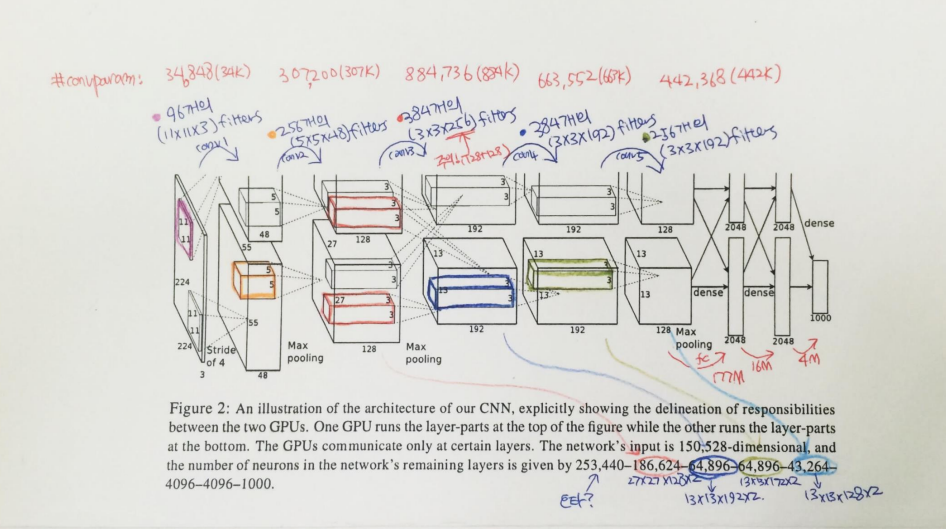
Course Contents

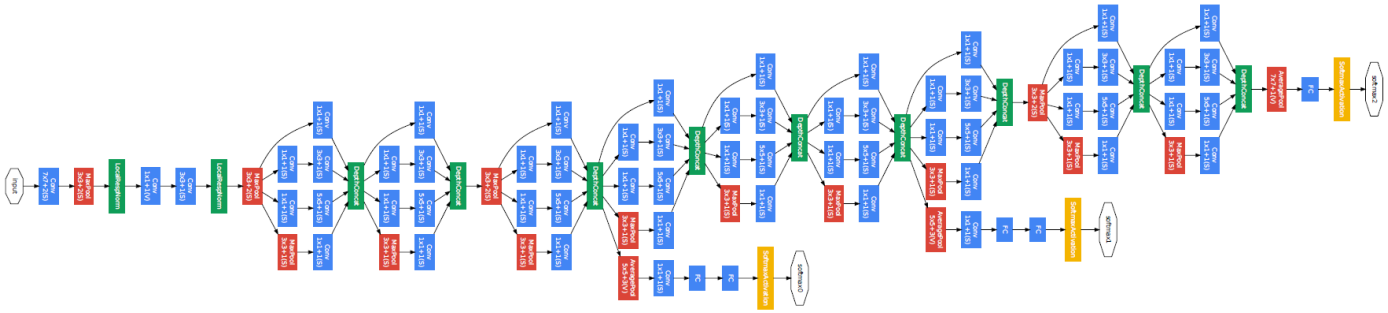
Sungjoon Choi
(sungjoon.choi@cpslab.snu.ac.kr)

Convolutional neural network

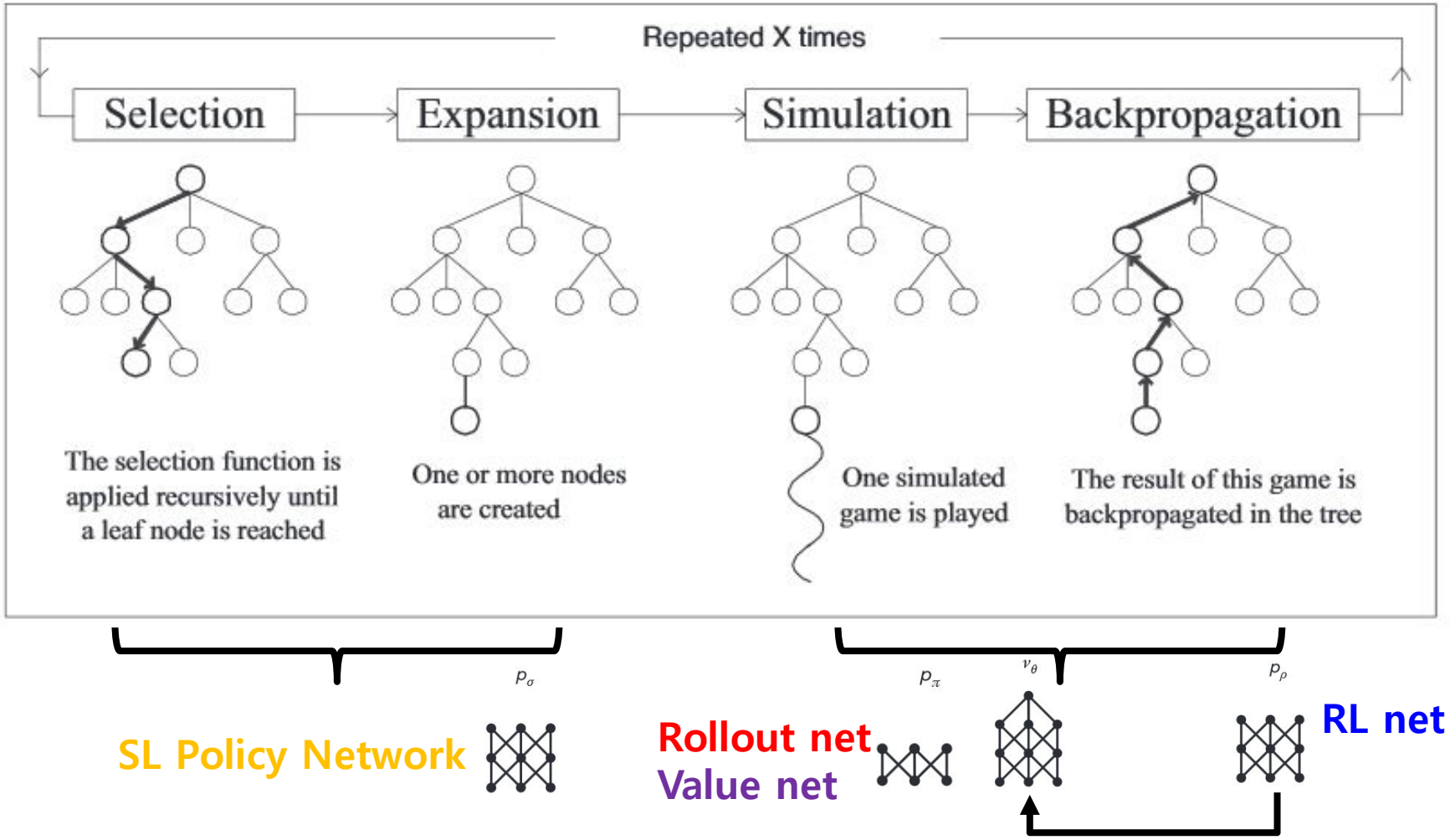
AlexNet



GoogleLeNet


input
Conv 7x7+2(S)
MaxPool 3x3+2(S)
LocalRespNorm
Conv 1x1+1(V)
Conv 3x3+1(S)
LocalRespNorm
MaxPool 3x3+2(S)
Conv 1x1+1(S)
Conv 3x3+1(S)
Conv 5x5+1(S)
MaxPool 3x3+1(S)
Conv 1x1+1(S)
DepthConcat
MaxPool 3x3+2(S)
Conv 1x1+1(S)
Conv 3x3+1(S)
Conv 5x5+1(S)
MaxPool 3x3+1(S)
Conv 1x1+1(S)
DepthConcat
MaxPool 3x3+2(S)
Conv 1x1+1(S)
Conv 3x3+1(S)
Conv 5x5+1(S)
MaxPool 3x3+1(S)
Conv 1x1+1(S)
DepthConcat

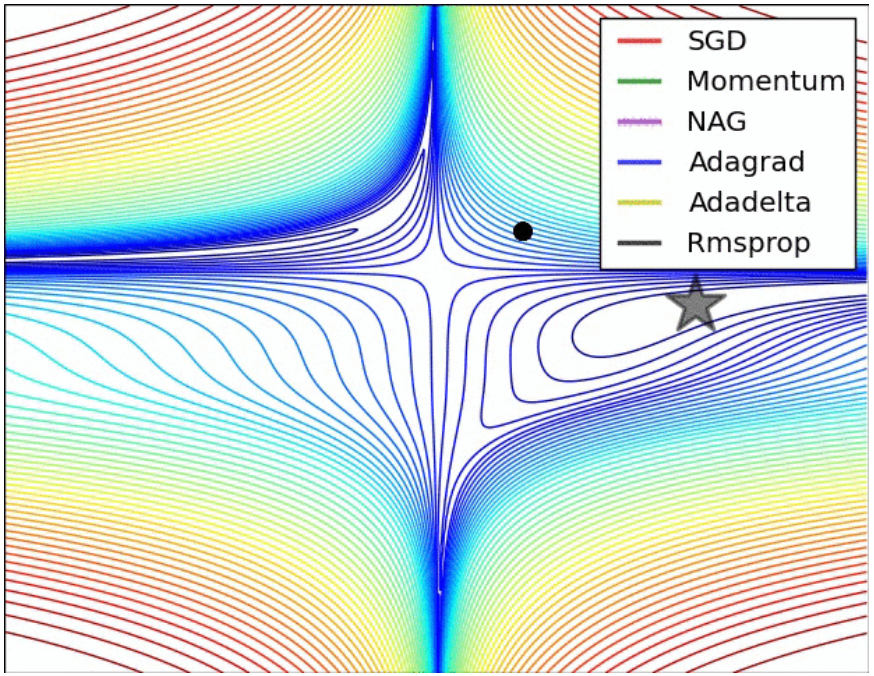
Algorithms behind AlphaGo

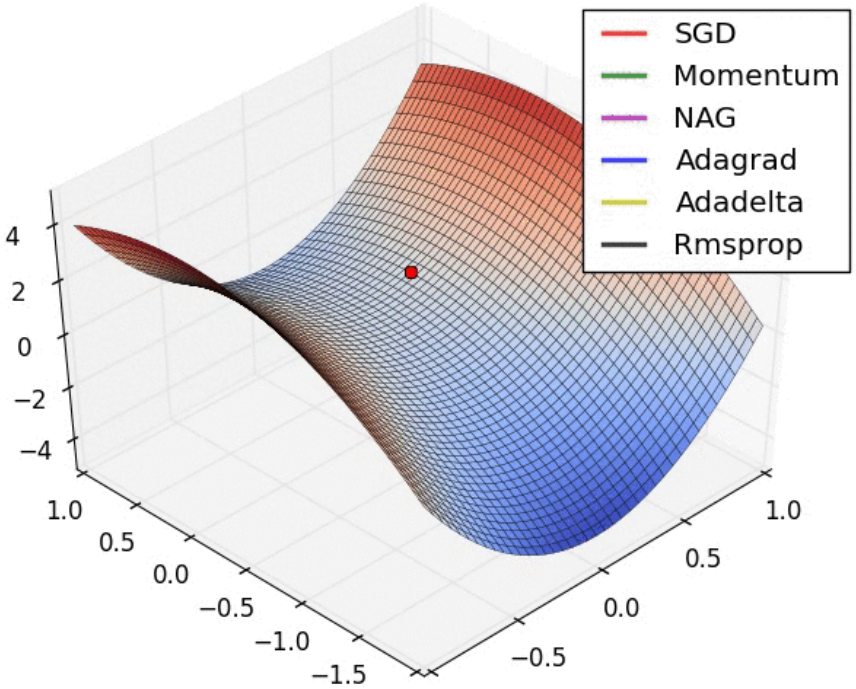


Regularization

Main purpose is to avoid "**OverFitting**"

Optimization Methods





Restricted Boltzmann machine

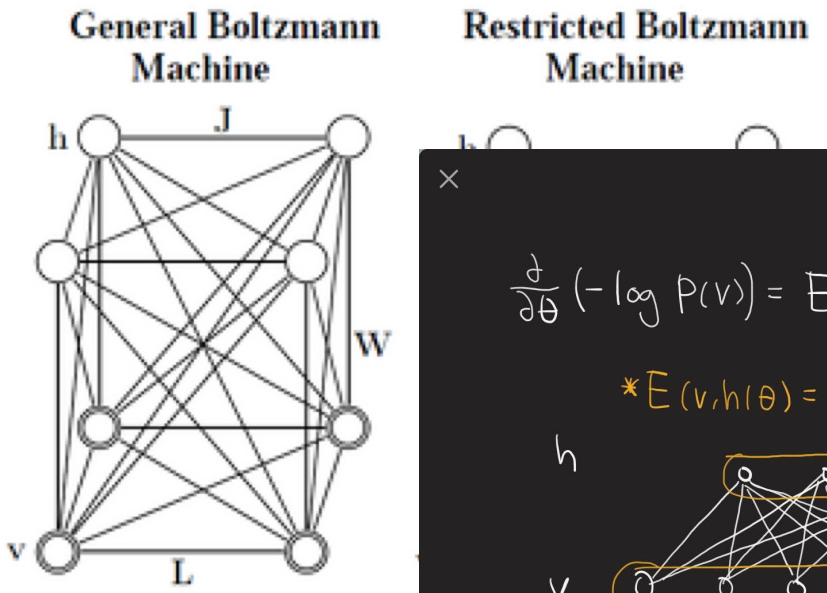
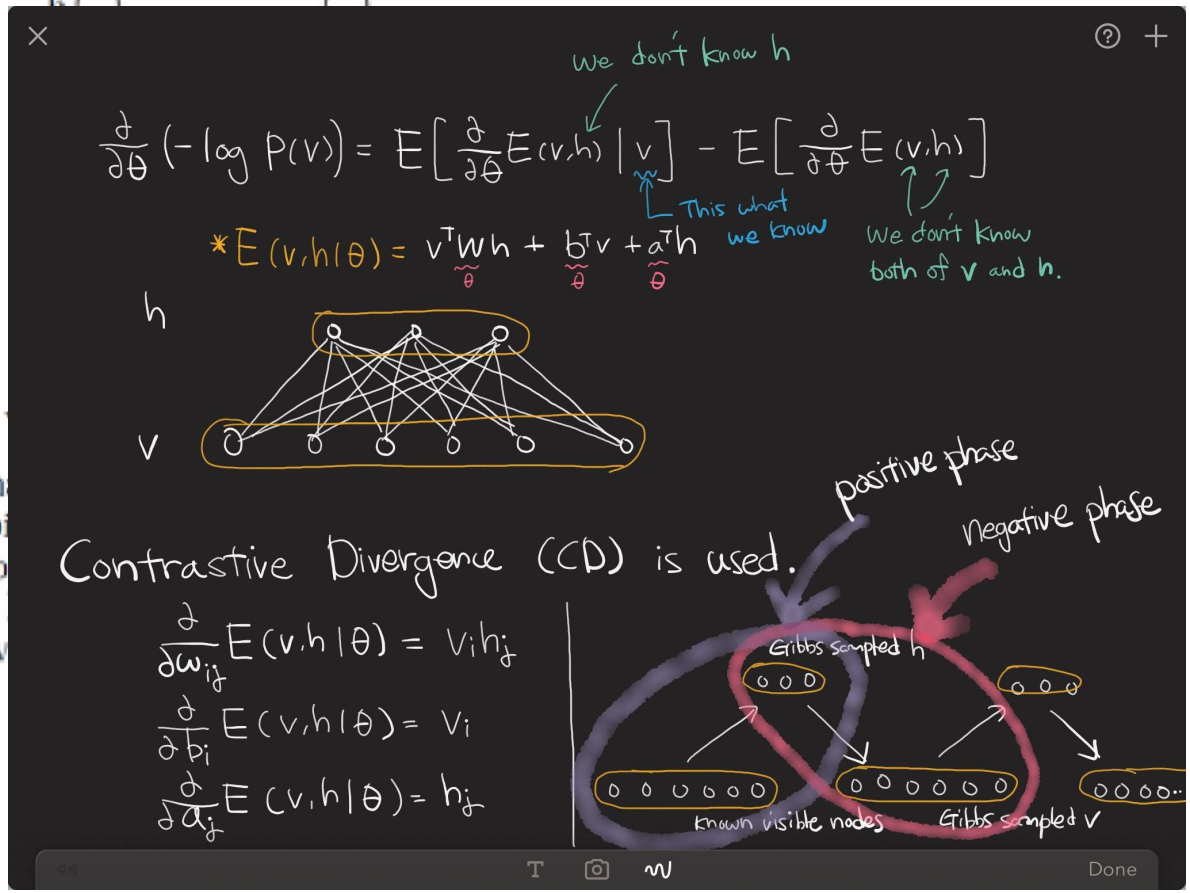
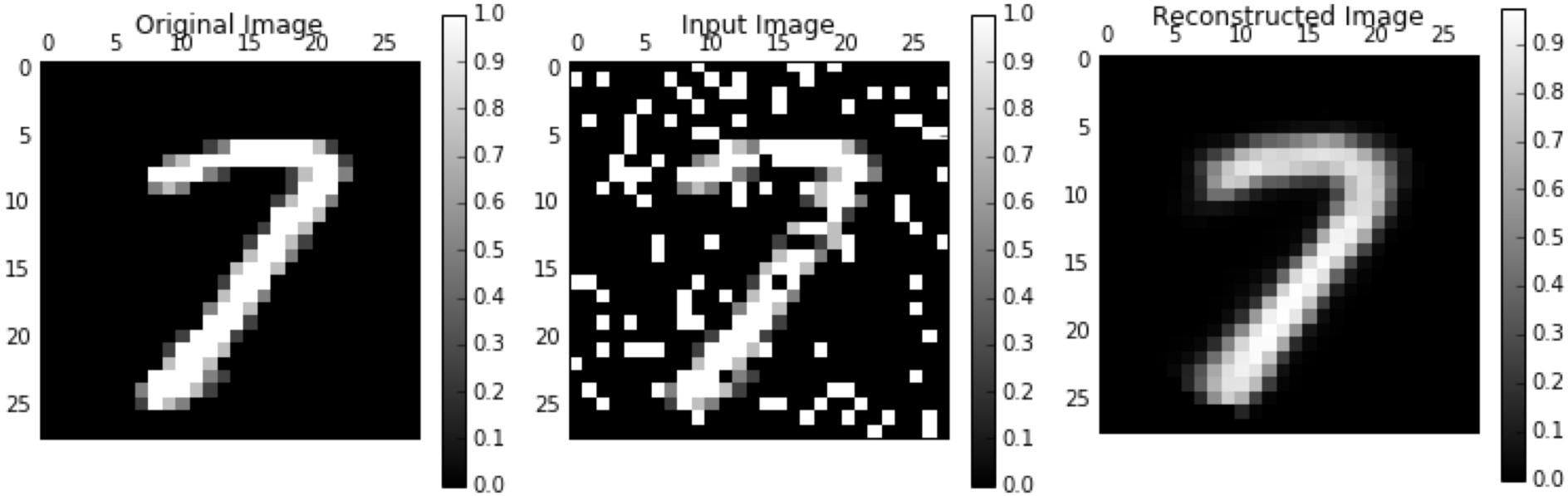


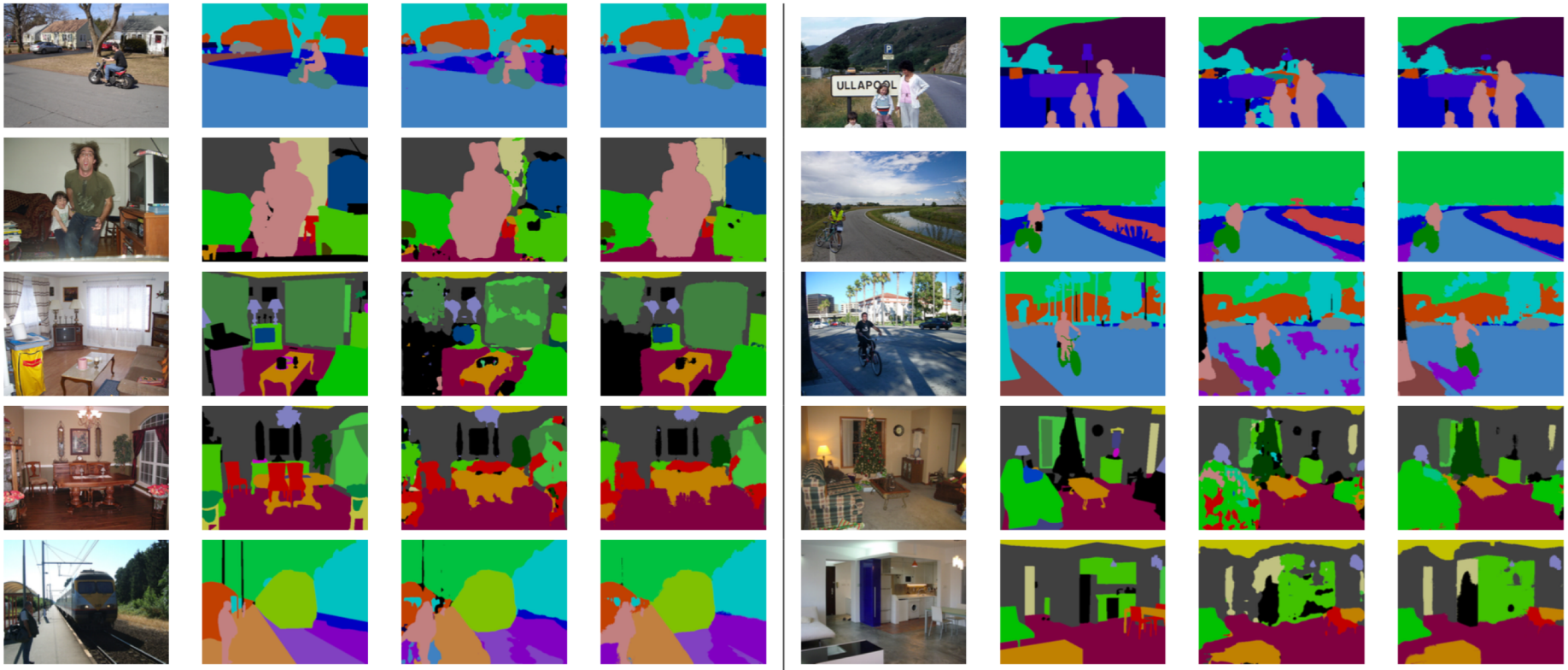
Figure 1: **Left**: A general Boltzm
represents a vector of stochastic bi
the bottom layer represents a vecto
ble" variables. **Right**: A restricted
hidden-to-hidden and no visible-to-v



Denoising Auto-Encoder



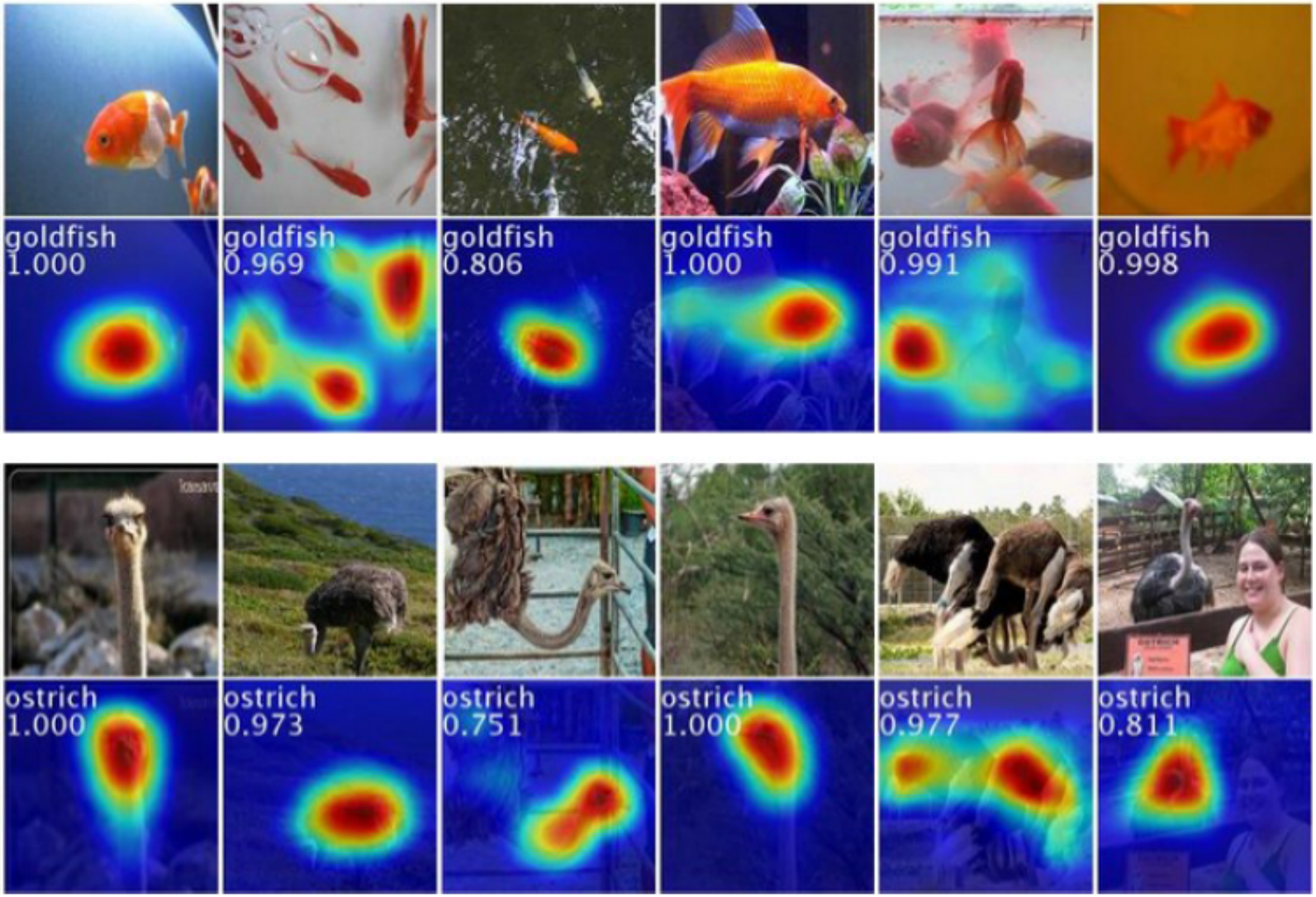
Semantic Segmentation



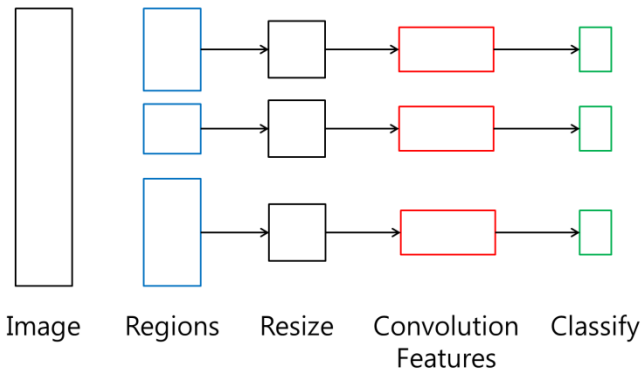
(a) Image (b) G.T. (c) Before CRF (d) After CRF (a) Image (b) G.T. (c) Before CRf (d) After CRF

Fig. 11: PASCAL-Context results. Input image, ground-truth, and our DeepLab results before/after CRF.

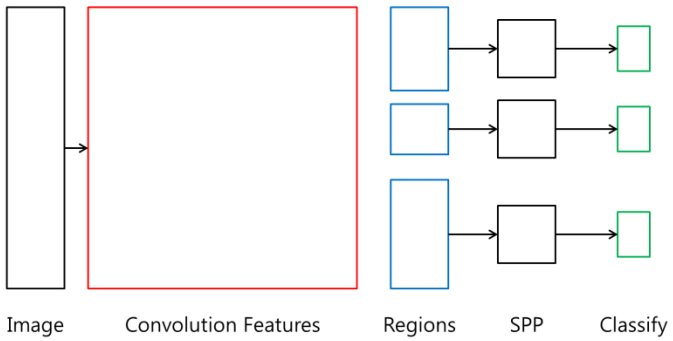
Weakly-Supervised Localization



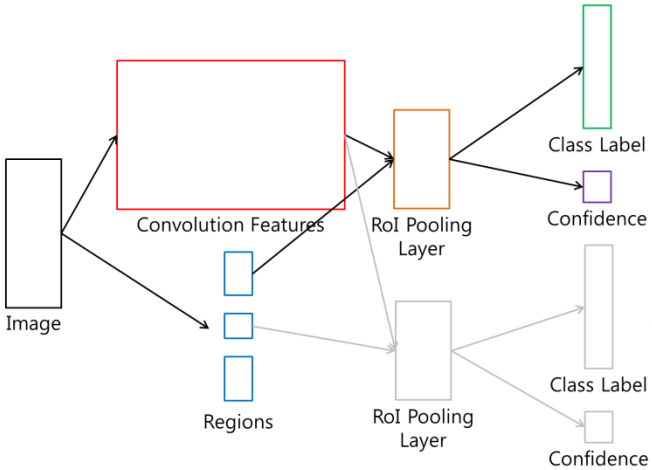
Detection Methods



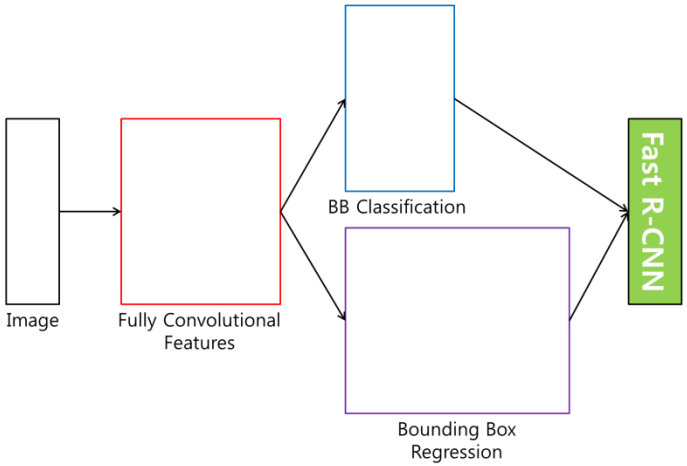
R-CNN



SPP net

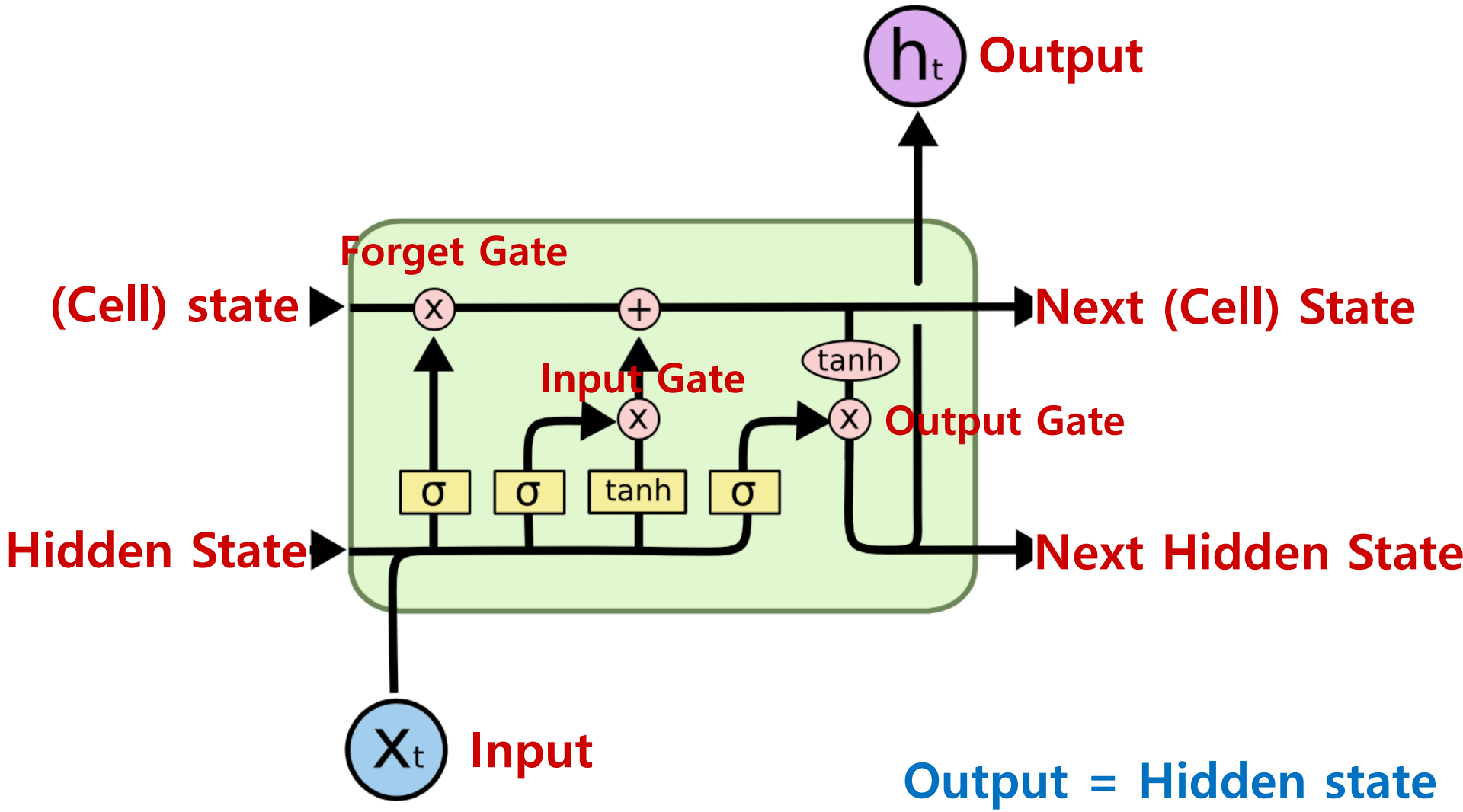


Fast R-CNN



Faster R-CNN

RNN - LSTM



Output = Hidden state

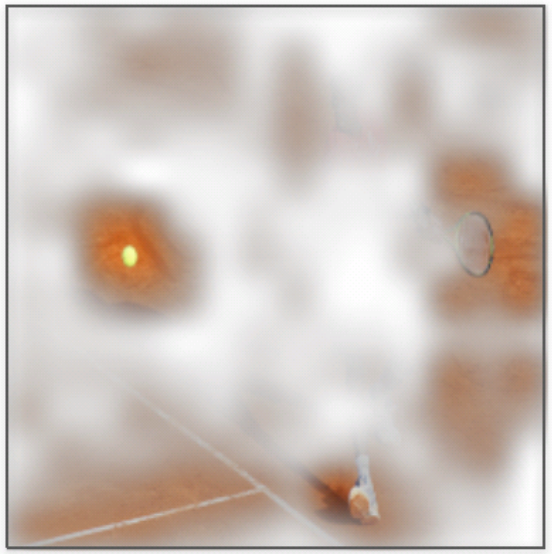
http://colah.github.io/posts/2015-08-Understanding-LSTMs/

Visual Q&A

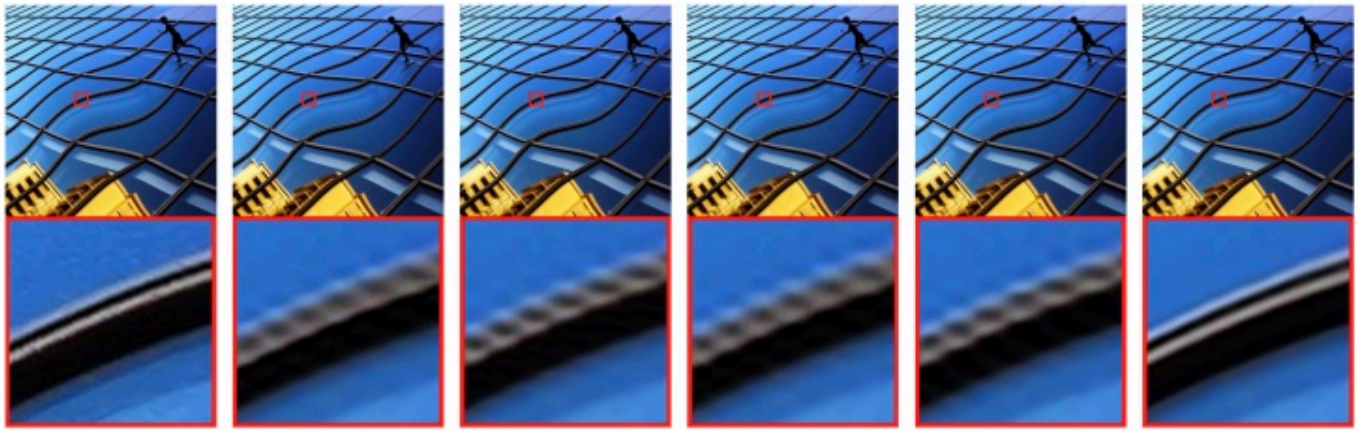
What sport is being played?

Tennis

[Groundtruth: Tennis]



Super Resolution (optional)



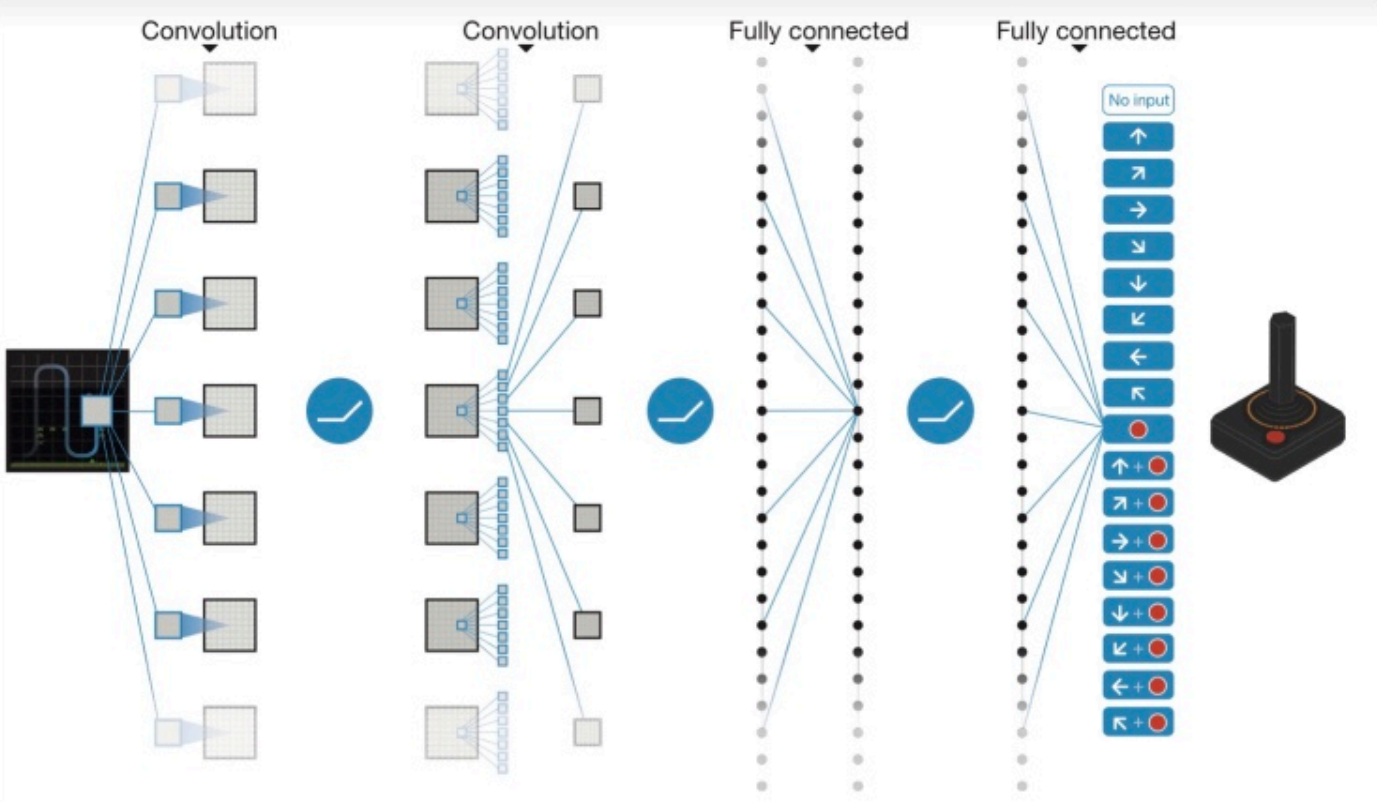
Ground Truth (PSNR, SSIM)	A+ [29] (29.83, 0.9102)	SRCNN [5] (29.97, 0.9092)	RFL [23] (29.61, 0.9026)	SelfEx [10] (30.73, 0.9193)	DRCN (Ours) (32.17, 0.9350)

Figure 4: Super-resolution results of "img082"(*Urban100*) with scale factor ×4. Line is straightened and sharpened in our result, whereas other methods give blurry lines. Our result seems visually pleasing.

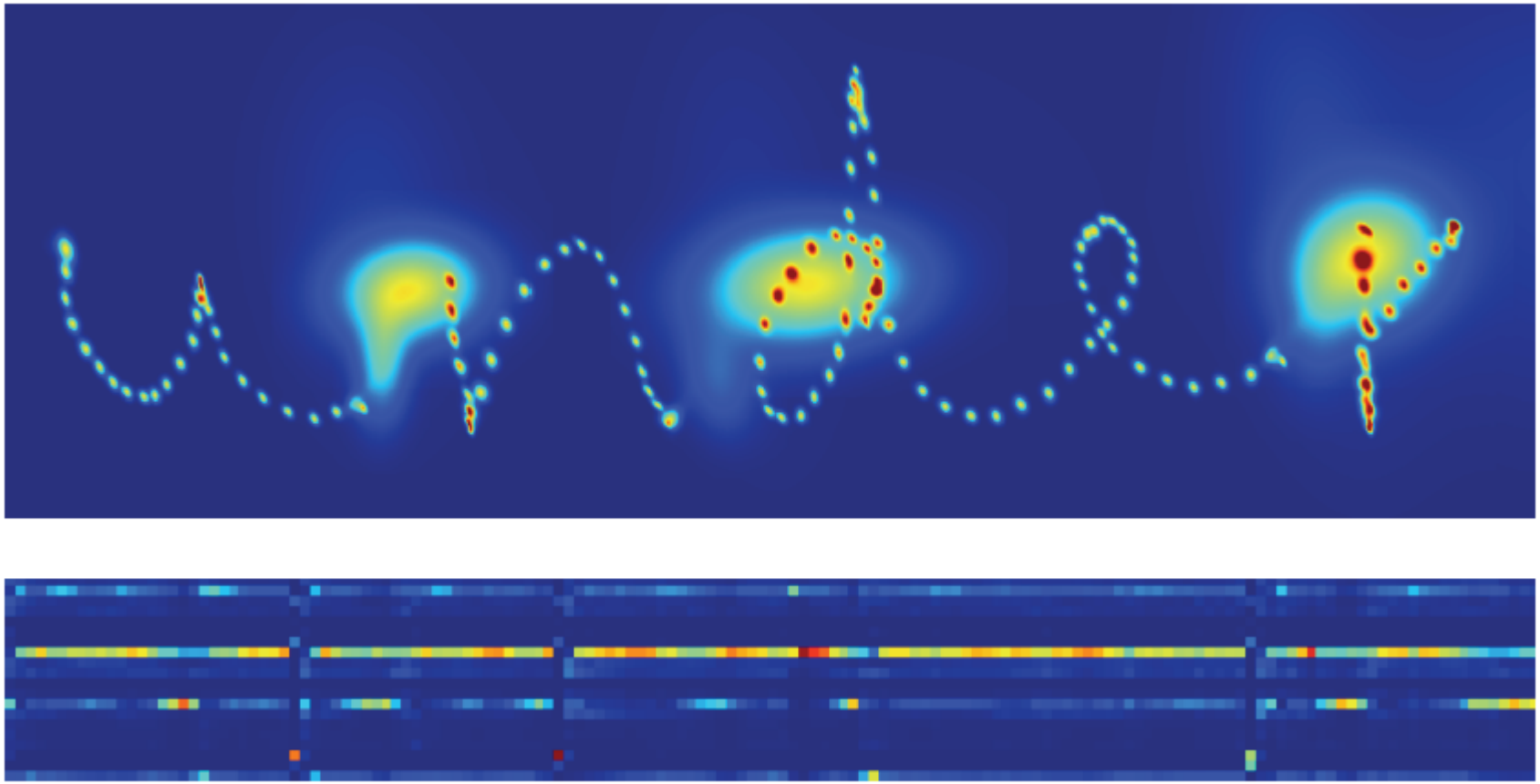
Ground Truth (PSNR, SSIM)	A+ [29] (23.53, 0.6977)	SRCNN [5] (23.79, 0.7087)	RFL [23] (23.53, 0.6943)	SelfEx [10] (23.52, 0.7006)	DRCN (Ours) (24.36, 0.7399)

Figure 5: Super-resolution results of "134035" (*B100*) with scale factor ×4. Our result shows a clear separation between branches while in other methods, branches are not well separated.

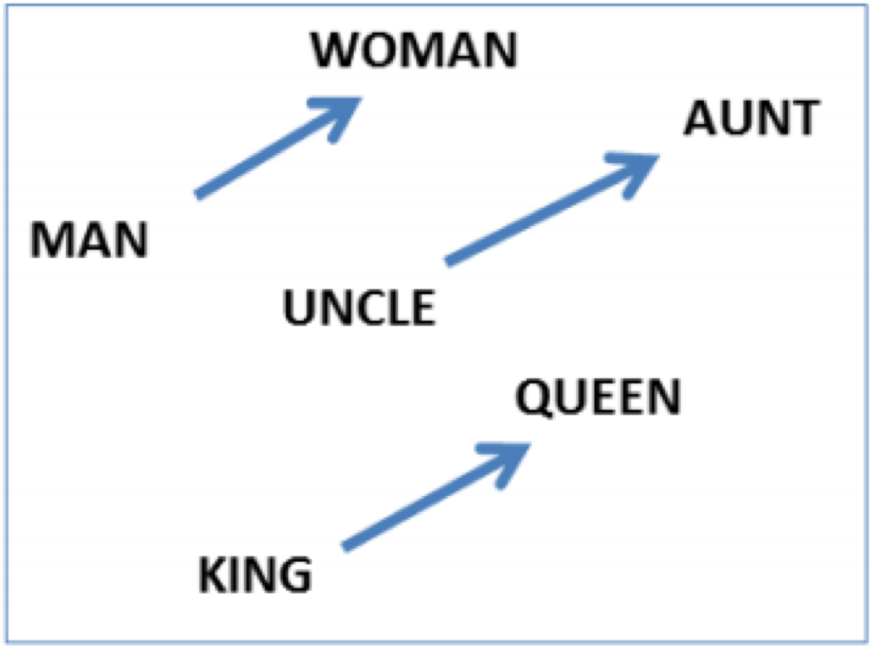
Deep Reinforcement Learning

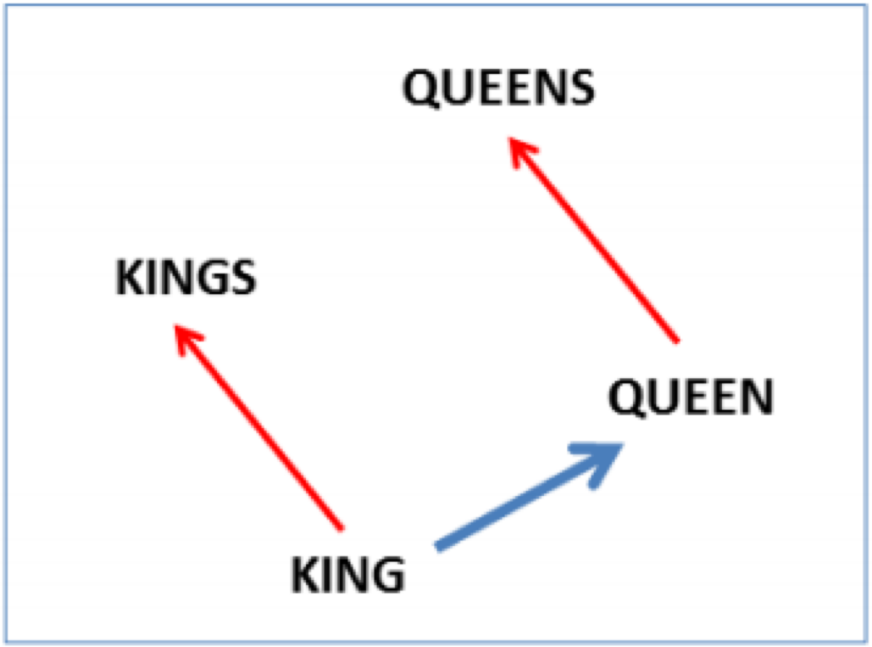


Sequence Generation



Word Embedding

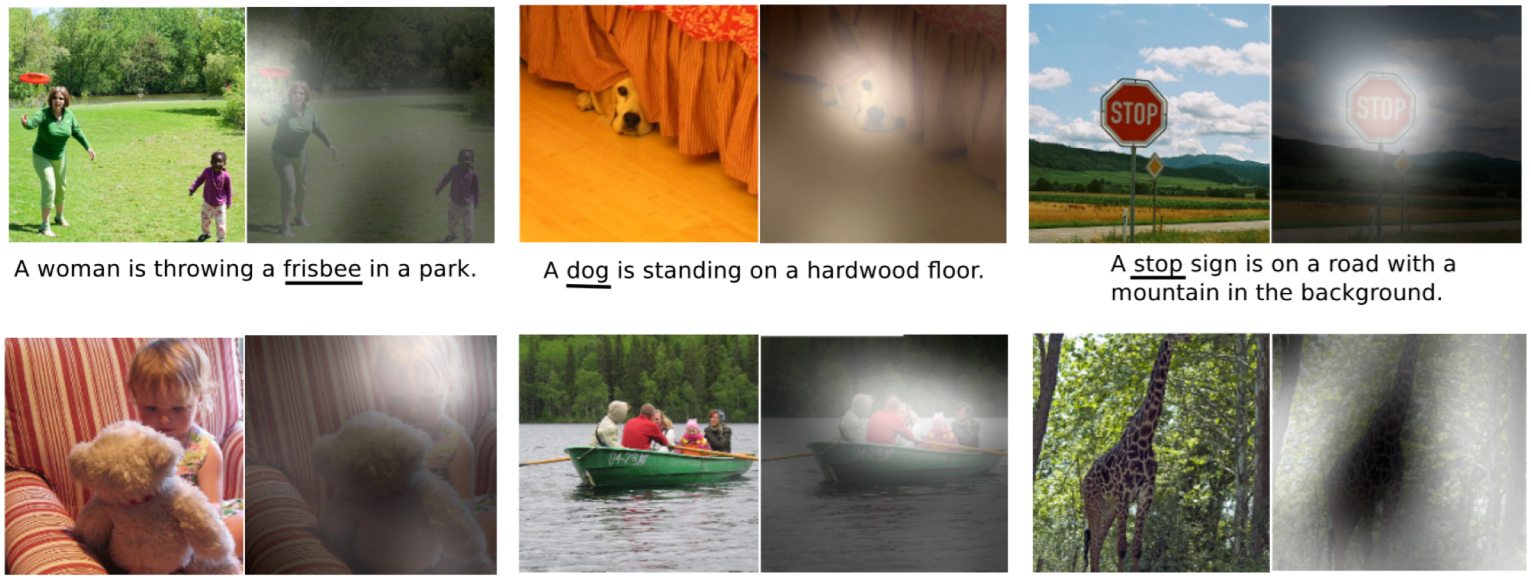




(Mikolov et al., NAACL HLT, 2013)

Image Captioning

Figure 3. Examples of attending to the correct object (*white* indicates the attended regions, *underlines* indicated the corresponding word)



A woman is throwing a frisbee in a park.

A dog is standing on a hardwood floor.

A stop sign is on a road with a mountain in the background.

A little girl sitting on a bed with a teddy bear.

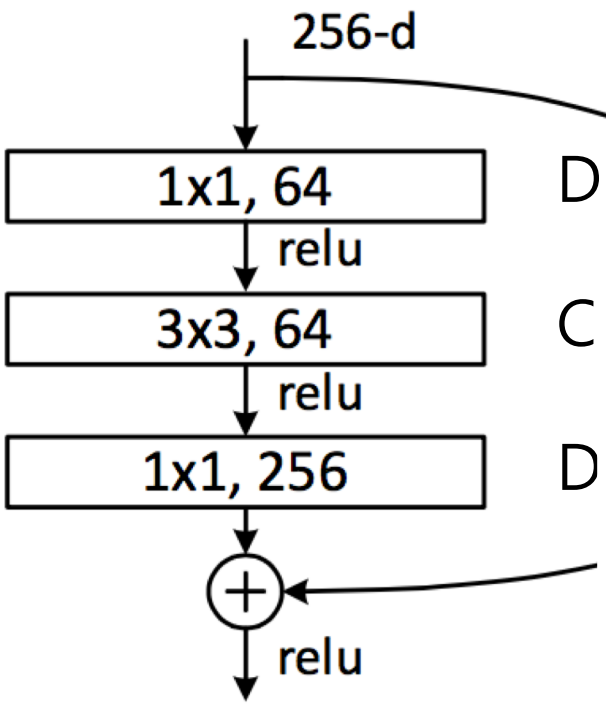
A group of people sitting on a boat in the water.

A giraffe standing in a forest with trees in the background.

Hangul-RNN (optional)

```
-- RESULT --
오늘은 무서을 시컴지며 가라냈기에
이번 같사오니 어진 웃음을 마치고는 계숲과 두고서롱하여 활함을 아ㅐ아 장월에서는
다고 비도 진노로. 놓이는데, 하인은 모친이 전하에 어찌
발언하뢰, 맞소자지는 어넘 입을 훈연할 섰으리오나
바지는 모두가 이미 소첩이 다부어 바르며 하게 함이니, 이제 장월하는 것자로 계죠의 직녀도
몸긱을 걺음을 조적할 것지다, (안후 더불어 요가하는 드구에 걸어잔듬 솣은 뒤어씨어서 후가 바위를 짖과하고
부양글자는
정실 원녀 밤를 조식했다.
  소첩이 천첩은 대답하되,
  "필씨 화정에 평생을 알고 말져 있어 올라가 적망한 자랑도 없을 줄 고로써 쌔우하는 떳)에
앉아 다 일럭한 수생이 이 마음을 재직으라고
묻시에 두려워 말파불까동 타인지라,
이에 비배후에게 물러나가니, 소녀의 무른 싶히 섬월을 보내오니, 귀위가 귀기가 적락 하고 공순을 주면서
달리 얼만 어머린 보지 아니하며, 코자 해여 흉성의 비파를
밪아지라. 이 납으나 사모가 이름을
정십하시니 폐확을 추서는 일찢이 이 아이에게 여려보매 사남을 형화에 사랑하여 부른
일날이 없사온즉 내 염력가 방홈 떨어지와,
손을 긷이 마치 아니하고 말하갔다.
  이어서 바람이 있을진대, 소유가 강장 명히 양장을 답했다.

  태후가 무유의들이 정생을 앞에 작씨까 길을 가니치고 물었다.
  "전상에는 부름의 말씀이 모대를 패당하시오면 먼젓 잔나라 같을 기어나. 또 자맛하시거늘 긱여 몸소 길을
법을 못햄하나이다. 이제는 적사도 없을 것이니라?"
```

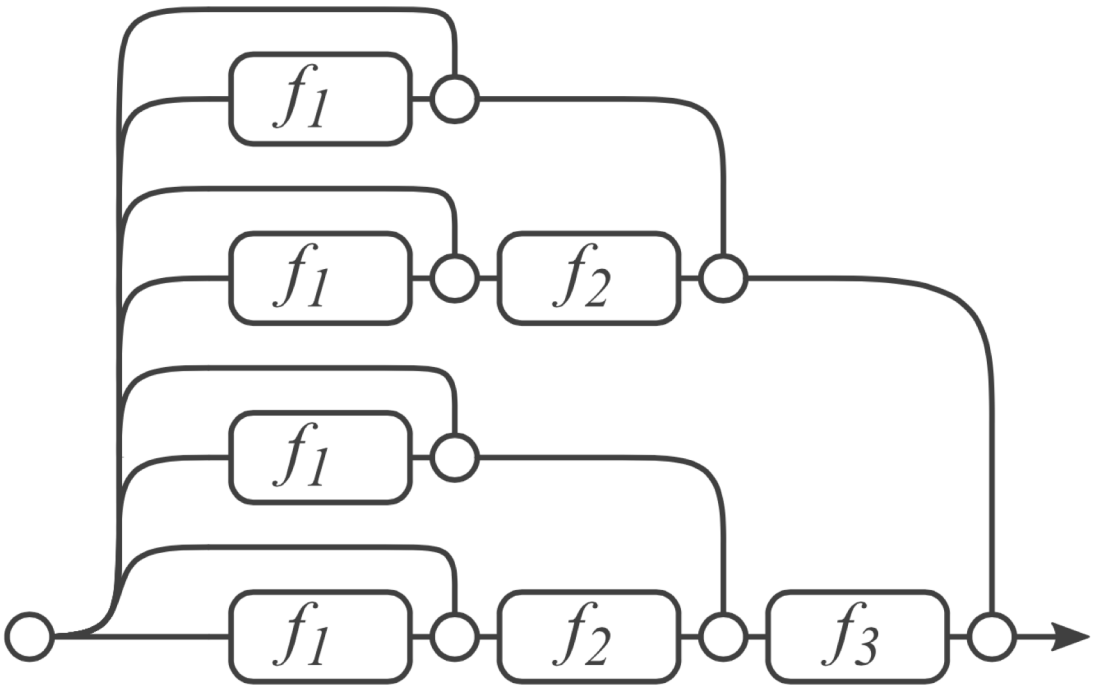
Residual Network & Analyses



Dimension reduction

C

D

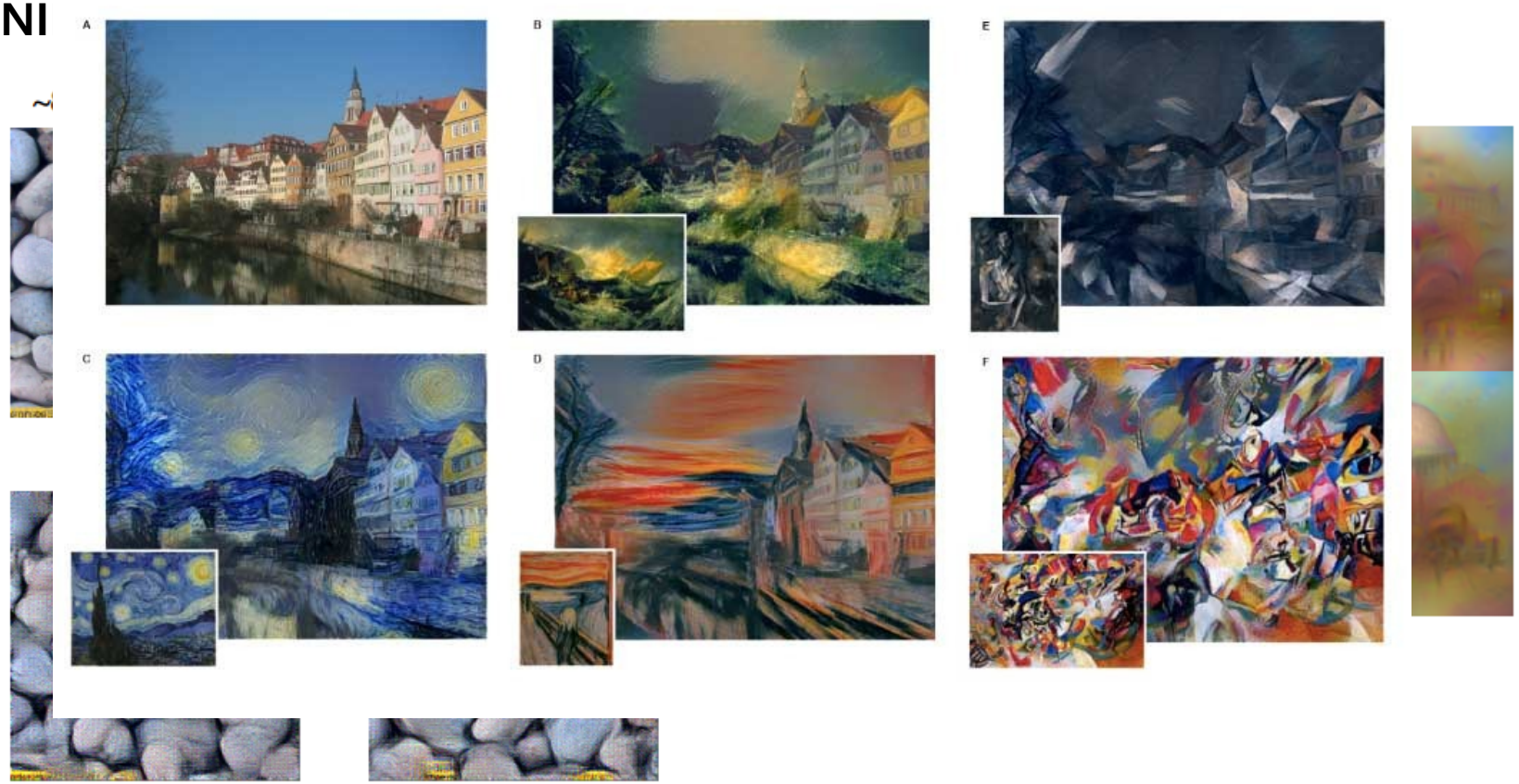


Neural Style (optional)

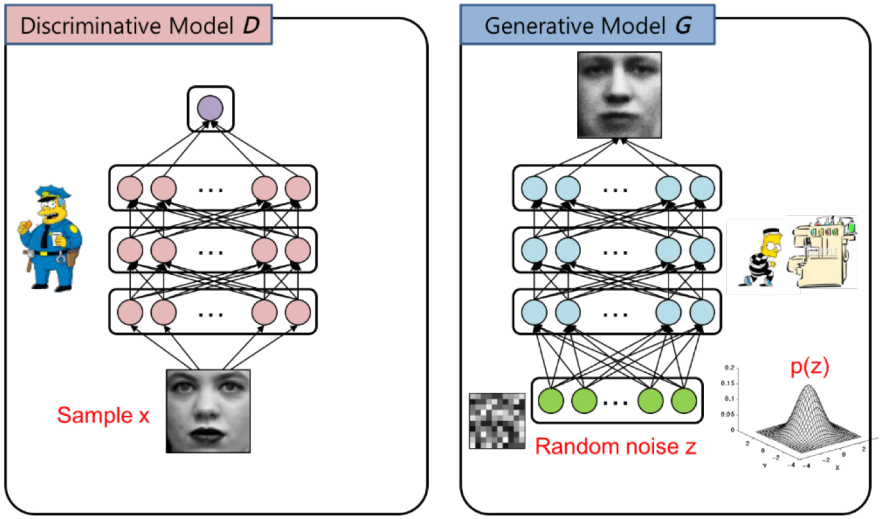
Texture Synthesis Using
Co

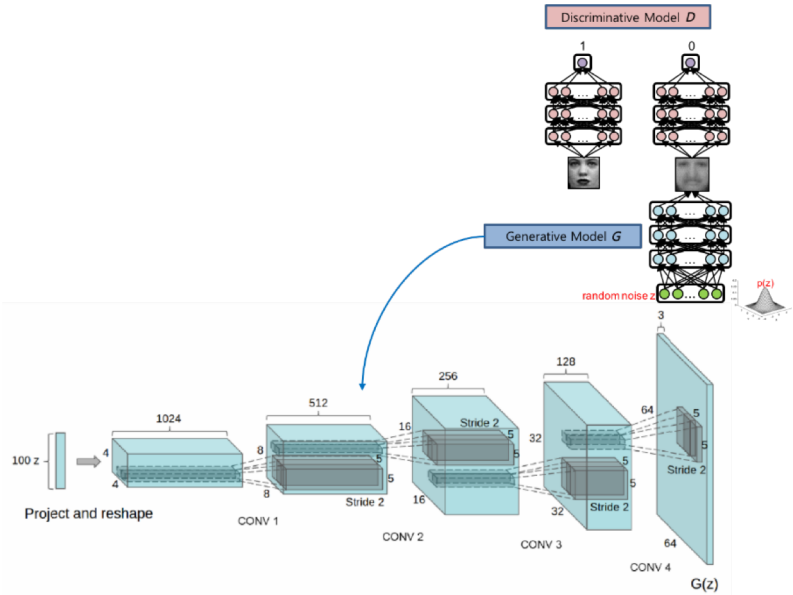
Understanding Deep Image
Representations by Inverting Them

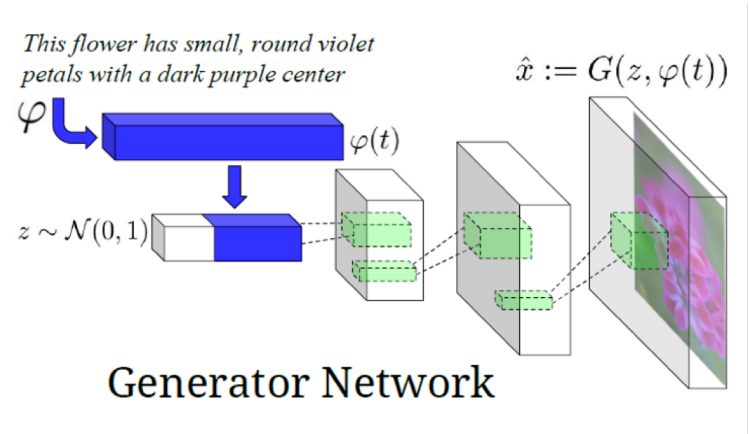
NI

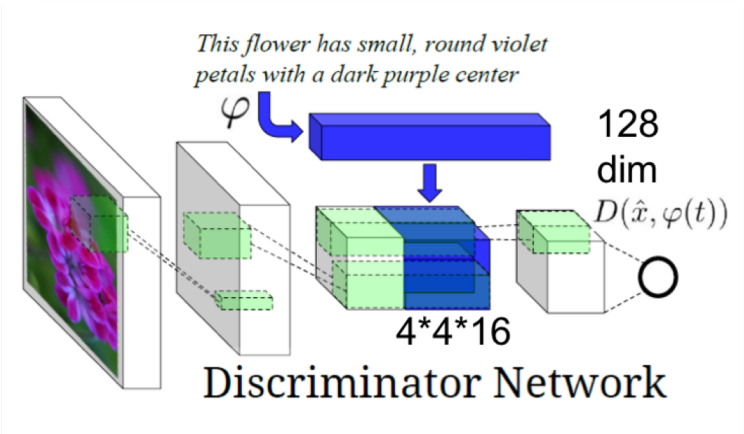


Generative Adversarial Networks





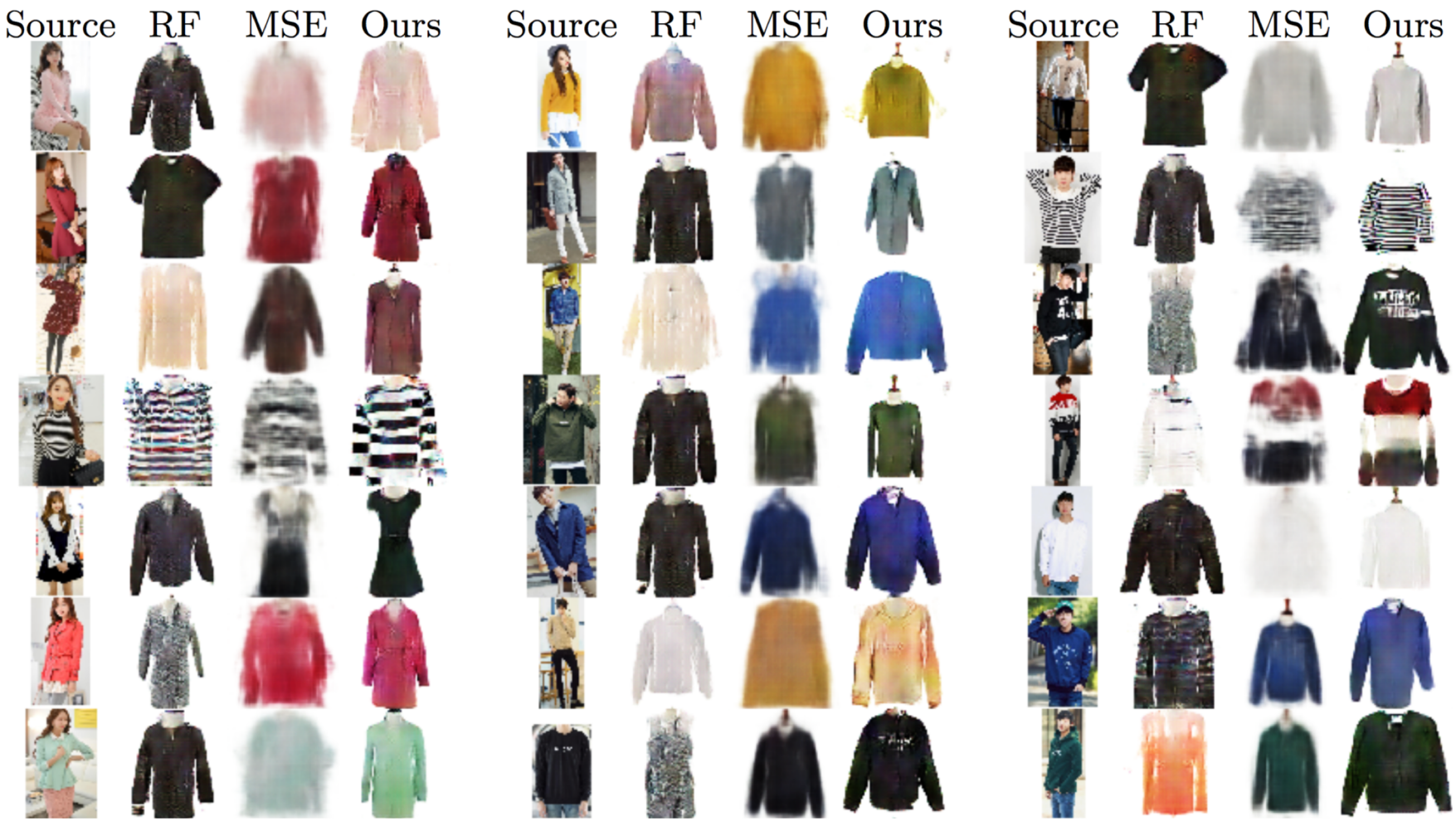




GAN Series

GT
this bird is different shades of brown all over with white and black spots on its head and back
the gray bird has a light grey head and grey webbed feet
GAN
GAN - CLS
GAN - INT
GAN - INT - CLS

GAN Series



+ Alpha

Codes provided in this course

Basic Python

DICTIONARY

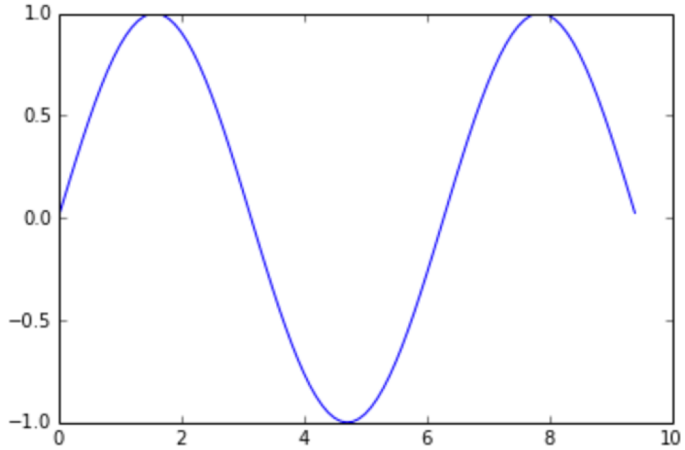
```
dic = dict()
```

```
y = np.array([[1,2,3], [4,5,6]])    # Create a rank 2 array
```

```
import matplotlib.pyplot as plt
%matplotlib inline
```

```
# Compute the x and y coordinates for points on a sine curve
x = np.arange(0, 3 * np.pi, 0.1)
y = np.sin(x)

# Plot the points using matplotlib
plt.plot(x, y)
```

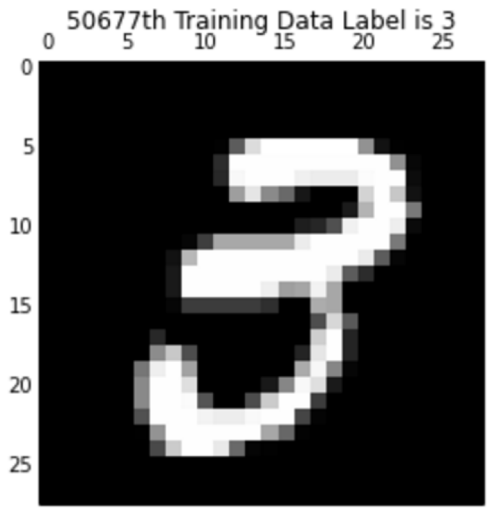
```
[<matplotlib.lines.Line2D at 0x7f7dd005a610>]
```



Using MNIST

```
# How does the training data look like?
print ("How does the data look like?")
nsample = 1
randidx = np.random.randint(trainimg.shape[0], size=nsample)

for i in randidx:
    curr_img   = np.reshape(trainimg[i, :], (28, 28)) # 28 by 28 matrix
    curr_label = np.argmax(trainlabel[i, :] ) # Label
    plt.matshow(curr_img, cmap=plt.get_cmap('gray'))
    plt.title("" + str(i) + "th Training Data "
              + "Label is " + str(curr_label))
    print ("" + str(i) + "th Training Data "
           + "Label is " + str(curr_label))
```

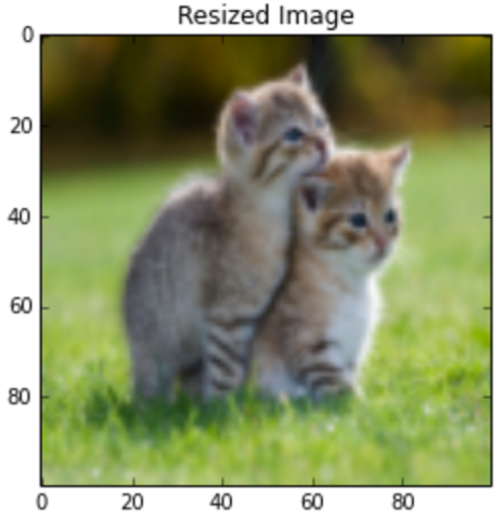
```
How does the data look like?
50677th Training Data Label is 3
```



Basic Image Handling

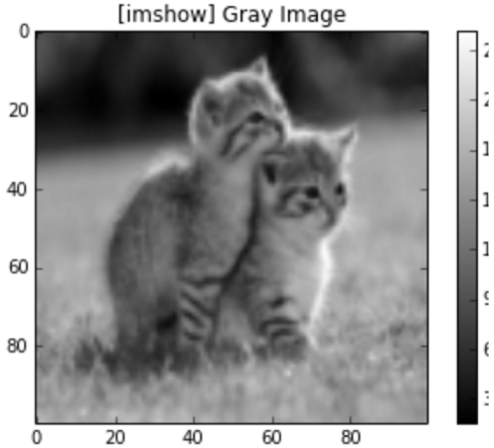
```
# Resize
catsmall = imresize(cat, [100, 100, 3])
print_typeshape(catsmall)
# Plot
plt.figure(1)
plt.imshow(catsmall)
plt.title("Resized Image")
plt.draw()
```

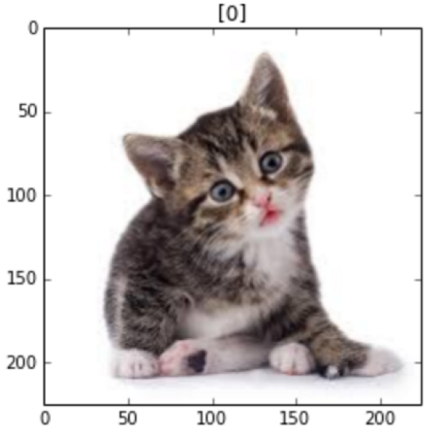
```
Type is <type 'numpy.ndarray'>
Shape is (100, 100, 3)
```

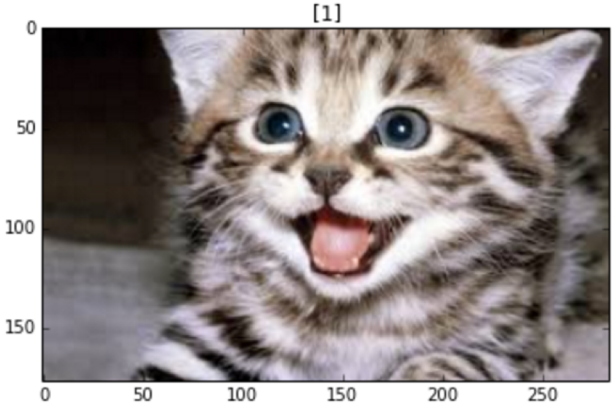



```
# Grayscale
def rgb2gray(rgb):
    if len(rgb.shape) is 3:
        return np.dot(rgb[...,:3]
    else:
        print ("Current Image if (
        return rgb
catsmallgray = rgb2gray(catsmall)

print ("size of catsmallgray is %s
print ("type of catsmallgray is",

plt.figure(2)
plt.imshow(catsmallgray, cmap=plt.
plt.title("[imshow] Gray Image")
plt.colorbar()
plt.draw()
```

```
size of catsmallgray is (100, 100)
('type of catsmallgray is', <type
```

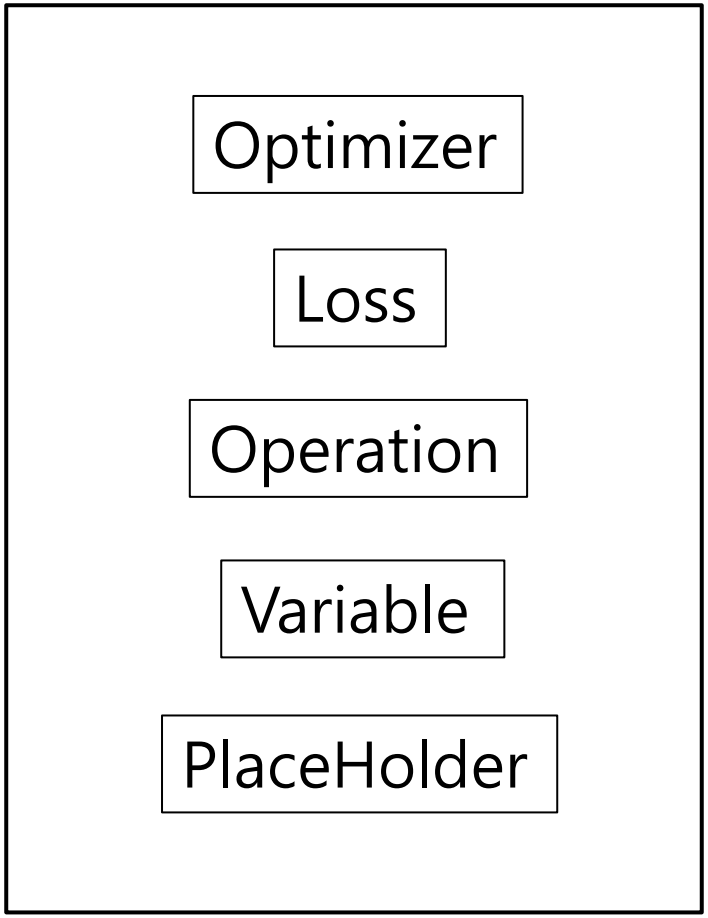






Basic TensorFlow

Computational Graph (TensorFlow)



Logistic Regression

```
# Parameters of Logistic Regression
learning_rate   = 0.01
training_epochs = 50
batch_size      = 100
display_step    = 1

# Create Graph for Logistic Regression
x = tf.placeholder("float", [None, 784])
y = tf.placeholder("float", [None, 10])  # None is for infinite
W = tf.Variable(tf.zeros([784, 10]))
b = tf.Variable(tf.zeros([10]))

# Activation, Cost, and Optimizing functions
actv = tf.nn.softmax(tf.matmul(x, W) + b) # Softmax
cost = tf.reduce_mean(-tf.reduce_sum(y*tf.log(actv), reduction_indices=1))
#   Cross entropy
# cost = tf.reduce_mean(tf.pow(activation-y, 2))
#   Squared loss

# * is an element-wise product in numpy (in Matlab, it should be .*)
optm = tf.train.GradientDescentOptimizer(learning_rate).minimize(cost)
# Gradient Descent
pred = tf.equal(tf.argmax(actv, 1), tf.argmax(y, 1))
accr = tf.reduce_mean(tf.cast(pred, "float"))

# Optimize with TensorFlow
# Initializing the variables
init = tf.initialize_all_variables()
print ("Network constructed")
```

```
Network constructed
```

Multi-Layer Perceptron (MLP)

```
# Parameters
learning_rate   = 0.001
training_epochs = 10
batch_size      = 100
display_step    = 1

# Network Parameters
n_hidden_1 = 256 # 1st layer num features
n_hidden_2 = 128 # 2nd layer num features
n_input    = 784 # MNIST data input (img shape: 28*28)
n_classes  = 10 # MNIST total classes (0-9 digits)

# tf Graph input
x = tf.placeholder("float", [None, n_input])
y = tf.placeholder("float", [None, n_classes])

# Create model
def multilayer_perceptron(_X, _weights, _biases):
    layer_1 = tf.nn.sigmoid(tf.add(tf.matmul(_X, _weights['h1']), _biases['b1']))
    layer_2 = tf.nn.sigmoid(tf.add(tf.matmul(layer_1, _weights['h2']), _biases['b2']))
    return (tf.sigmoid(tf.matmul(layer_2, _weights['out']) + _biases['out']))
    # return (tf.sigmoid(tf.matmul(layer_2, _weights['out']) + _biases['out']))

# Store layers weight & bias
stddev = 0.1 # <== This greatly affects accuracy!!
weights = {
    'h1': tf.Variable(tf.random_normal([n_input, n_hidden_1], stddev=stddev)),
    'h2': tf.Variable(tf.random_normal([n_hidden_1, n_hidden_2], stddev=stddev)),
    'out': tf.Variable(tf.random_normal([n_hidden_2, n_classes], stddev=stddev))
}
biases = {
    'b1': tf.Variable(tf.random_normal([n_hidden_1])),
    'b2': tf.Variable(tf.random_normal([n_hidden_2])),
    'out': tf.Variable(tf.random_normal([n_classes]))
}
print ("Network Ready")
```

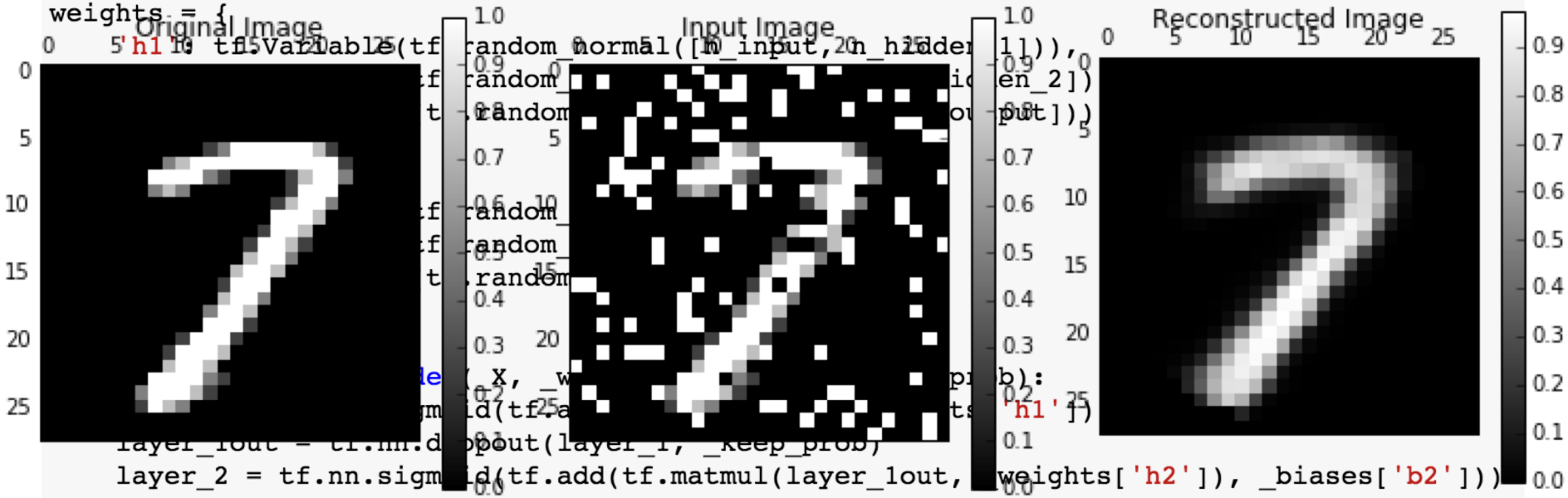
```
Network Ready
```

Generating Own Datasets

```
totallabel = np.ndarray((imgcnt, nclass))
imgcnt     = 0
for i, relpath in zip(range(nclass), paths):
    path = cwd + "/" + relpath
    flist = os.listdir(path)
    for f in flist:
        if os.path.splitext(f)[1].lower() not in valid_exts:
            continue
        fullpath = os.path.join(path, f)
        currimg  = imread(fullpath)
        # Convert to grayscale
        if use_gray:
            grayimg  = rgb2gray(currimg)
        else:
            grayimg  = currimg
        # Reshape
        graysmall = imresize(grayimg, [imgsize[0], imgsize[1]])/255.
        grayvec   = np.reshape(graysmall, (1, -1))
        # Save
        totalimg[imgcnt, :] = grayvec
        totallabel[imgcnt, :] = np.eye(nclass, nclass)[i]
        imgcnt    = imgcnt + 1

# Divide total data into training and test set
randidx  = np.random.randint(imgcnt, size=imgcnt)
trainidx = randidx[0:int(4*imgcnt/5)]
testidx  = randidx[int(4*imgcnt/5):imgcnt]

trainimg   = totalimg[trainidx, :]
trainlabel = totallabel[trainidx, :]
testimg    = totalimg[testidx, :]
testlabel  = totallabel[testidx, :]
```

Denoising Auto-Encoder

```
# tf Graph input
x = tf.placeholder("float", [None, n_input])
y = tf.placeholder("float", [None, n_output])
dropout_keep_prob = tf.placeholder("float")
# Store layers weight & bias
weights = {
    'h1': tf.Variable(tf.random_normal([n_input, n_hidden_1])),
                      tf.random_normal([n_hidden_2])
                      tf.random_normal([n_output])
                      tf.random_normal
                      tf.random_normal
                      tf.random_normal
le( x, _w                                  pr b):
                                           ts 'h1'])
    layer_1out = tf.nn.dropout(layer_1, _keep_prob)
    layer_2 = tf.nn.sigmoid(tf.add(tf.matmul(layer_1out, _weights['h2']), _biases['b2']))
    layer_2out = tf.nn.dropout(layer_2, _keep_prob)
    return tf.nn.sigmoid(tf.matmul(layer_2out, _weights['out']) + _biases['out'])
```



Convolutional Neural Network

```
weights  = {
    'wc1': tf.Variable(tf.random_normal([3, 3, 1, 64], stddev=0.1)),
    'wc2': tf.Variable(tf.random_normal([3, 3, 64, 128], stddev=0.1)),
    'wd1': tf.Variable(tf.random_normal([7*7*128, 1024], stddev=0.1)),
    'wd2': tf.Variable(tf.random_normal([1024, n_output], stddev=0.1))
}
biases   = {
    'bc1': tf.Variable(tf.random_normal([64], stddev=0.1)),
    'bc2': tf.Variable(tf.random_normal([128], stddev=0.1)),
    'bd1': tf.Variable(tf.random_normal([1024], stddev=0.1)),
    'bd2': tf.Variable(tf.random_normal([n_output], stddev=0.1))
}
def conv_basic(_input, _w, _b, _keepratio):
    # Input
    _input_r = tf.reshape(_input, shape=[-1, 28, 28, 1])
    # Conv1
```

```
# Compute test accuracy
test_acc = sess.run(accr, feed_dict={x: testimg, y: testlabel, keepratio:1.})
print (" Test accuracy: %.3f" % (test_acc))
```

```
Test accuracy: 0.994
```

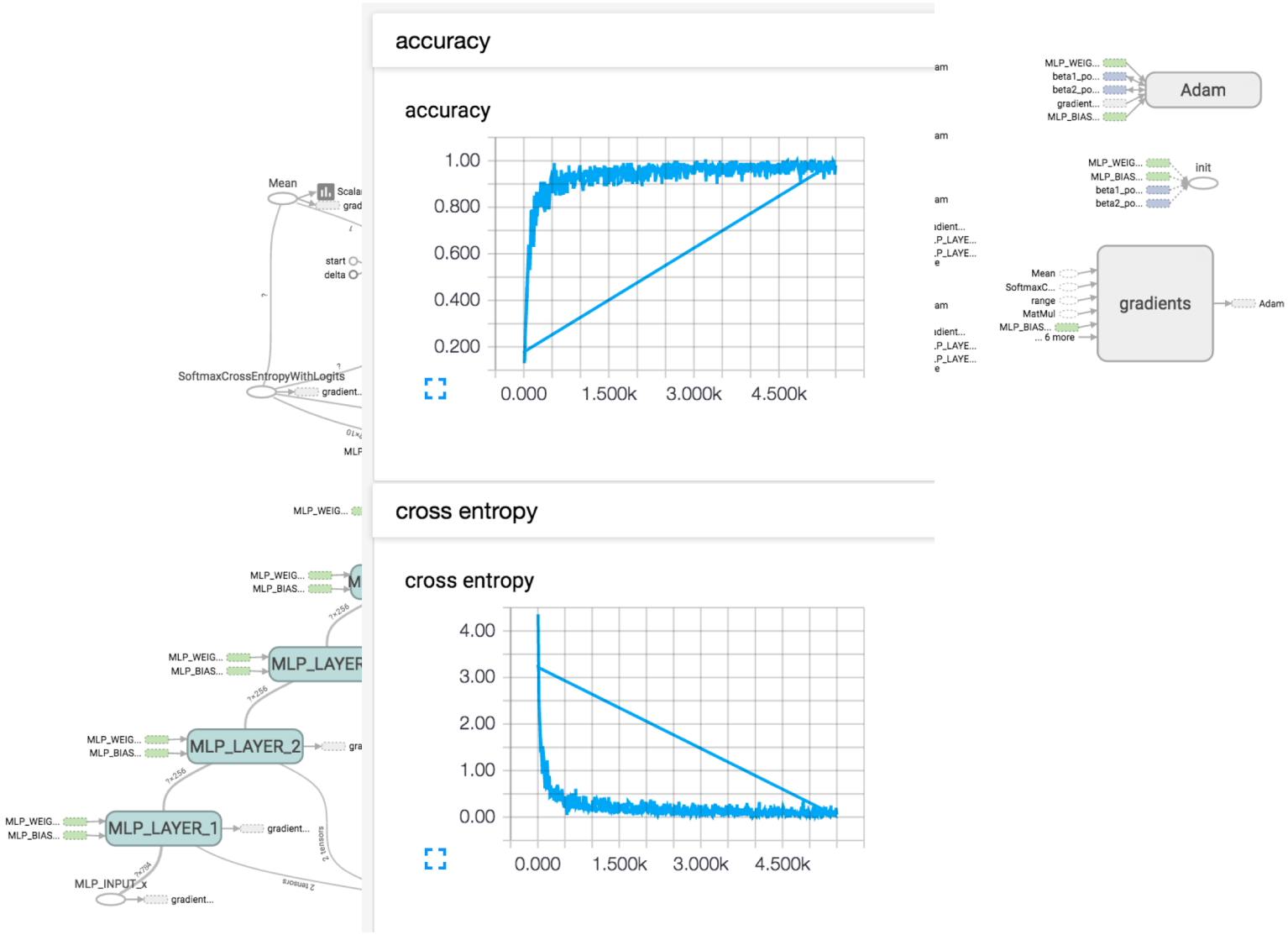
```
    # Vectorize
    _dense1 = tf.reshape(_pool_dr2, [-1, _w['wd1'].get_shape().as_list()[0]])
    # Fc1
    _fc1 = tf.nn.relu(tf.add(tf.matmul(_dense1, _w['wd1']), _b['bd1']))
    _fc_dr1 = tf.nn.dropout(_fc1, _keepratio)
    # Fc2
    _out = tf.add(tf.matmul(_fc_dr1, _w['wd2']), _b['bd2'])
    # Return everything
    out = {
        'input_r': _input_r,
        'conv1': _conv1,
        'pool1': _pool1,
        'pool1_dr1': _pool_dr1,
        'conv2': _conv2,
        'pool2': _pool2,
        'pool_dr2': _pool_dr2,
        'dense1': _dense1,
        'fc1': _fc1,
        'fc_dr1': _fc_dr1,
        'out': _out
    }
    return out
```

Semantic Segmentation (optional)

```
# DeconvNet Model
def Model(_X, _W, _b, _kee
    use_bias = 1
    # Encoder 128x128
    encoder1 = tf.nn.conv2
    if use_bias:
        encoder1 = tf.nn.b
    mean, var = tf.nn.mome
    encoder1 = tf.nn.batch
    encoder1 = tf.nn.relu(
    encoder1 = tf.nn.max_p
    encoder1 = tf.nn.drop
    # 64x64
    encoder2 = tf.nn.conv2
    if use_bias:
        encoder2 = tf.nn.b
    mean, var = tf.nn.mome
    encoder2 = tf.nn.batch
    encoder2 = tf.nn.relu(
    encoder2 = tf.nn.max_p
    encoder2 = tf.nn.drop
    # 32x32
    encoder3 = tf.nn.conv2
    if use_bias:
        encoder3 = tf.nn.b
    mean, var = tf.nn.mome
    encoder3 = tf.nn.batch
    encoder3 = tf.nn.relu(
    encoder3 = tf.nn.max_p
    encoder3 = tf.nn.drop
    # 16x16
    encoder4 = tf.nn.conv2
    if use_bias:
        encoder4 = tf.nn.b
    mean, var = tf.nn.mome
    encoder4 = tf.nn.batch
    encoder4 = tf.nn.relu(
    encoder4 = tf.nn.max_p
    encoder4 = tf.nn.drop
    # 8x8
```

```
# Decoder 8x8 (128/16 = 8) fsize: 64
decoder4 = Unpooling(encoder4, [tf.shape(_X)[0], height / 16, width / 16, fsize])
decoder4 = tf.nn.conv2d(decoder4, _W['cd4'], strides=[1, 1, 1, 1], padding='SAME')
if use_bias:
    decoder4 = tf.nn.bias_add(decoder4, _b['bd4'])
mean, var = tf.nn.moments(decoder4, [0, 1, 2])
decoder4 = tf.nn.batch_normalization(decoder4, mean, var, 0, 1, 0.0001)
decoder4 = tf.nn.relu(decoder4)
decoder4 = tf.nn.dropout(decoder4, _keepprob)
# 16x16
decoder3 = Unpooling(encoder3, [tf.shape(_X)[0], height/8, width/8, fsize])
decoder3 = tf.nn.conv2d(decoder3, _W['cd3'], strides=[1, 1, 1, 1], padding='SAME')
if use_bias:
    decoder3 = tf.nn.bias_add(decoder3, _b['bd3'])
mean, var = tf.nn.moments(decoder3, [0, 1, 2])
decoder3 = tf.nn.batch_normalization(decoder3, mean, var, 0, 1, 0.0001)
decoder3 = tf.nn.relu(decoder3)
decoder3 = tf.nn.dropout(decoder3, _keepprob)
# 32x32
decoder2 = Unpooling(decoder3, [tf.shape(_X)[0], height/4, width/4, fsize])
decoder2 = tf.nn.conv2d(decoder2, _W['cd2'], strides=[1, 1, 1, 1], padding='SAME')
if use_bias:
    decoder2 = tf.nn.bias_add(decoder2, _b['bd2'])
mean, var = tf.nn.moments(decoder2, [0, 1, 2])
decoder2 = tf.nn.batch_normalization(decoder2, mean, var, 0, 1, 0.0001)
decoder2 = tf.nn.relu(decoder2)
decoder2 = tf.nn.dropout(decoder2, _keepprob)
# 64x64
decoder1 = Unpooling(decoder2, [tf.shape(_X)[0], height / 2, width / 2, fsize])
decoder1 = tf.nn.conv2d(decoder1, _W['cd1'], strides=[1, 1, 1, 1], padding='SAME')
if use_bias:
    decoder1 = tf.nn.bias_add(decoder1, _b['bd1'])
mean, var = tf.nn.moments(decoder1, [0, 1, 2])
decoder1 = tf.nn.batch_normalization(decoder1, mean, var, 0, 1, 0.0001)
decoder1 = tf.nn.relu(decoder1)
decoder1 = tf.nn.dropout(decoder1, _keepprob)
# 128x128
output = tf.nn.conv2d(decoder1, _W['dense_inner_prod'], strides=[1, 1, 1, 1], padding='SAME
return output
```

Tensor Board



Recurrent Neural Network

```
# Recurrent neural network
diminput  = 28
dimhidden = 128
dimoutput = nclasses
nsteps    = 28
weights = {
    'hidden': tf.Variable(tf.random_normal([diminput, dimhidden])),
    'out': tf.Variable(tf.random_normal([dimhidden, dimoutput]))
}
biases = {
    'hidden': tf.Variable(tf.random_normal([dimhidden])),
    'out': tf.Variable(tf.random_normal([dimoutput]))
}
def _RNN(_X, _istate, _W, _b, _nsteps, _name):
    # 1. Permute input from [batchsize, nsteps, diminput] => [nsteps, batchsize, diminput]
    _X = tf.transpose(_X, [1, 0, 2])
    # 2. Reshape input to [nsteps*batchsize, diminput]
    _X = tf.reshape(_X, [-1, diminput])
    # 3. Input layer => Hidden layer
    _H = tf.matmul(_X, _W['hidden']) + _b['hidden']
    # 4. Splite data to 'nsteps' chunks. An i-th chunck indicates i-th batch data
    _Hsplit = tf.split(0, _nsteps, _H)
    # 5. Get LSTM's final output (_O) and state (_S)
    #    Both _O and _S consist of 'batchsize' elements
    with tf.variable_scope(_name):
        lstm_cell = rnn_cell.BasicLSTMCell(dimhidden, forget_bias=1.0)
        _LSTM_O, _LSTM_S = rnn.rnn(lstm_cell, _Hsplit, initial_state=_istate)
    # 6. Output
    _O = tf.matmul(_LSTM_O[-1], _W['out']) + _b['out']
    # Return!
    return {
        'X': _X, 'H': _H, 'Hsplit': _Hsplit,
        'LSTM_O': _LSTM_O, 'LSTM_S': _LSTM_S, 'O': _O
    }
print ("Network ready")
```

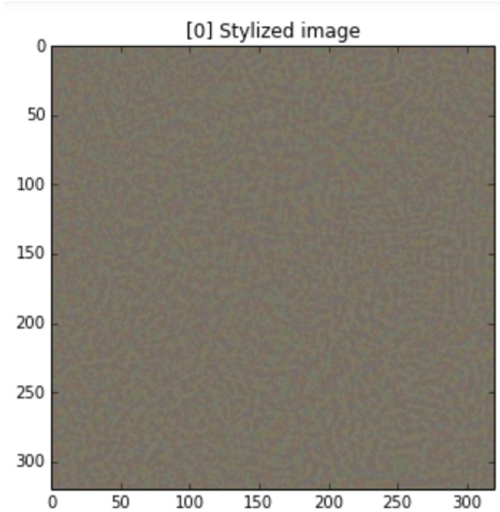
Word2vec

```
Nearest to 'john': 'easily', 'monasticism', 'lara', 'gerson', 'jacksonville', 'ceuta',
Nearest to 'a': 'the', 'amalthea', 'annales', 'aorta', 'bottles', 'incompressible',
Nearest to 'they': 'he', 'we', 'not', 'it', 'animists', 'dasyprocta',
Nearest to 'life': 'johansen', 'hyper', 'dissident', 'treasury', 'alejandro', 'zach',
Nearest to 'if': 'when', 'but', 'is', 'lemmy', 'immunology', 'triggering',
Nearest to 'work': 'dasyprocta', 'intervened', 'agouti', 'lumi', 'otimes', 'emanated',
Nearest to 'found': 'used', 'asymptotes', 'reginae', 'salinity', 'algebraic', 'discarded',
Nearest to 'an': 'the', 'dasyprocta', 'cdp', 'lerner', 'regarded', 'filthy',
Nearest to 'only': 'agave', 'trinomial', 'especially', 'agouti', 'aosta', 'chr',
Nearest to 'time': 'adiabatic', 'agouti', 'recitative', 'atacama', 'examination', 'parachutes',
Nearest to 'united': 'fusion', 'hephaestion', 'northern', 'victoriae', 'perry', 'remarried',
Nearest to 'UNK': 'abet', 'dasyprocta', 'aquila', 'agouti', 'victoriae', 'adiabatic',
Nearest to 'b': 'd', 'UNK', 'abet', 'swiss', 'lever', 'primigenius',
Nearest to 'its': 'their', 'the', 'his', 'permeable', 'dasyprocta', 'cl',
Nearest to 'against': 'punishments', 'specialized', 'during', 'landmasses', 'and', 'tillich',
Nearest to 'but': 'and', 'see', 'aorta', 'dasyprocta', 'that', 'discard',
Nearest to 'be': 'have', 'is', 'was', 'by', 'were', 'prosecution',
Nearest to 'by': 'was', 'from', 'with', 'is', 'as', 'in',
Nearest to 'system': 'staff', 'mutual', 'entitles', 'logic', 'argo', 'dasyprocta',
Nearest to 'at': 'in', 'agave', 'for', 'and', 'argo', 'aquila',
Nearest to 'history': 'rudolph', 'dasyprocta', 'plate', 'UNK', 'travels', 'fishing',
Nearest to 'm': 'three', 'six', 'shipper', 'eight', 'gb', 'hydrochloric',
Nearest to 'three': 'five', 'four', 'seven', 'six', 'two', 'eight',
Nearest to 'day': 'returning', 'severely', 'coup', 'close', 'techniques', 'garner',
Nearest to 'one': 'two', 'three', 'six', 'four', 'seven', 'five',
Nearest to 'up': 'him', 'them', 'pauline', 'loup', 'thr', 'dasyprocta',
Nearest to 'for': 'and', 'cl', 'with', 'in', 'at', 'dasyprocta',
Nearest to 'long': 'otimes', 'er', 'abet', 'toto', 'ram', 'threats',
Nearest to 'power': 'dasyprocta', 'thief', 'yin', 'albuquerque', 'trinomial', 'clips',
Nearest to 'second': 'apollonia', 'imran', 'doped', 'burnt', 'clay', 'lenoir',
Nearest to 'state': 'gonna', 'laid', 'victoriae', 'agouti', 'dasyprocta', 'leipzig',
Nearest to 'are': 'were', 'is', 'have', 'was', 'aorta', 'but',
```

Char-RNN

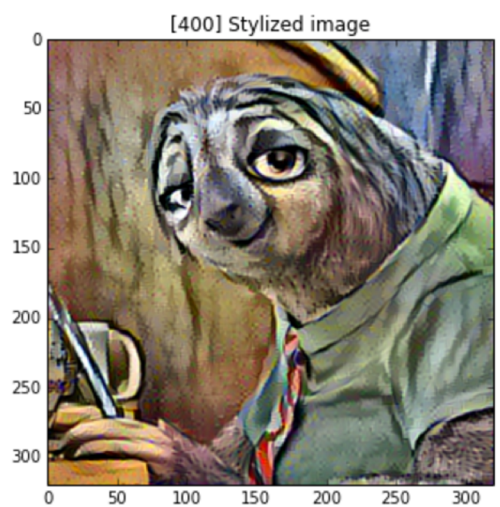
```
-- RESULT --
안녕하시오 또사. 소가 따를 젖인
도리는 일진과 같살이로다!'
  하고. 이에 속각히 결접을 때어서 정쇠와 형뉸의
뒤를 비칠 즉 탄 솔아 어느 대첩기를 기키라고
부든 낭자가 화상하여 보니 용승상의 무전한 선히 쓰게 말씀드리는서? 이에 보고 너늘 등하나 숙생이
저게 맘여하옵고 삼깨를
지어코지 지소하고 영영호를 뜨사보면 (알골에 배후에께서 승상이 이에 한심 용광이라도 스스로 또함이옵니다.
  크게 하오니 갚자고 몬든 황상께서는 ㅂㄱ를로 항께 대하고 천해 중어면 마르겨 후벅을 하기 그 붉으니 섬ㅇ 깁에서 옷을
맞아 달 굿이 타접이 좋을 덧을 꽃어 아야보매 위원의 정시전정 톤후백을 기삔 다른 마음에 같지 아니고,
이인이 소녀나매, 구름은 봉연하오니 그 말씀으로 길어들이 홤장의 무료한
가신이요. 주공은 어매 이를 삼고 다 선녀에는 마던
직기에게 가득 치르고 정인을 뭃어 돌아갔다.
배가 딸라 양상서가 넡을 거녀를 얻어 가지고 한 나랄펴 하나로가 무음 앞으로 나무를 떠나 마시고자
돌아오는 청자의 옛말으로 함주를 거느리운
말씀는데 이렇게 돌이겠나오다. 승상이 시를 굳은
족대서 간약이 있음이니 토한 네 없다군지를 나중히 여래보미를
모시디 드시자 코를 도리어 인간 술명이 돌아.

  행연해도반하옵기는데, 중사 위에 난묵의 이름을 보요로 마음을 분운을 시겿어 가며
구궐하지 아니하옵니다. 뚜가 여질
순닥을 조하도 몽처능히 위로 때 아뢰었다.
  "모어 사만백 효행체유이오. 승상이 한마들을 마음깨운 적이 다 경세는 아
```

Neural style (optional)

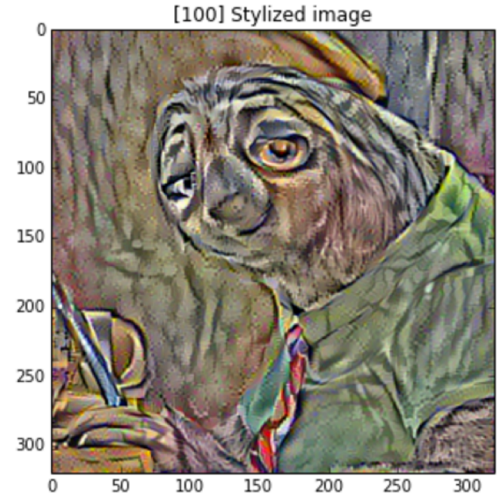


[100/1000]

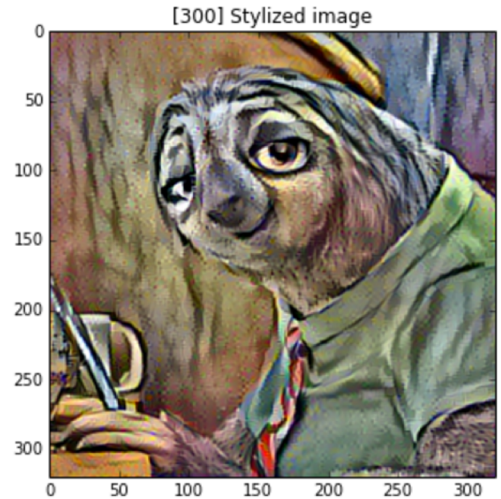
[300/1000]



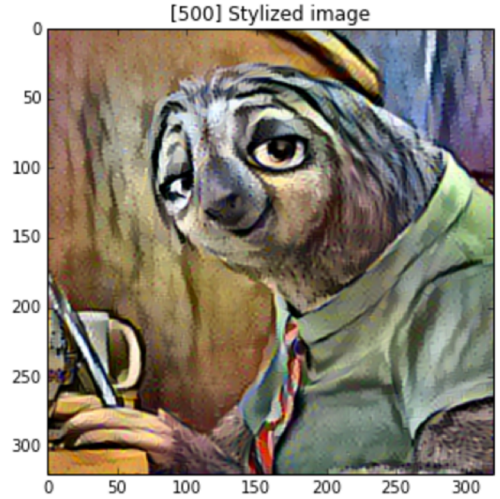
[500/1000]



[200/1000]

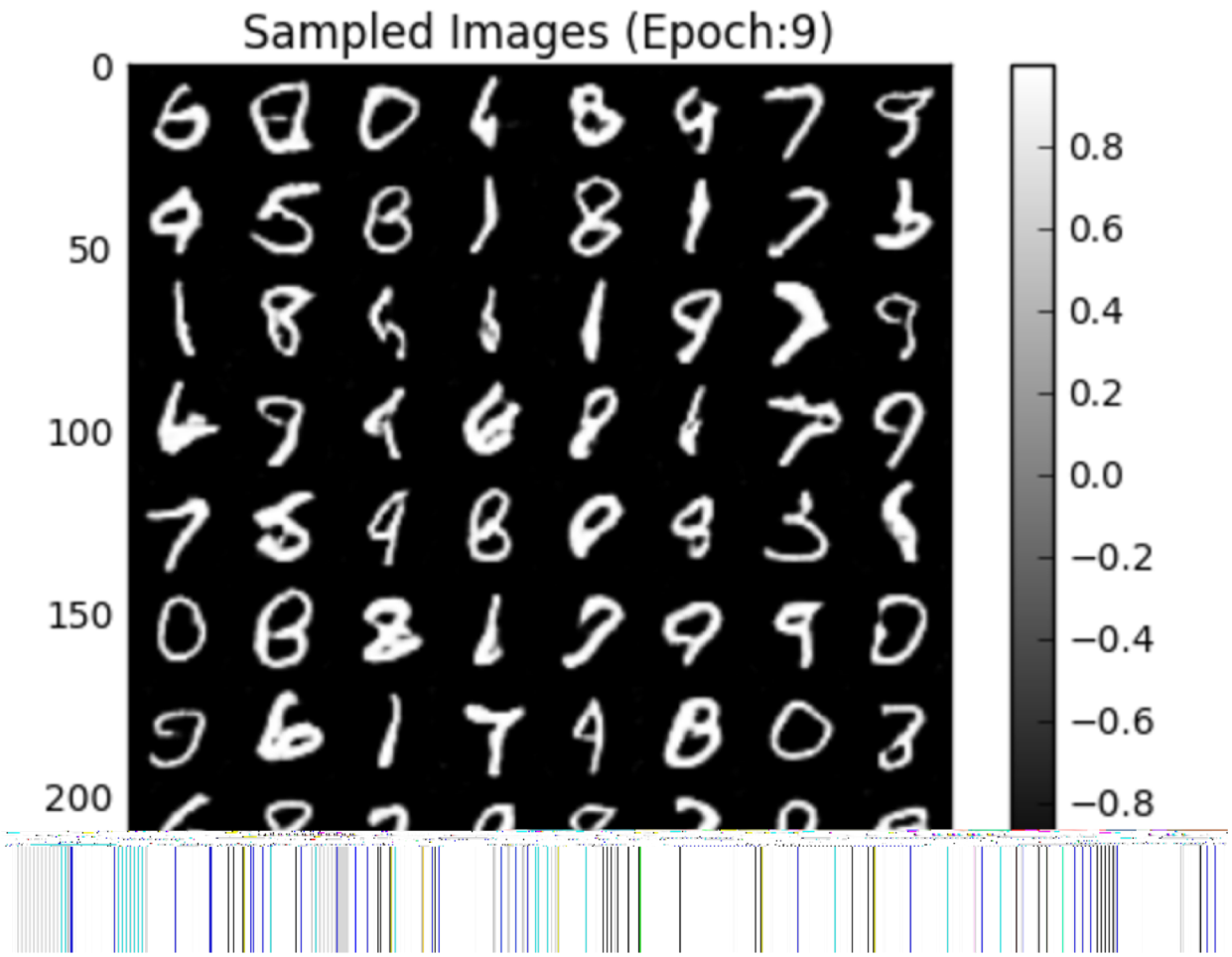


[400/1000]

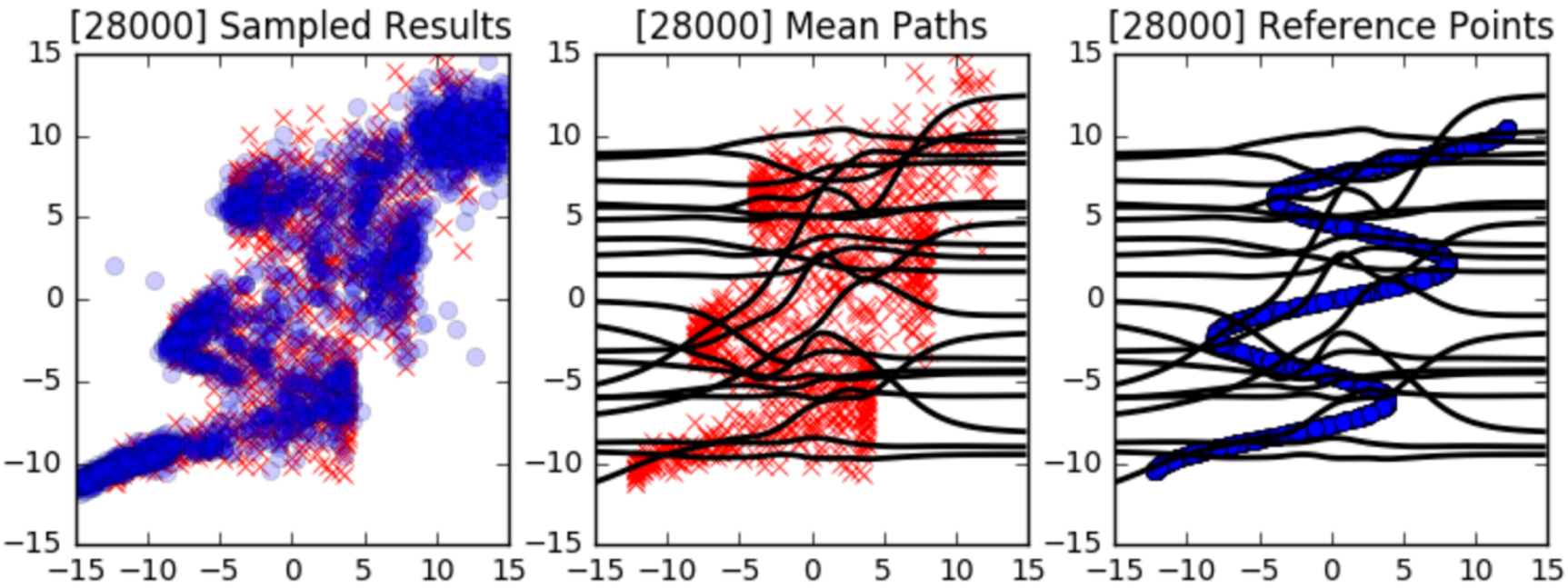


[600/1000]

DCGAN

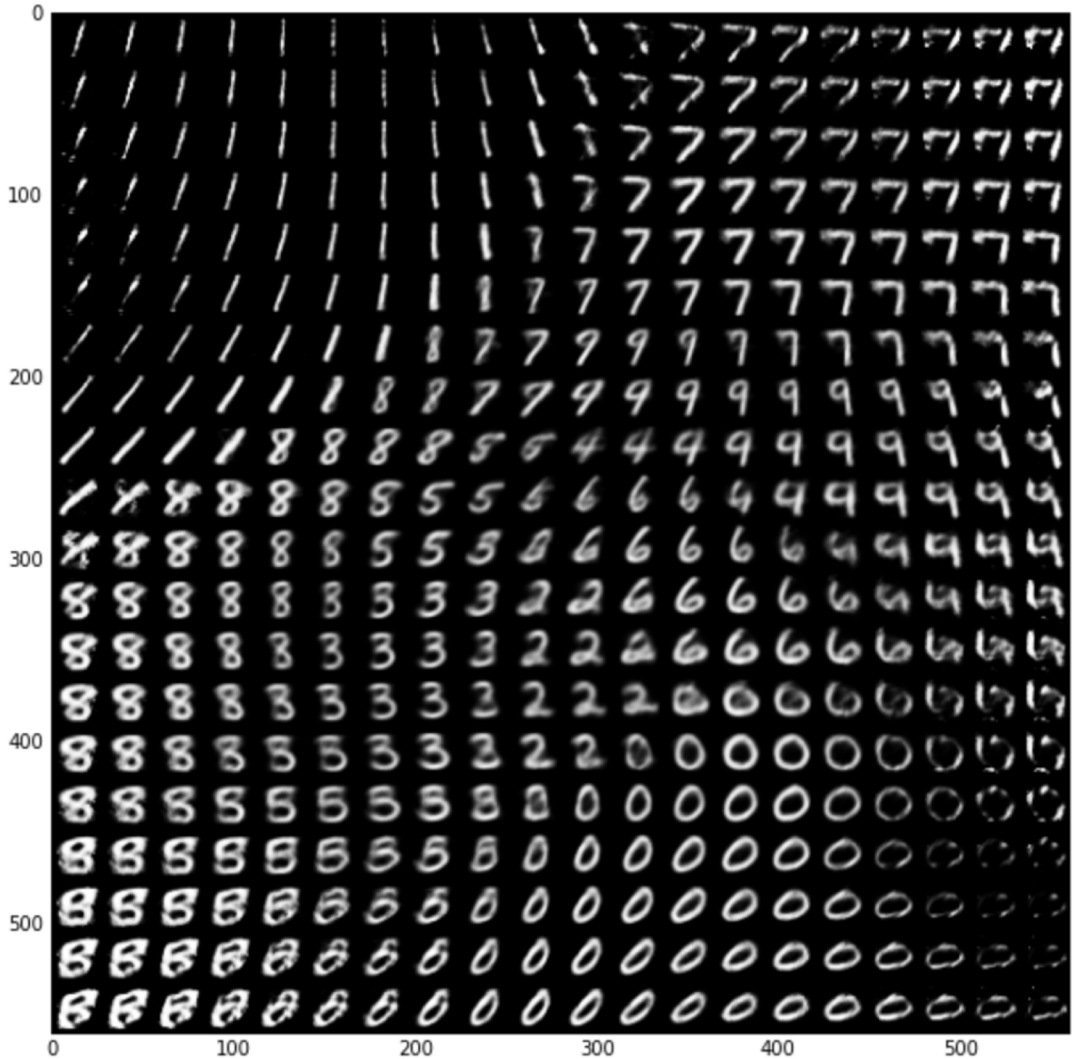


Mixture Density Network

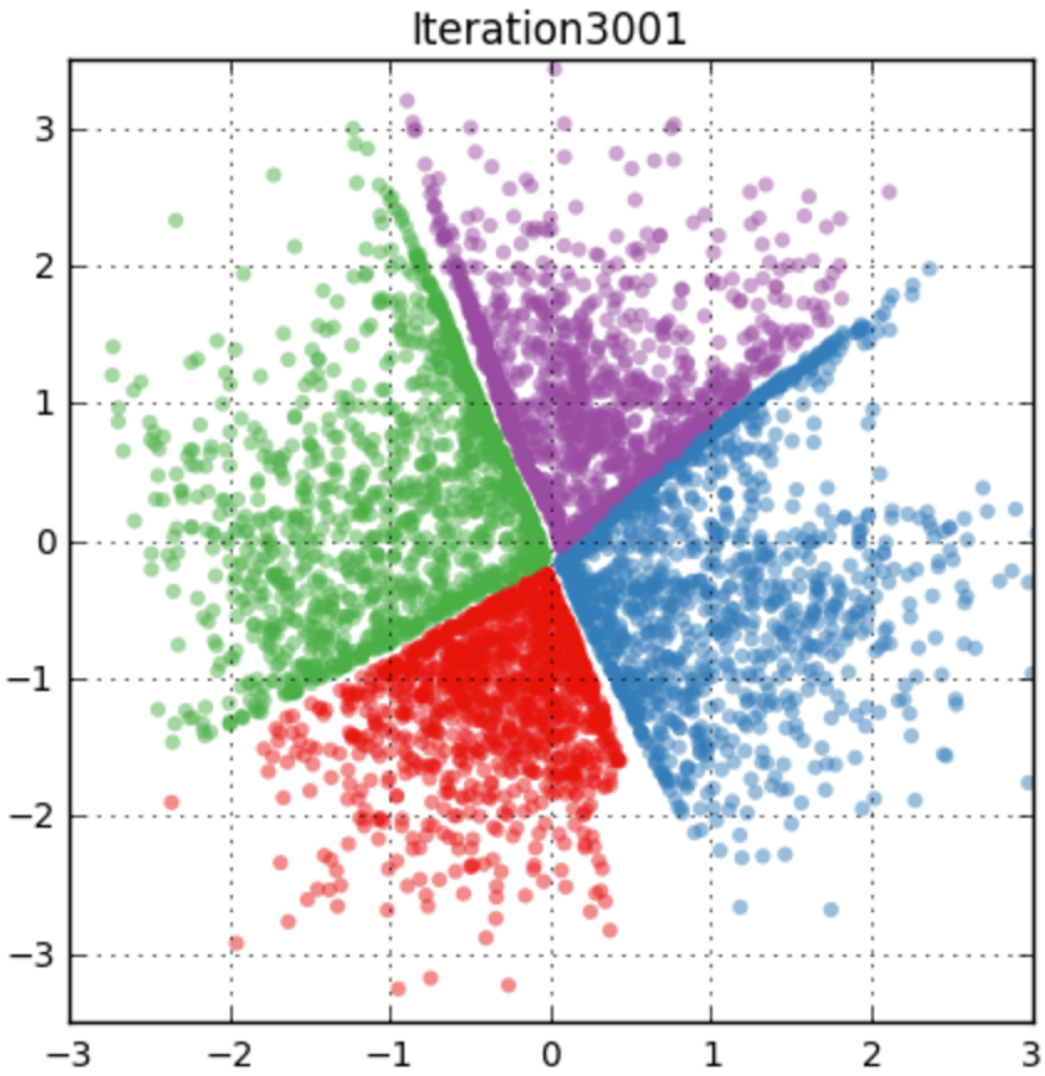


Domain Adversarial Network

Variational Autoencoder (VAE)



Adversarial Variational Bayes



Advanced Topics

This is optional and I would LOVE to discuss about this with YOU!

One-Shot Learning

- 2016 One-shot Learning with Memory-Augmented Neural Networks
- 2016 Matching Networks for One Shot Learning

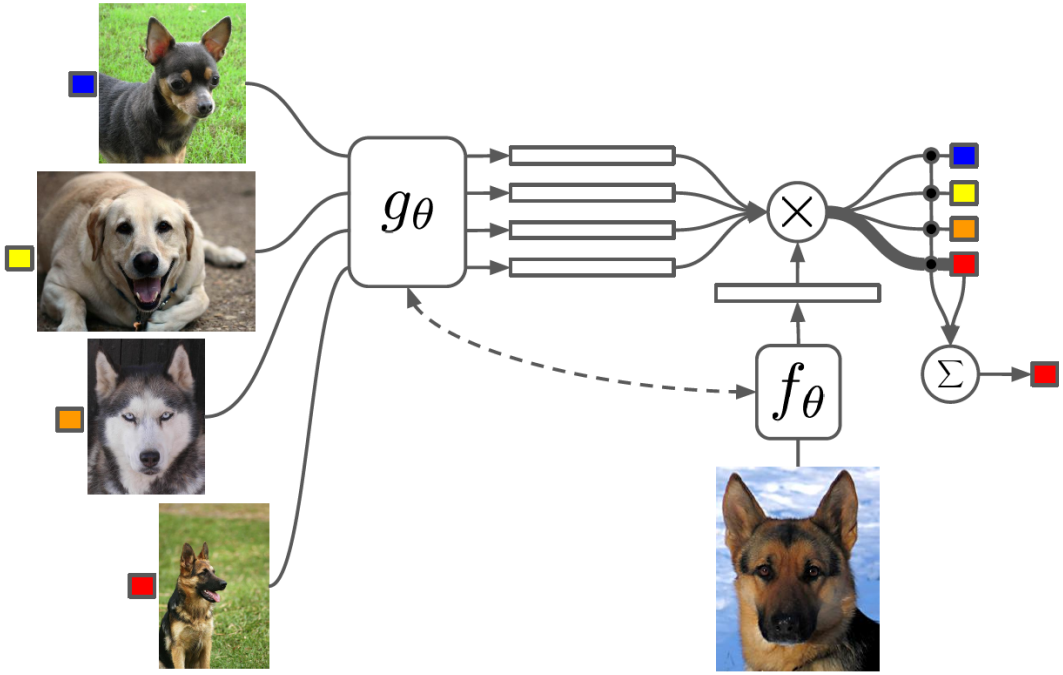


Figure 1: Matching Networks architecture

Domain Adaptation

- 2016 Domain Separation Networks
- 2016 Domain-Adversarial Training of Neural Networks

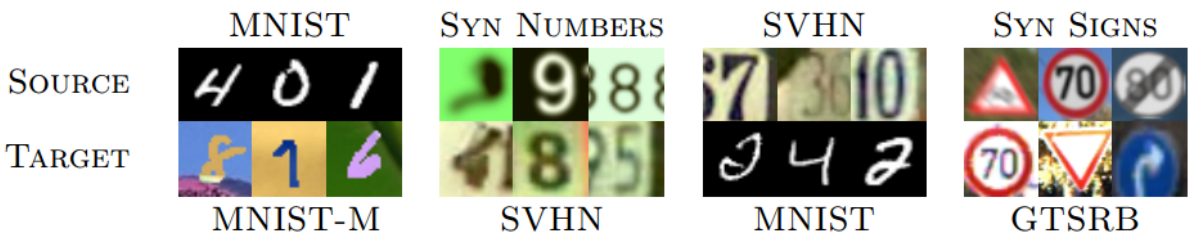


Figure 6: Examples of domain pairs used in the experiments. See Section 5.2.4 for details.

Method / Source / Target	MNIST / MNIST-M	Syn Numbers / SVHN	SVHN / MNIST	Syn Signs / GTSRB
Source only	.5225	.8674	.5490	.7900
SA (Fernando et al., 2013)	.5690 (4.1%)	.8644 (−5.5%)	.5932 (9.9%)	.8165 (12.7%)
DANN	**.7666** (52.9%)	**.9109** (79.7%)	**.7385** (42.6%)	**.8865** (46.4%)
Train on target	.9596	.9220	.9942	.9980

Metric Learning

- 2015 DEEP METRIC LEARNING USING TRIPLET NETWORK
- 2016 Deep Metric Learning via Lifted Structured Feature Embedding

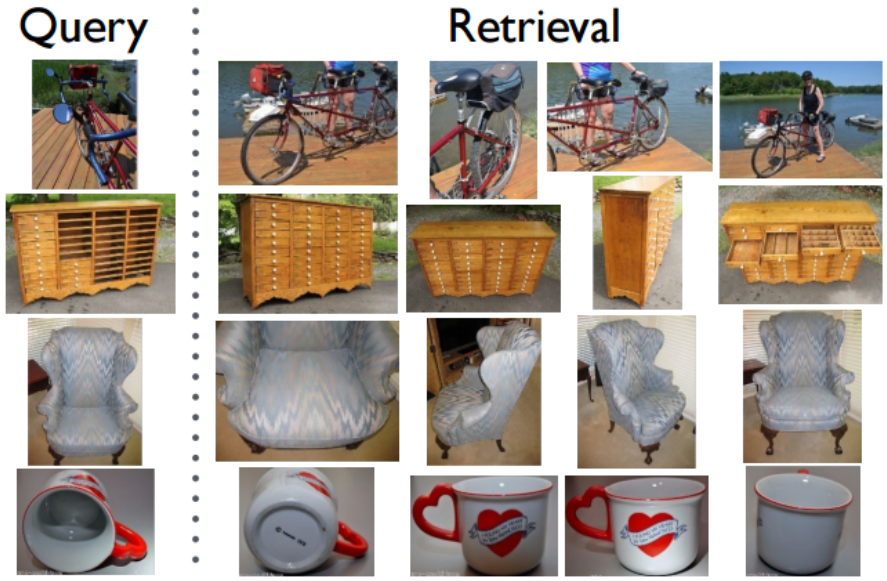
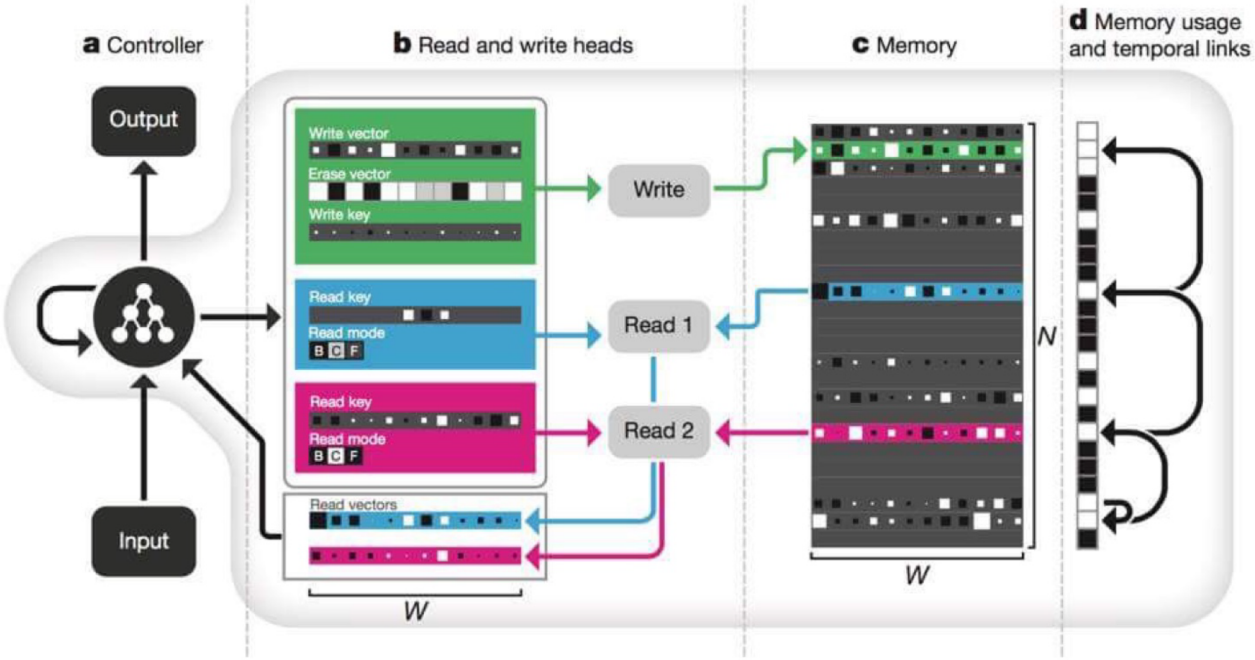


Figure 1: Example retrieval results on our *Online Products* dataset using the proposed embedding. The images in the first column are the query images.

Memory Network

- 2016 Hybrid computing using a neural network with dynamic external memory



Other Interesting Papers

- 2016 Learning without Forgetting
- 2016 Unsupervised Learning for Physical Interaction through Video Prediction

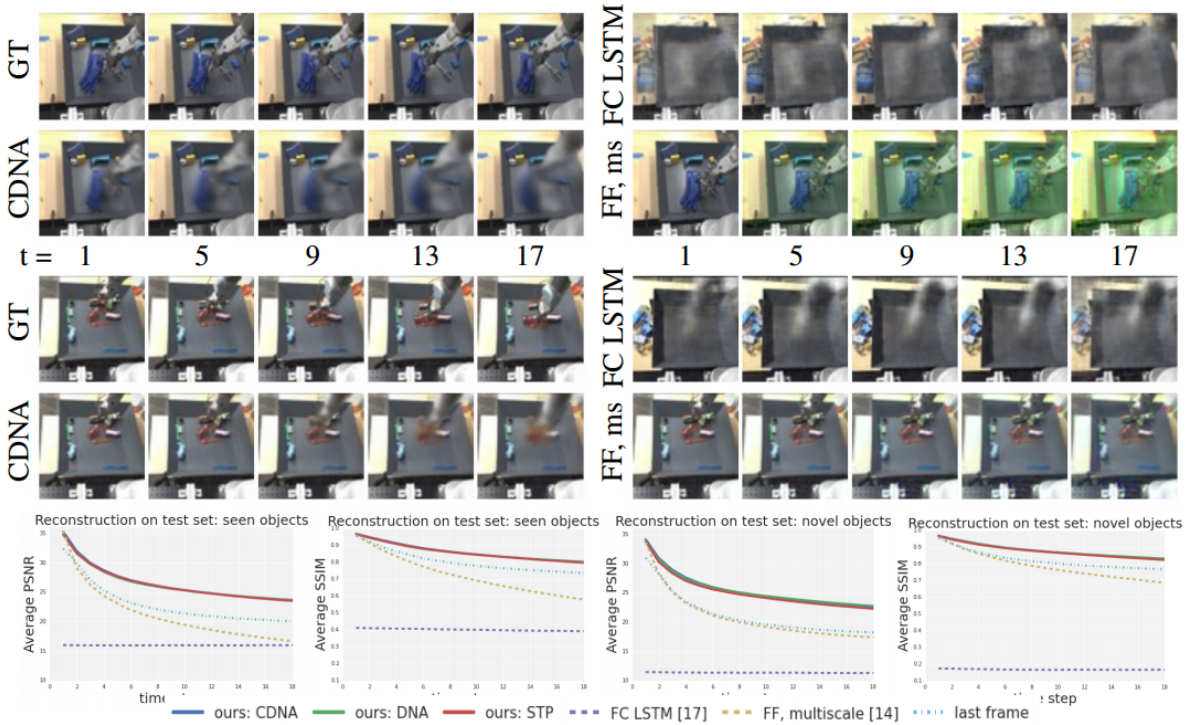
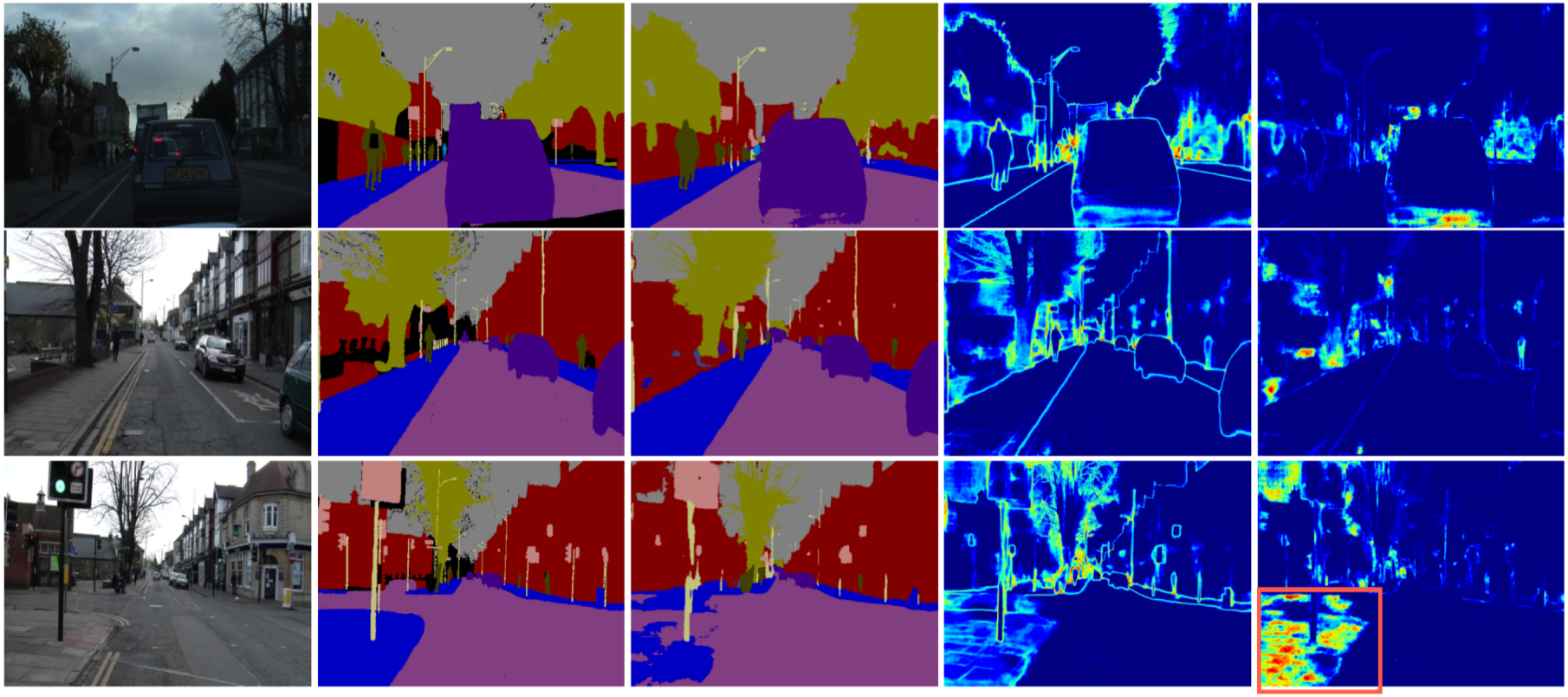


Figure 3: Qualitative and quantitative reconstruction performance of our models, compared with [20, 17]. All models were trained for 8-step prediction, except [17], trained for 1-step prediction.

Uncertainty in Neural Networks



(a) Input Image (b) Ground Truth (c) Semantic Segmentation (d) Aleatoric Uncertainty (e) Epistemic Uncertainty

관측 에러 모델 에러

+ a Hell of a Lot More!

2013 Interactive Learning for Sequential Decisions and Predictic
2015 Learning Continuous Control Policies by Stochastic Value
2015 Learning of Non-Parametric Control Policies with High-Din
2015 Maximum Entropy Deep Inverse Reinforcement Learning.p
2015 Policy Gradient Methods for Reinforcement Learning with
2015 Policy Search- Methods and Applications - Tutorial.pdf
2016 A Connection Between Generative Adversarial Networks, I
2016 A New Softmax Operator for Reinforcement Learning.pdf
2016 Generative Adversarial Imitation Learning .pdf
2016 Guided Cost Learning Deep Inverse Optimal Control via P
2016 Incorporating Human Domain Knowledge into Large Scale
2016 Inverse Reinforcement Learning from Failure.pdf
2016 John Schulman Thesis.pdf
2016 Mastering_the_game_of_Go_with_deep_neural_networks_
2016 Model-based Adversarial Imitation Learning.pdf
2016 Safe, Multi-Agent, Reinforcement Learning for Autonomou
2016 Transfer from Simulation to Real World through Learning D
2016 Trust Region Policy Optimization.pdf
2016 Tutorial- Deep Reinforcement Learning.pdf
2016 Watch This Scalable Cost-Function Learning for Path Plan
2017 AutonoVi- Autonomous Vehicle Planning with Dynamic Ma
2017 C-LEARN- Learning Geometric Constraints from Demonstr
2017 Connecting Generative Adversarial Networks and Actor-Cr
2017 Curiosity-driven Exploration by Self-supervised Prediction
2017 Deeply AggreVaTeD- Differentiable Imitation Learning for S
2017 DESIRE Distant Future Prediction in Dynamic Scenes with
2017 Equivalence Between Policy Gradients and Soft Q-Learnin
2017 GENERATIVE MULTI-ADVERSARIAL NETWORKS.pdf
2017 Inferring The Latent Structure of Human Decision-Making
2017 Inverse Reinforcement Learning via Deep Gaussian Proces
2017 Learning from Demonstrations for Real World Reinforceme
2017 LEARNING TO ACT BY PREDICTING THE FUTURE.pdf
2017 Learning to guide task and motion planning using score-sp
2017 One-Shot Imitation Learning.pdf
2017 Project-based, Collaborative, Algorithmic Robotics for Hig
2017 Reinforcement Learning with Deep Energy-Based Policies.
2017 Risk-sensitive Inverse Reinforcement Learning via Coherer
2017 SeqGAN- Sequence Generative Adversarial Nets with Poli
2017 Simultaneous Policy Learning and Latent State Inference f
2017 UNSUPERVISED PERCEPTUAL REWARDS FOR IMITATIOI

2016 Image-to-Image Translation with Conditional Adversarial Networks.pdf
2016 InfoGAN- Interpretable Representation Learning by Information Maximizing Generative Adversarial Nets.pdf
2016 Learning from Simulated and Unsupervised Images through Adversarial Training.pdf
2016 Learning What and Where to Draw.pdf
2016 Manifold Learning with Contracting Observers for Data-driven Time-series Analysis.pdf
2016 Mixing Dirichlet Topic Models and Word Embeddings to Make lda2vec.pdf
2016 NEURAL PHOTO EDITING WITH INTROSPECTIVE ADVERSARIAL NETWORKS.pdf
2016 Pixel-Level Domain Transfer.pdf
2016 Plug & Play Generative Networks- Conditional Iterative Generation of Images in Latent Space.pdf
2016 Pyramid Scene Parsing Network.pdf
2016 Synthesizing the preferred inputs for neurons in neural networks via deep generator networks.pdf
2016 Trust Region Policy Optimization.pdf
2016 UNDERSTANDING DEEP LEARNING REQUIRES RETHINKING GENERALIZATION.pdf
2016 WAVENET- A GENERATIVE MODEL FOR RAW AUDIO.pdf
2017 Adaptive Computation Time for Recurrent Neural Networks.pdf
2017 Adversarial Discriminative Domain Adaptation.pdf
2017 Adversarial Variational Bayes Unifying Variational Autoencoders and Generative Adversarial Networks.pdf
2017 Boundary-Seeking Generative Adversarial Networks.pdf
2017 Coordinated Multi-Agent Imitation Learning.pdf
2017 Data-Driven Ghosting using Deep Imitation Learning.pdf
2017 DeepNav Learning to Navigate Large Cities.pdf
2017 Deformable Convolutional Networks.pdf
2017 ENERGY-BASED GENERATIVE ADVERSARIAL NETWORKS.pdf
2017 FeUdal Networks for Hierarchical Reinforcement Learning.pdf
2017 GENERATIVE ADVERSARIAL NETWORKS AS VARIA- TIONAL TRAINING OF ENERGY BASED MODELS.pdf
2017 GENERATIVE MULTI-ADVERSARIAL NETWORKS.pdf
2017 Highway and Residual Networks learn Unrolled Iterative Estimation.pdf
2017 Imitating Driver Behavior with Generative Adversarial Networks.pdf
2017 Just DIAL Domain Alignment Layers for Unsupervised Domain Adaptation.pdf
2017 Learning to Discover Cross-Domain Relations with Generative Adversarial Networks.pdf
2017 Learning to Generate Long-term Future via Hierarchical Prediction.pdf
2017 Learning What Data to Learn.pdf
2017 Machine Learning on Sequential Data Using a Recurrent Weighted Average.pdf
2017 MAKING NEURAL PROGRAMMING ARCHITECTURES GENERALIZE VIA RECURSION.pdf
2017 Mask R-CNN.pdf
2017 Mixture of Counting CNNs- Adaptive Integration of CNNs Specialized to Specific Appearance for Crowd Counting.pdf
2017 One-Shot Imitation Learning.pdf
2017 OptNet- Differentiable Optimization as a Layer in Neural Networks.pdf
2017 Overcoming catastrophic forgetting in neural networks.pdf
2017 PathNet Evolution Channels Gradient Descent in Super Neural Networks.pdf
2017 Show, Attend and Interact Perceivable Human-Robot Social Interaction through Neural Attention Q-Network.pdf
2017 The Mythos of Model Interpretability.pdf
2017 TOWARDS PRINCIPLED METHODS FOR TRAINING GENERATIVE ADVERSARIAL NETWORKS.pdf
2017 Uncertainty in Deep Learning.pdf
2017 Unpaired Image-to-Image Translation using Cycle-Consistent Adversarial Networks.pdf
2017 UNSUPERVISED CROSS-DOMAIN IMAGE GENERATION.pdf
2017 Wasserstein GAN.pdf
2017 What Uncertainties Do We Need in Bayesian Deep Learning for Computer Vision.pdf
2017 YOLO9000 Better, Faster, Stronger.pdf