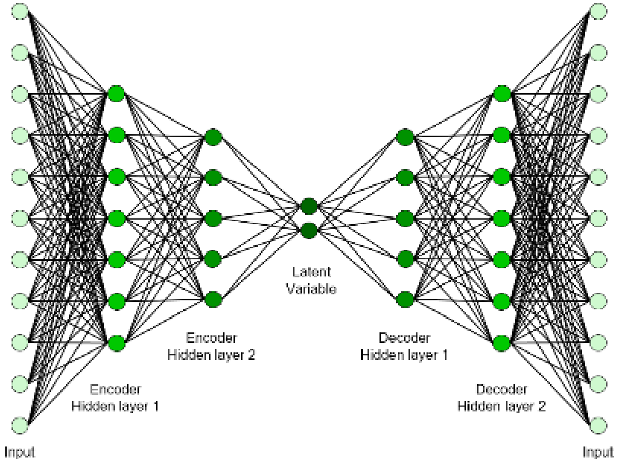
Convolutional Autoencoder

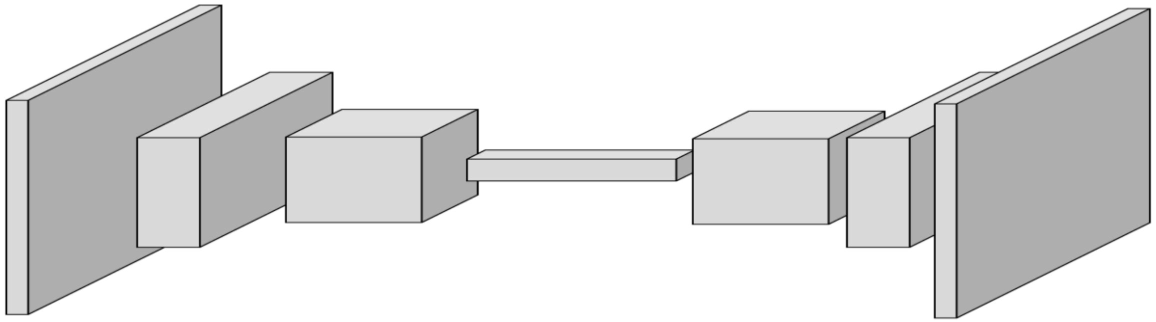
Industrial AI Lab.

Prof. Seungchul Lee

Convolutional Autoencoder

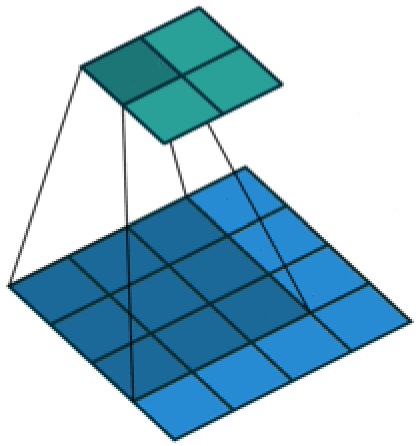
- Motivation: image to autoencoder ?
- Convolutional autoencoder extends the basic structure of the simple autoencoder by changing the fully connected layers to convolution layers.
  - the network of encoder change to convolution layers
  - the network of decoder change to transposed convolutional layers
    - A transposed 2-D convolution layer upsamples feature maps.
    - This layer is sometimes incorrectly known as a "deconvolution" or "deconv" layer.
    - This layer is the transpose of convolution and does not perform deconvolution.





tf.keras.models.Conv2D

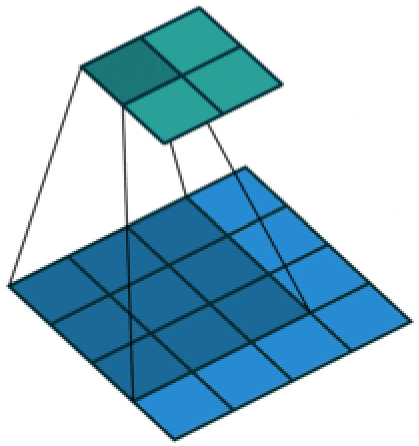
- Encoder
- Padding



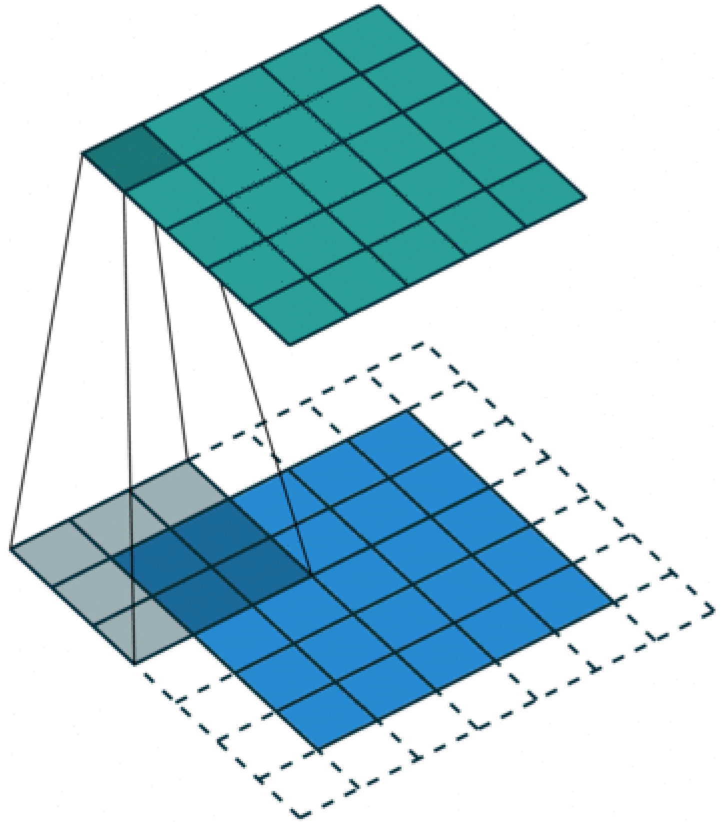
padding = 'VALID'
strides = [1, 1, 1, 1]

tf.keras.models.Conv2D

- Encoder
- Padding



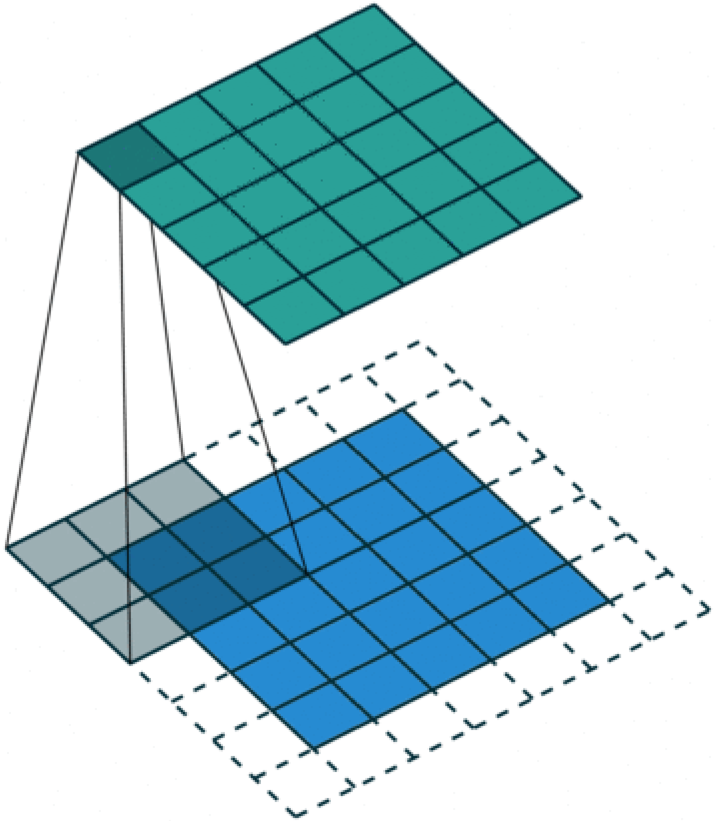
padding = ‘VALID’
strides = [1, 1, 1, 1]



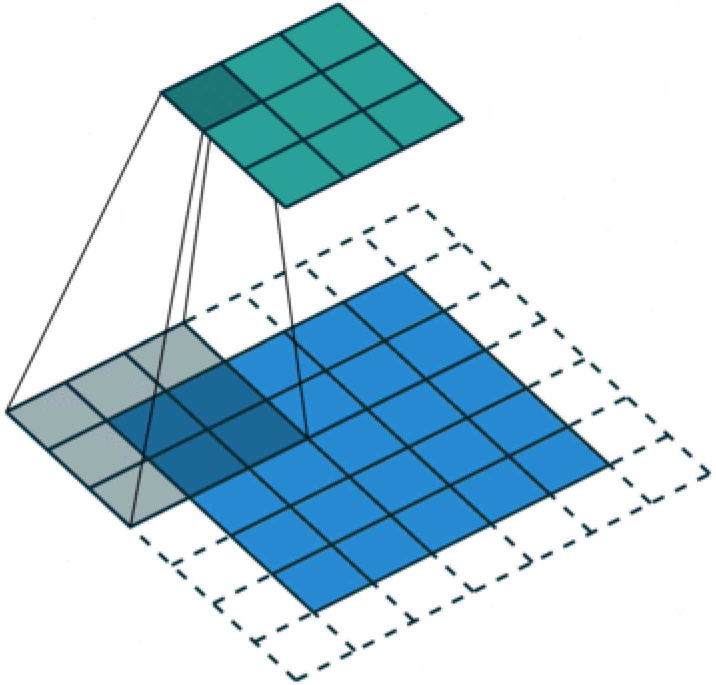
padding = ‘SAME’
strides = [1, 1, 1, 1]

tf.keras.models.Conv2D

- Encoder
- Stride



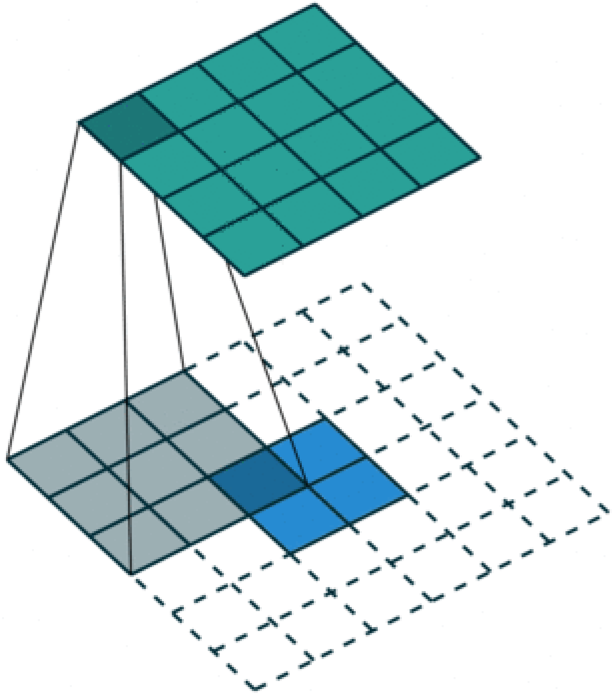
padding = 'SAME'
strides = [1, 1, 1, 1]



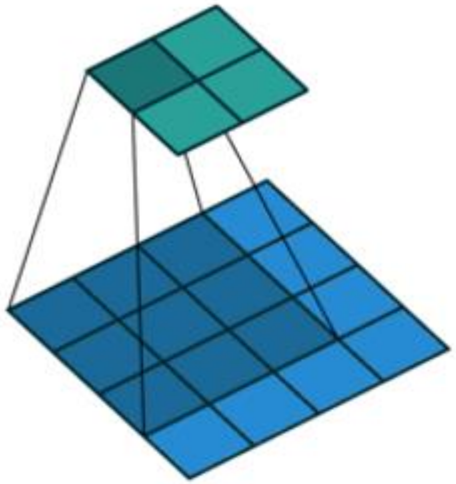
padding = 'SAME'
strides = [1, 2, 2, 1]

tf.keras.models.Conv2DTranspose

- Decoder
- Stride



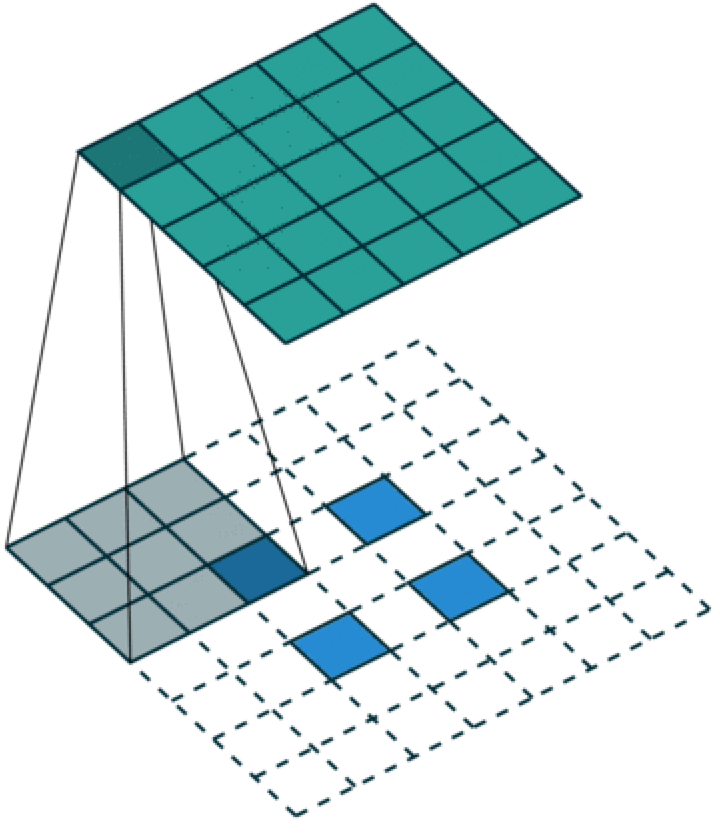
padding = 'VALID'
strides = (1,1)



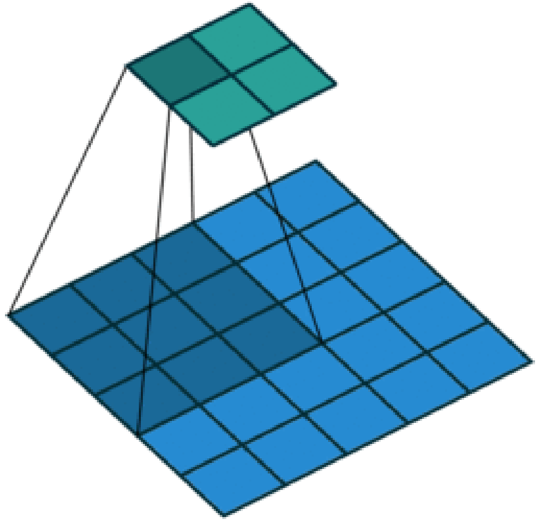
padding = 'VALID'
strides = (1,1)

tf.keras.models.Conv2DTranspose

- Decoder
- Stride



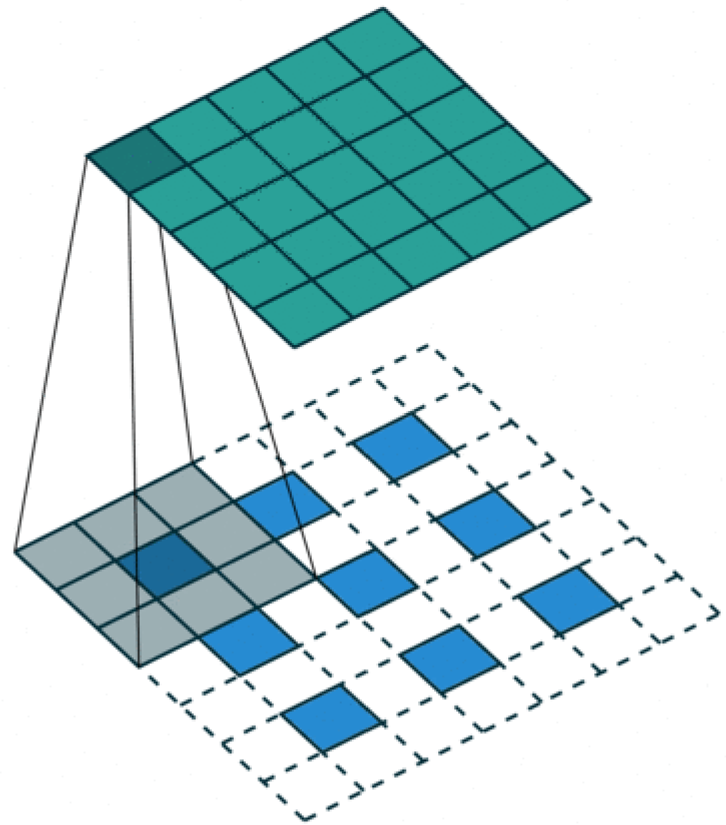
padding = ‘VALID’
strides = (2,2)



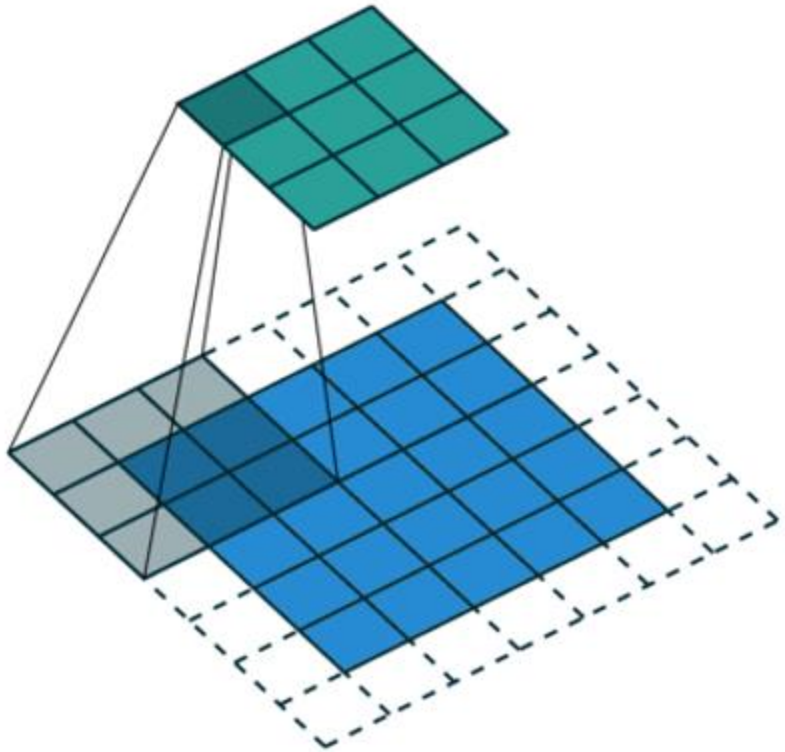
padding = ‘VALID’
strides = (2,2)

tf.keras.models.Conv2DTranspose

- Decoder
- Stride



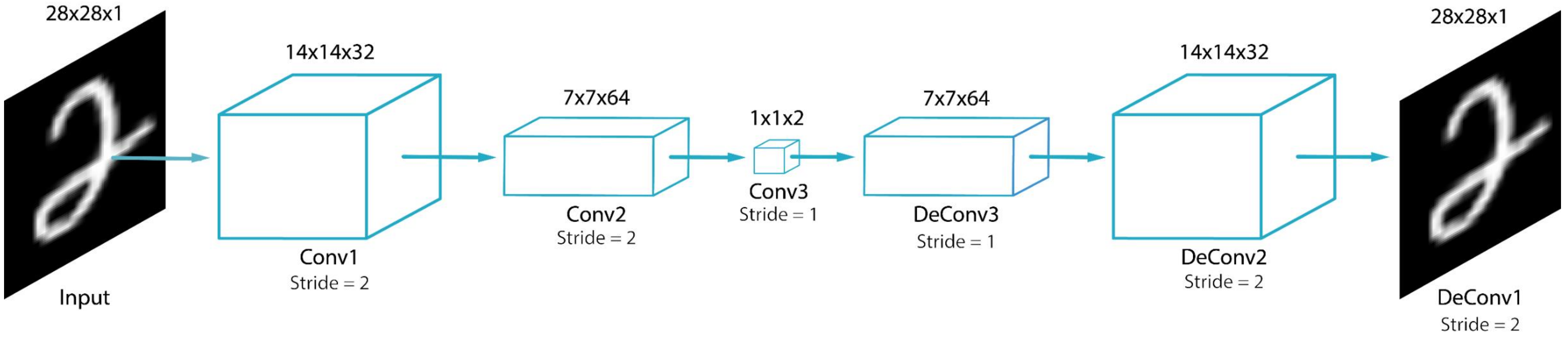
padding = 'SAME'
strides = (2,2)



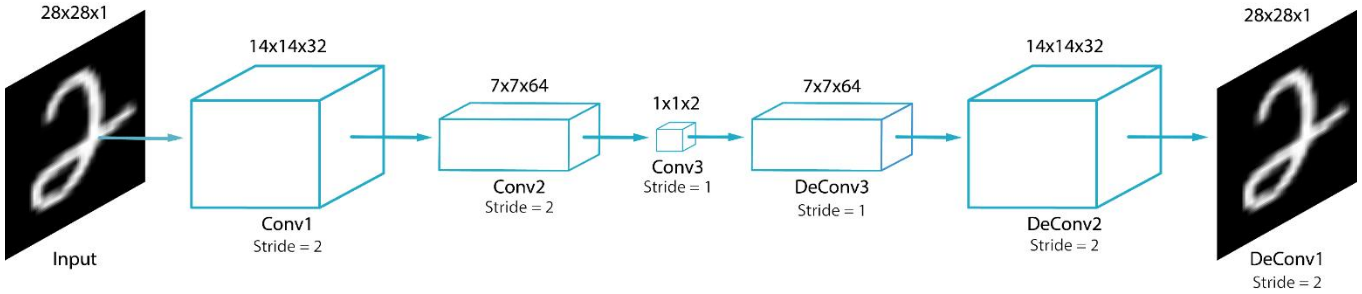
padding = 'SAME'
strides = (2,2)

CAE Implementation

- Fully convolutional
- Note that no dense layer is used

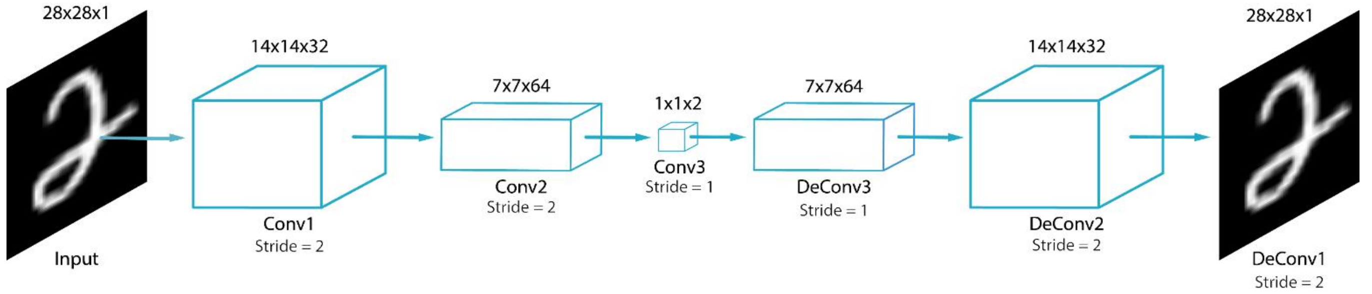


CAE Implementation



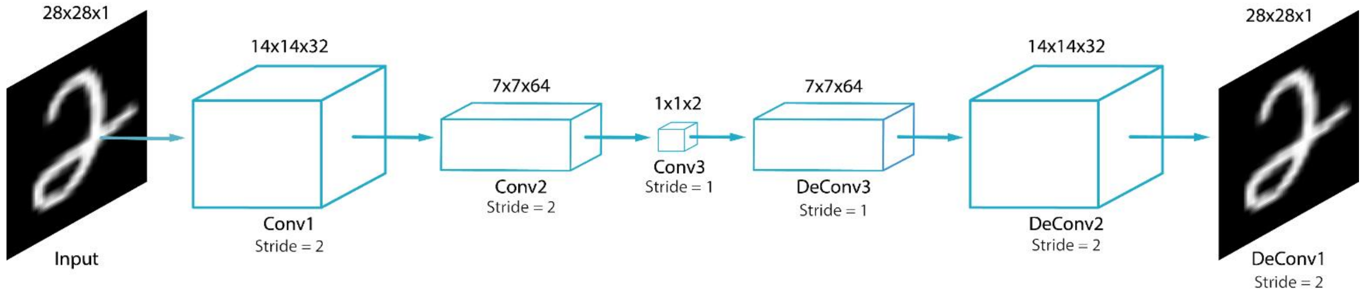

```
encoder = tf.keras.models.Sequential([
    tf.keras.layers.Conv2D(filters = 32,
                           kernel_size = (3,3),
                           strides = (2,2),
                           activation = 'relu',
                           padding = 'SAME',
                           input_shape = (28, 28, 1)),

    tf.keras.layers.Conv2D(filters = 64,
                           kernel_size = (3,3),
                           strides = (2,2),
                           activation = 'relu',
                           padding = 'SAME',
                           input_shape = (14, 14, 32)),

    tf.keras.layers.Conv2D(filters = 2,
                           kernel_size = (7,7),
                           padding = 'VALID',
                           input_shape = (7,7,64))
])
```

CAE Implementation




```python
decoder = tf.keras.models.Sequential([
    tf.keras.layers.Conv2DTranspose(filters = 64,
                                    kernel_size = (7,7),
                                    strides = (1,1),
                                    activation = 'relu',
                                    padding = 'VALID',
                                    input_shape = (1, 1, 2)),

    tf.keras.layers.Conv2DTranspose(filters = 32,
                                    kernel_size = (3,3),
                                    strides = (2,2),
                                    activation = 'relu',
                                    padding = 'SAME',
                                    input_shape = (7, 7, 64)),

    tf.keras.layers.Conv2DTranspose(filters = 1,
                                    kernel_size = (7,7),
                                    strides = (2,2),
                                    padding = 'SAME',
                                    input_shape = (14,14,32))
])
```

CAE Implementation

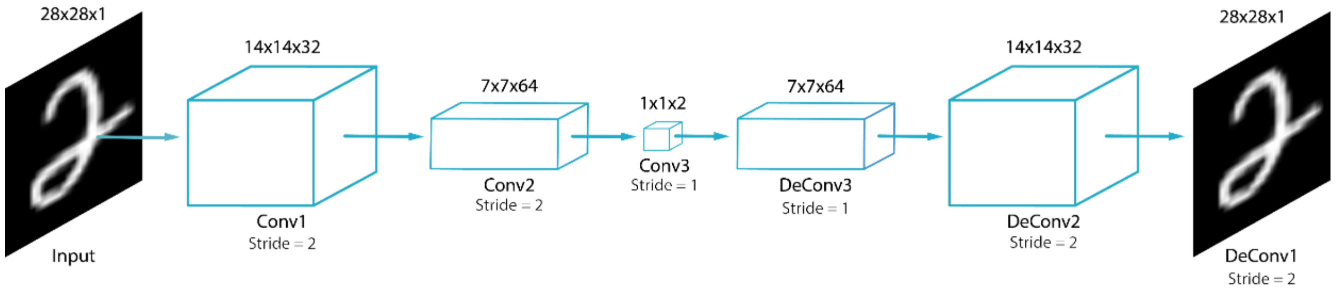



```
latent = encoder.output
result = decoder(latent)

model = tf.keras.Model(inputs = encoder.input,
                       outputs = result)
```

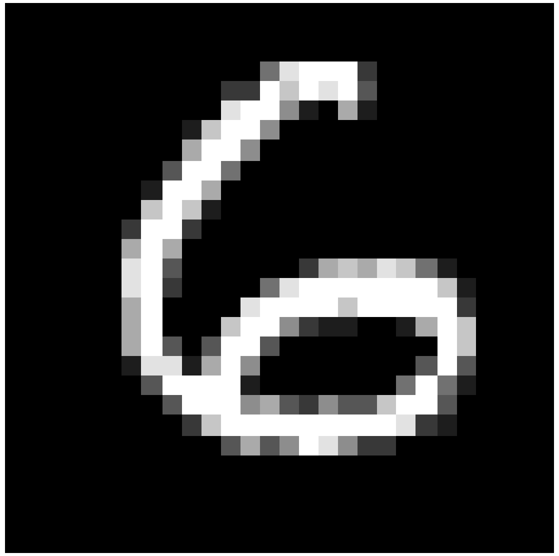
```
model.compile(optimizer = 'adam',
              loss = 'mean_squared_error')
```

```
model.fit(train_x, train_x, epochs = 10)
```

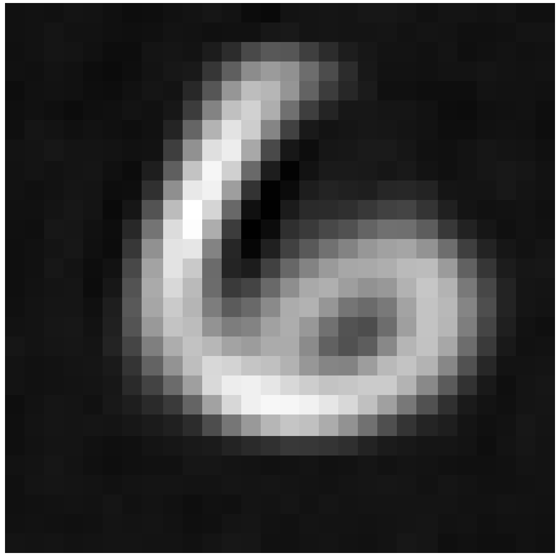
Reconstruction Result



Input image



Reconstructed image



Fully Convolutional Network (FCN)

Industrial AI Lab.

Prof. Seungchul Lee

POSTECH

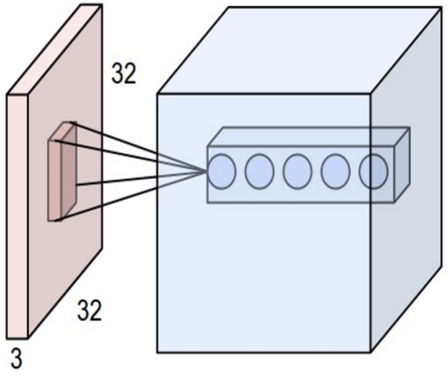
Deep Learning for Computer Vision: Review

Foundations

- Why computer vision?
- Representing images
- Convolutions for feature extraction

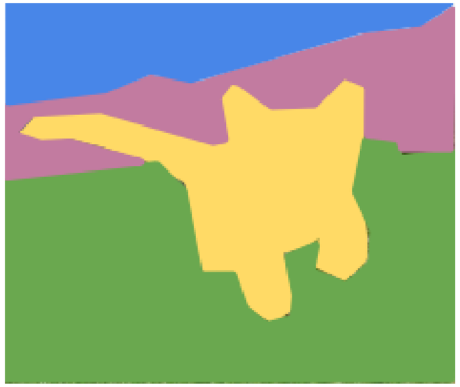
CNNs

- CNN architecture
- Application to classification: ImageNet



Applications

- Segmentation, object detection, image captioning
- Visualization



Segmentation

- Segmentation task is different from classification task because it requires predicting a class for each pixel of the input image, instead of only 1 class for the whole input.
- Segment images into regions with different semantic categories. These semantic regions label and predict objects at the pixel level

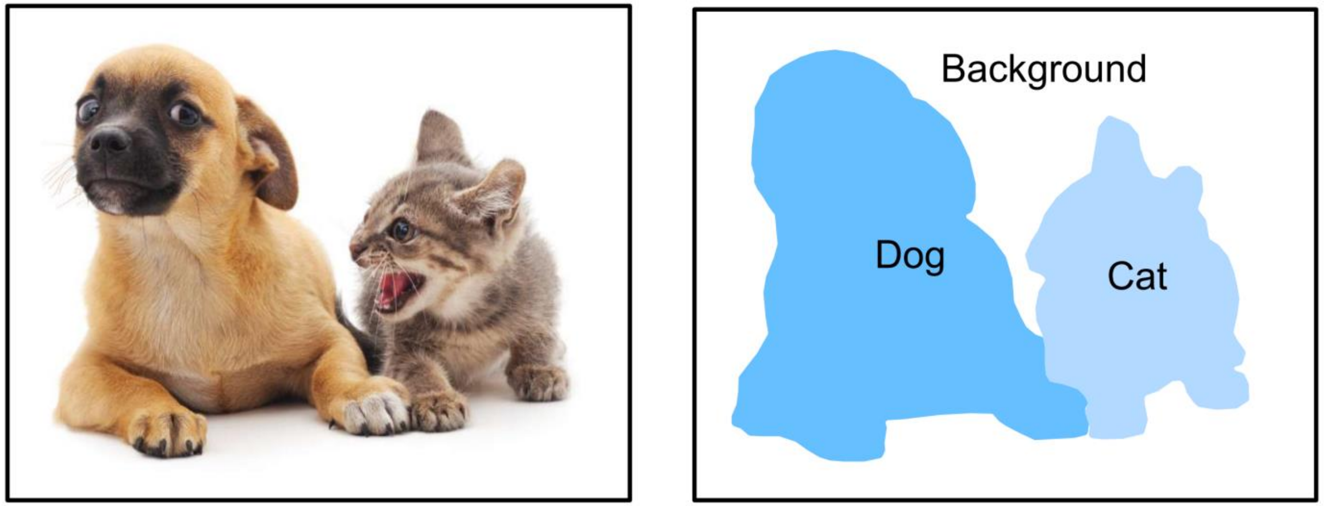


Image from http://d2l.ai/

Segmentation

- Segmentation task is different from classification task because it requires predicting a class for each pixel of the input image, instead of only 1 class for the whole input.
- Segment images into regions with different semantic categories. These semantic regions label and predict objects at the pixel level
- Classification needs to understand what is in the input (namely, the context).
- However, in order to predict what is in the input for each pixel, segmentation needs to recover not only what is in the input, but also where.

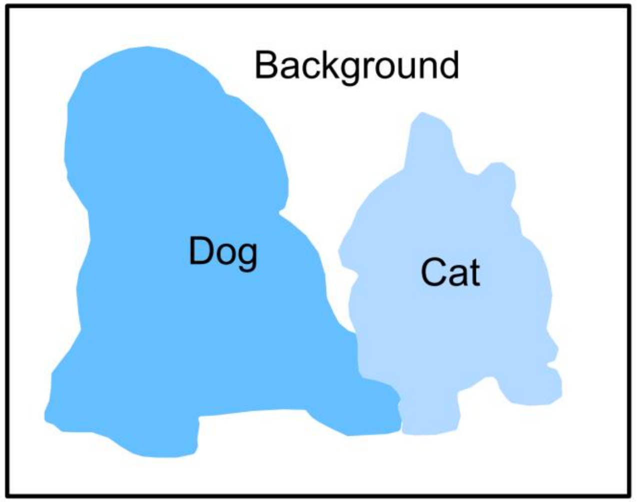


Image from http://d2l.ai/

Semantic Segmentation: FCNs

- FCN uses a convolutional neural network to transform image pixels to pixel categories.

- Network designed with all convolutional layers, with down-sampling and up-sampling operations

- Given a position on the spatial dimension, the output of the channel dimension will be a category prediction of the pixel corresponding to the location.

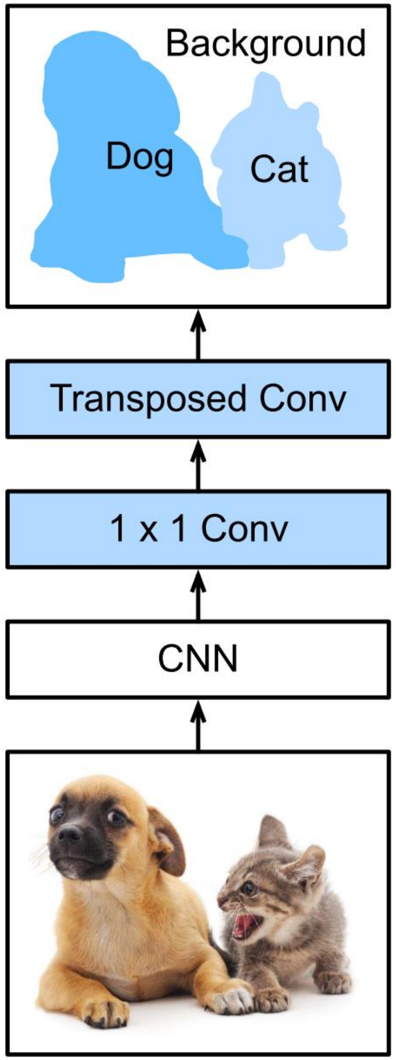
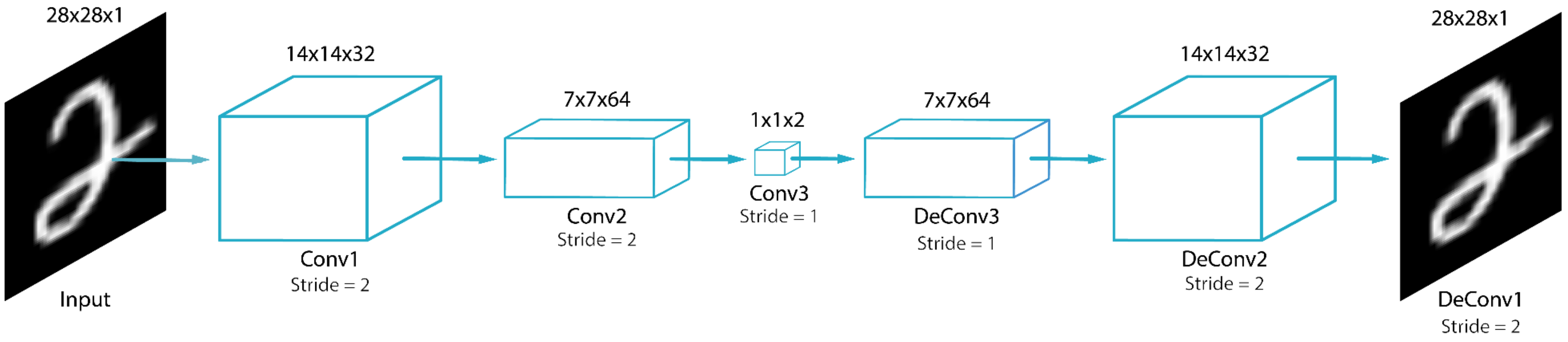
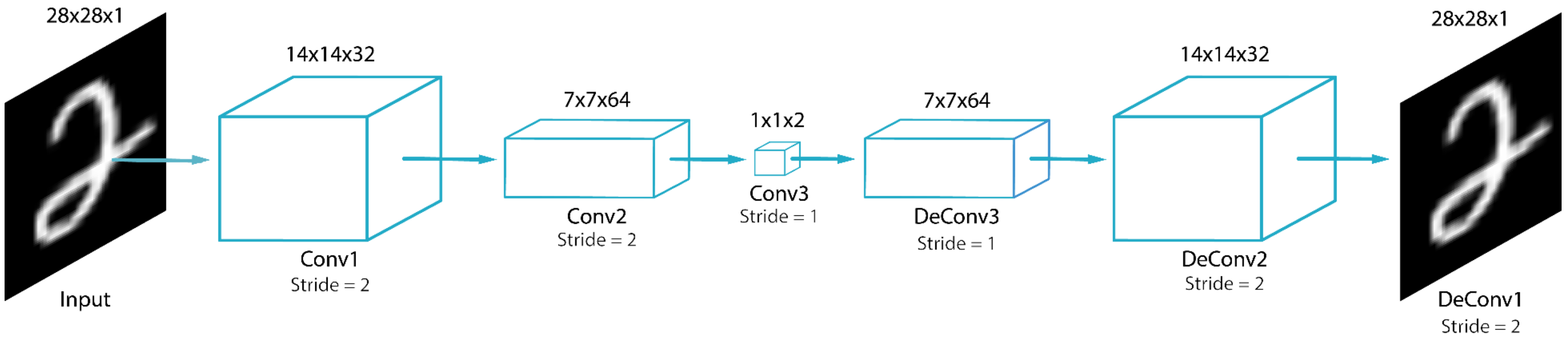


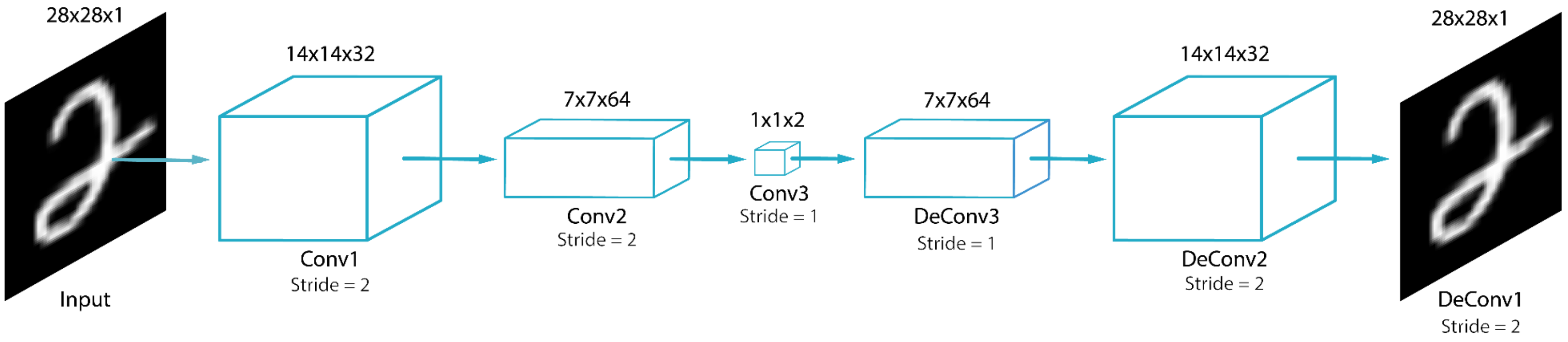
Image from http://d2l.ai/

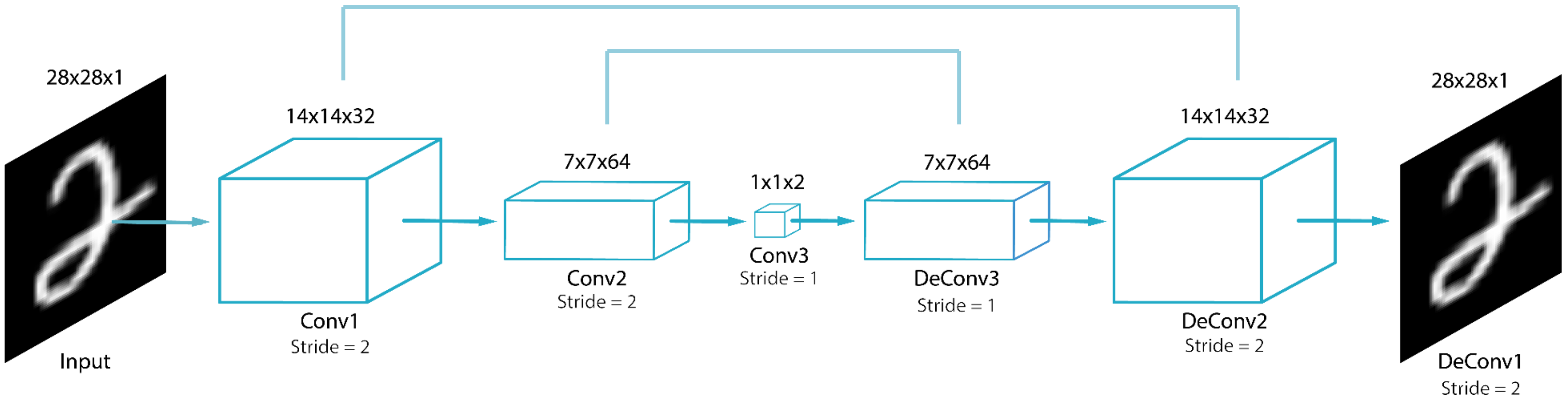
From CAE to FCN





From CAE to FCN

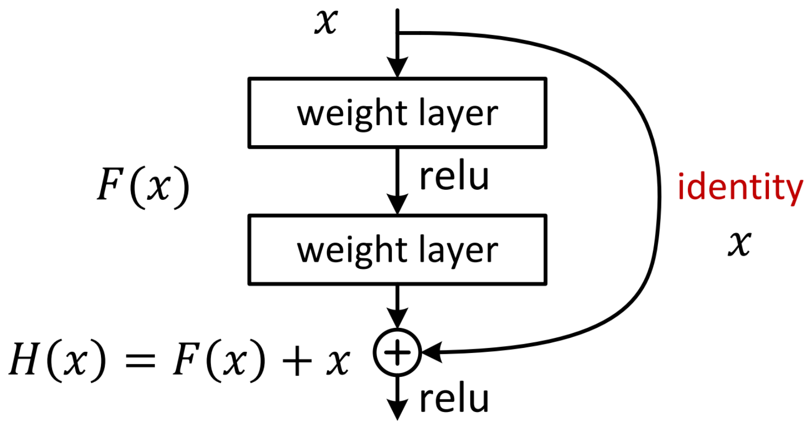




Skip Connection

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

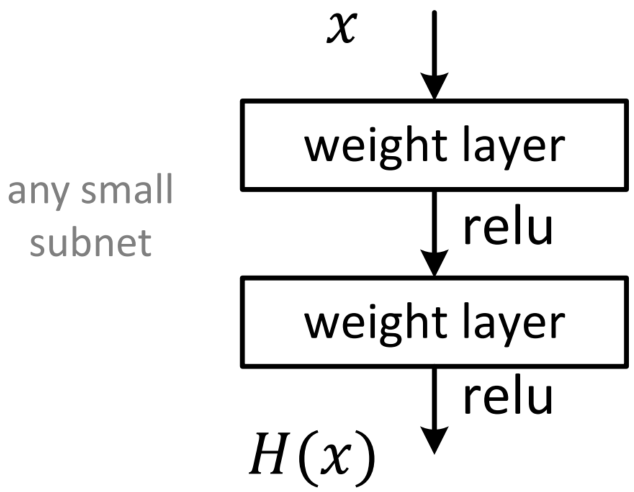
- Here, it is often used to transfer local information by summing feature maps from the downsampling path with feature maps from the upsampling path.
  - Merging features from various resolution levels helps combining context information with spatial information.



ResNet (Deep Residual Learning)

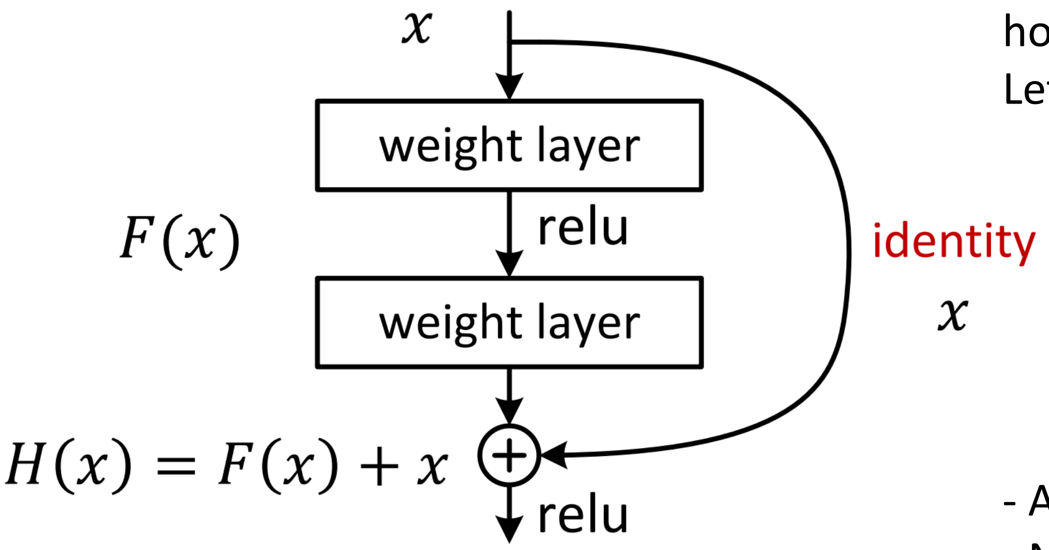
- He, Kaiming, et al. “Deep residual learning for image recognition.” CVPR. 2016.
- Plane net

$H(x)$ 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

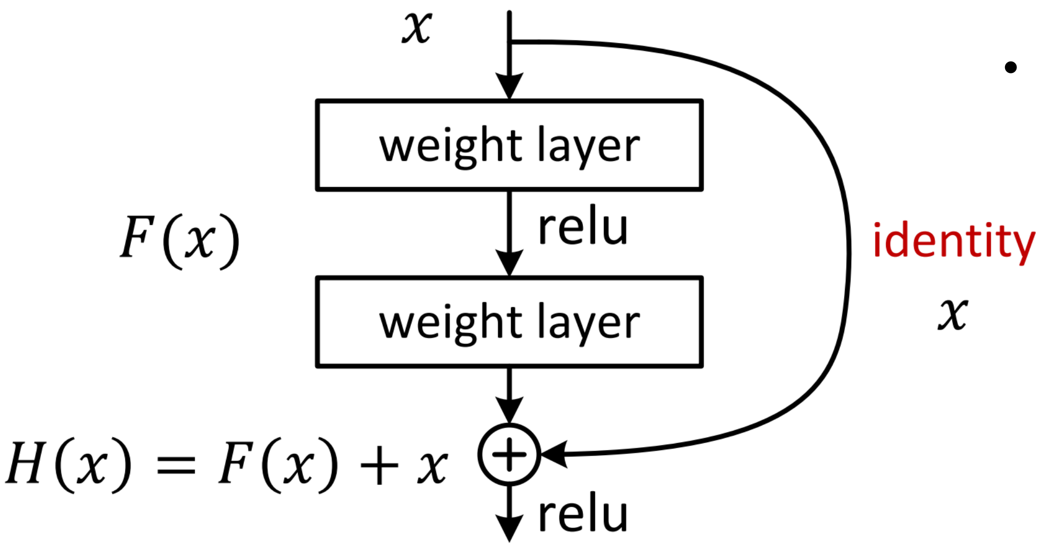


$H(x)$ is any desired mapping,
~~hope the small subnet fit $H(x)$~~
hope the small subnet fit $F(x)$
Let $H(x) = F(x) + x$

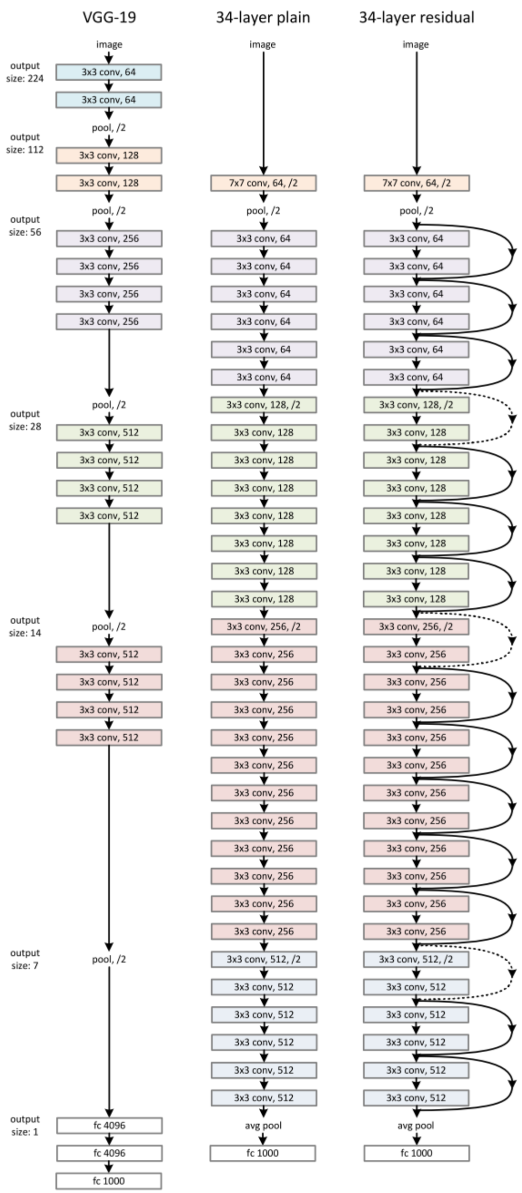
- A direct connection between 2 non-consecutive layers
- No gradient vanishing

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

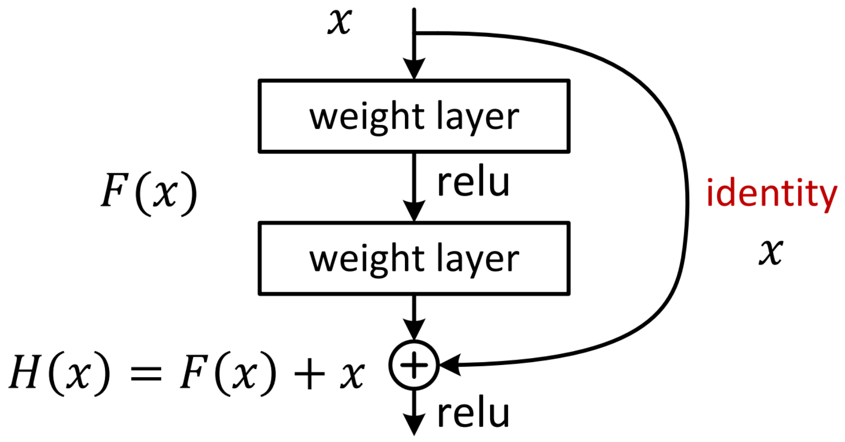


- If identity were optimal, easy to set weights as 0
- If optimal mapping is closer to identity, easier to find small fluctuations



Residual Net

```python
def residual_net(x):
    conv1 = tf.keras.layers.Conv2D(filters = 32,
                                   kernel_size = (3, 3),
                                   padding = "SAME",
                                   activation = 'relu')(x)


    conv2 = tf.keras.layers.Conv2D(filters = 32,
                                   kernel_size = (3, 3),
                                   padding = "SAME",
                                   activation = 'relu')(conv1)

    maxp2 = tf.keras.layers.MaxPool2D(pool_size = (2, 2),
                                      strides = 2)(conv2 + x)

    flat = tf.keras.layers.Flatten()(maxp2)

    hidden = tf.keras.layers.Dense(units = n_hidden,
                                   activation='relu')(flat)

    output = tf.keras.layers.Dense(units = n_output)(hidden)

    return output
```

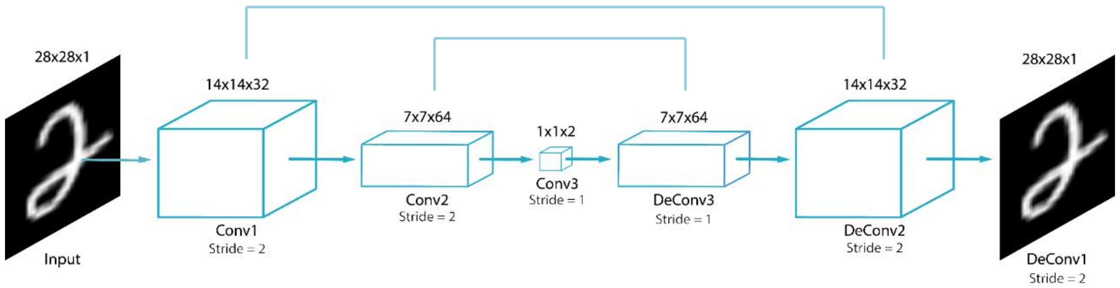


Fully Convolutional Networks (FCNs)

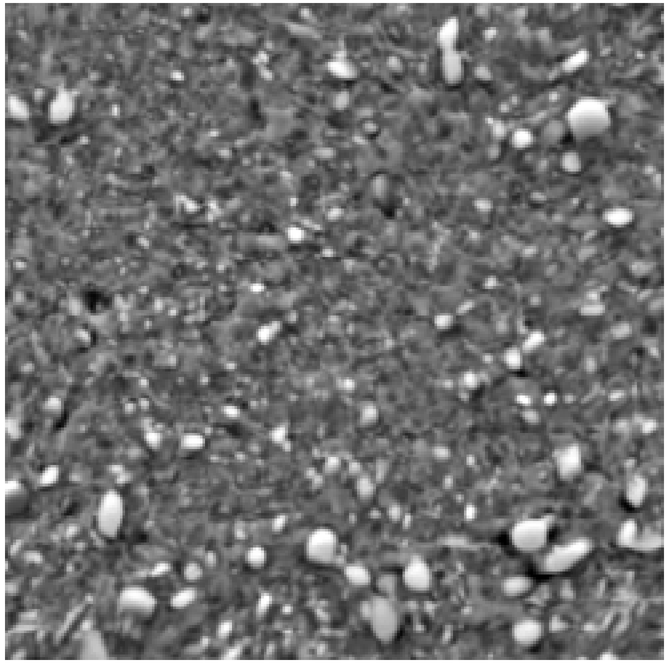
- To obtain a segmentation map (output), segmentation networks usually have 2 parts
  - Downsampling path: capture semantic/contextual information
  - Upsampling path: recover spatial information
- The downsampling path is used to extract and interpret the context (what), while the upsampling path is used to enable precise localization (where).

- Furthermore, to fully recover the fine-grained spatial information lost in the pooling or downsampling layers, we often use skip connections.

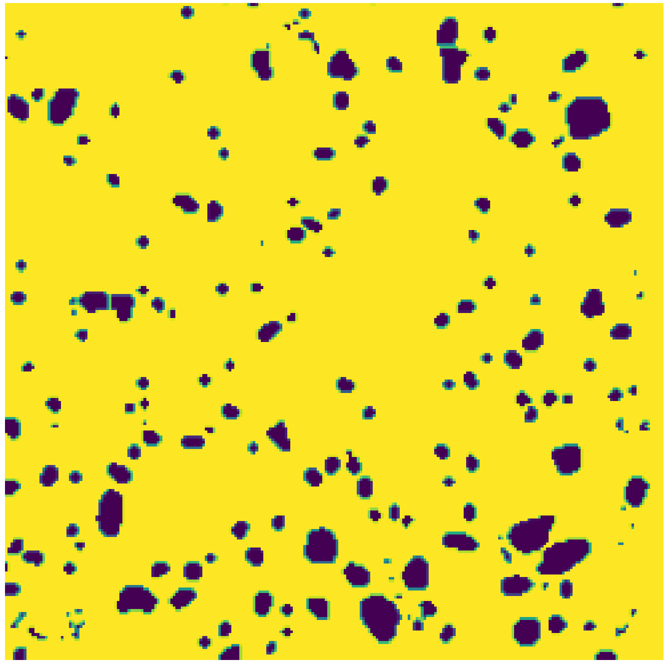
- Network can work regardless of the original image size, without requiring any fixed number of units at any stage.



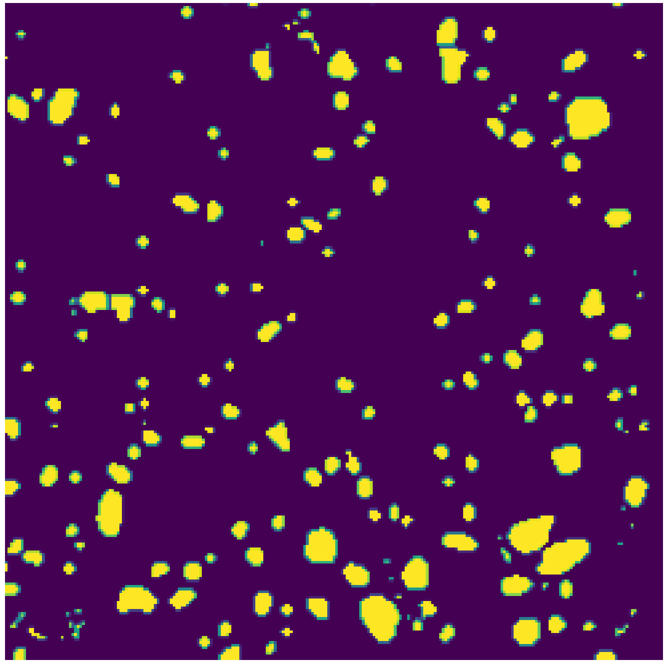
Segmented (Labeled) Images



input

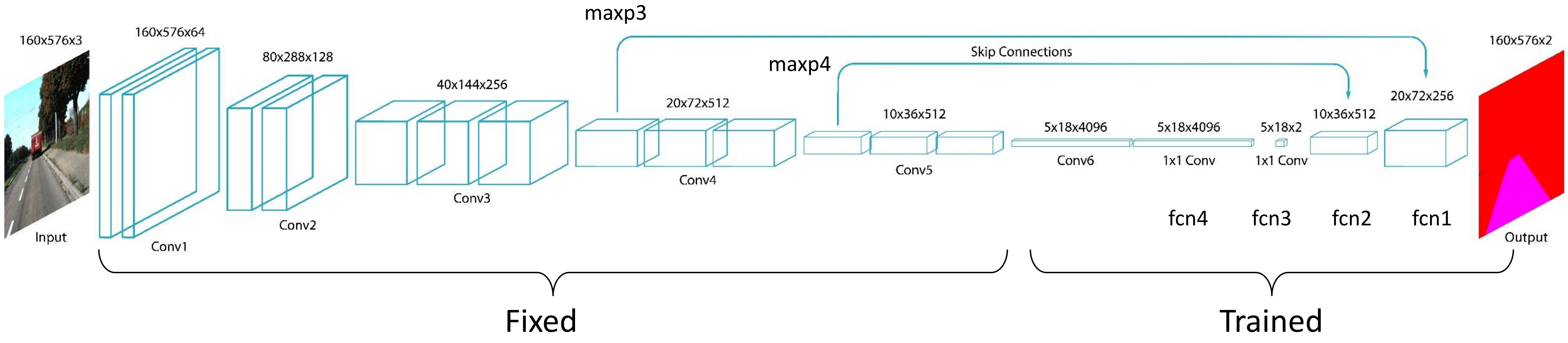


output

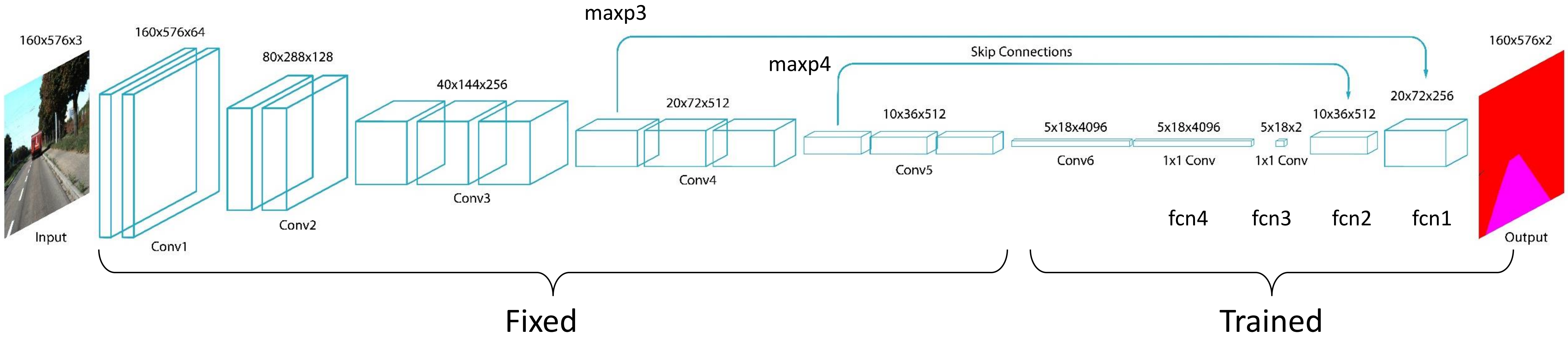


output

FCN Architecture



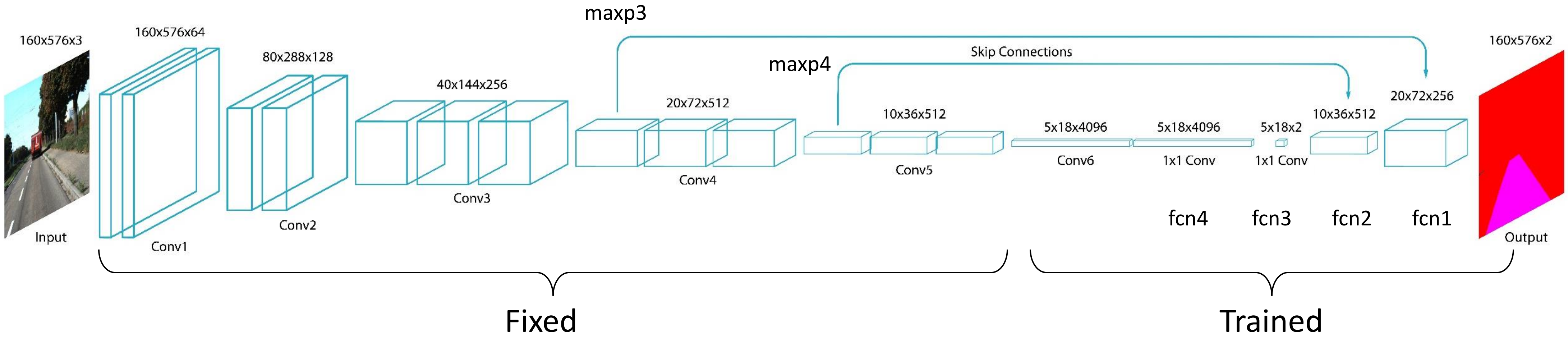
FCN Architecture




```
model_type = tf.keras.applications.vgg16
base_model = model_type.VGG16()
base_model.trainable = False
base_model.summary()
```

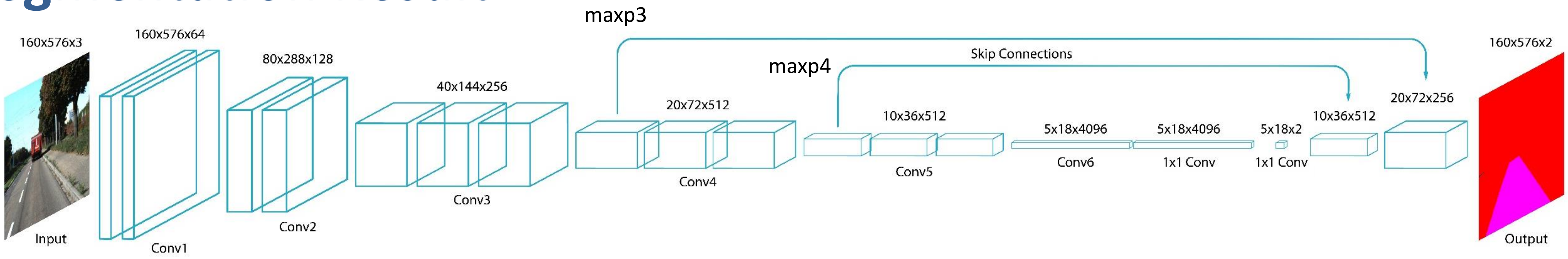
```
map5 = base_model.layers[-5].output

# sixth convolution layer
conv6 = tf.keras.layers.Conv2D(filters = 4096,
                               kernel_size = (7,7),
                               padding = 'SAME',
                               activation = 'relu')(map5)

# 1x1 convolution layers
fcn4 = tf.keras.layers.Conv2D(filters = 4096,
                              kernel_size = (1,1),
                              padding = 'SAME',
                              activation = 'relu')(conv6)

fcn3 = tf.keras.layers.Conv2D(filters = 2,
                              kernel_size = (1,1),
                              padding = 'SAME',
                              activation = 'relu')(fcn4)
```

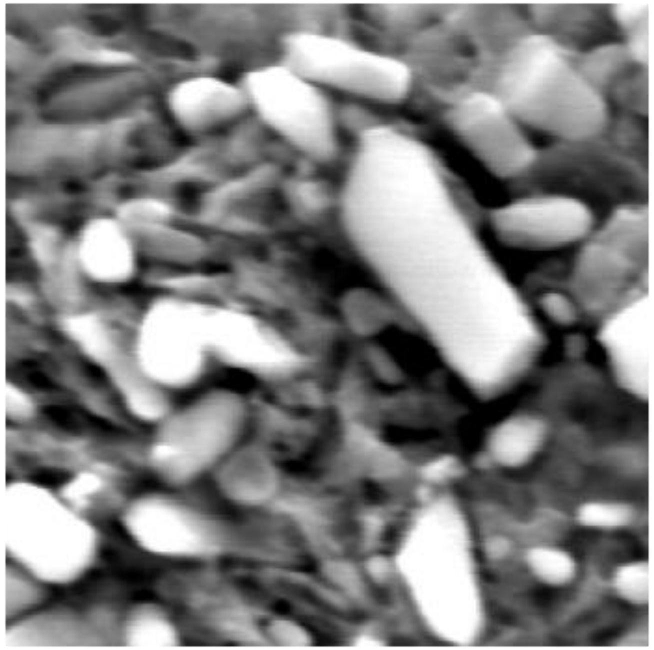
FCN Architecture



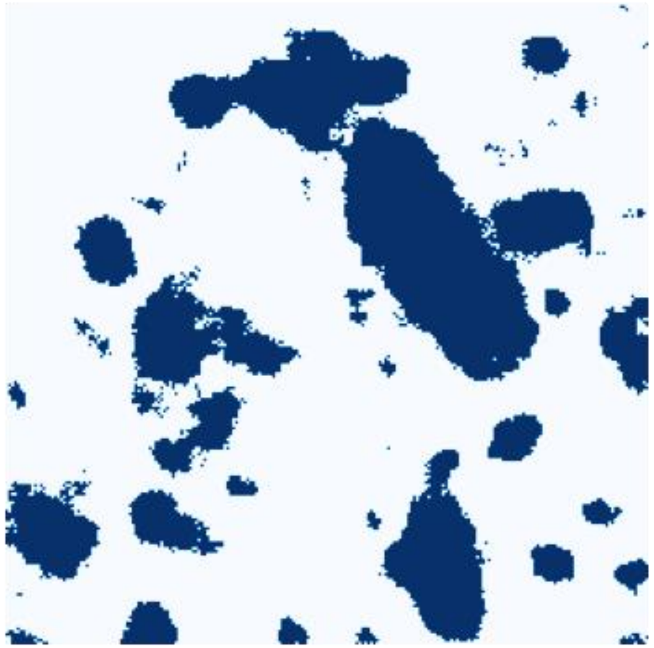

```
# Upsampling layers
fcn2 =  tf.keras.layers.Conv2DTranspose(filters = 512,
                                        kernel_size = (4,4),
                                        strides = (2,2),
                                        padding = 'SAME')(fcn3)

fcn1 =  tf.keras.layers.Conv2DTranspose(filters = 256,
                                        kernel_size = (4,4),
                                        strides = (2,2),
                                        padding = 'SAME')(fcn2 + base_model.layers[14].output)

output =  tf.keras.layers.Conv2DTranspose(filters = 2,
                                          kernel_size = (16,16),
                                          strides = (8,8),
                                          padding = 'SAME',
                                          activation = 'softmax')(fcn1 + base_model.layers[10].output)

model = tf.keras.Model(inputs = base_model.inputs, outputs = output)
```

Segmentation Result

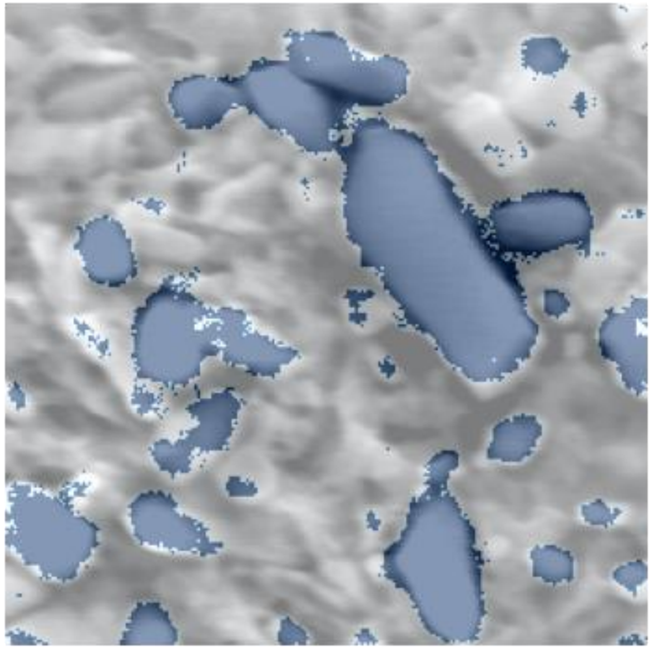




input



Segmentation output



overlapping