='relu')(x)
x = Dropout(0.1)(x)
x = Dense(128, activation='relu')(x)
```



Share Parameters Across the Two Branches

```
siamese_a = base_network(input_a)
siamese_b = base_network(input_b)

distance =
        Lambda(euclidean_distance,
                    output_shape=
        eucl_dist_output_shape)
        ([siamese_a, siamese_b])

model = Model([input_a, input_b],
distance)
```



Resources for Siamese Networks

- **Papers**
  - DeepFace: Closing the Gap to Human-Level Performance in Face Verification
    - Yaniv Taigman, Ming Yang, Marc'Aurelio Ranzato, Lior Wolf
  - FaceNet: A Unified Embedding for Face Recognition and Clustering
    - Florian Schroff, Dmitry Kalenichenko, James Philbin
- **Code**
  - https://github.com/keras-team/keras/blob/master/examples/mnist_siamese.py

Summary

- The shift from better features to better network topologies
- We reviewed four:
  - Split-transform-merge connections for reduced parameters
  - Residual connections for higher accuracy thru depth
  - Encode/Decode networks for segmentation
  - Siamese networks for learning similarity