Image Semantic Segmentation in Deep Learning

CT Hsieh

Reference

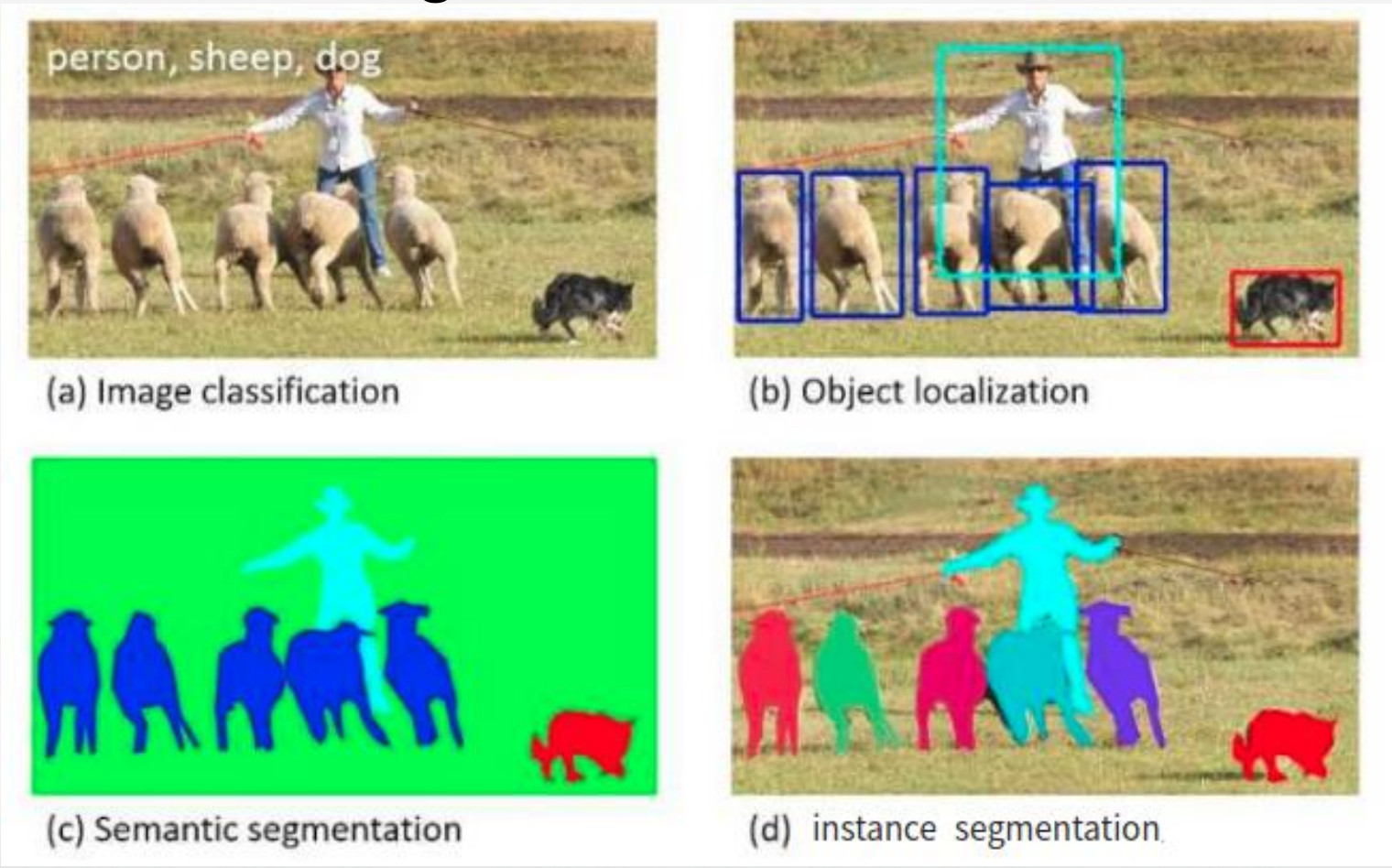
- https://github.com/mrgloom/awesome-semantic-segmentation
- http://blog.qure.ai/notes/semantic-segmentation-deep-learning-review
- https://www.slideshare.net/xavigiro/image-segmentation-d3l1-2017-upc-deep-learning-for-computer-vision
- https://www.slideshare.net/mitmul/a-brief-introduction-to-recent-segmentation-methods
- https://www.youtube.com/watch?v=le1NvGzDWDk

Reference

- Reference Code from Github
  - Fully Convolutional Network
    - https://github.com/shekkizh/FCN.tensorflow
  - U-Net
    - https://github.com/jakeret/tf_unet

Definition

- Semantic Segmentation

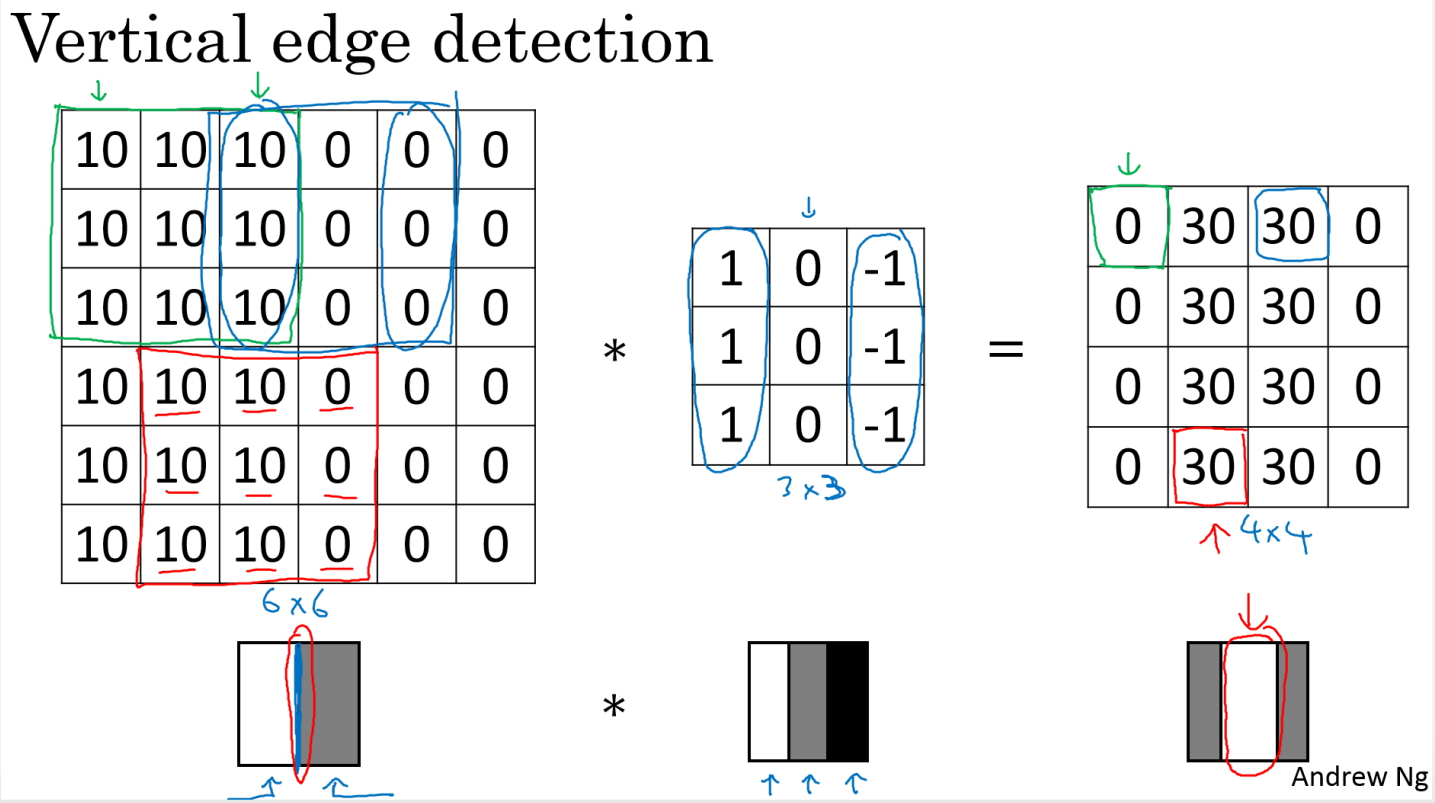


Outline

- Related Work
  - Related Guide to Computer Vision
  - Why Go into Deeper
- Machine Learning Method
  - Conditional Random Field
- Deep Learning Method
  - Fully Convolutional Network and Others
- Discuss

Related Work

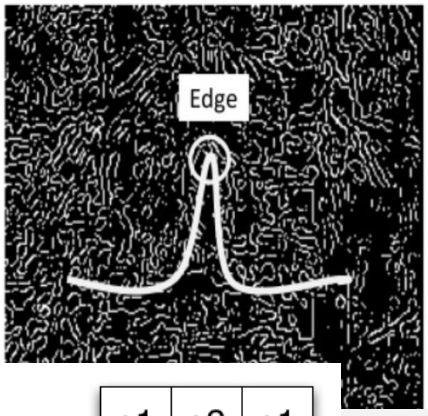
- All about filter
  - Feature Extraction, Blur, Sharpen, … etc



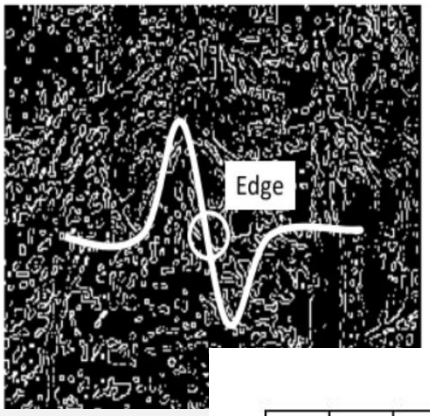
Edge Detection

- How it works

- ***Gradient Based***
  Detects edges by looking for the maxima or minima of the first derivate of the image



- ***Laplacian based***
  Detect edges by searching for the zero crossing of the second derivate of the image



-1	0	+1
-2	0	+2
-1	0	+1

x filter

+1	+2	+1
0	0	0
-1	-2	-1

y filter

0	-1	0
-1	4	-1
0	-1	0

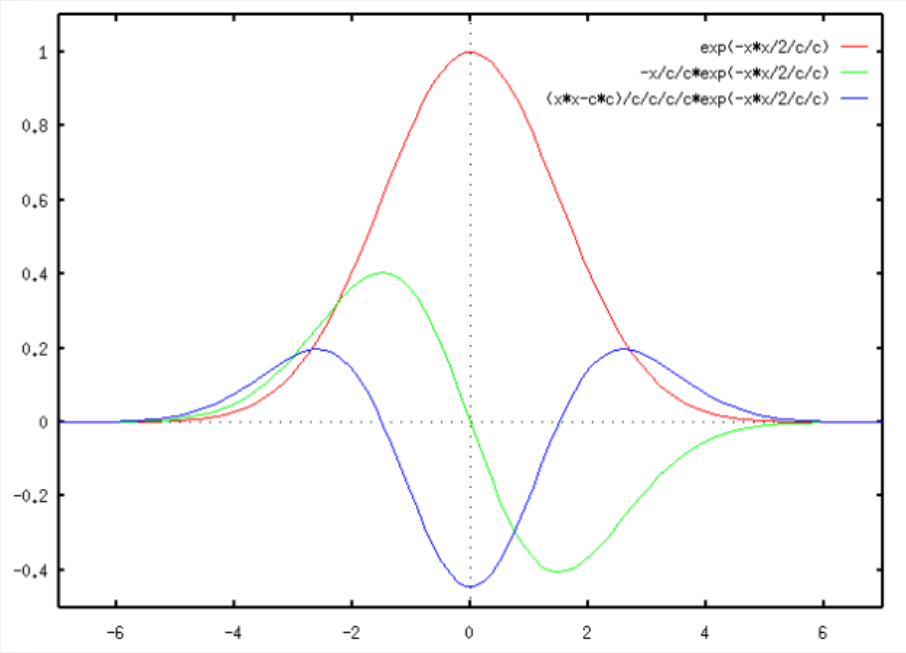
The laplacian operator

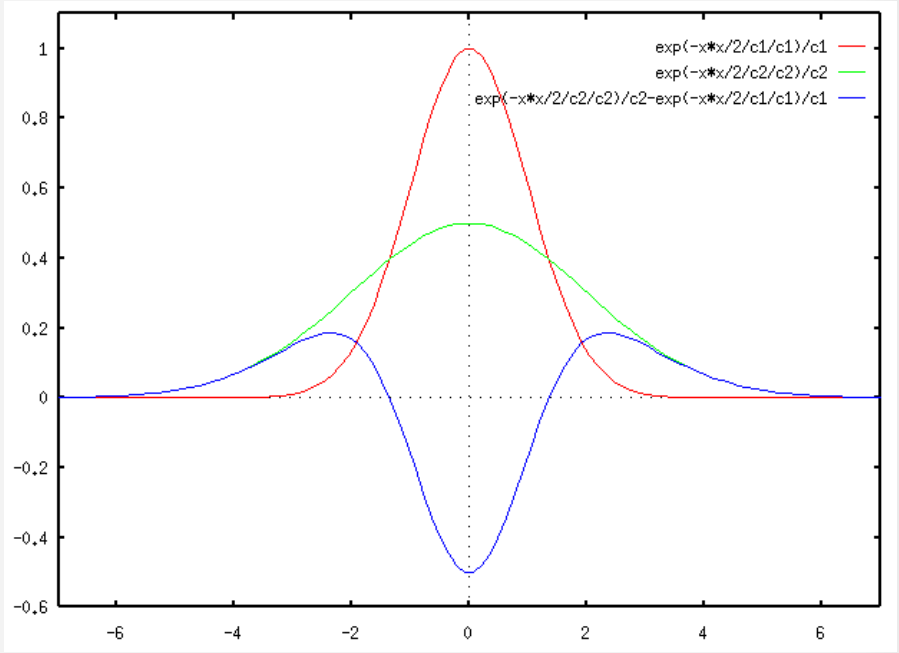
-1	-1	-1
-1	8	-1
-1	-1	-1

The laplacian operator (include diagonals)

Edge Detection

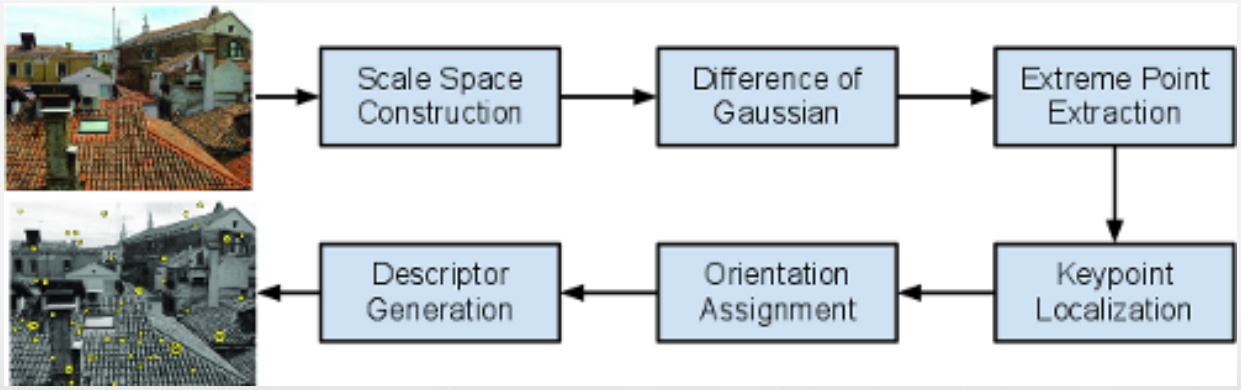
- Other Methods
  - Mexican Hat Wavelet: LoG and DoG





Hand Made CV Algorithm

- Local Feature Based Recognition
  - SIFT, SURF, ORB, … etc
- Method
  - Detection: Find Candidate Point or Region
  - Description: Turn Features to Vector
  - Matching: Find Minimum Distance Pair



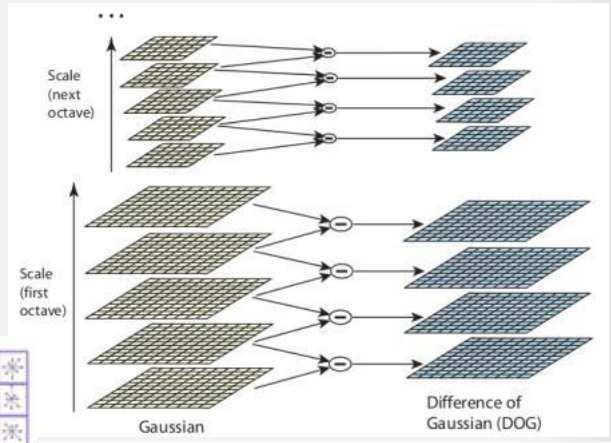
Local Feature Based

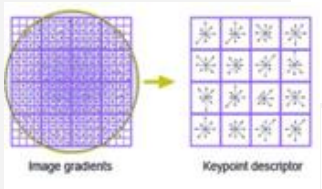
- Compute Gradient of Gray
  - Magnitude
  - Orientation

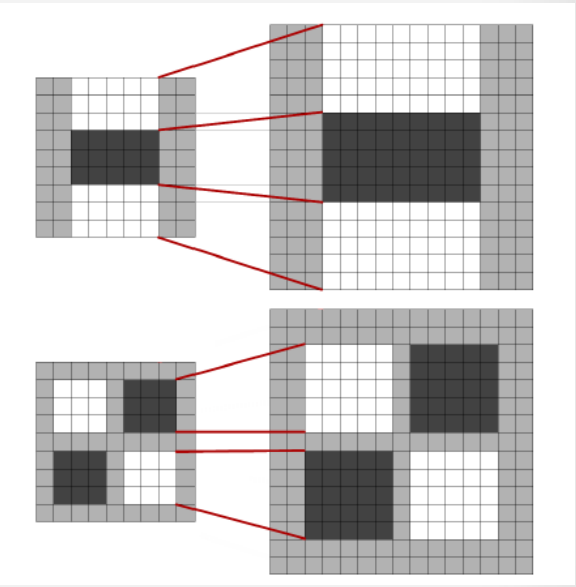
$$m(x,y) = \sqrt{(L(x+1,y) - L(x-1,y))^2 + (L(x,y+1) - L(x,y-1))^2}$$

$$\theta(x,y) = \tan^{-1}((L(x,y+1) - L(x,y-1))/(L(x+1,y) - L(x-1,y)))$$

- Description
  - SIFT: Statistic Method
  - SURF: Harr-Like Feature
- Orientation Invariance
  - Normalize Orientation
- Scale Invariance
  - SIFT: Change Image Size - Strike
  - SURF: Change Filter Size - Inception

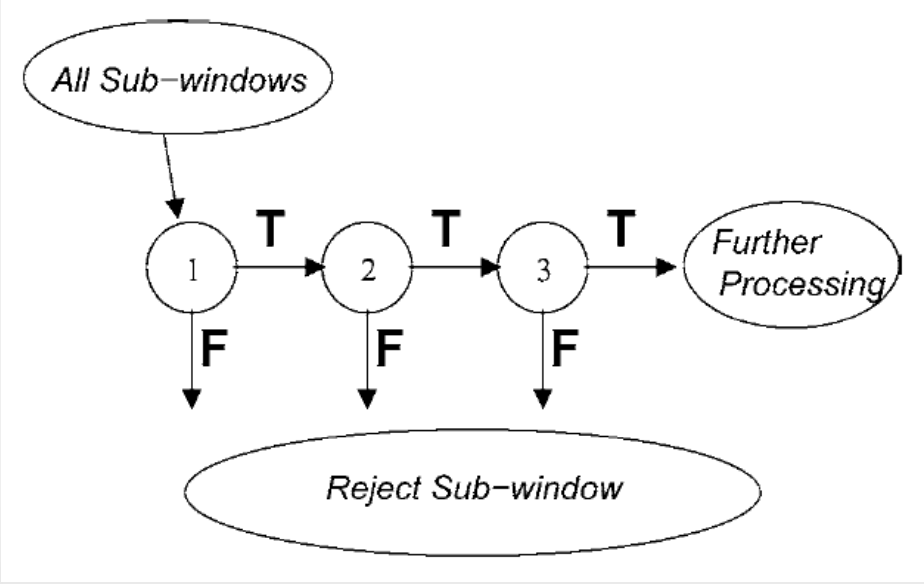


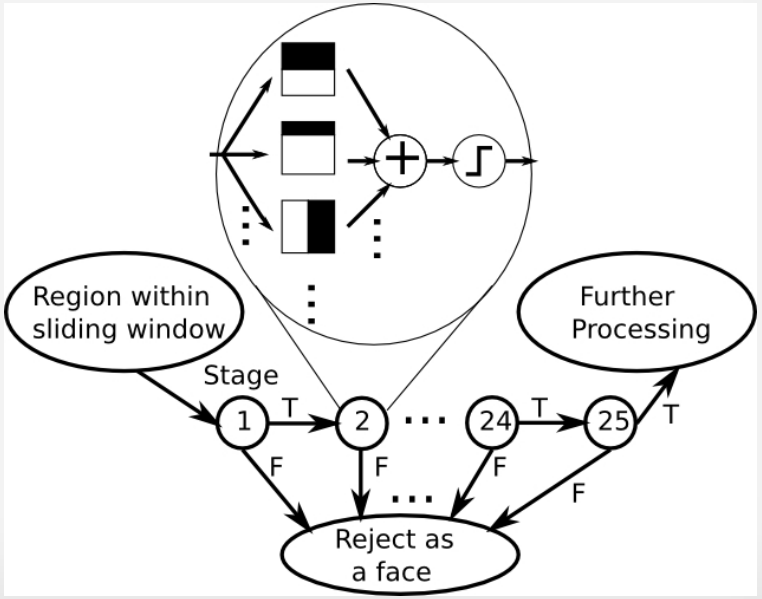




Machine Learning CV Algorithm

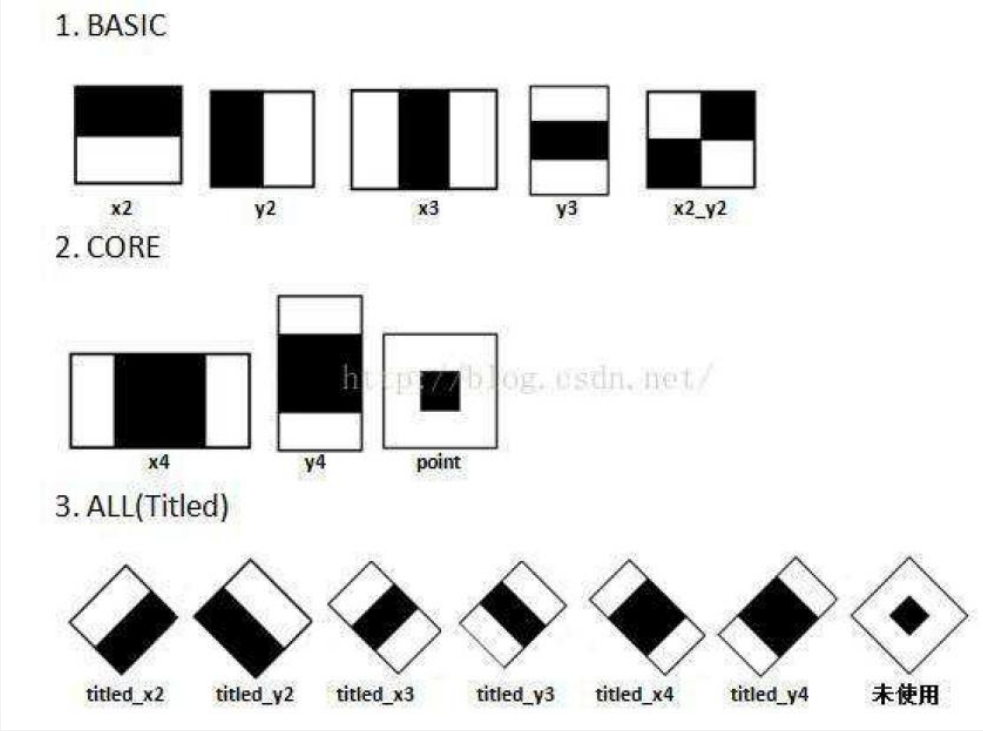
- Cascade Classifier for Face Detection
  - OpenCV Example, No Candidate
  - Prevent FN, More Power Near Root
  - Also Use Scaled Harr-Like Feature





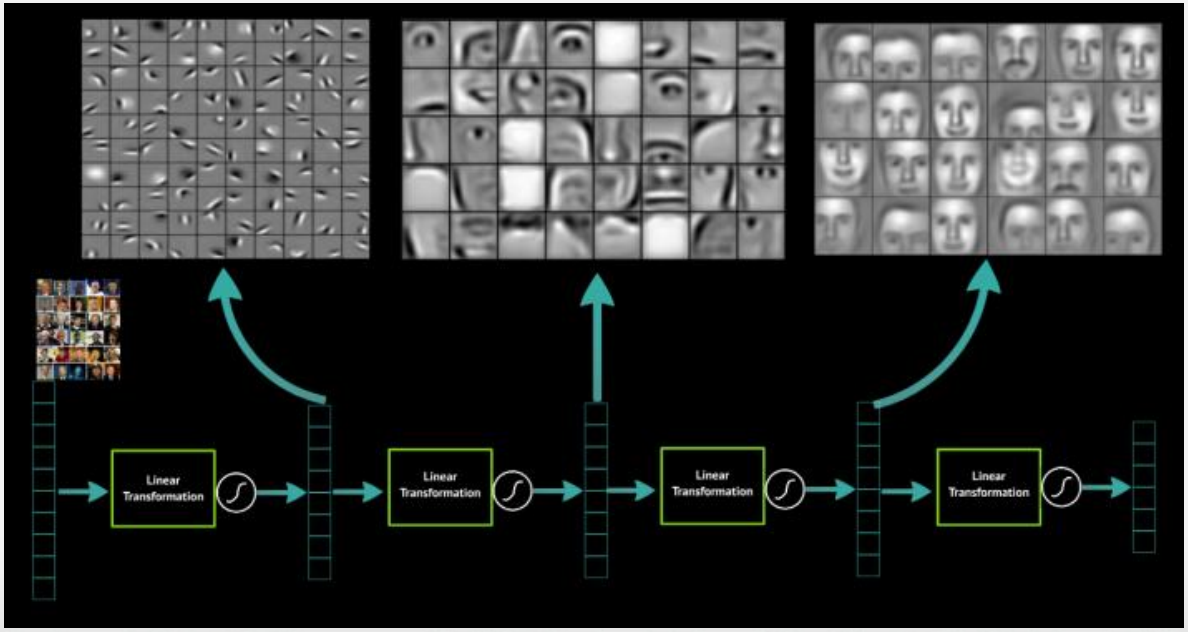
Harr-Like Feature

- Cascade Classifier
  - Harr-Like Feature



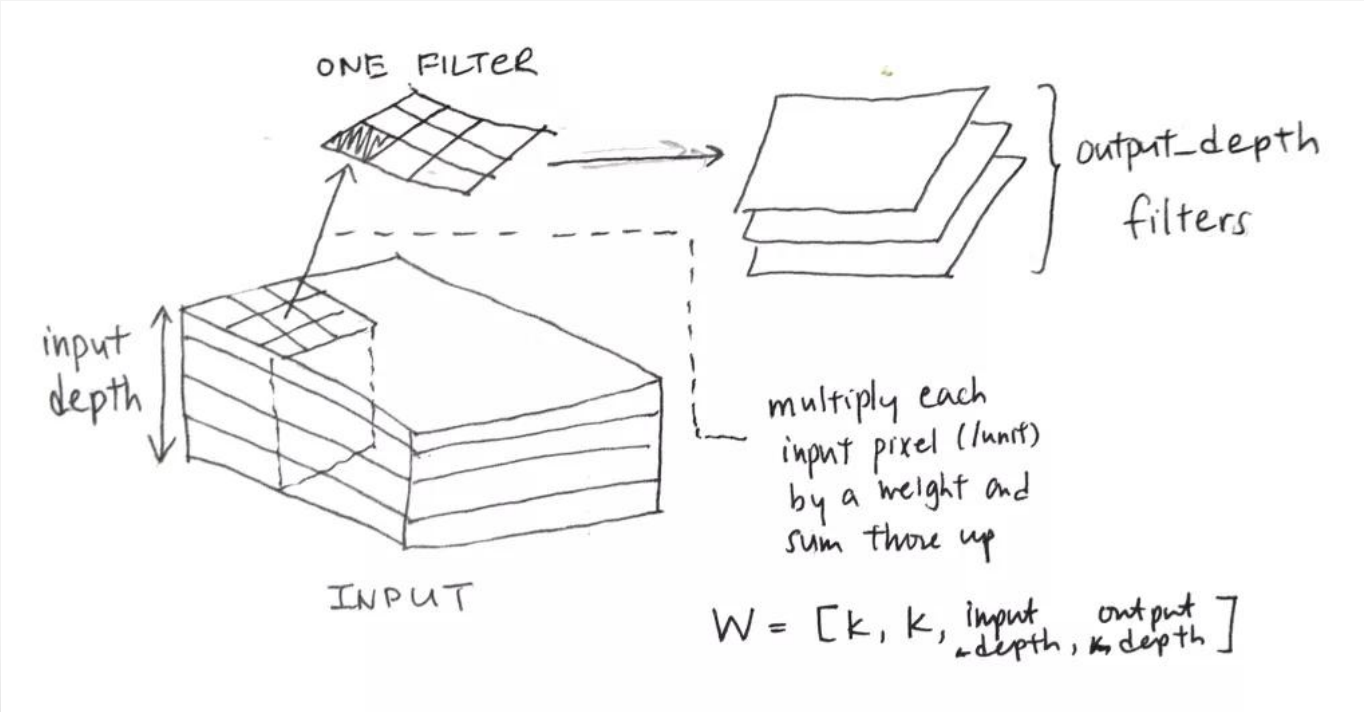
Into the Deep

- CNN for Classification
  - LeNet, AlexNet, VGG, GoogLeNet (Inception), ResNet, DenseNet
  - Learned Filter Weight and bias



Convolutional

- tf.nn.conv2D: CH.in / (F.hXF.w) / CH.out (3D Filter)
  - For each output: CH.in X (F.hXF.w) Weights + 1 Bias
  - Total: CH.in X (F.hXF.w) X CH.out Weights + 1 X CH.out Bias
- Think about MRI: 3D Image, 4D Filter

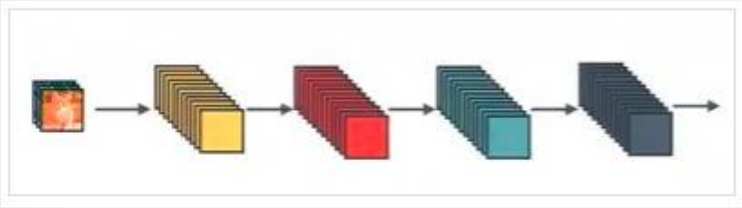


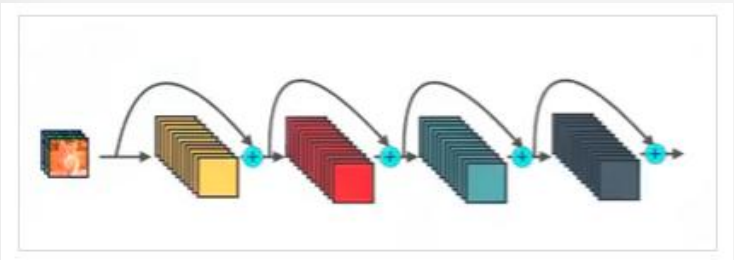
Why Go Deeper

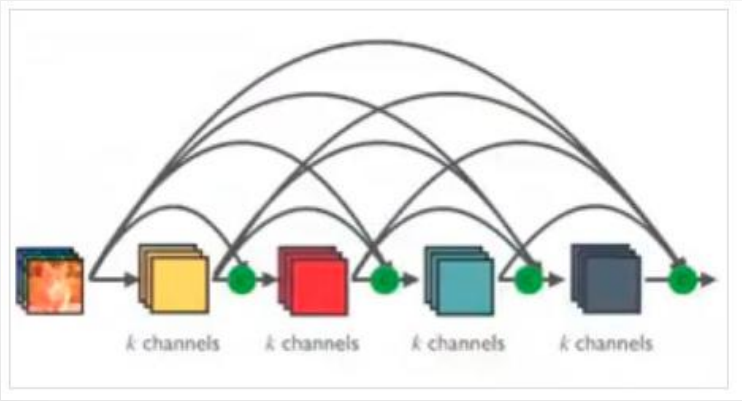
- Hand Made CV
  - Usually must select Candidate
- Traditional Machine Learning CV
  - Usually Use Fixed Filter with different scale
- Into the deep
  - Freedom!
  - Even Learn Filter
  - Deeper, more complicated

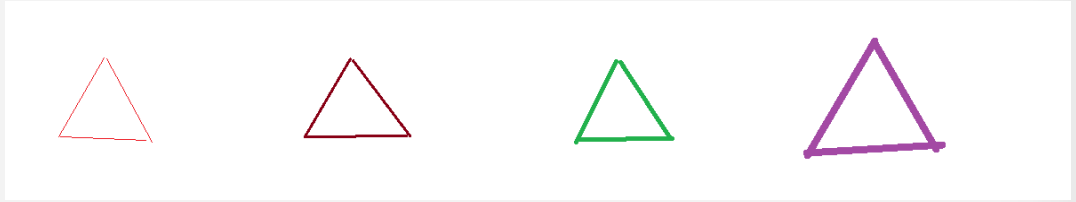
All About Degree of Freedom

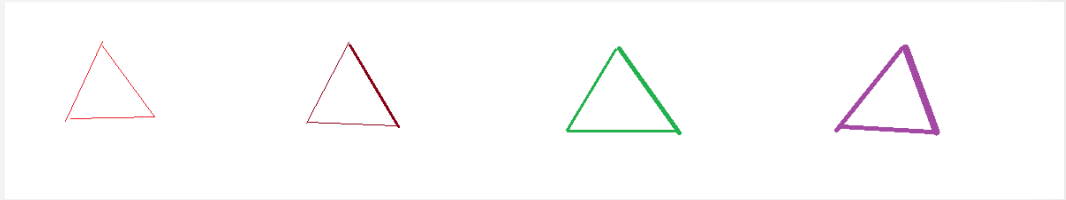
- Why ResNet / DenseNet Work: CVPR 16/17 Best Paper

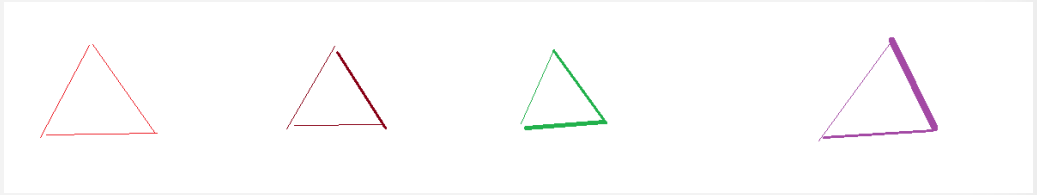






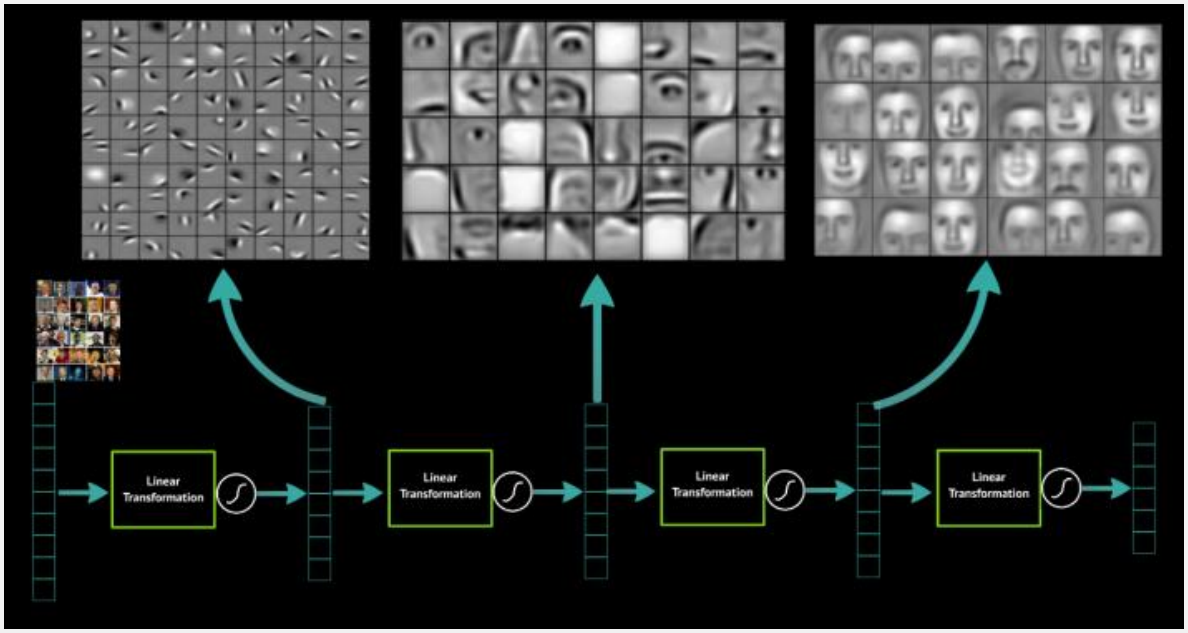






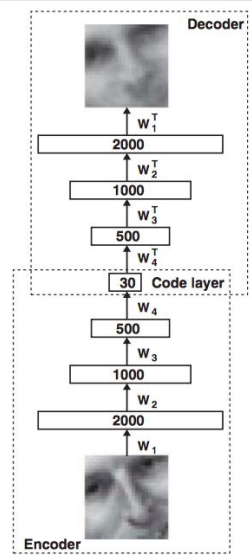
All About Degree of Freedom

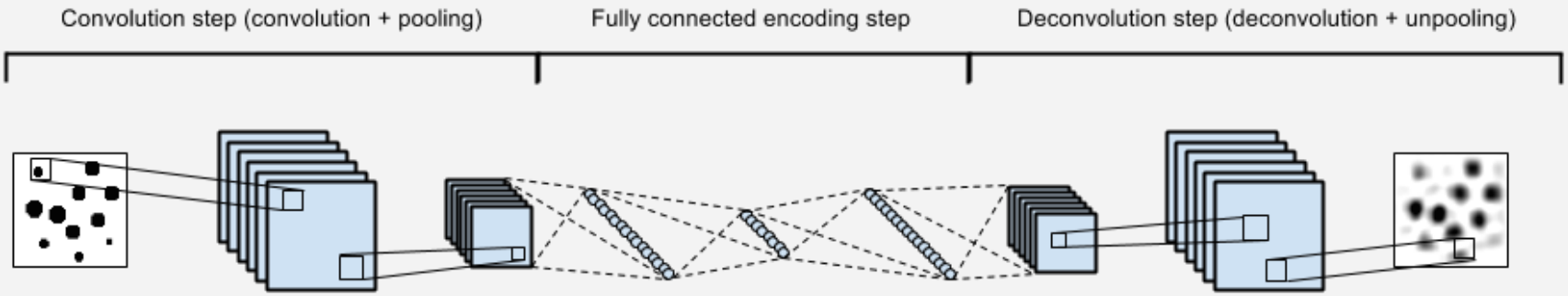
- Problem in Deeper
  - Large Scale Feature, Much Less Freedom without short cut



Related Work

- Auto-Encoder: Unsupervised Learning
  - Dimension Reduction and Reconstruct
  - Just like PCA





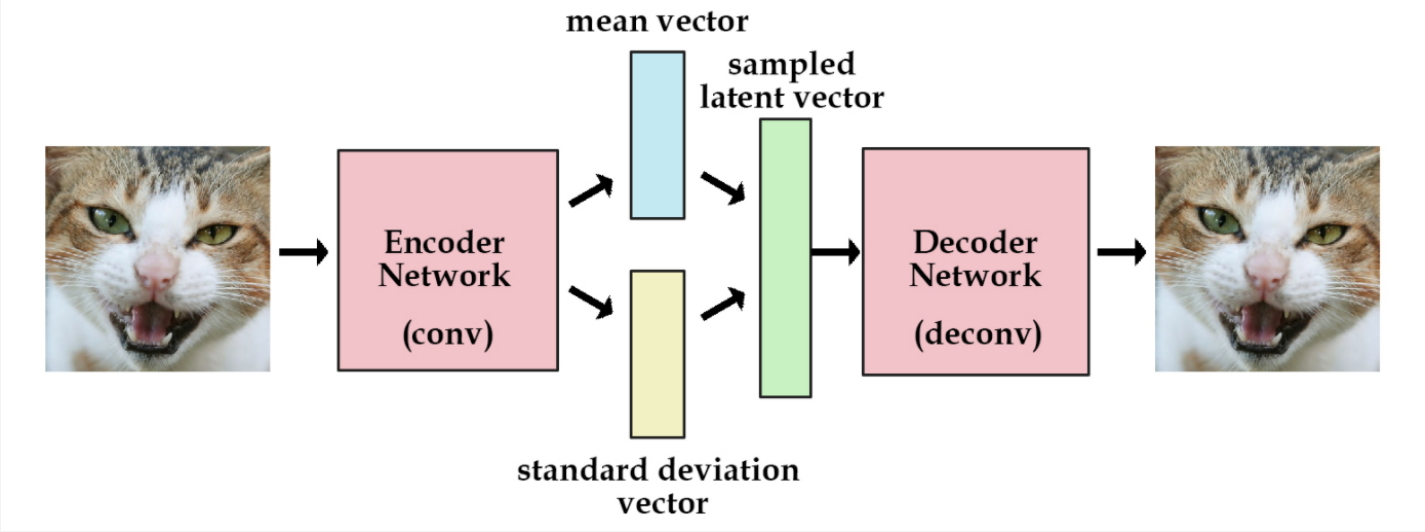
Related Work

$$D_{\mathrm{KL}}(P\|Q) = -\sum_i P(i)\,\log\frac{Q(i)}{P(i)}$$

$$D_{\mathrm{KL}}(P\|Q) = \sum_i P(i)\,\log\frac{P(i)}{Q(i)}$$

$$D_{\mathrm{KL}}(P\|Q) = \int_X \log\left(\frac{dP}{dQ}\right)\frac{dP}{dQ}\,dQ$$

- VAE – Variance Autoencoder
  - Upgrade Version Autoencoder
  - Kullback–Leibler divergence (Related Entropy)




```
# z_mean and z_stddev are two vectors generated by encoder network
latent_loss = 0.5 * tf.reduce_sum(tf.square(z_mean) + tf.square(z_stddev) - tf.log(tf.square(z_stddev)) - 1,1)

samples = tf.random_normal([batchsize,n_z],0,1,dtype=tf.float32)
sampled_z = z_mean + (z_stddev * samples)
```

Related Work

- Sparse Autoencoder

$$\Omega_{sparsity} = \sum_{i=1}^{D} \rho \log(\frac{\rho}{\hat{\rho}_i}) + (1-\rho) \log(\frac{1-\rho}{1-\hat{\rho}_i})$$

$\hat{\rho}_i$: average output activation value of a neuron i

$$\Omega_{weights} = \frac{1}{2} \sum_{l}^{L} \sum_{j}^{n} \sum_{i}^{k} (w_{ji}^{(l)})^2$$

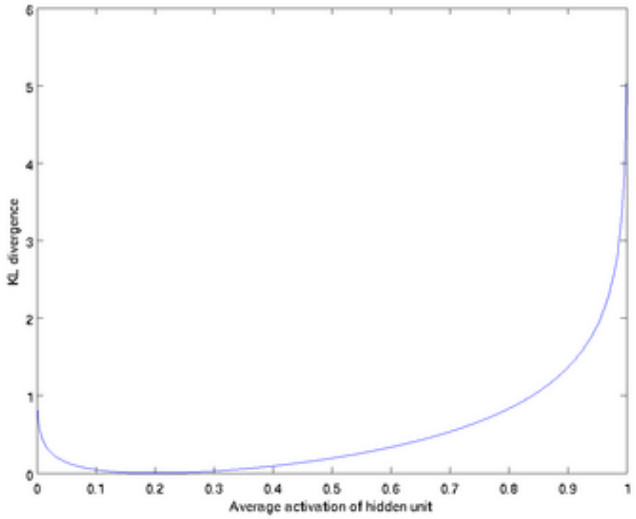
L : number of the hidden layers

n : number of observations

k : number of variables in training data

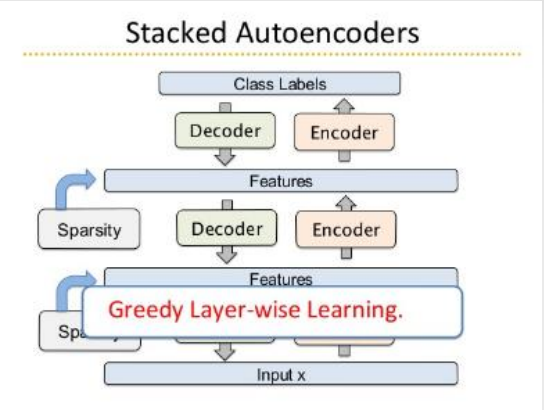
$$E = \Omega_{mse} + \beta * \Omega_{sparsity} + \lambda * \Omega_{weights}$$

```
%matplotlib inline
import numpy as np
import tensorflow as tf
import matplotlib.pyplot as plt
from tensorflow.examples.tutorials.mnist import input_data
mnist = input_data.read_data_sets("MNIST_data/", one_hot = True)

rho_hat = np.linspace(0 + 1e-2, 1 - 1e-2, 100)
rho = 0.2
kl_div = rho * np.log(rho/rho_hat) + (1 - rho) * np.log((1 - rho) / (1 - rho_h
at))
plt.plot(rho_hat, kl_div)
plt.xlabel("rho_hat")
plt.ylabel("kl_div")
```

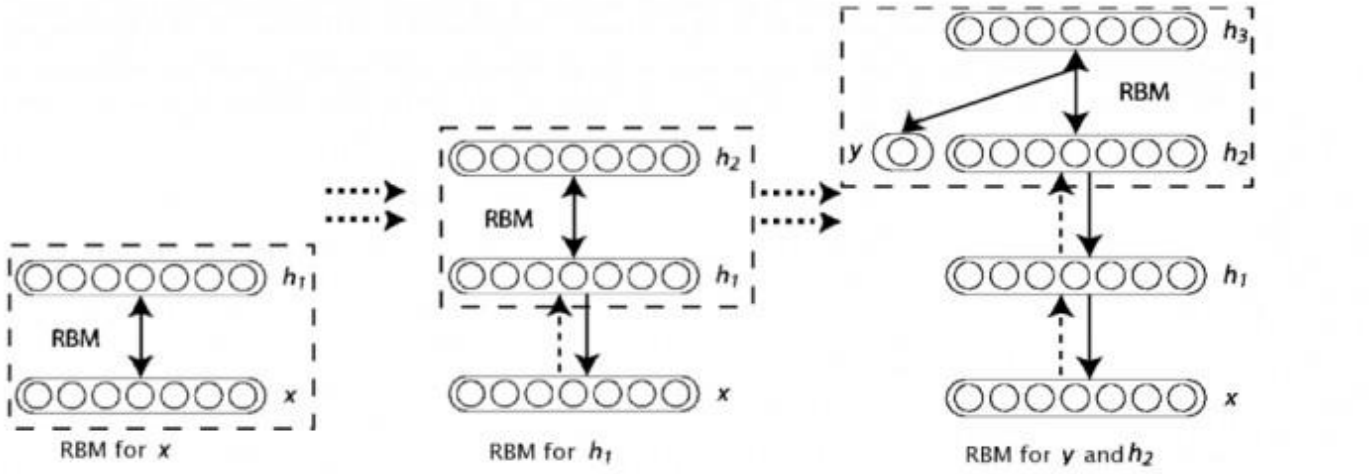


Related Work

- Stacked Autoencoder



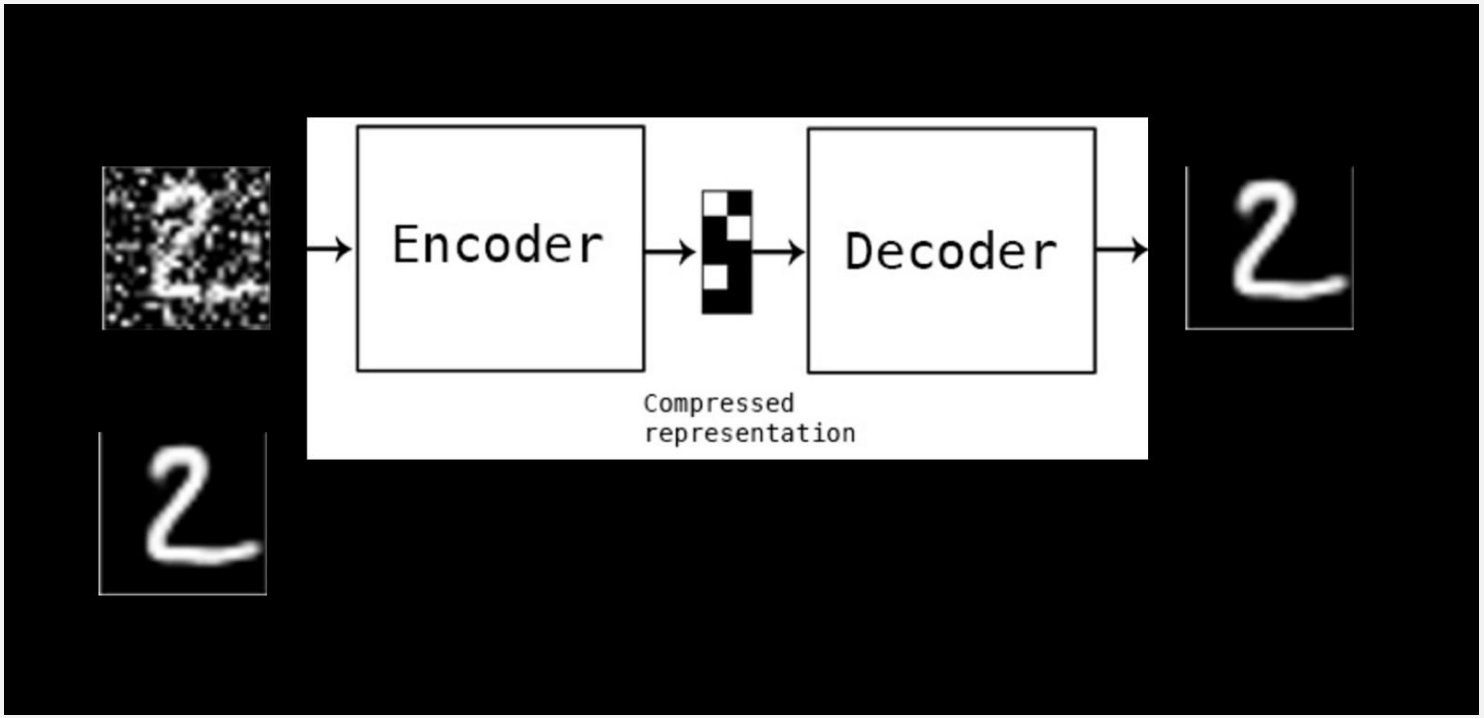
Greedy Layer-Wise Pre-Training



Stacking Restricted Boltzmann Machines (RBM) → Deep Belief Network (DBN)
→ Supervised deep neural network

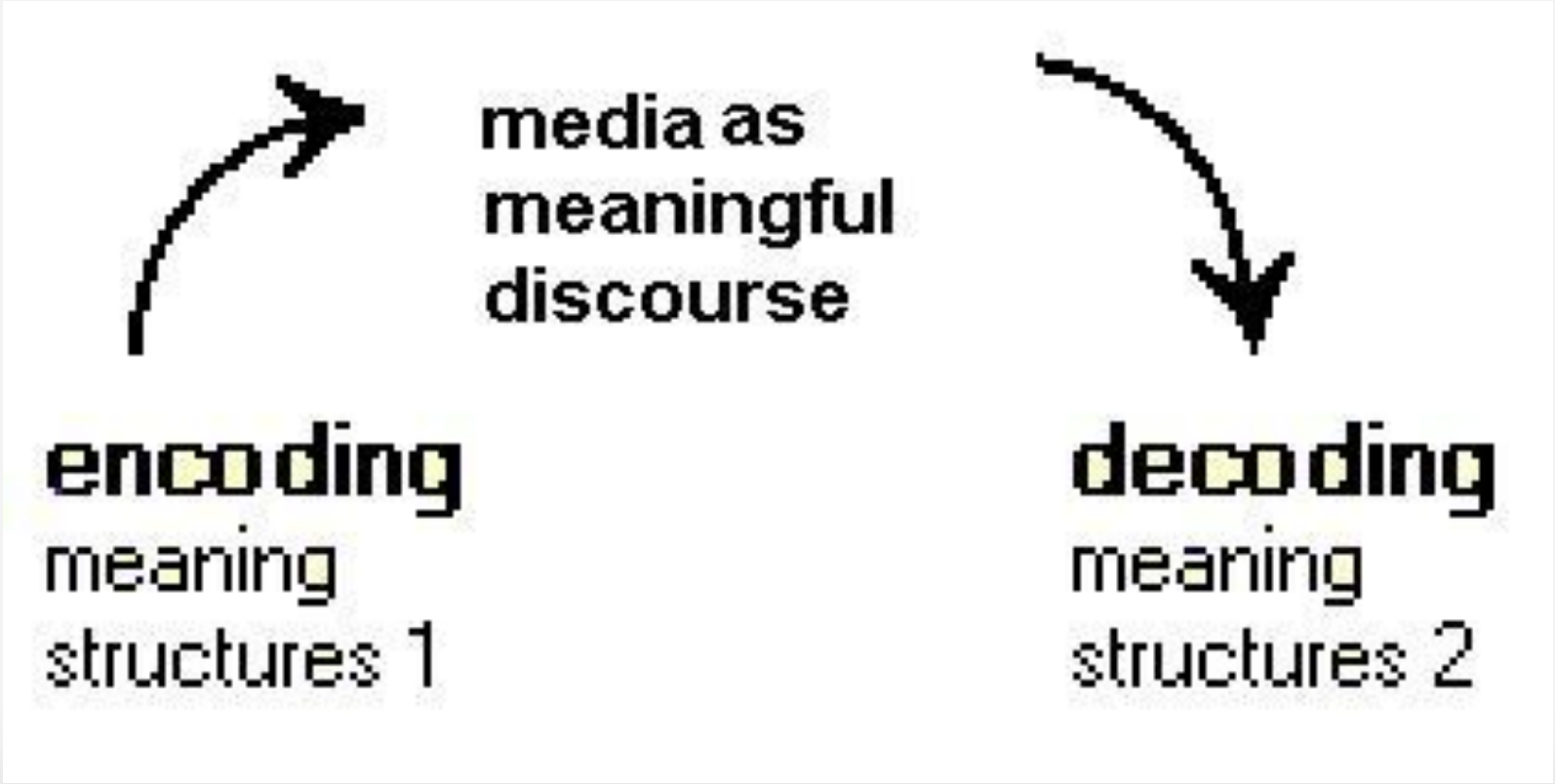
Related Work

- Auto-Encoder for De-noise

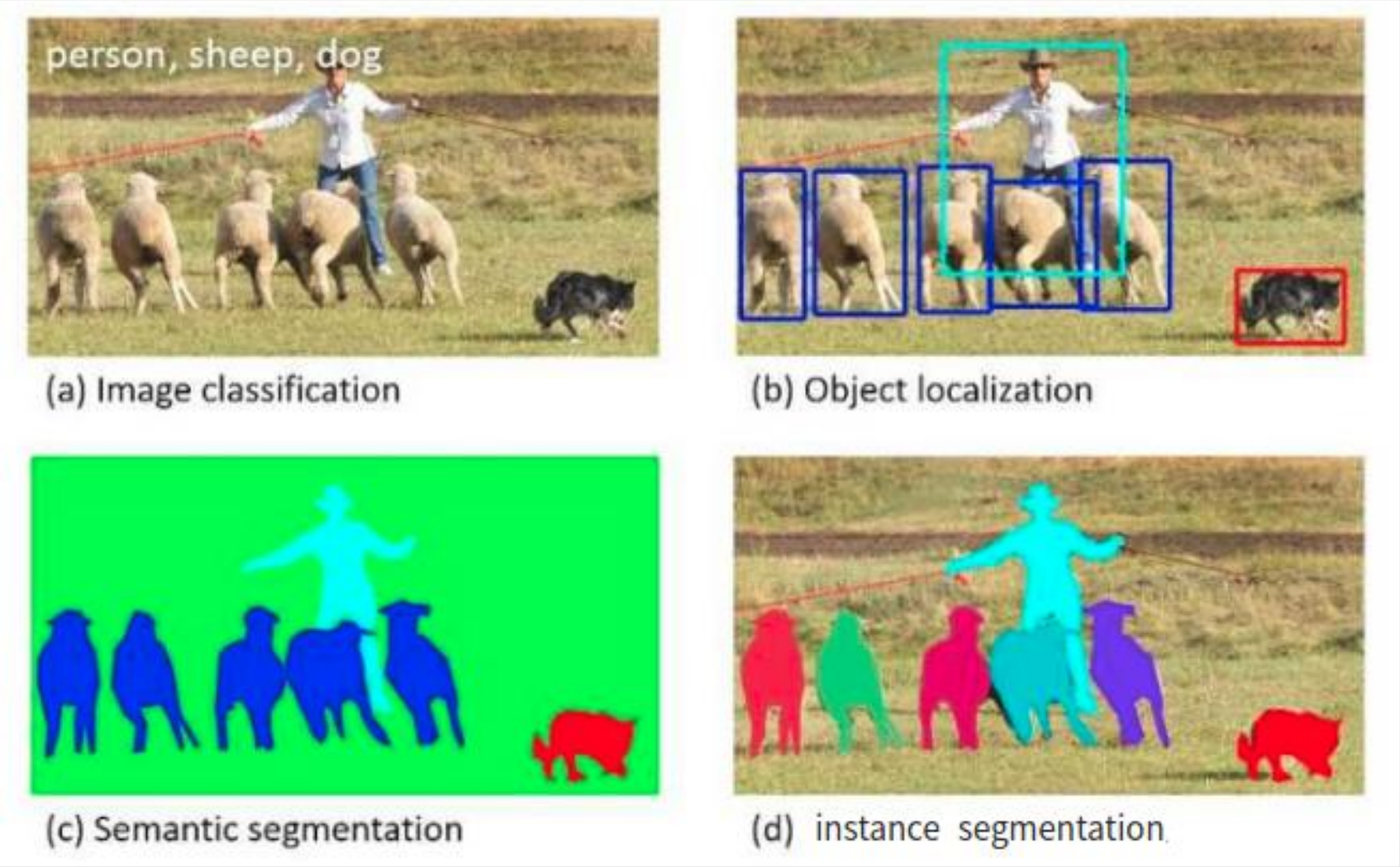


Related Work

- Encoding-Decoding Structure
  - Can be Supervised Learning



Traditional Semantic Segmentation

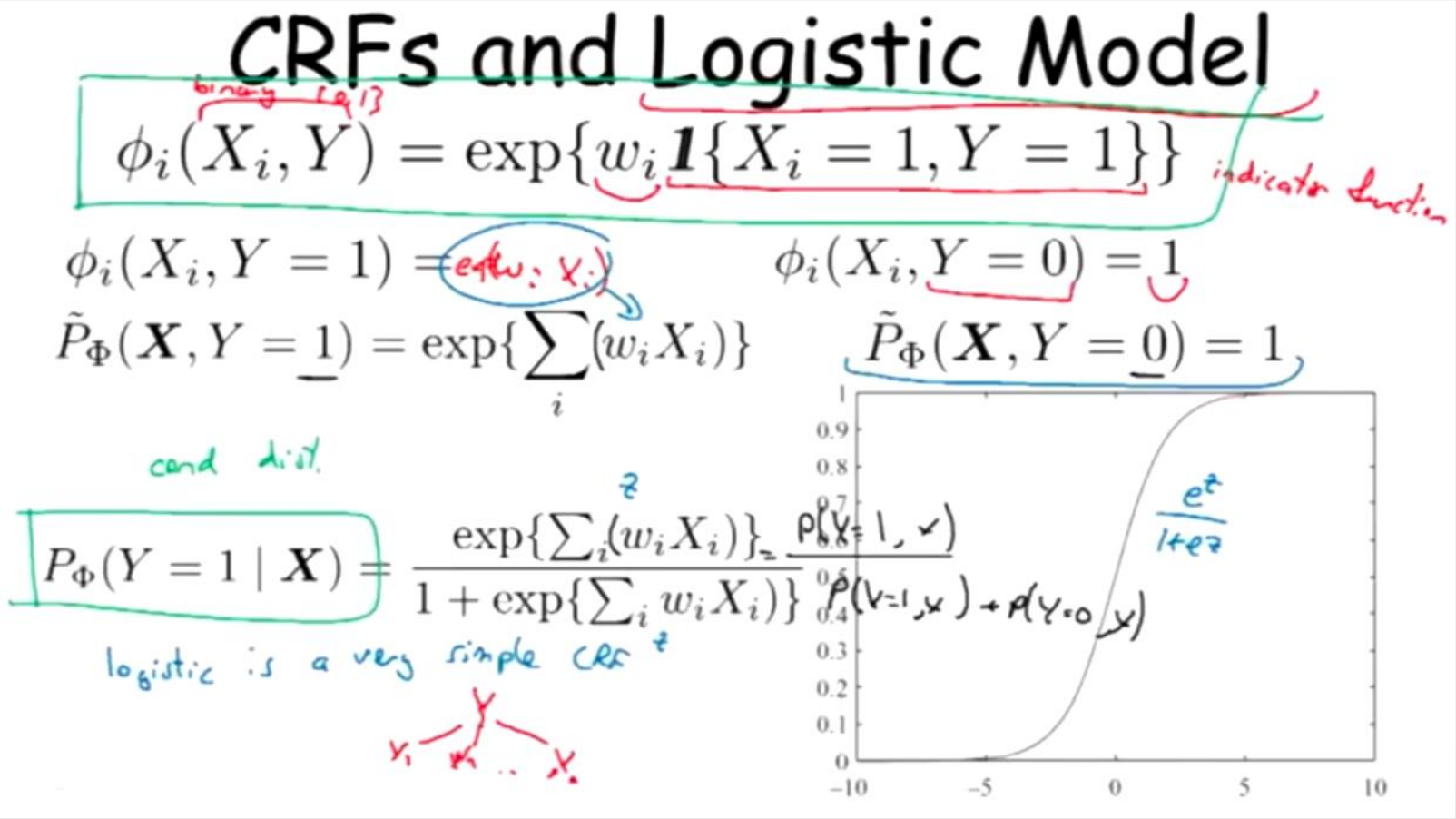


Segmentation

- Blob Detection
  - Without Semantic – Color Difference Based
  - Can be preprocess of semantic segmentation
  - Laplacian Of Gaussian
  - Maximally Stable Extreme Regions
- Region Growing
  - Seed Point and then growning

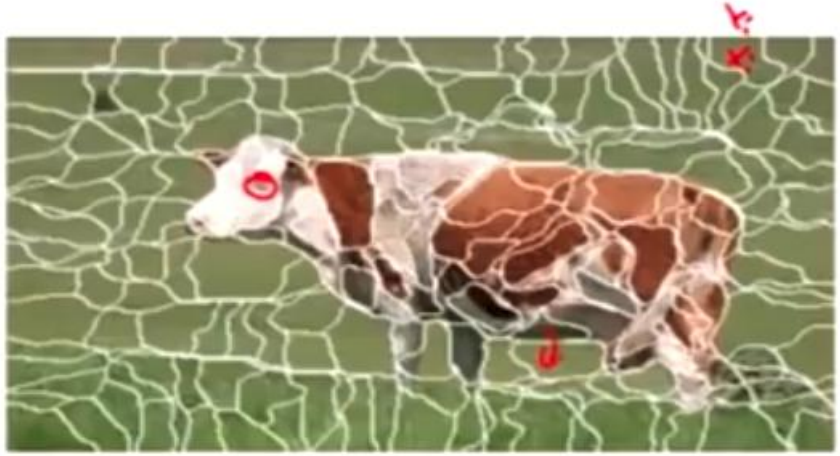
Machine Learning Method

- Conditional Random Field



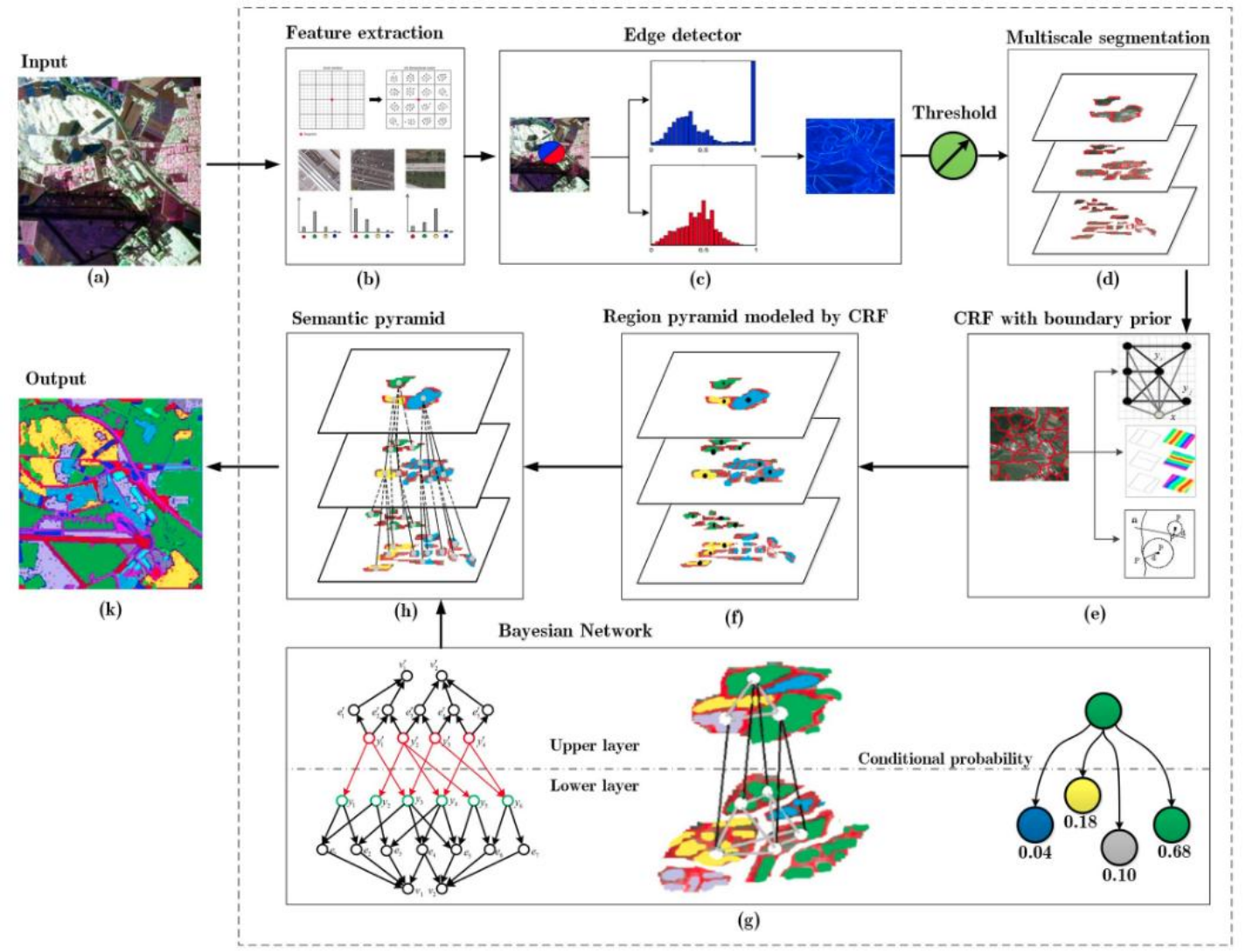
Conditional Random Field

CRFs for Image Segmentation



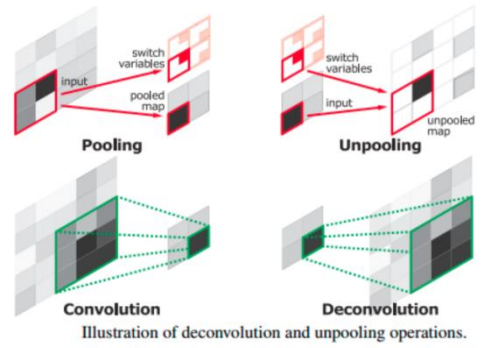
- Node factors can use any features of the image
  - Color histograms
  - Texture features
  - Discriminative patches
- Features can be in or out of the superpixel
- Correlations don't matter
- Can train a discriminative classifier (SVM, boosting) to improve performance

Conditional Random Field

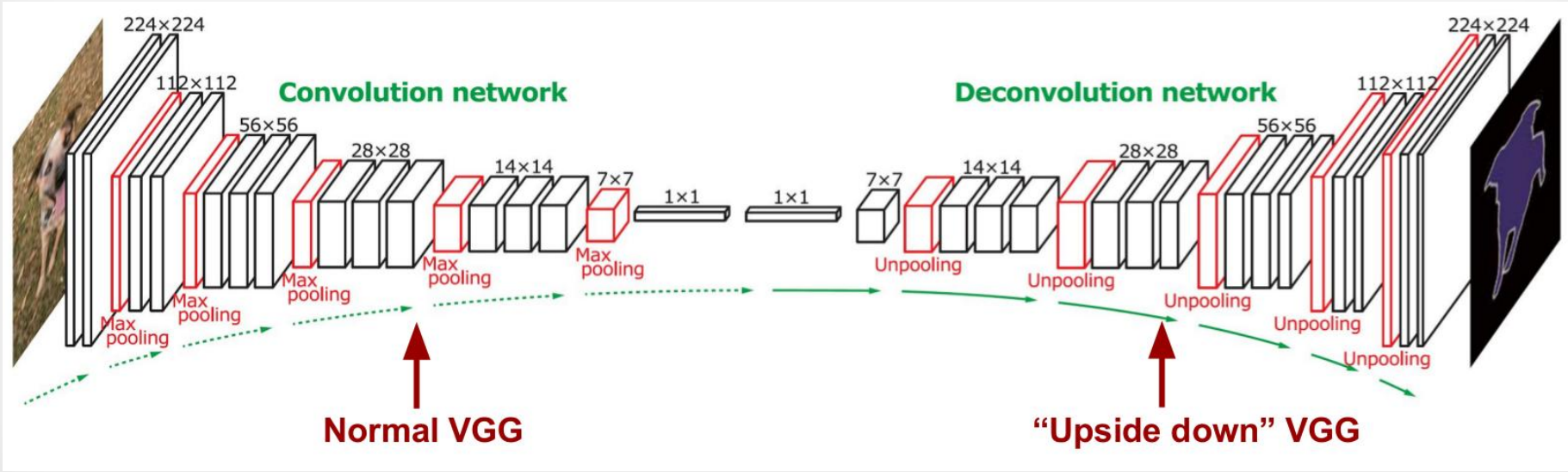


Into the Deep

Just Mirrored



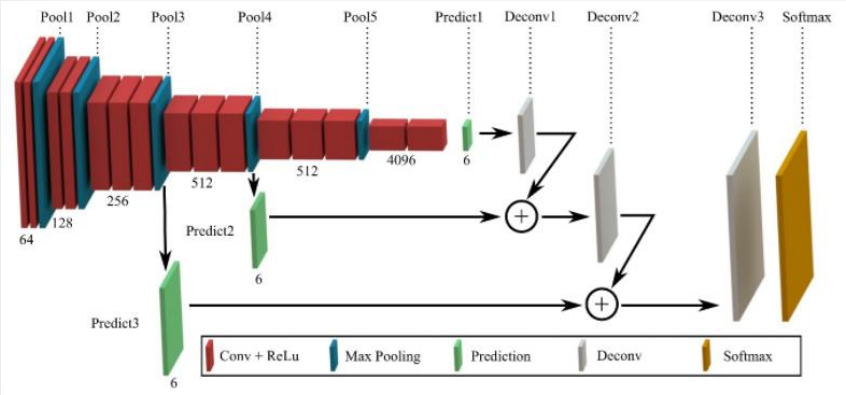
- DeconvNet
  - Learning Deconvolution Network for Semantic Segmentation, ICCV 2015
  - VGG-16(conv+ReLu+MaxPool) + mirrored VGG (Unpooling + deconv+ReLu)
  - Encoder Decoder Architecture: Supervised Learning

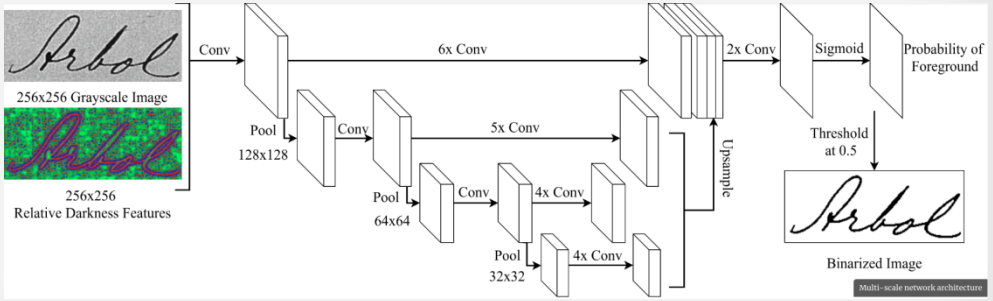


Fully Convolutional Network

- Not New, you can see it is a CNN without FC
- Use Pre-trained Existing Deep CNN, Remove FC and add Deconv
- However, it is an important paper for Semantic Segmentation

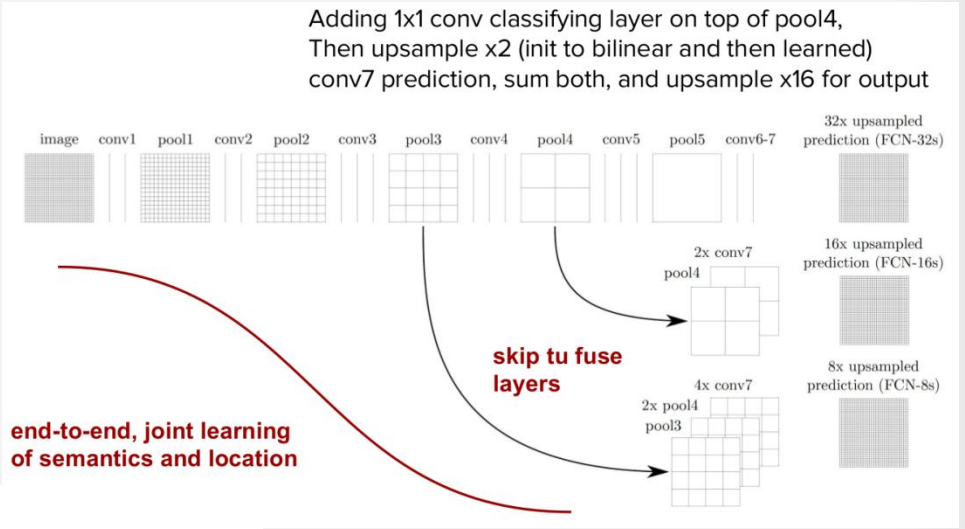
There's no such thing as a "fully convolutional network" as opposed to a "convolutional network" and this is a stupid distinction made by exclusively recent authors.

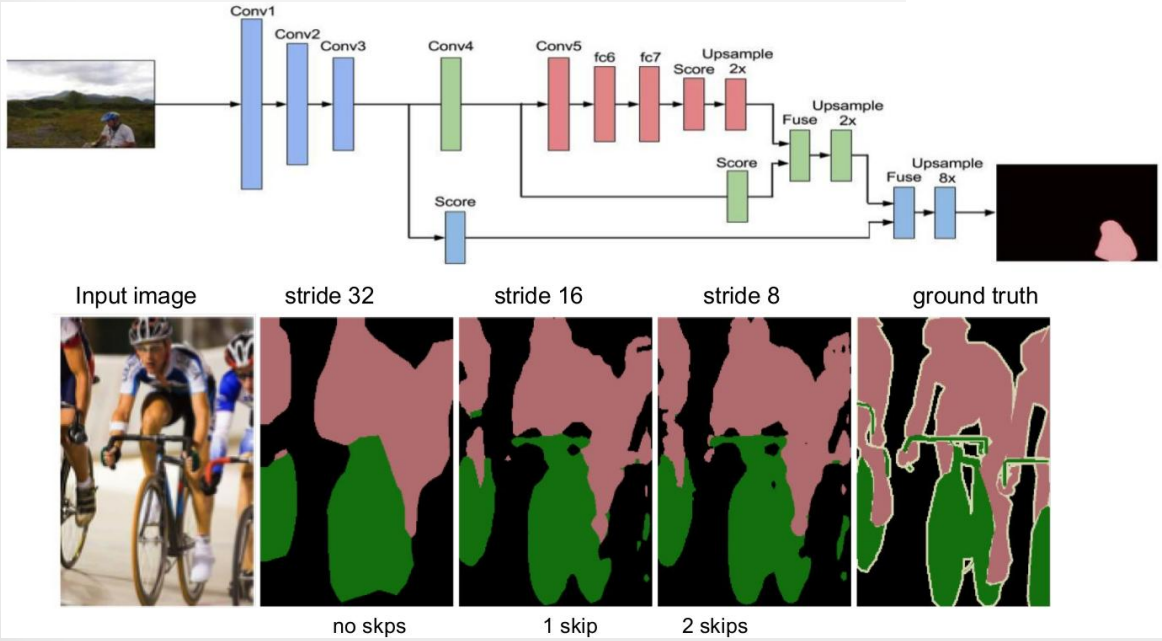




Use Skip Connection

- And add Short cut

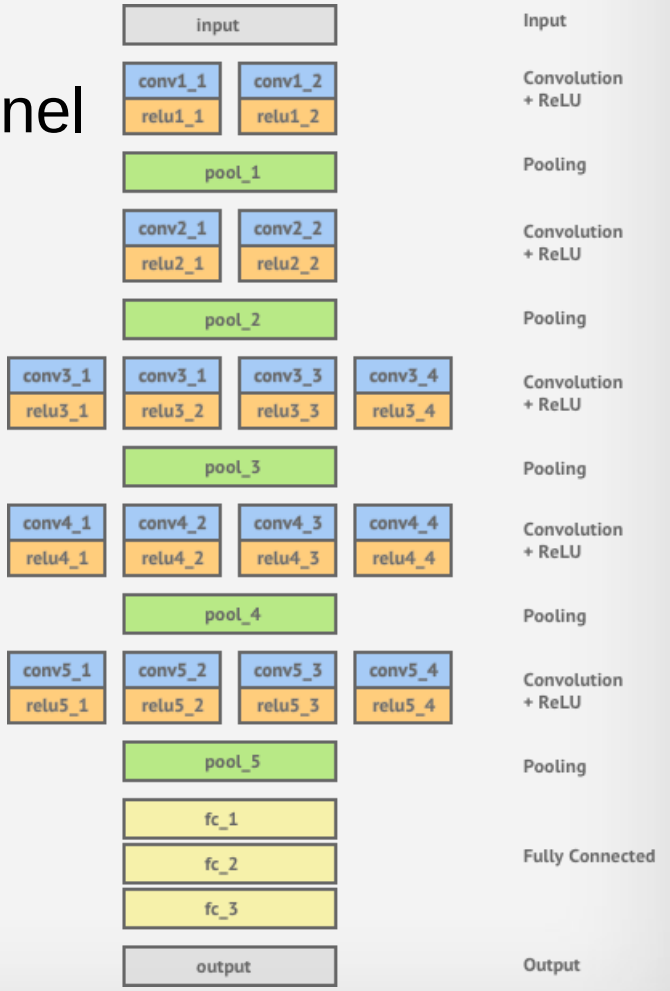




FCN Code

- First Section - Use VGG 19 till pool 5
- Input Image size 224 X 224 X 3 Channel

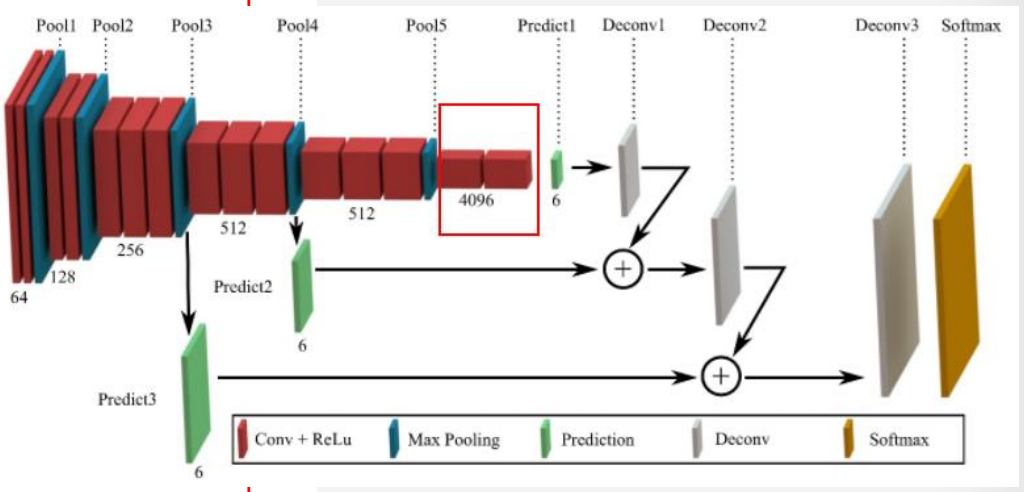
```
MODEL_URL = 'http://www.vlfeat.org/matconvnet/models/beta16/imagenet-vgg-verydeep-19.mat'

MAX_ITERATION = int(1e5 + 1)
NUM_OF_CLASSESS = 151
IMAGE_SIZE = 224


def vgg_net(weights, image):
    layers = (
        'conv1_1', 'relu1_1', 'conv1_2', 'relu1_2', 'pool1',

        'conv2_1', 'relu2_1', 'conv2_2', 'relu2_2', 'pool2',

        'conv3_1', 'relu3_1', 'conv3_2', 'relu3_2', 'conv3_3',
        'relu3_3', 'conv3_4', 'relu3_4', 'pool3',

        'conv4_1', 'relu4_1', 'conv4_2', 'relu4_2', 'conv4_3',
        'relu4_3', 'conv4_4', 'relu4_4', 'pool4',

        'conv5_1', 'relu5_1', 'conv5_2', 'relu5_2', 'conv5_3',
        'relu5_3', 'conv5_4', 'relu5_4'
    )

```



FCN Code

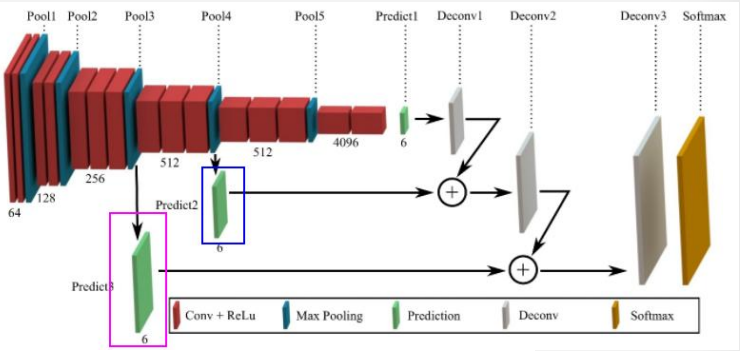
- Second Section: Same Dimension Conv2D

```
with tf.variable_scope("inference"):
    image_net = vgg_net(weights, processed_image)
    conv_final_layer = image_net["conv5_3"]

    pool5 = utils.max_pool_2x2(conv_final_layer)

    W6 = utils.weight_variable([7, 7, 512, 4096], name="W6")
    b6 = utils.bias_variable([4096], name="b6")
    conv6 = utils.conv2d_basic(pool5, W6, b6)
    relu6 = tf.nn.relu(conv6, name="relu6")
    if FLAGS.debug:
        utils.add_activation_summary(relu6)
    relu_dropout6 = tf.nn.dropout(relu6, keep_prob=keep_prob)

    W7 = utils.weight_variable([1, 1, 4096, 4096], name="W7")
    b7 = utils.bias_variable([4096], name="b7")
    conv7 = utils.conv2d_basic(relu_dropout6, W7, b7)
    relu7 = tf.nn.relu(conv7, name="relu7")
    if FLAGS.debug:
        utils.add_activation_summary(relu7)
    relu_dropout7 = tf.nn.dropout(relu7, keep_prob=keep_prob)

    W8 = utils.weight_variable([1, 1, 4096, NUM_OF_CLASSESS], name="W8")
    b8 = utils.bias_variable([NUM_OF_CLASSESS], name="b8")
    conv8 = utils.conv2d_basic(relu_dropout7, W8, b8)
    # annotation_pred1 = tf.argmax(conv8, dimension=3, name="prediction1")

    # now to upscale to actual image size
    deconv_shape1 = image_net["pool4"].get_shape()
    W_t1 = utils.weight_variable([4, 4, deconv_shape1[3].value, NUM_OF_CLASSESS], name="W_t1")
    b_t1 = utils.bias_variable([deconv_shape1[3].value], name="b_t1")
    conv_t1 = utils.conv2d_transpose_strided(conv8, W_t1, b_t1, output_shape=tf.shape(image_net["pool4"]))
    fuse_1 = tf.add(conv_t1, image_net["pool4"], name="fuse_1")
```



FCN Code

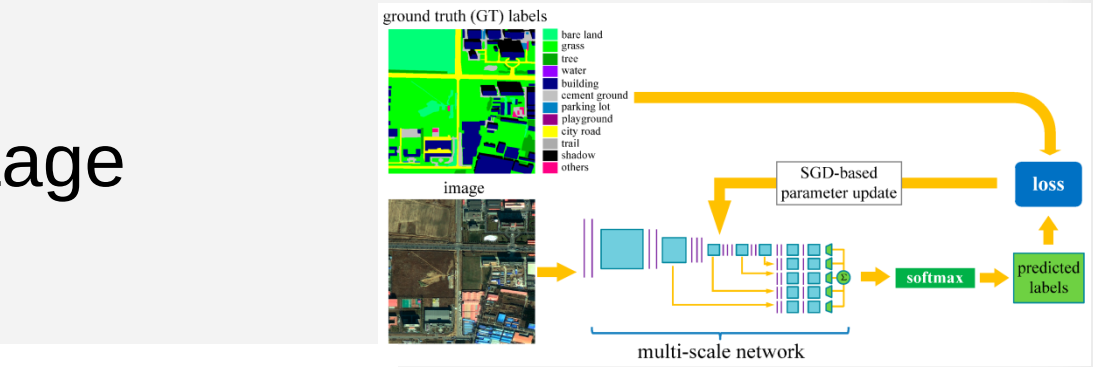
- Last Section - Upscale

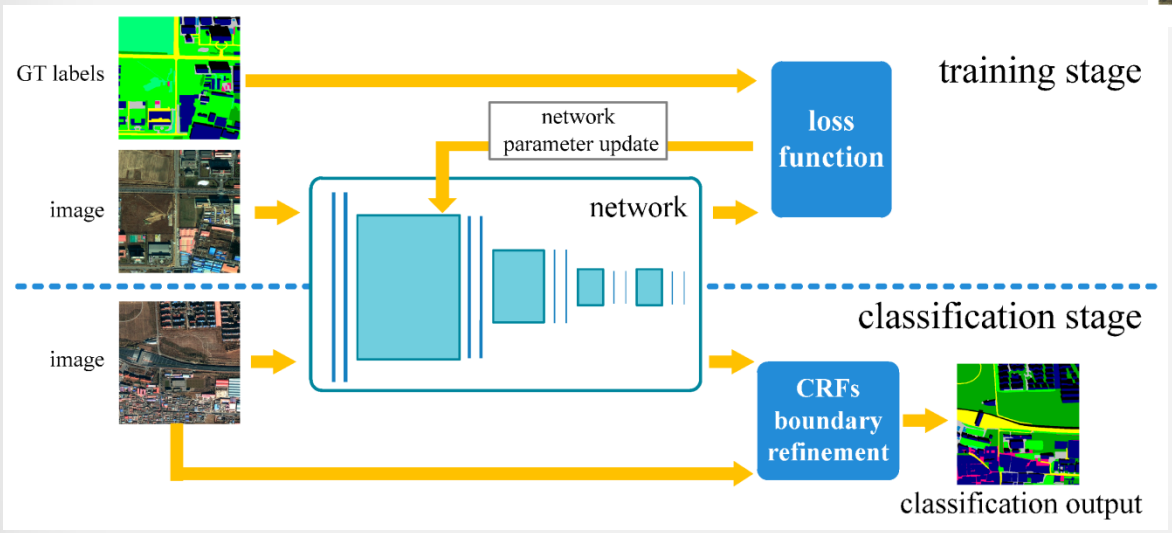



```
# now to upscale to actual image size
deconv_shape1 = image_net["pool4"].get_shape()
W_t1 = utils.weight_variable([4, 4, deconv_shape1[3].value, NUM_OF_CLASSESS], name="W_t1")
b_t1 = utils.bias_variable([deconv_shape1[3].value], name="b_t1")
conv_t1 = utils.conv2d_transpose_strided(conv8, W_t1, b_t1, output_shape=tf.shape(image_net["pool4"]))
fuse_1 = tf.add(conv_t1, image_net["pool4"], name="fuse_1")

deconv_shape2 = image_net["pool3"].get_shape()
W_t2 = utils.weight_variable([4, 4, deconv_shape2[3].value, deconv_shape1[3].value], name="W_t2")
b_t2 = utils.bias_variable([deconv_shape2[3].value], name="b_t2")
conv_t2 = utils.conv2d_transpose_strided(fuse_1, W_t2, b_t2, output_shape=tf.shape(image_net["pool3"]))
fuse_2 = tf.add(conv_t2, image_net["pool3"], name="fuse_2")

shape = tf.shape(image)
deconv_shape3 = tf.stack([shape[0], shape[1], shape[2], NUM_OF_CLASSESS])
W_t3 = utils.weight_variable([16, 16, NUM_OF_CLASSESS, deconv_shape2[3].value], name="W_t3")
b_t3 = utils.bias_variable([NUM_OF_CLASSESS], name="b_t3")
conv_t3 = utils.conv2d_transpose_strided(fuse_2, W_t3, b_t3, output_shape=deconv_shape3, stride=8)

annotation_pred = tf.argmax(conv_t3, dimension=3, name="prediction")
```

Deep Learning Structure

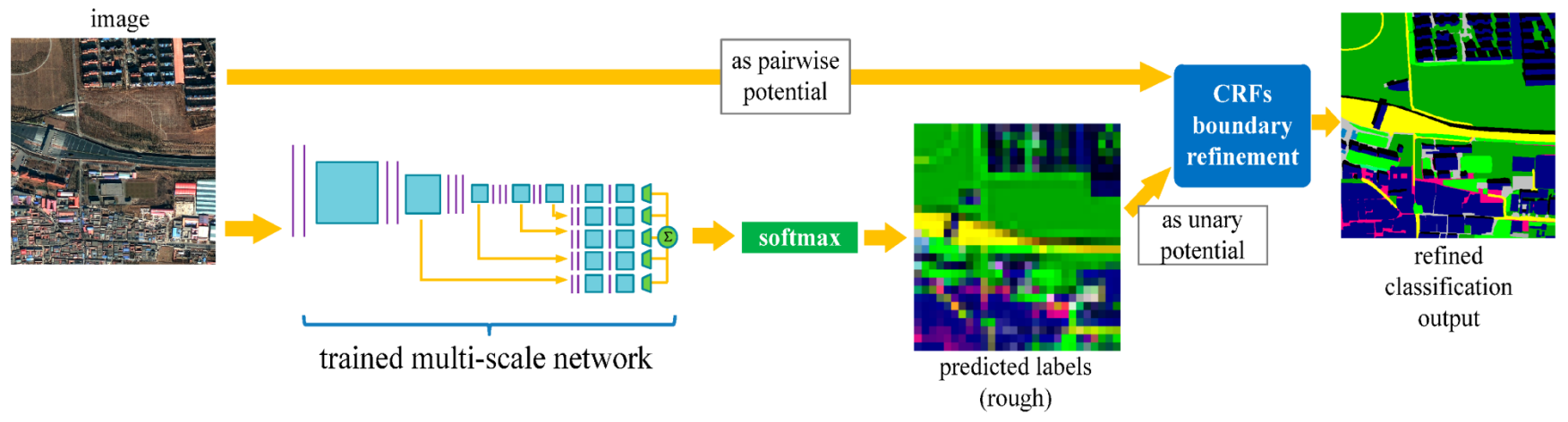
- Deep Learning Structure - Supervised
  - Training stage
  - Classification stage





Typical Architecture of Semantic Segmentation in Deep Learning

- Downsampling path: extract coarse features
- Upsampling path: recover input image resolution
- Skip connections: recovers detailed information
- Post-processing(optional): refine predictions



Another Famous Structure

- U-Net: Winner of
  - CAD Caries challenge ISBI 2015
  - Cell tracking challenge ISBI 2015

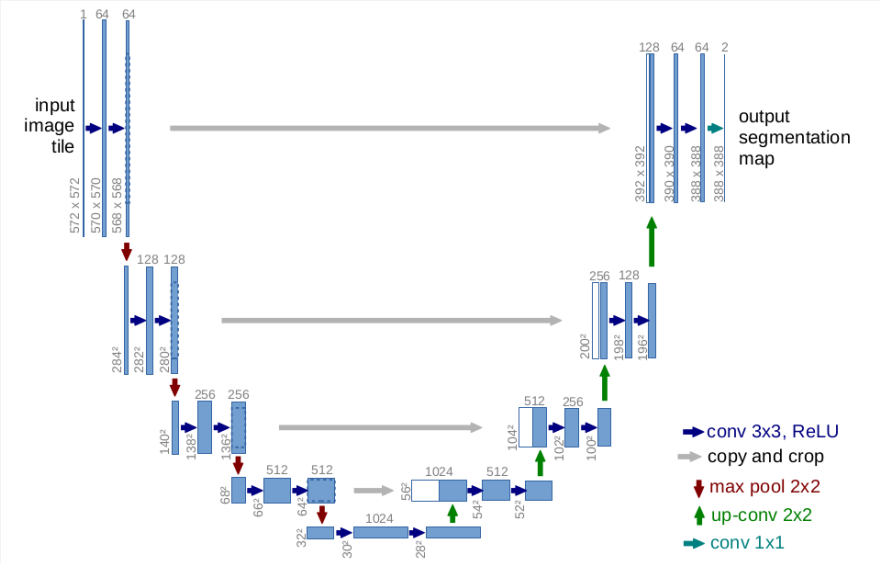
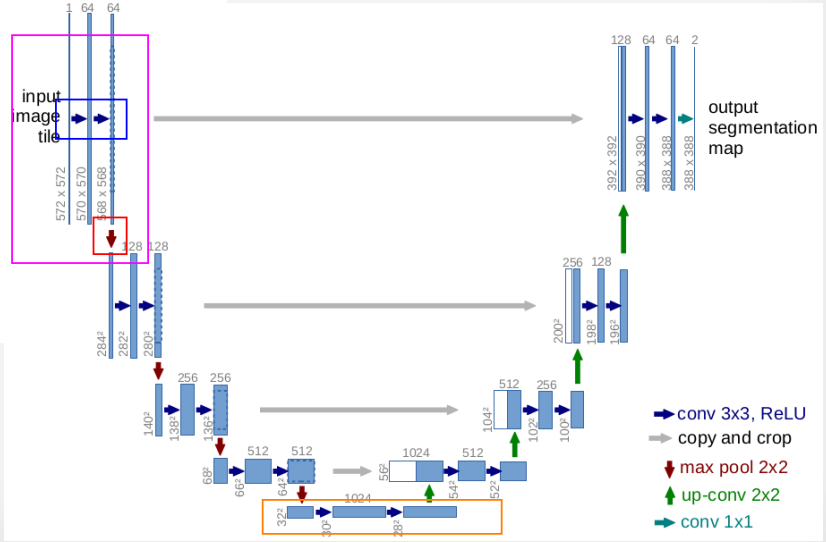


Fig. 1. U-net architecture (example for 32x32 pixels in the lowest resolution). Each blue box corresponds to a multi-channel feature map. The number of channels is denoted on top of the box. The x-y-size is provided at the lower left edge of the box. White boxes represent copied feature maps. The arrows denote the different operations.

U-Net Code

- Contracting / Encoding / Down Layers

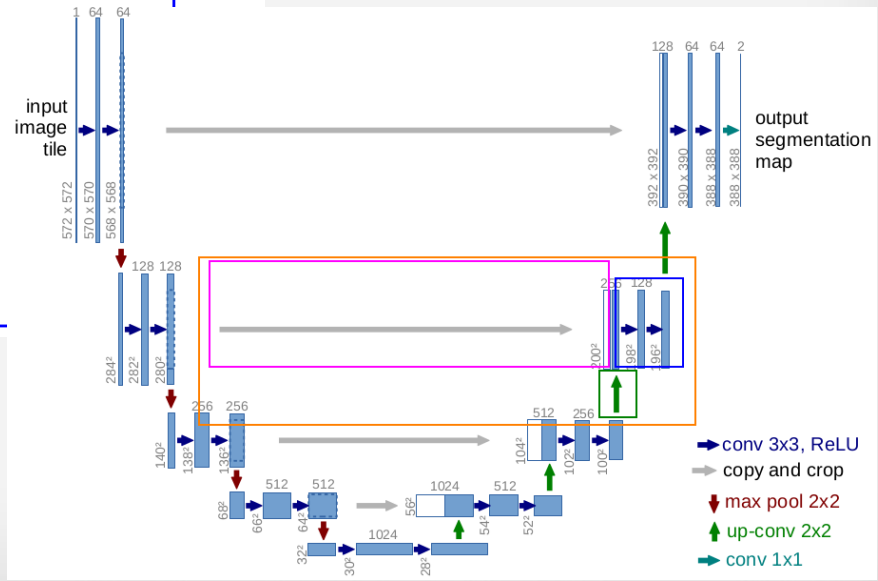
```
# down layers
for layer in range(0, layers):
    features = 2**layer*features_root
    stddev = np.sqrt(2 / (filter_size**2 * features))
    if layer == 0:
        w1 = weight_variable([filter_size, filter_size, channels, features], stddev)
    else:
        w1 = weight_variable([filter_size, filter_size, features//2, features], stddev)

    w2 = weight_variable([filter_size, filter_size, features, features], stddev)
    b1 = bias_variable([features])
    b2 = bias_variable([features])

    conv1 = conv2d(in_node, w1, keep_prob)
    tmp_h_conv = tf.nn.relu(conv1 + b1)
    conv2 = conv2d(tmp_h_conv, w2, keep_prob)
    dw_h_convs[layer] = tf.nn.relu(conv2 + b2)

    weights.append((w1, w2))
    biases.append((b1, b2))
    convs.append((conv1, conv2))

    size -= 4
    if layer < layers-1:
        pools[layer] = max_pool(dw_h_convs[layer], pool_size)
        in_node = pools[layer]
        size /= 2
```



U-Net Code

- Expansive / Decoding / Up Layers

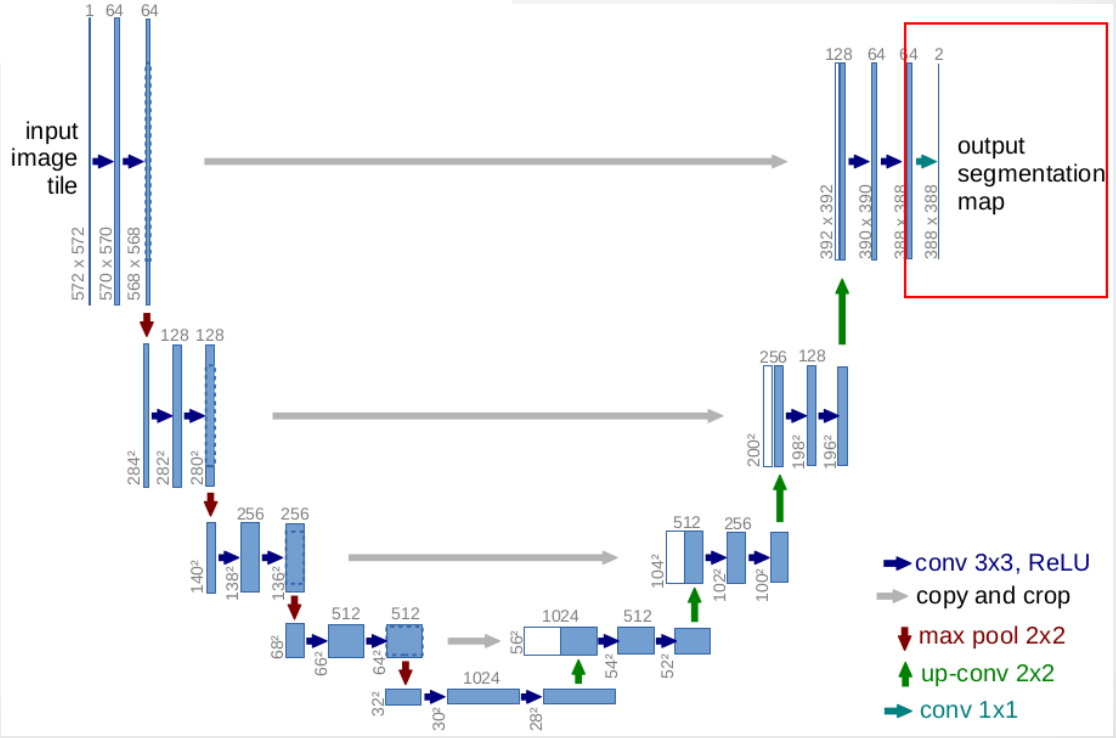
```
# up layers
for layer in range(layers-2, -1, -1):
    features = 2**(layer+1)*features_root
    stddev = np.sqrt(2 / (filter_size**2 * features))

    wd = weight_variable_devonc([pool_size, pool_size, features//2, features], stddev)
    bd = bias_variable([features//2])
    h_deconv = tf.nn.relu(deconv2d(in_node, wd, pool_size) + bd)
    h_deconv_concat = crop_and_concat(dw_h_convs[layer], h_deconv)
    deconv[layer] = h_deconv_concat

    w1 = weight_variable([filter_size, filter_size, features, features//2], stddev)
    w2 = weight_variable([filter_size, filter_size, features//2, features//2], stddev)
    b1 = bias_variable([features//2])
    b2 = bias_variable([features//2])

    conv1 = conv2d(h_deconv_concat, w1, keep_prob)
    h_conv = tf.nn.relu(conv1 + b1)
    conv2 = conv2d(h_conv, w2, keep_prob)
    in_node = tf.nn.relu(conv2 + b2)
    up_h_convs[layer] = in_node

    weights.append((w1, w2))
    biases.append((b1, b2))
    convs.append((conv1, conv2))

    size *= 2
    size -= 4
```



U-Net Code

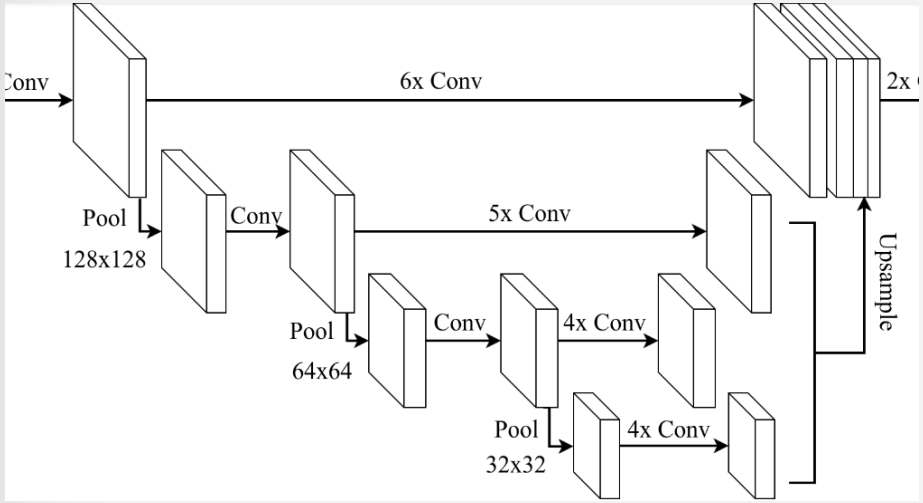
- Output Map

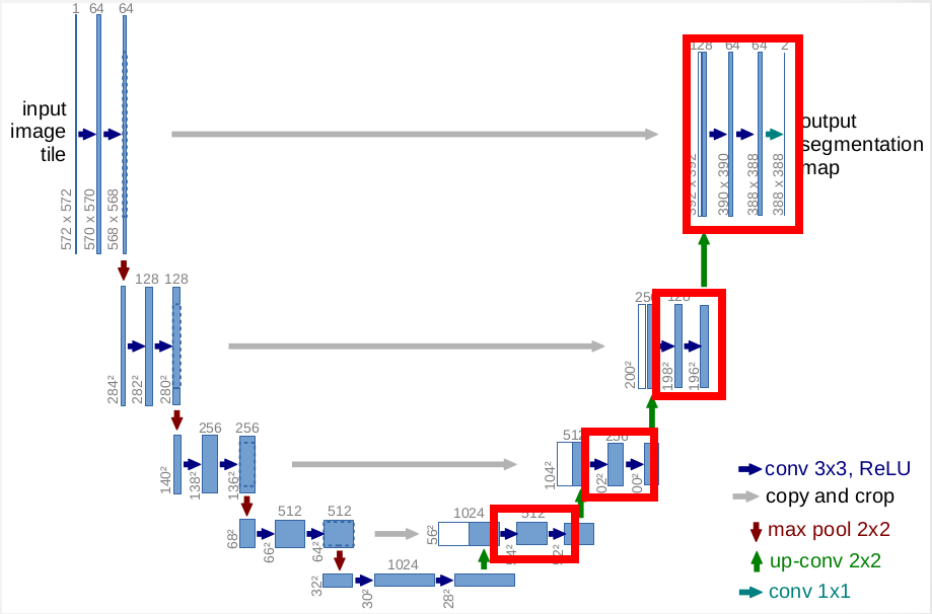
```
# Output Map
weight = weight_variable([1, 1, features_root, n_class], stddev)
bias = bias_variable([n_class])
conv = conv2d(in_node, weight, tf.constant(1.0))
output_map = tf.nn.relu(conv + bias)
up_h_convs["out"] = output_map
```



Compare

- U-Net VS FCN
  - Add convolutions in the updampling path (symmetric net)
  - There is NO convolutions in the updampling path in FCN





Compare

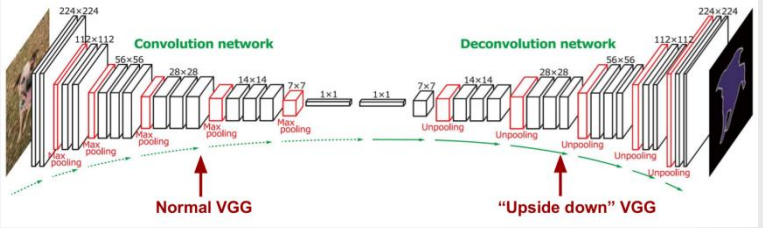
- Short Cut Method: U-Net VS FCN
  - FCN: Add

```
# now to upscale to actual image size
deconv_shape1 = image_net["pool4"].get_shape()
W_t1 = utils.weight_variable([4, 4, deconv_shape1[3].value, NUM_OF_CLASSESS], name="W_t1")
b_t1 = utils.bias_variable([deconv_shape1[3].value], name="b_t1")
conv_t1 = utils.conv2d_transpose_strided(conv8, W_t1, b_t1, output_shape=tf.shape(image_net["pool4"]))
fuse_1 = tf.add(conv_t1, image_net["pool4"], name="fuse_1")

deconv_shape2 = image_net["pool3"].get_shape()
W_t2 = utils.weight_variable([4, 4, deconv_shape2[3].value, deconv_shape1[3].value], name="W_t2")
b_t2 = utils.bias_variable([deconv_shape2[3].value], name="b_t2")
conv_t2 = utils.conv2d_transpose_strided(fuse_1, W_t2, b_t2, output_shape=tf.shape(image_net["pool3"]))
fuse_2 = tf.add(conv_t2, image_net["pool3"], name="fuse_2")
```

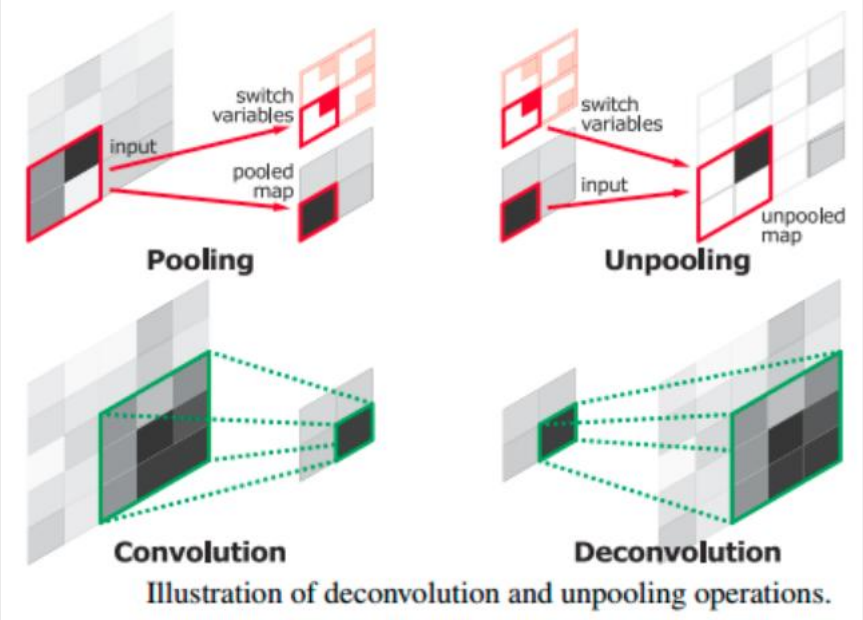
  - U-Net: Crop and Concat

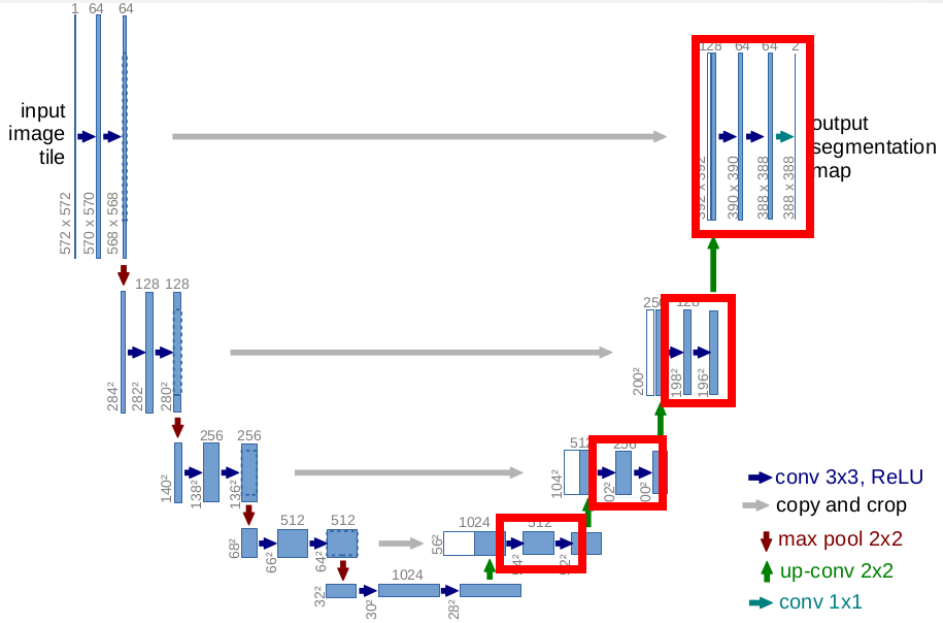
```
def crop_and_concat(x1,x2):
    x1_shape = tf.shape(x1)
    x2_shape = tf.shape(x2)
    # offsets for the top left corner of the crop
    offsets = [0, (x1_shape[1] - x2_shape[1]) // 2, (x1_shape[2] - x2_shape[2]) // 2, 0]
    size = [-1, x2_shape[1], x2_shape[2], -1]
    x1_crop = tf.slice(x1, offsets, size)
    return tf.concat([x1_crop, x2], 3)
```

Compare



- U-Net VS DeconvNet
  - U-Net Add convolutions in the up sampling path (symmetric net)
  - DeconvNet Use Deconvolution (mirror net), where U-Net does NOT (But used in upscale: up-conv 2X2)





Other Tricks in U-Net

- Augmentation: Overlap-tile Strategy
- Loss Function: Weight Map with DIC
- Parameters Initialize: Gaussian Distribution

In deep networks with many convolutional layers and different paths through the network, a good initialization of the weights is extremely important. Otherwise, parts of the network might give excessive activations, while other parts never contribute. Ideally the initial weights should be adapted such that each feature map in the network has approximately unit variance. For a network with our architecture (alternating convolution and ReLU layers) this can be achieved by drawing the initial weights from a Gaussian distribution with a standard deviation of $\sqrt{2/N}$, where N denotes the number of incoming nodes of one neuron [5]. E.g. for a 3x3 convolution and 64 feature channels in the previous layer $N = 9 \cdot 64 = 576$.

SegNet

- Architecture
  - Max Unpooling

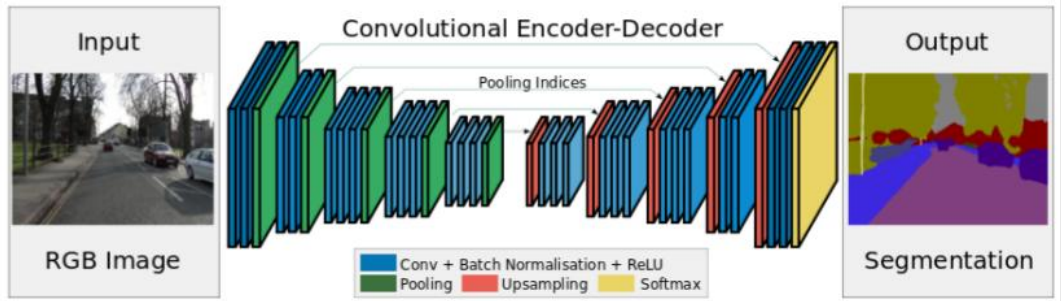


Figure : The SegNet Architecture

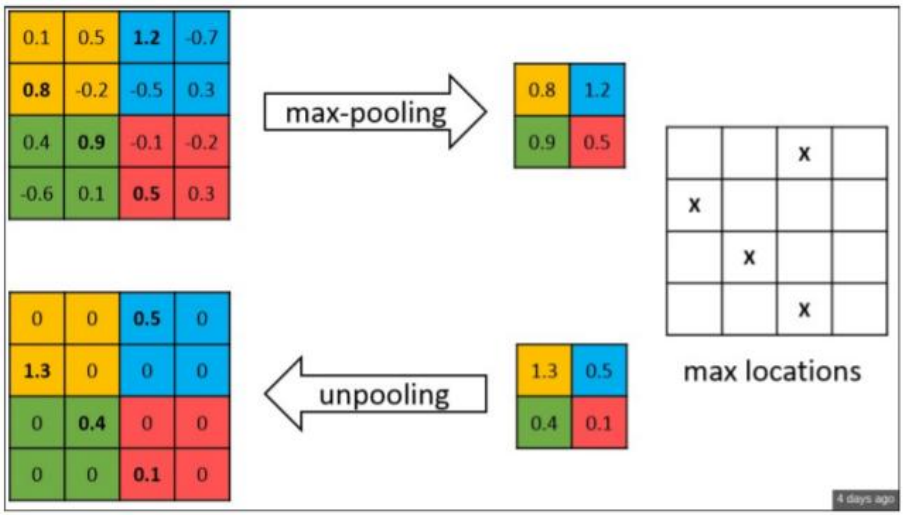
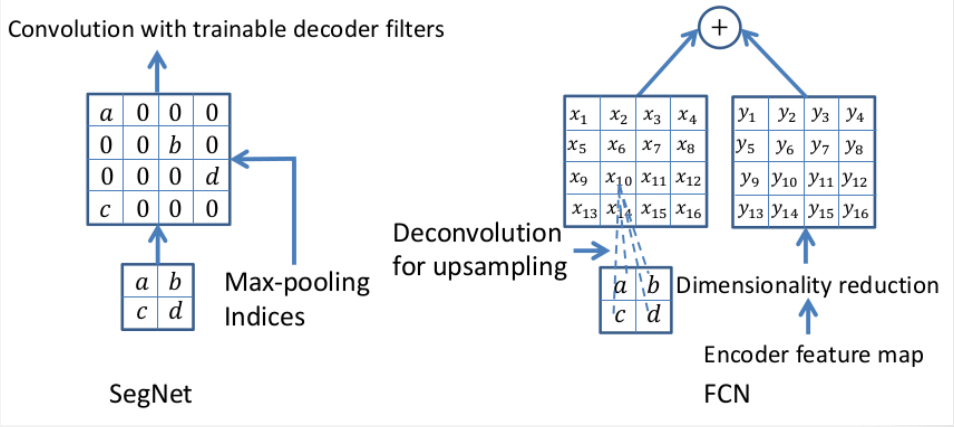
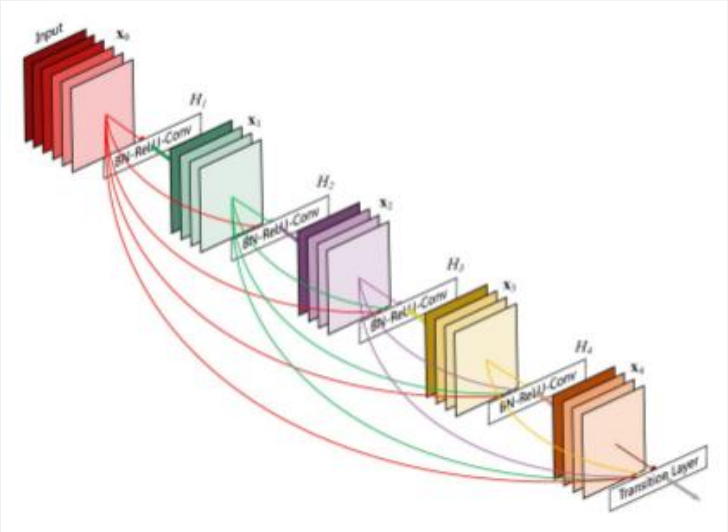


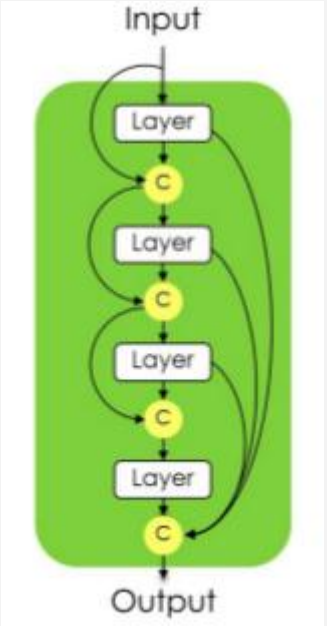
Figure : Max Unpooling

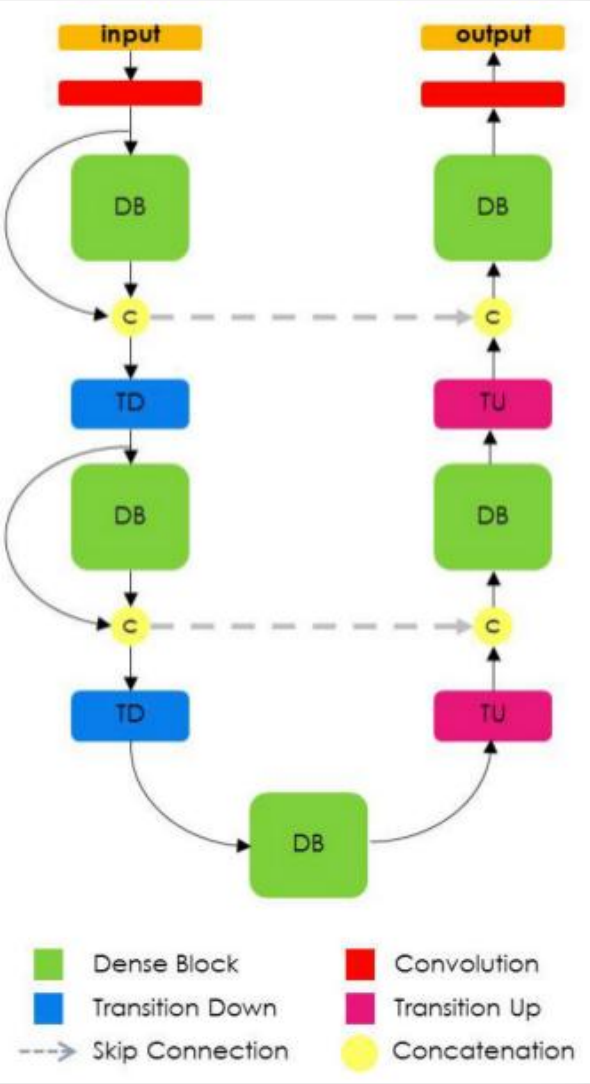


Fully Convolutional DenseNets

- Dense block
  - Dense Short cut

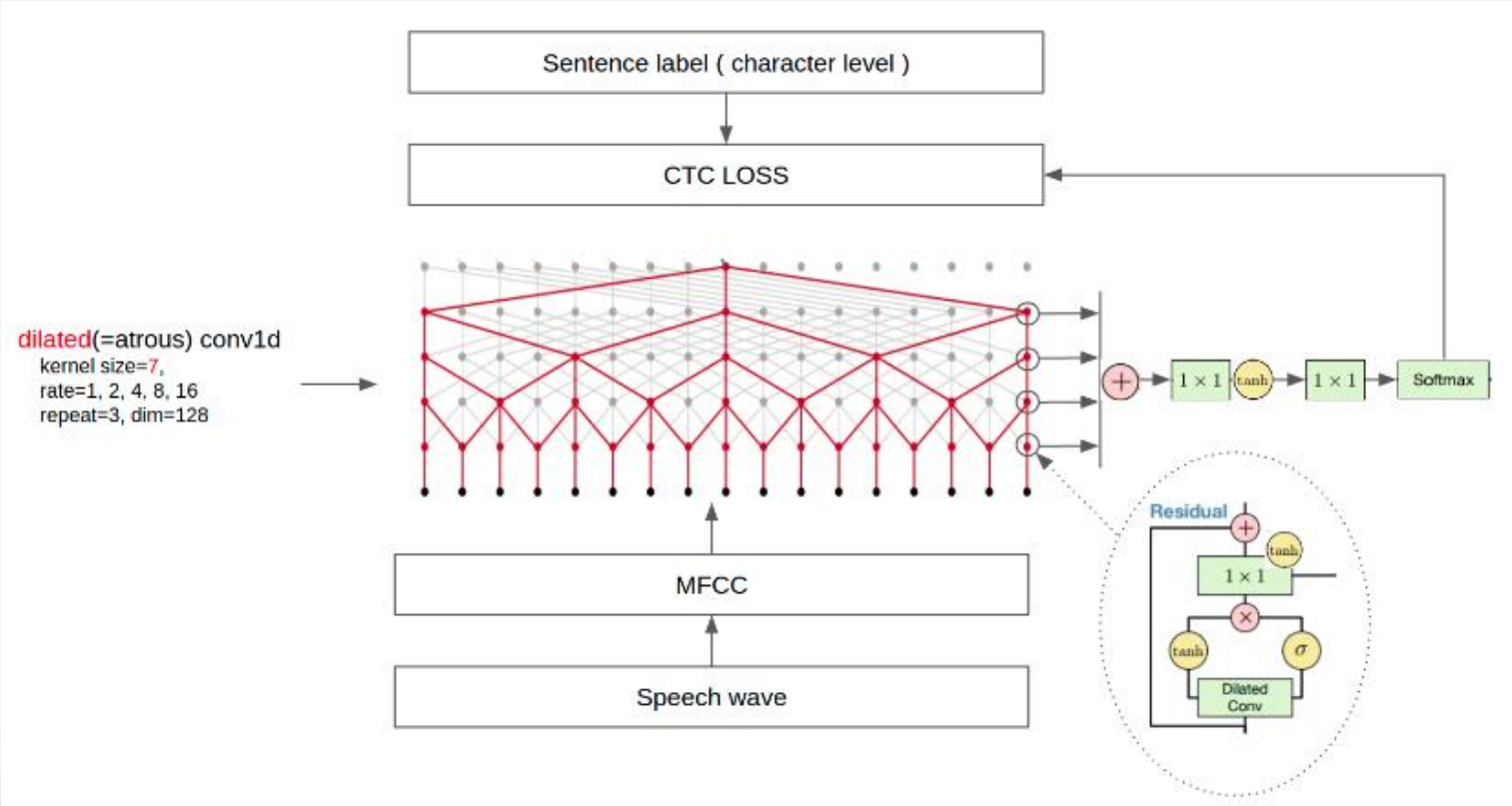






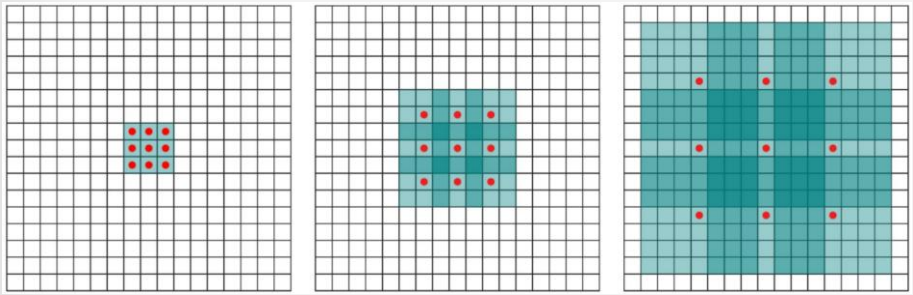
Dilated Convolution

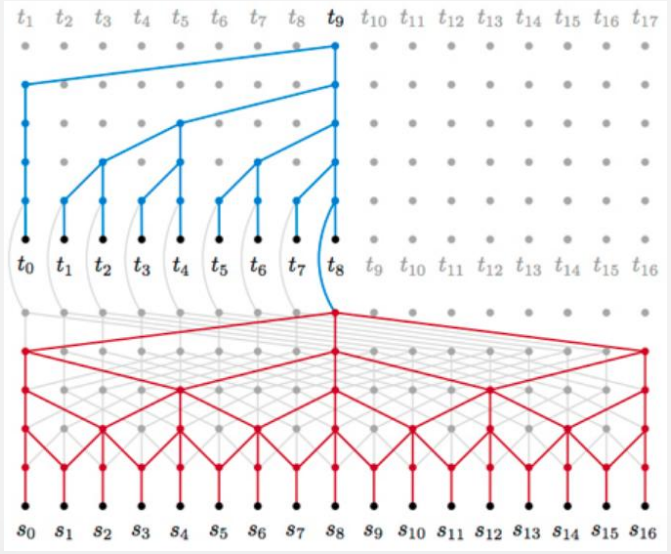
- DeepLab, DilatedNet and Refine Net
  - 1D Example: Speech-to-Text-WaveNet



Dilated Convolutional

- 2D Example





S: stride ; k: kernel (filter)

$F(0) = 1$

$F(1) = F(0) + (S+1)^{1-1} \times (k-1)$

$F(2) = F(1) + (S+1)^{2-1} \times (k-1)$

$F(i) = F(i-1) + (S+1)^{i-1} \times (k-1)$

$F(n) = F(n-1) + (S+1)^{n-1} \times (k-1)$

$F(n+1) = F(n) + (S+1)^{n} \times (k-1)$

$= 1 + (S+1)^{1-1}(k-1) + \cdots + (S+1)^{n}(k-1)$

$= 1 + (k-1) \sum_{i=1}^{n+1} (S+1)^{n-1}$

$= 1 + (k-1) \times \frac{(S+1)^{0}\left[(S+1)^{n+1} - 1\right]}{(S+1) - 1}$

$= 1 + (k-1) \times \frac{(S+1)^{n+1} - 1}{S}$

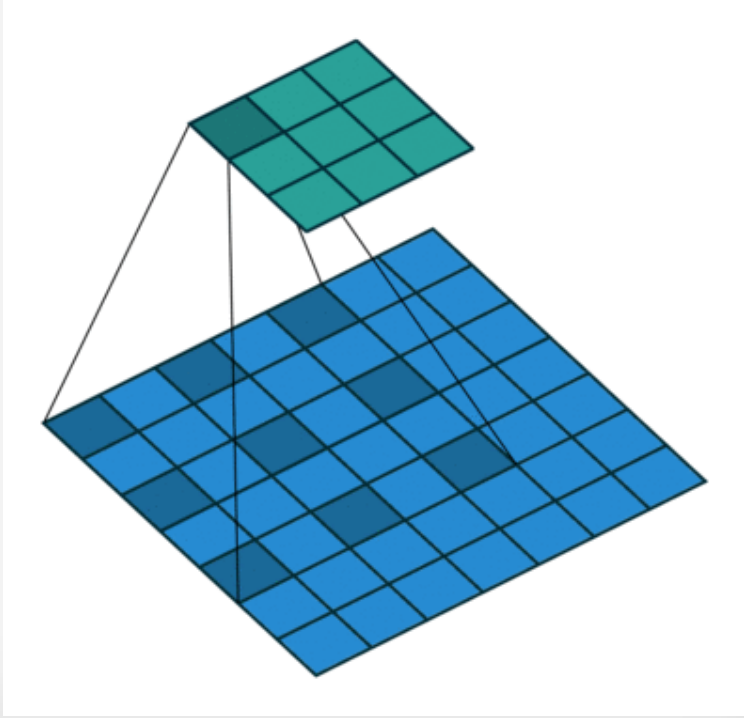
$k=3$, $S=1$ 代入

$F(n+1) = 1 + 2 \times \frac{2^{n+1} - 1}{S} = 1 + 2 \times \frac{2^{n+1} - 1}{1}$

$= 2^{n+2} - 1$

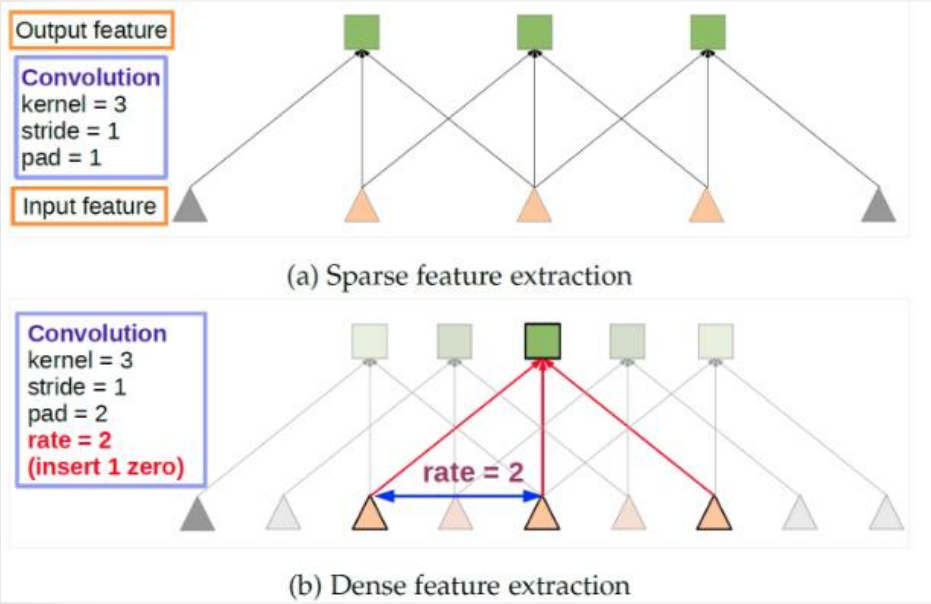
Dilated Convolutional

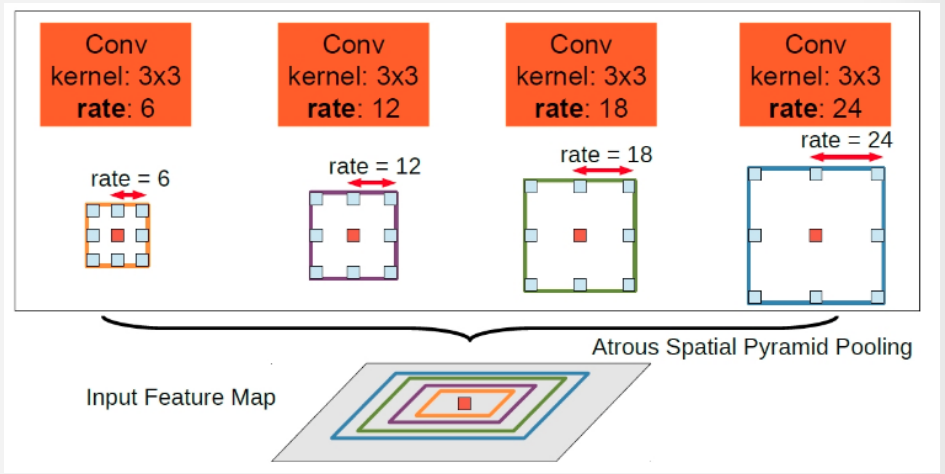
- 2D Example
  - https://github.com/vdumoulin/conv_arithmetic
  - Dilated / Atrous Convolutional
  - Meaning in CV
    - Detect Scaled Feature
    - Which Less Loss
      - Compared to
      - filter with smaller image



Atrous Convolution and DeepLab

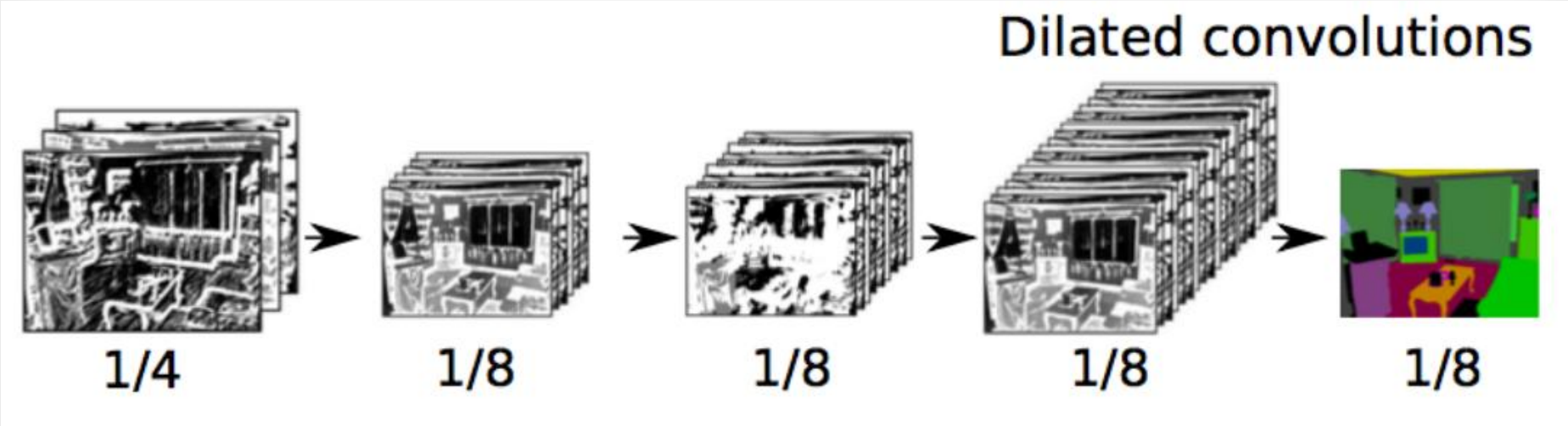
- Similar to Dilated Convolutional but defined pad
- DeepLab = atrous convolutions + spatial pyramid + CRF





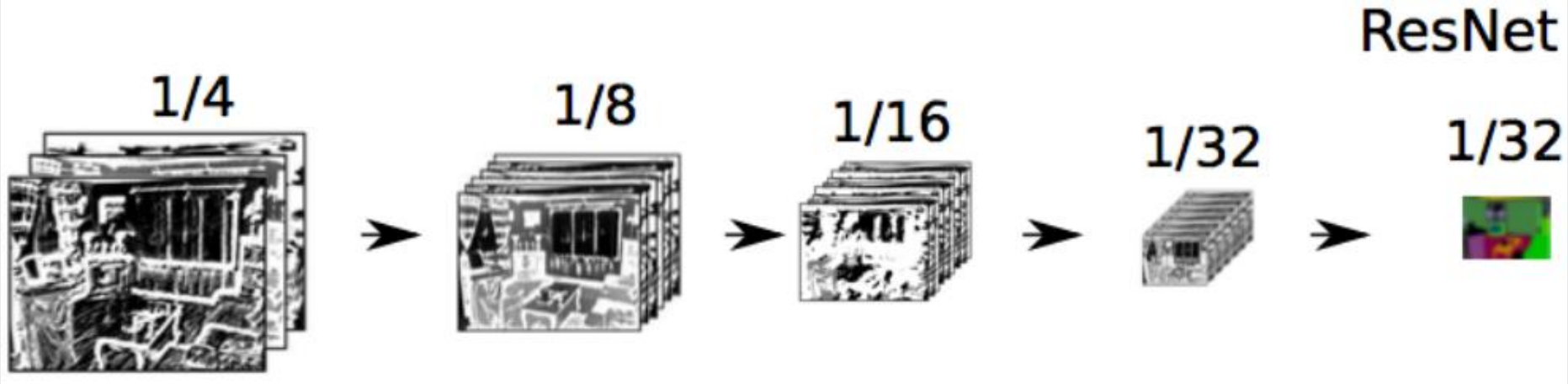
Different of DilatedNet and RefineNet

- DilatedNet
  - Only Apply DilatedNet in Previous Net
  - Till 1/8, otherwise traditional convolutional
  - You can think only apply RefineNet before 1/8



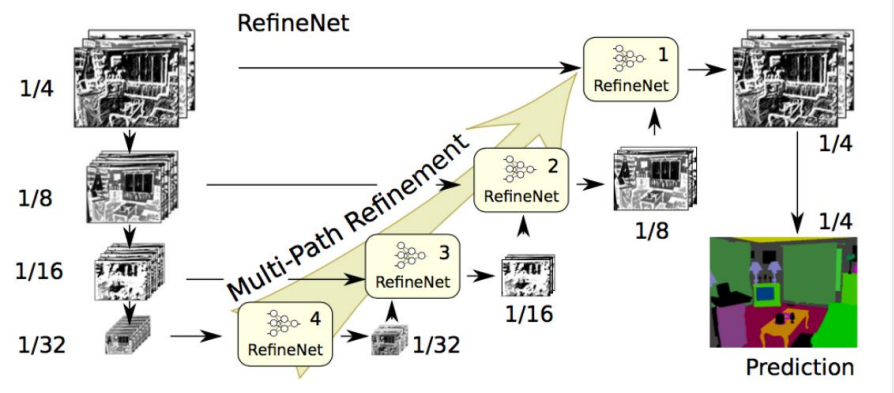
Different of DilatedNet and RefineNet

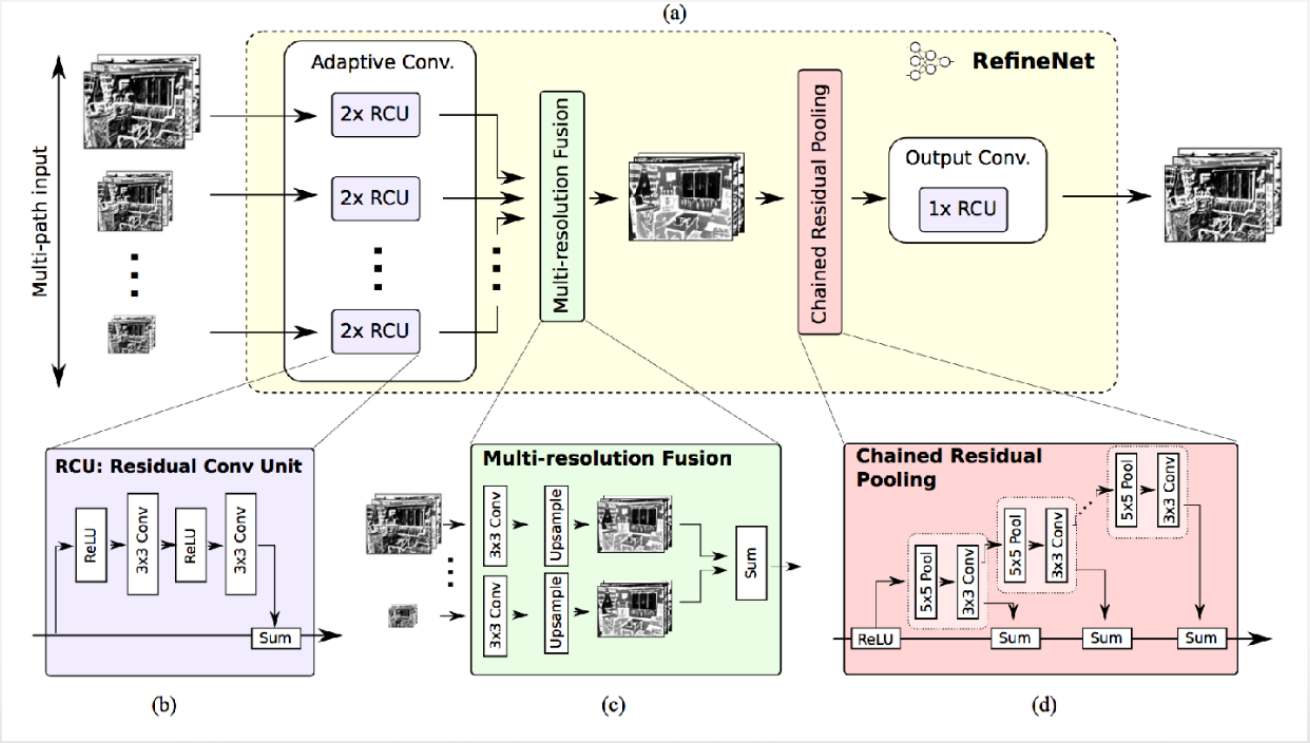
- RefineNet
  - All Dilated Convolutional
  - 1/32, Maybe due to structure / image size



RefineNet

- Structure





Conclusion

- More Structure
  - https://github.com/mrgloom/awesome-semantic-segmentation
- My Experience: FCN and U-Net
  - DataSet: Ultrasound Nerve Segmentation
  - https://www.kaggle.com/c/ultrasound-nerve-segmentation/data
  - Not good enough due to time and computing limitation
  - Deeper is more important than filter number

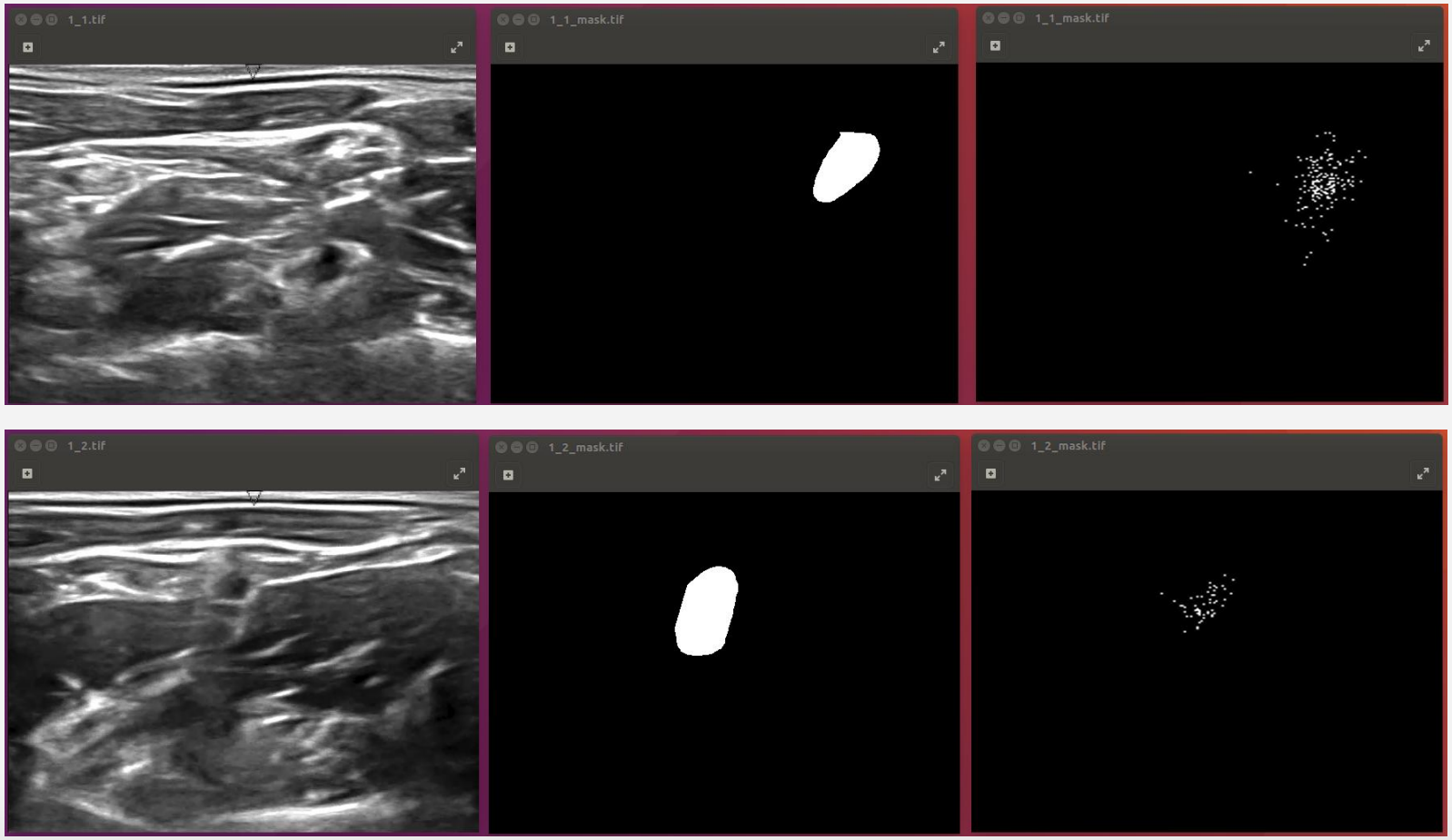
FCN Result

- Settings
  - Input X: resize to 224X224X3
  - Input Y: 224X224X1 Integer for Class
  - In this project, Class Number is 2
  - 0 for background and 1 for hilight
  - Training Epoch is 200 due to time limits

```
""" Assign Values """
""" Org Image """
OrgWidth = 580
OrgHight = 420
OrgChannel = 3
""" Dst Image """
DstWidth = 224
DstHeight = 224
DstChanel = 3
""" HyperParameters """
DropOutKeep = 0.8
LearnRate = 0.000001
BatchSize = 32
NumEpoch = 200
""" Otehr Parameters """
""" Number Of Class """
NumOfClass = 2
""" File System """
FolderToTrain = "./train/"
FolderToTest = "./test/"
FolderToTestOut = "./testout/"
FileTailToCheck = ".tif"
MaskKeyWord = "_mask"
""" FCN Train Result """
FcnLog = "./fcn_log/"
FcnModel = "./fcn_model/"
```

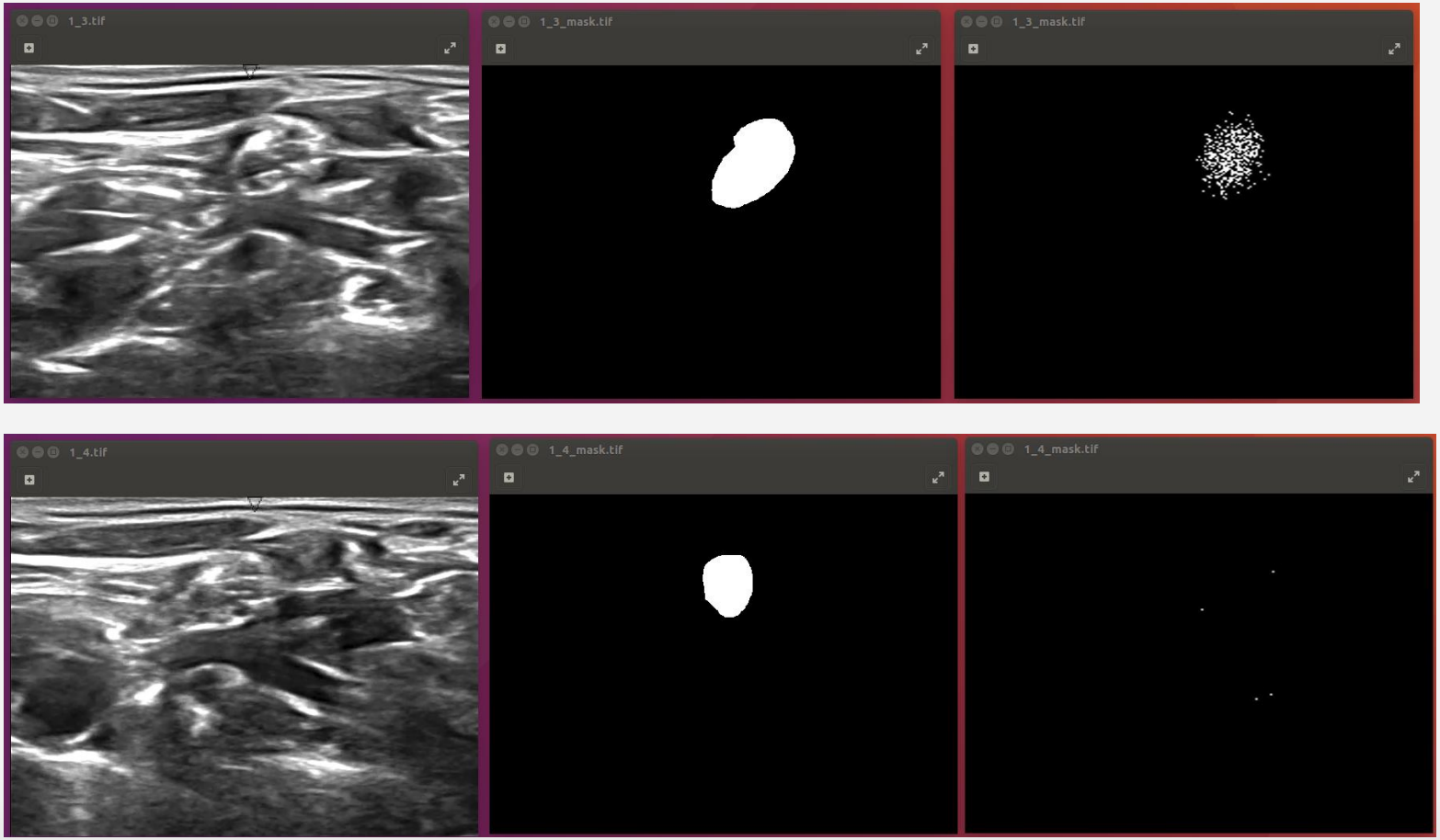
Training Result

- Train Result Example as following (X / Y / Pred) (01/02):



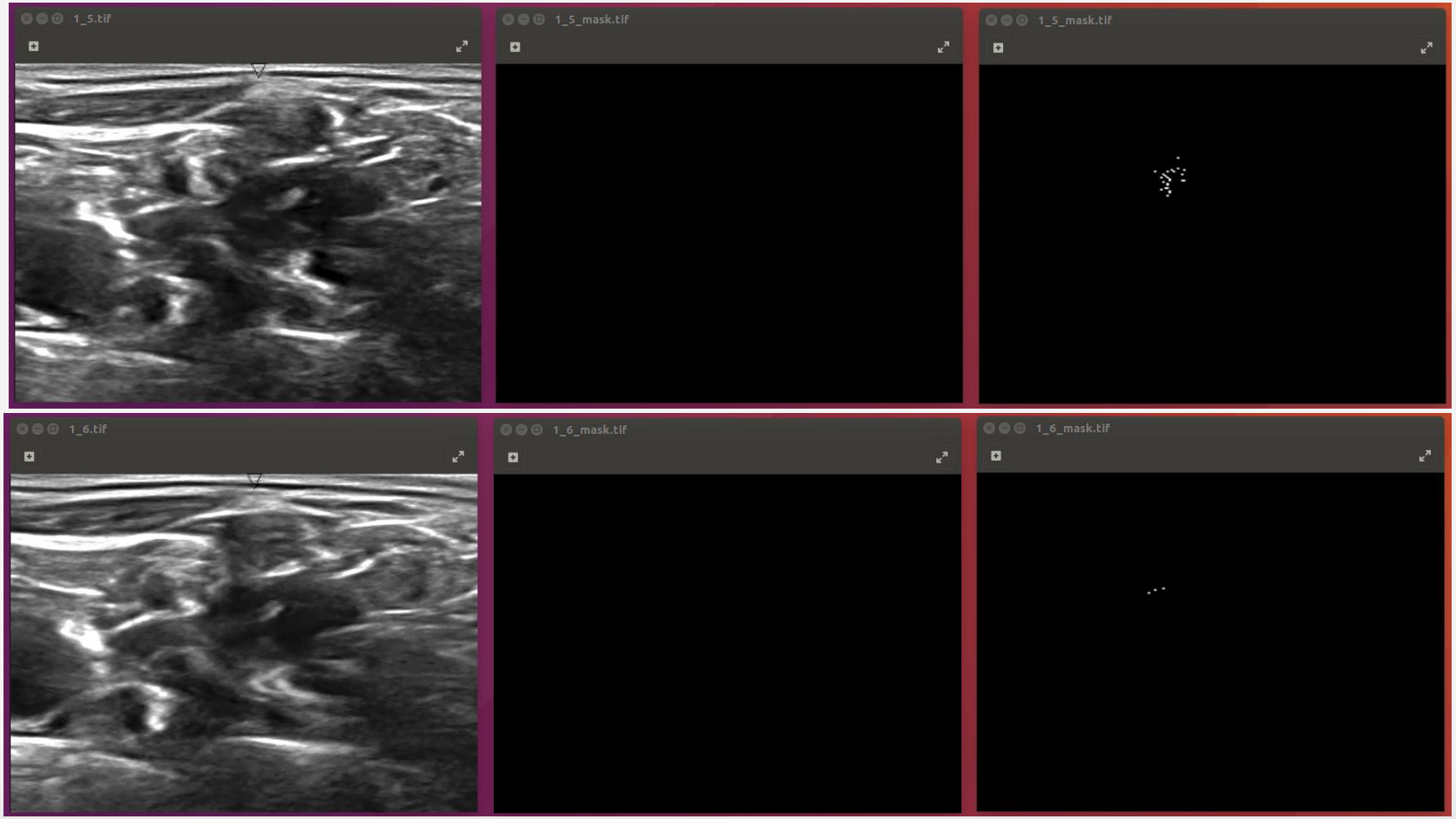
Training Result

- Train Result Example as following (X / Y / Pred) (03/04):



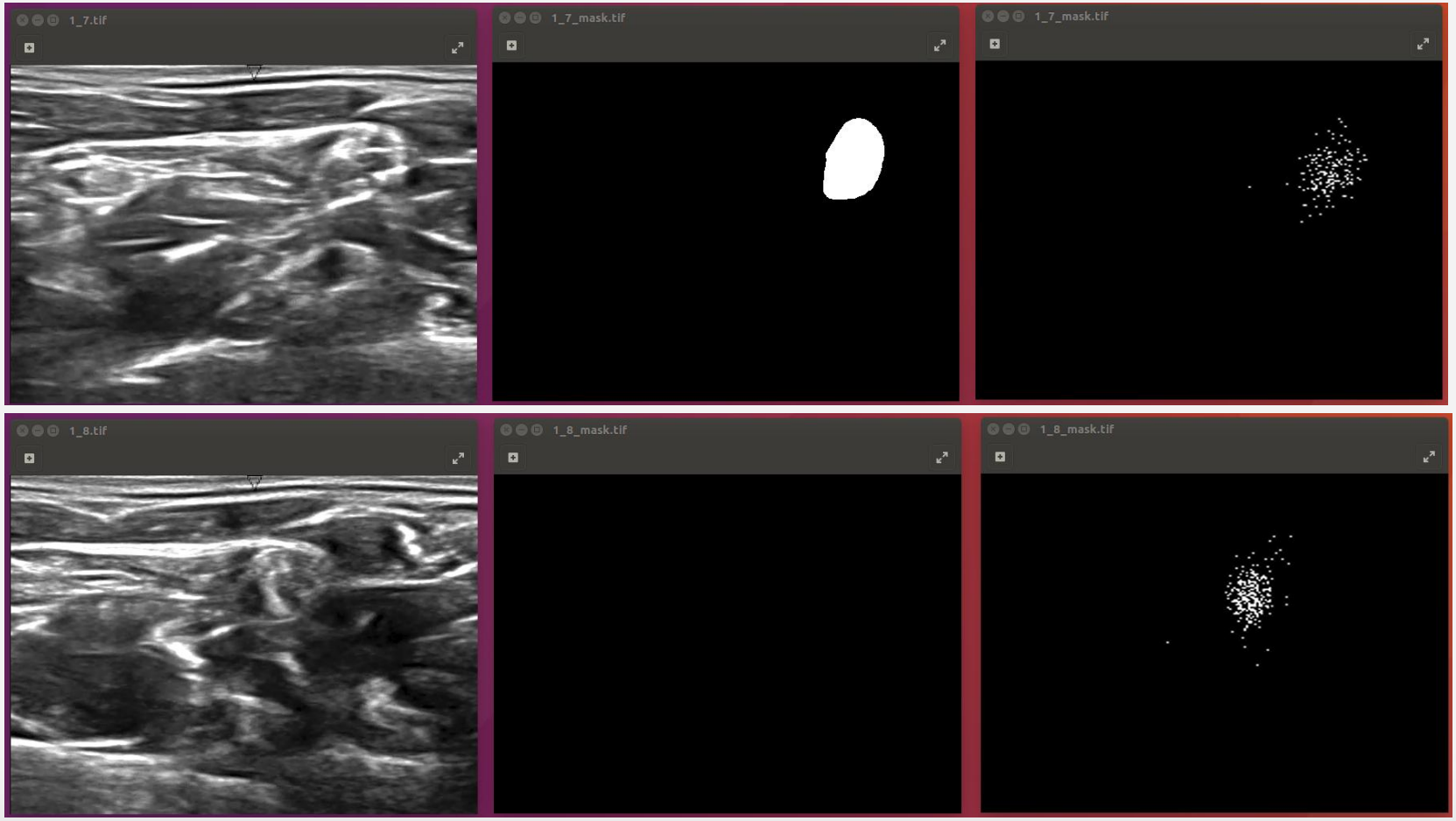
Training Result

- Train Result Example as following (X / Y / Pred) (05/06):



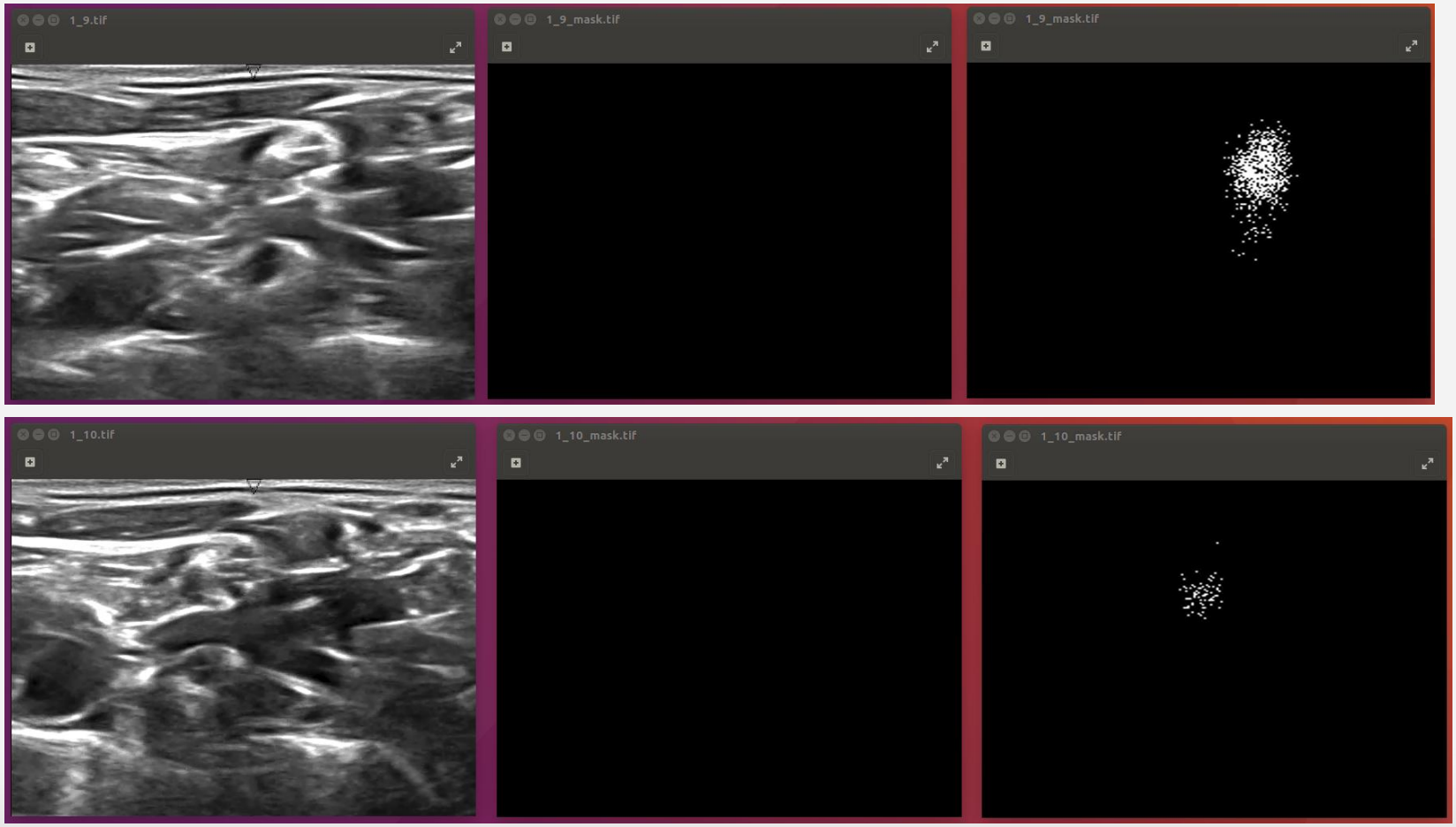
Training Result

- Train Result Example as following (X / Y / Pred) (07/08):



Training Result

- Train Result Example as following (X / Y / Pred) (09/10):



FCN Discuss

- Due to memory limit, the size of batch =32 is small.
- Example 01~03 and 07 is Positive and Predict seems several separated spot
- Example 04 is Positive but it seems predict none
- Example 05/06 are Negative and Predict still output several highlight
- Example 08~10 are Negative But it predict many hilight pixels
- The result is not good enough.
- Batch size is too small to converge but it seems better than Unet due to pre-trained Model
- May be bigger batch size and more tranning epoch are needed.

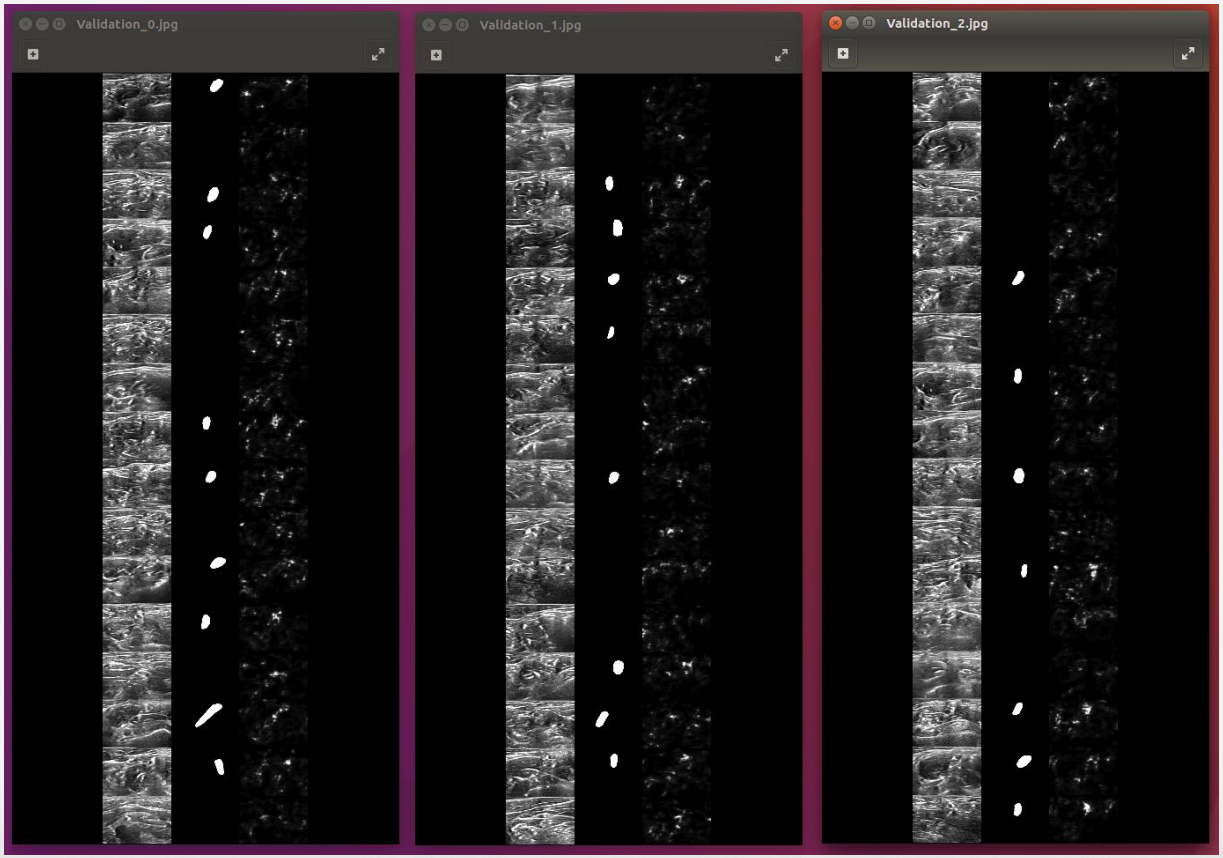
U-Net Result

- Settings
  - Input X Original Image Size
  - Input Y is two class float size (Height X Width X 2)
  - Positive one and Negative One
  - Reduced size of U-Net with Depth 3, Root Size 8
  - Training Epoch is 400 due to time limit

```
""" Assign Values """
""" Org Image """
OrgWidth = 580
OrgHight = 420
OrgChannel = 3
""" U-Net Initial Structure """
UnetDepth = 3
UnetFilterRoot = 8
""" HyperParameters """
DropOutKeep = 0.6
LearnRate = 0.0001
BatchSize = 16
NumEpoch = 400
""" Otehr Parameters """
""" Mini Batch to Summary """
IterOfSummary = 150
""" Number Of Class """
NumOfClass = 2
""" File System """
FolderToTrain = "./train/"
FolderToTest = "./test/"
FolderToTestOut = "./testout/"
FileTailToCheck = ".tif"
MaskKeyWord = "_mask"
""" UNet Train Result """
UnetLog = "./unet_log/"
UnetModel = "./unet_model/"
```

Training Result

- Train Result Example as following (X / Y / Pred) (Not converge):



Discuss

- Parameter will find a weight ?
  - Mirror vs Symmetric vs Non-Symmetric
  - No matter use Deconv. or Conv.
  - No matter user Just Add or Crop-Concat
  - All May Work
  - All about Degree of Freedom Of Structure
  - Which better?
  - Compare to Object Detection?

Future

- What I cannot create, I do not understand.
  - Richard Feynman
- Future - Not only predict, but also create
- GAN + Auto-encoder

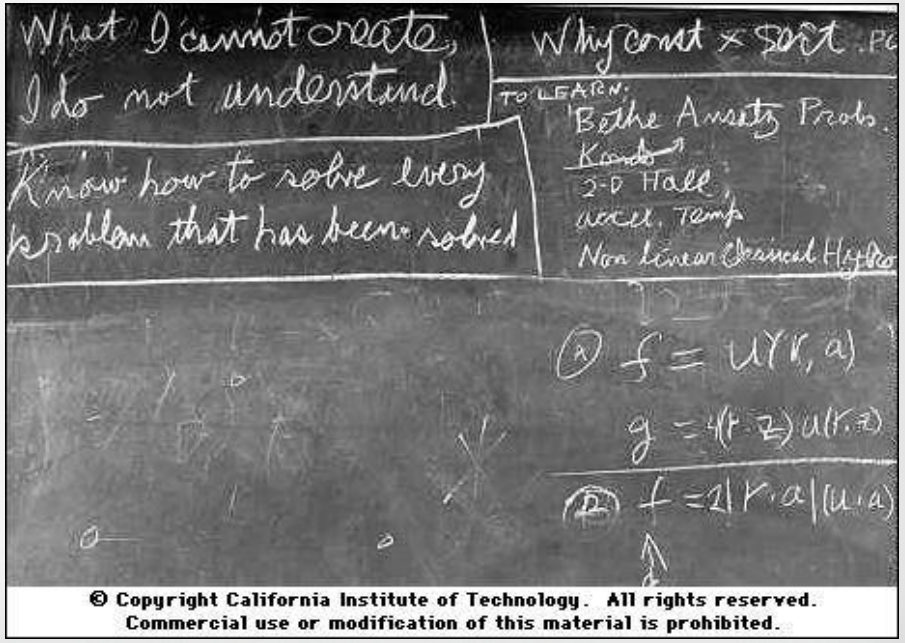
Adversarial Autoencoders

Alireza Makhzani
University of Toronto
makhzani@psi.toronto.edu

Jonathon Shlens & Navdeep Jaitly
Google Brain
{shlens,ndjaitly}@google.com

Ian Goodfellow
OpenAI
goodfellow.ian@gmail.com

Brendan Frey
University of Toronto
frey@psi.toronto.edu



Future

- AI Process
  - Hand Script Algorithm
  - Defined Model and Feature
  - Defined Model
  - Auto Find Model ?