UNSUPERVISED IMAGE SEGMENTATION BY BACKPROPAGATION

AIST Asako Kanezaki
National Institute of Advanced Industrial Science and Technology (AIST)

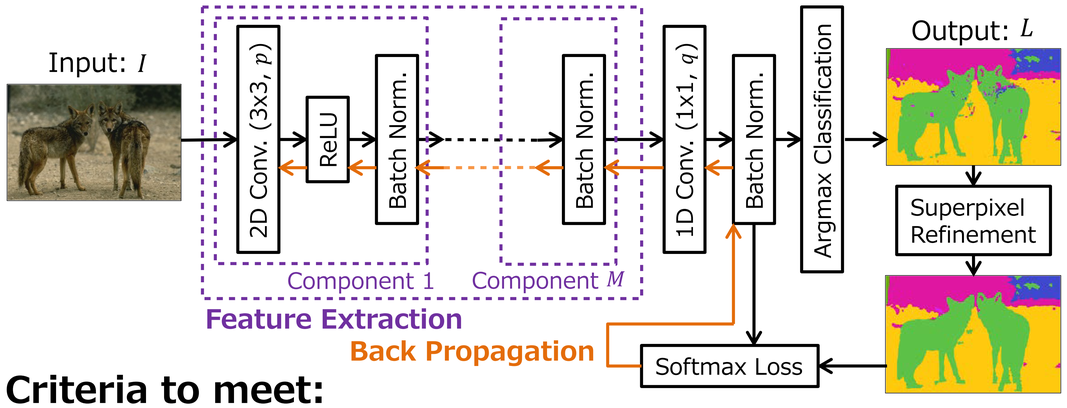
https://kanezaki.github.io/pytorch-unsupervised-segmentation/

OVERVIEW

- No need to train.
- No pre-trained weights required.
- Our CNN only requires a test image as input and then it outputs segmentation labels.

image ➡ CNN ⇢ labels

METHOD



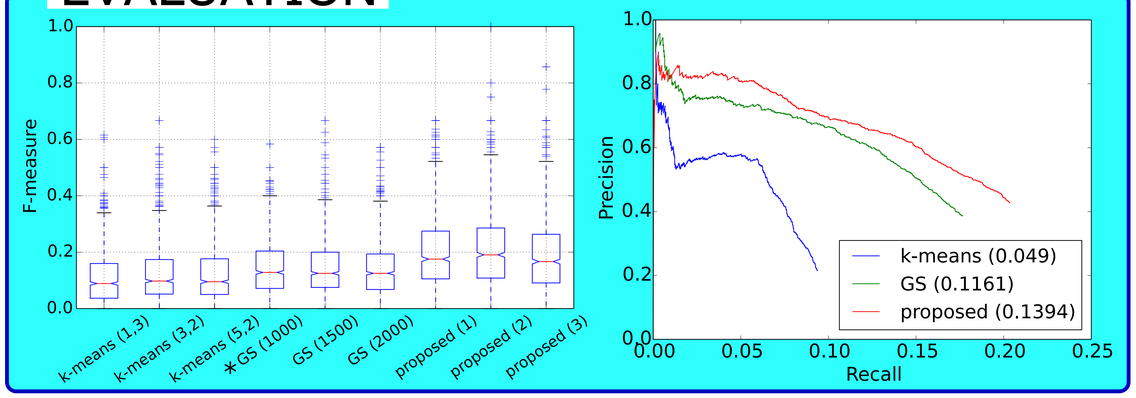
Criteria to meet:

(a) pixels of similar features are desired to be assigned the same label

(b) spatially continuous pixels are desired to be assigned the same label

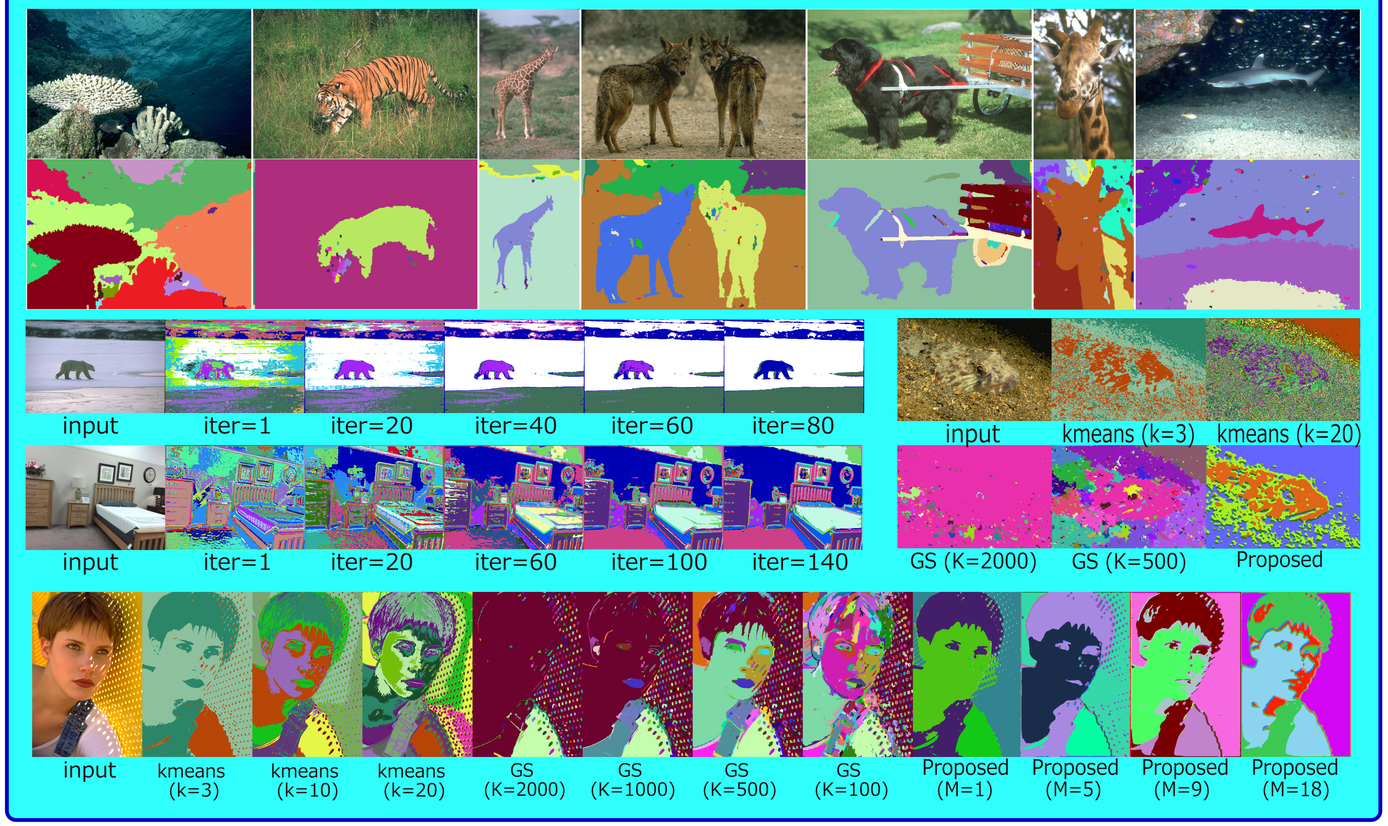
(c) the number of unique labels is desired to be large

EVALUATION



*Felzenszwalb and Huttenlocher, "Efficient graph-based image segmentation," Int. J. of Computer Vision, vol. 59, no. 2, 2004.

RESULTS



Code is available:

https://kanezaki.github.io/pytorch-unsupervised-segmentation/

The actual code consists of few lines. ➡

Algorithm 1: Unsupervised image segmentation

Input: $\mathcal{I} = \{\boldsymbol{v}_n \in \mathbb{R}^3\}_{n=1}^N$ // RGB image
Output: $\mathcal{L} = \{c_n \in \mathbb{Z}\}_{n=1}^N$ // Label image
$\{W_m, \boldsymbol{b}_m\}_{m=1}^M \leftarrow$ Init() // Initialize
$\{W_c, \boldsymbol{b}_c\} \leftarrow$ Init() // Initialize
$\{\mathcal{S}_k\}_{k=1}^K \leftarrow$ GetSuperPixels($\{\boldsymbol{v}_n\}_{n=1}^N$)
for $t = 1$ **to** T **do**
  $\{\boldsymbol{x}_n\}_{n=1}^N \leftarrow$ GetFeats($\{\boldsymbol{v}_n\}_{n=1}^N, \{W_m, \boldsymbol{b}_m\}_{m=1}^M$)
  $\{\boldsymbol{y}_n\}_{n=1}^N \leftarrow \{ W_c \boldsymbol{x}_n + \boldsymbol{b}_c \}_{n=1}^N$
  $\{\boldsymbol{y}'_n\}_{n=1}^N \leftarrow$ Norm($\{\boldsymbol{y}_n\}_{n=1}^N$) // Batch norm.
  $\{c_n\}_{n=1}^N \leftarrow \{ \arg\max \boldsymbol{y}'_n \}_{n=1}^N$ // Assign labels
  for $k = 1$ **to** K **do**
    $c_{\max} \leftarrow \arg\max |c_n|_{n \in \mathcal{S}_k}$
    $c'_n \leftarrow c_{\max}$ **for** $n \in \mathcal{S}_k$
  $\mathscr{L} \leftarrow$ SoftmaxLoss($\{\boldsymbol{y}'_n, c'_n\}_{n=1}^N$)
  $\{W_m, \boldsymbol{b}_m\}_{m=1}^M, \{W_c, \boldsymbol{b}_c\} \leftarrow$ Update($\mathscr{L}$)

```
# load image
im = cv2.imread(args.input)
data = Variable(torch.from_numpy(np.array([im.transpose((2,0,1)).astype('float32')/255.]) ))

# superpixels
labels = segmentation.slic(im, compactness=args.compactness, n_segments=args.num_superpixels)
labels = labels.reshape(im.shape[0]*im.shape[1])
u_labels = np.unique(labels)
l_inds = []
for i in range(len(u_labels)):
    l_inds.append( np.where( labels == u_labels[ i ] )[ 0 ] )

# CNN training
model = MyNet( data.size(1) )
loss_fn = torch.nn.CrossEntropyLoss()
optimizer = optim.SGD(model.parameters(), lr=args.lr, momentum=0.9)
for batch_idx in range(args.maxIter):
    optimizer.zero_grad()
    output = model( data )[ 0 ]
    output = output.permute( 1, 2, 0 ).contiguous().view( -1, args.nChannel )
    ignore, target = torch.max( output, 1 )
    im_target = target.data.cpu().numpy()
    nLabels = len(np.unique(im_target))
    for i in range(len(l_inds)):
        labels_per_sp = im_target[ l_inds[ i ] ]
        u_labels_per_sp = np.unique( labels_per_sp )
        hist = np.zeros( len(u_labels_per_sp) )
        for j in range(len(hist)):
            hist[ j ] = len( np.where( labels_per_sp == u_labels_per_sp[ j ] )[ 0 ] )
        im_target[ l_inds[ i ] ] = u_labels_per_sp[ np.argmax( hist ) ]
    target = Variable( torch.from_numpy( im_target ) )
    loss = loss_fn(output, target)
    loss.backward()
    optimizer.step()
```