

# Human Parsing with Contextualized Convolutional Neural Network

Project Final Evaluation

## Team:

Team Name: Illuminati

Team Number: 28

Team Members:

1. Pradeep Yarlagadda ( 20161164 )
2. K.L.N.Saketh ( 20161226 )
3. Koganti Sai Venkat ( 20161238 )

## Github Link:

<https://github.com/agentcap/CoCNN>

## Presentation Link:

[Link to presentation Slides](#)

## Introduction

### Problem Statement

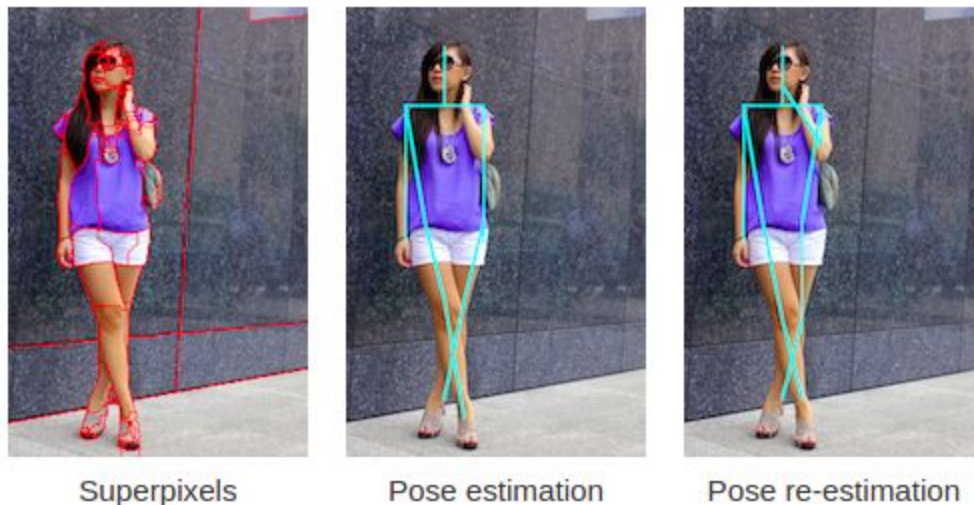
- Human parsing refers to decomposing a human image into semantic clothes/body regions and labeling them.
- The problem is to perform human parsing on the input human image and produce the correspondingly-sized pixel-wise labeling map in a fully end-to-end way
- It has an important role in general human-centric analysis, clothing style recognition, automatic product recommendation, and many other areas.



### Existing Methods

- They are based on human pose estimation, non-parametric label transferring and active template regression.
- There are also some methods which perform the task using Semantic Segmentation with CNN.

### Clothing Parsing ( YamaGuchi ) : (Prerequisite => Pose Estimation)



### Limitations with these methods

- **None of the previous methods has achieved excellent dense prediction (that is equal sized labeled output)** over raw image pixels in a fully end-to-end way.
- The **outputs are very coarse** and not optimal for the required fine-grained segmentation. CNN can only predict the very low-resolution labeling, such as eight times down-sampled prediction.
- These methods often take **complicated preprocessing** as the requisite, such as reliable human pose estimation, bottom-up hypothesis generation etc. This makes the **system vulnerable to potential errors of the requisites**.
- **Diverse contextual information** and **mutual relationships among the key components** of human parsing should be well addressed during prediction. This is **not performed** well in the existing methods.

## Advantages of the Proposed Method

- It well **integrates the different types of context into a unified network** to capture contextual information.
- **Does not depend on any other requisites.**
- Predicts a **dense output** that is correspondingly-sized pixel-wise labelling, there is no reduction in the size of the image.

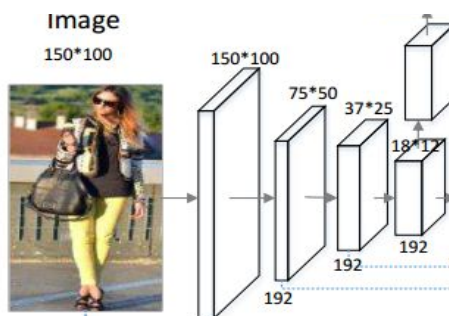
## Proposed Method

### Overview

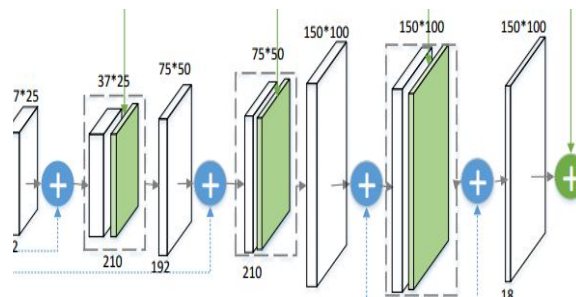
In this method , we integrate four different components into a unified network :

#### 1. Cross - Layer Context :

- The basic **local-to-global-to-local structure** captures the cross-layer context. It simultaneously considers the **local fine details** and **global structure information**.
- The feature maps are down-sampled three times to consider different scales , the early convolutional layers with high spatial resolutions (e.g., 150×100) capture more local details
- The ones with low spatial resolutions (e.g.,  $18 \times 12$ ) capture more structure information with high-level semantics.
- We transform the coarse outputs (e.g., with resolution  $18 \times 12$ ) to dense outputs (e.g., with resolution  $37 \times 25$ ) with up-sampling interpolation of factor 2.
- We combine the local fine details and the high level structure information by **cross-layer aggregation** ( element-wise summation -> shown using blue circle in image ) of early fine layers and up-sampled deep layers.
- **Loss function** is the **sum of cross-entropy terms** for all pixels in the output map.



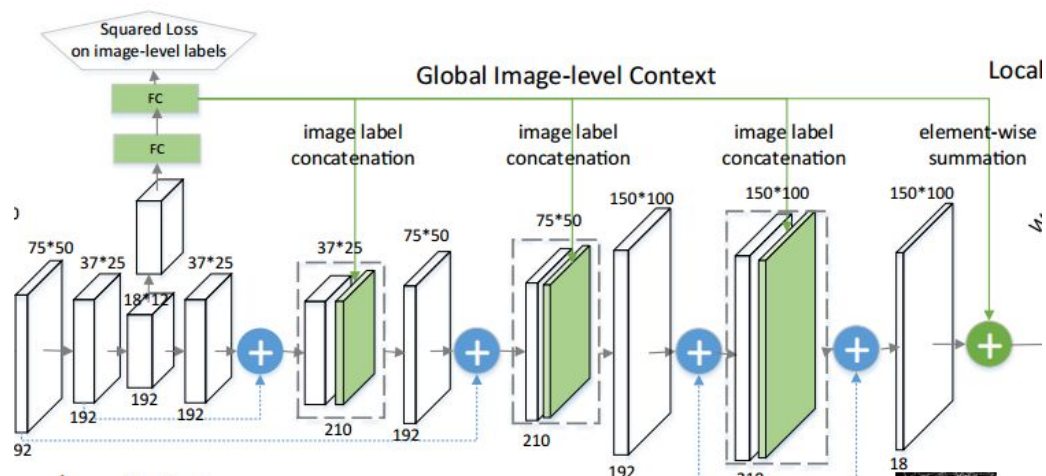
Local-to-global



Global-to-Local

## 2. Global Image-Level Context :

- In this **global image label prediction is done** using fully-connected layer and C-way softmax which produces a probability distribution over the C class labels. In our case C=18.
- We use **sum of squared loss** in this process.
- We append the **output of the FCN to the intermediate convolutional layers** and do **element-wise aggregation**.
- The motive behind this is that if the class c has a low probability of appearing in the image, the corresponding pixel-wise probability will be suppressed.
- This is implemented as shown in the image , highlighted as green components.



Global Image-Level Context ( Green components in the image )

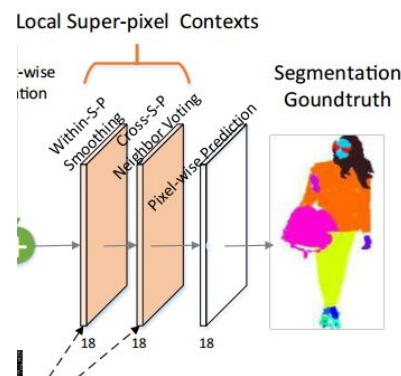
### 3. Local Super Pixel Context :

- We integrate within-super-pixel smoothing and the cross-super-pixel neighborhood voting into the training and testing process to respect the local detailed information
- We first divide the image into super pixels (over-segmentation) using **entropy rate based segmentation** (explained later).
- Then we apply the methods mentioned above :
- **Within Super-Pixel smoothing** : In this we perform **smoothing of the predicted confidence maps** using the super-pixels generated.
- We consider each super-pixel and average the corresponding predicted values in the 18 maps generated.

$$\tilde{x}_{i,j,c} = \frac{1}{||s_{ij}||} \sum_{(i',j') \in s_{ij}} x_{i',j',c},$$

- Here  $||s_{ij}||$  is the number of pixels inside the super pixel , the summation is over all the predicted confidences. This is for the cth confidence map.
- **Cross-super-pixel Neighborhood Voting** : After smoothing confidences within each super-pixel, we can **take the neighboring larger regions into account for better inference**, and exploit more statistical structures and correlations between different super-pixels.
- This is done using the cross neighbourhood voting which is taking the weightage of neighbouring super pixels.
- We first compute a concatenation of bag-of-words from RGB, Lab and HOG descriptor for each super-pixel, and the **feature of each super-pixel can be denoted as  $b_s$** .
- Let  **$D_s$  be the neighbourhood** around super-pixel  $s$  ,  **$s'$  represent the neighbours** , then we do the **voting as follows** where  **$x_s$  has the predicted confidences of super pixel  $s$  and  $x_{s'}$  has those of  $s'$** .

$$\bar{x}_s = (1 - \alpha)\tilde{x}_s + \alpha \sum_{s' \in D_s} \frac{\exp(-||b_s - b_{s'}||^2)}{\sum_{\hat{s} \in D_s} \exp(-||b_s - b_{\hat{s}}||^2)} \tilde{x}_{s'}.$$



## Super Pixel Segmentation :

To perform this task we have considered the following algorithms :

### Entropy rate based segmentation :

- In this method the segmentation is formulated as a graph partitioning problem.
- Each pixel is associated with a vertex in the graph and the edges in the graph show the relationship between adjacent pixels using pairwise potentials.
- Now we formulate a energy function as follows :

$$\text{Min. } H(A) + B(A)$$

Here  $H(A)$  is used to encourage compact and homogeneous clusters ,  $B(A)$  is used to encourage clusters with similar sizes.

- To model  $H(A)$  and  $B(A)$  the concept of entropy is used. Entropy measures randomness in the system hence minimizing it means that we are encouraging less randomness in the system.
- $H(A)$  is the entropy over the set of colors are used in the superpixel, hence minimizing it means that colors used are similar.
- $B(A)$  is the entropy over the set of superpixel sizes that are used.Hence minimizing it implies that the superpixels will have similar sizes.



- g) Therefore superpixel segmentation is nothing but energy minimization for the above graph.

Results :

input image.



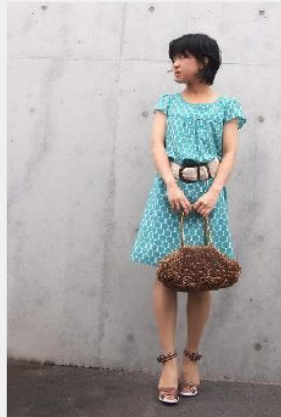
superpixel boundary map



randomly-colored superpixels



input image.



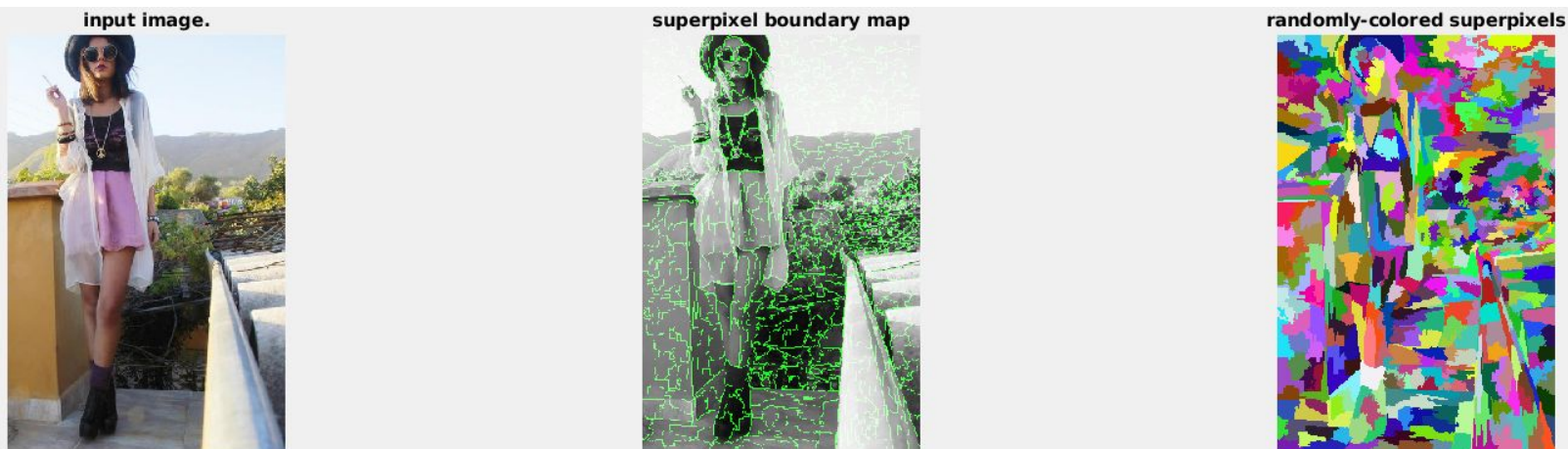
superpixel boundary map



randomly-colored superpixels





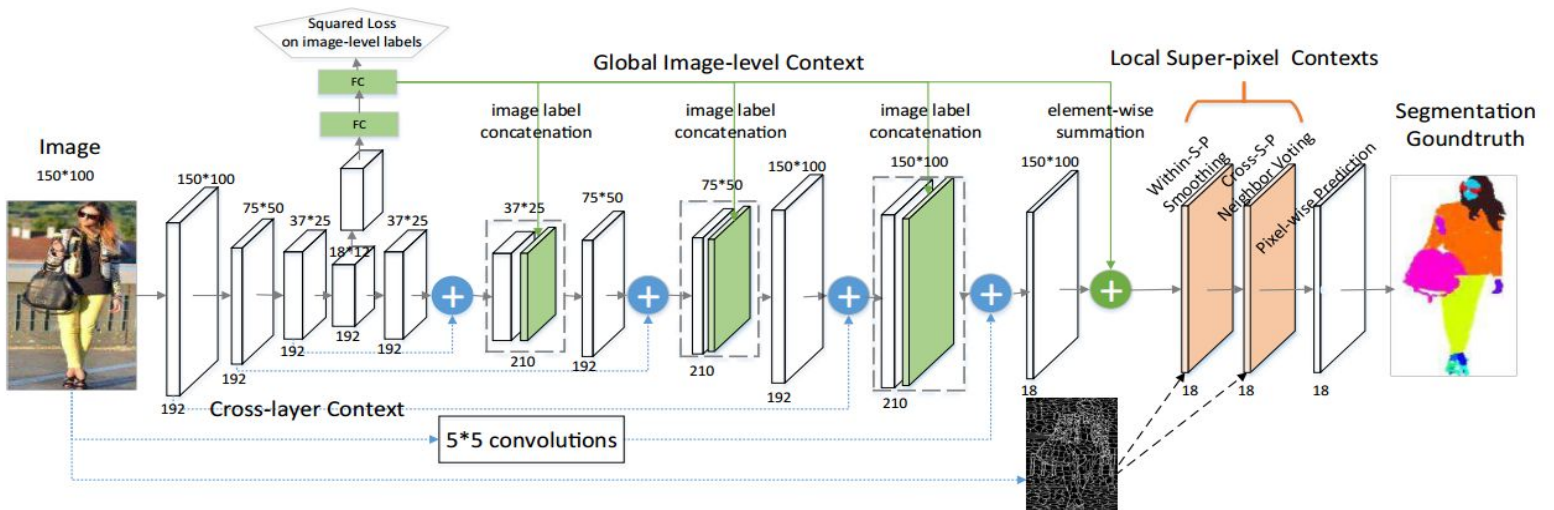


## Simple Linear Iterative Clustering (SLIC) :

1. In this method first a feature space is built in which points are plotted such that each point corresponds to a pixel.
2. The feature of each point is  $(l, a, b, x, y)$  that is the LAB colors and the coordinates of each pixel.
3. Then k-means clustering is performed in this space. The k clusters obtained are nothing but the required super-pixels.

After using the network using both the methods we found out that Entropy rate based super pixel segmentation is giving better results , hence we selected it.

## Overall Architecture of Our Co-CNN :



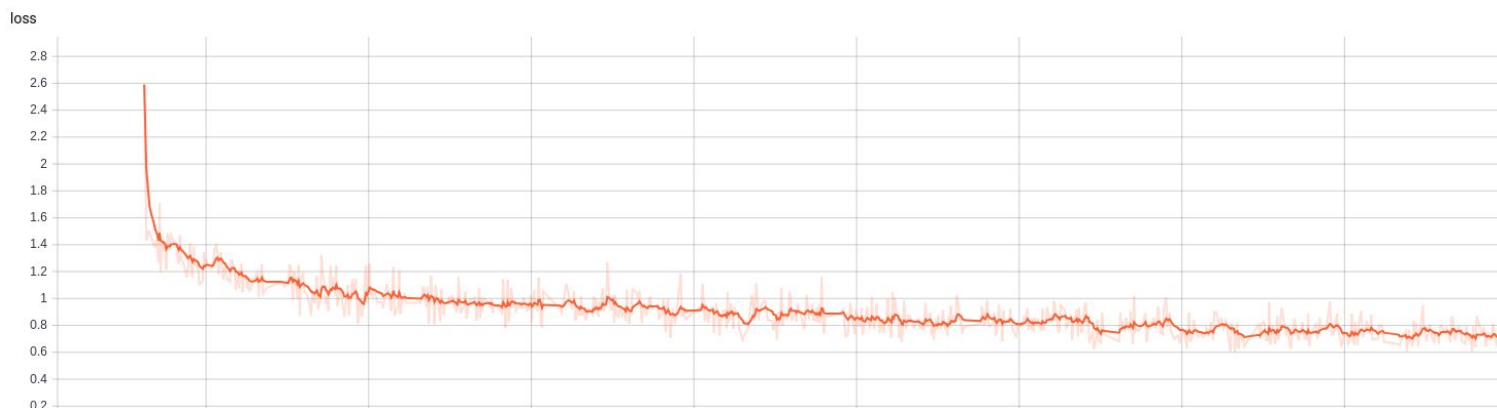
## Configuration of ourCNN

component	type	kernel size/stride	output size
local-to-global	convolution	$5 \times 5/1$	$150 \times 100 \times 128$
	convolution	$5 \times 5/1$	$150 \times 100 \times 192$
	max pool	$3 \times 3/2$	$75 \times 50 \times 192$
	convolution	$5 \times 5/1$	$75 \times 50 \times 192$
	convolution	$5 \times 5/1$	$75 \times 50 \times 192$
	max pool	$3 \times 3/2$	$37 \times 25 \times 192$
	convolution	$5 \times 5/1$	$37 \times 25 \times 192$
	convolution	$5 \times 5/1$	$37 \times 25 \times 192$
	max pool	$3 \times 3/2$	$18 \times 12 \times 192$
	convolution	$5 \times 5/1$	$18 \times 12 \times 192$
	convolution	$5 \times 5/1$	$18 \times 12 \times 192$
	convolution	$5 \times 5/1$	$18 \times 12 \times 192$
image-level label prediction	convolution	$1 \times 1/1$	$18 \times 12 \times 96$
	FC (dropout 30%)		$1 \times 1 \times 1024$
	FC		$1 \times 1 \times 18$
	Squared Loss		$1 \times 1 \times 18$
global-to-local	upsampling	$2 \times 2/2$	$37 \times 25 \times 192$
	convolution	$5 \times 5/1$	$37 \times 25 \times 192$
	element sum		$37 \times 25 \times 192$
	concat		$37 \times 25 \times 210$
	convolution	$5 \times 5/1$	$37 \times 25 \times 192$
	upsampling	$2 \times 2/2$	$75 \times 50 \times 192$
	convolution	$3 \times 3/1$	$75 \times 50 \times 192$
	element sum		$75 \times 50 \times 192$
	concat		$75 \times 50 \times 210$
	convolution	$5 \times 5/1$	$75 \times 50 \times 192$
	upsampling	$2 \times 2/2$	$150 \times 100 \times 192$
	convolution	$5 \times 5/1$	$150 \times 100 \times 192$
	element sum		$150 \times 100 \times 192$
	concat		$150 \times 100 \times 210$
	convolution	$5 \times 5/1$	$150 \times 100 \times 192$
	convolution (image)	$5 \times 5/1$	$150 \times 100 \times 192$
	element sum		$150 \times 100 \times 192$
	convolution	$3 \times 3/1$	$150 \times 100 \times 256$
prediction	convolution	$1 \times 1/1$	$150 \times 100 \times 18$
	element sum		$150 \times 100 \times 18$
	convolution	$1 \times 1/1$	$150 \times 100 \times 18$
super-pixel	within-S-P smoothing		$150 \times 100 \times 18$
	cross-S-P voting		$150 \times 100 \times 18$
	Softmax Loss		$150 \times 100 \times 18$

## Model Summary

```
CoCNN(
  (conv1): Conv2d(3, 128, kernel_size=(5, 5), stride=(1, 1), padding=(2, 2))
  (conv2): Conv2d(128, 192, kernel_size=(5, 5), stride=(1, 1), padding=(2, 2))
  (conv3): Conv2d(192, 192, kernel_size=(5, 5), stride=(1, 1), padding=(2, 2))
  (conv4): Conv2d(192, 192, kernel_size=(5, 5), stride=(1, 1), padding=(2, 2))
  (conv5): Conv2d(192, 192, kernel_size=(5, 5), stride=(1, 1), padding=(2, 2))
  (conv6): Conv2d(192, 192, kernel_size=(5, 5), stride=(1, 1), padding=(2, 2))
  (conv7): Conv2d(192, 192, kernel_size=(5, 5), stride=(1, 1), padding=(2, 2))
  (conv8): Conv2d(192, 192, kernel_size=(5, 5), stride=(1, 1), padding=(2, 2))
  (pool): MaxPool2d(kernel_size=3, stride=2, padding=0, dilation=1, ceil_mode=True)
  (conv9): Conv2d(192, 96, kernel_size=(1, 1), stride=(1, 1), padding=(2, 2))
  (fc1): Linear(in_features=20736, out_features=1024, bias=True)
  (dropout): Dropout(p=0.3)
  (fc2): Linear(in_features=1024, out_features=18, bias=True)
  (upsample1): Upsample(size=(37, 25), mode=nearest)
  (conv10): Conv2d(192, 192, kernel_size=(5, 5), stride=(1, 1), padding=(2, 2))
  (conv11): Conv2d(210, 192, kernel_size=(5, 5), stride=(1, 1), padding=(2, 2))
  (upsample2): Upsample(size=(75, 50), mode=nearest)
  (conv12): Conv2d(192, 192, kernel_size=(3, 3), stride=(1, 1), padding=(1, 1))
  (conv13): Conv2d(210, 192, kernel_size=(5, 5), stride=(1, 1), padding=(2, 2))
  (upsample3): Upsample(size=(150, 100), mode=nearest)
  (conv14): Conv2d(192, 192, kernel_size=(5, 5), stride=(1, 1), padding=(2, 2))
  (conv15): Conv2d(210, 192, kernel_size=(5, 5), stride=(1, 1), padding=(2, 2))
  (conv16): Conv2d(3, 192, kernel_size=(3, 3), stride=(1, 1), padding=(1, 1))
  (conv17): Conv2d(192, 256, kernel_size=(3, 3), stride=(1, 1), padding=(1, 1))
  (conv18): Conv2d(256, 18, kernel_size=(1, 1), stride=(1, 1), padding=(2, 2))
  (conv19): Conv2d(18, 18, kernel_size=(1, 1), stride=(1, 1), padding=(2, 2))
)
```

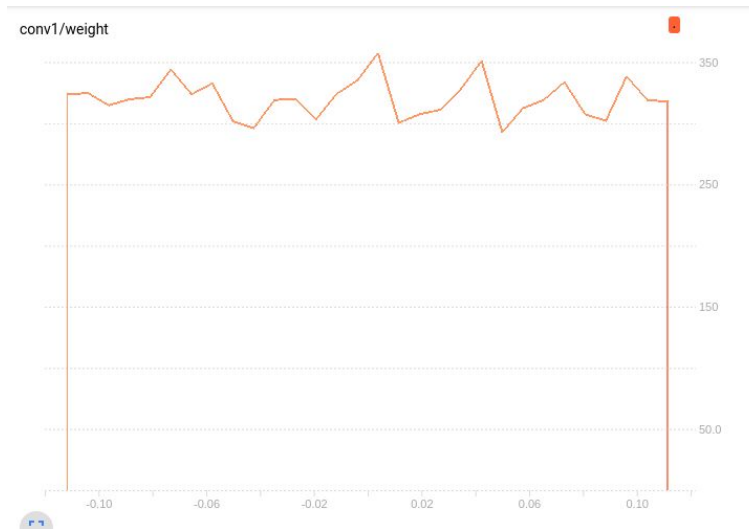
## Visualization of Model Parameters:



Plot Showing the change in loss as the model is training



Variation of Bias of Conv1



Variation of weights of Conv1





Variations of Weights of Fully Connected Layer 1



Variations of Bias of Fully Connected Layer 1



## Code Snippet:

We have implemented the Network using PyTorch.

```
class CoCNN(nn.Module):
    def __init__(self):
        super(CoCNN, self).__init__()
        ''' Local-to-Global'''
        self.conv1 = nn.Conv2d(3, 128, 5, 1, 2)
        self.conv2 = nn.Conv2d(128, 192, 5, 1, 2)

        self.conv3 = nn.Conv2d(192, 192, 5, 1, 2)
        self.conv4 = nn.Conv2d(192, 192, 5, 1, 2)

        self.conv5 = nn.Conv2d(192, 192, 5, 1, 2)
        self.conv6 = nn.Conv2d(192, 192, 5, 1, 2)

        self.conv7 = nn.Conv2d(192, 192, 5, 1, 2)
        self.conv8 = nn.Conv2d(192, 192, 5, 1, 2)

        self.pool = nn.MaxPool2d(3, 2, ceil_mode=True)

        ''' Image-Level label prediction'''
        self.conv9 = nn.Conv2d(192, 96, 1, 1, 0)
        self.fc1 = nn.Linear(18 * 12 * 96, 1024)
        self.dropout = nn.Dropout(0.3)
        self.fc2 = nn.Linear(1024, 18)

        ''' Global-to-Local '''
        self.upsample1 = nn.Upsample(size=(37, 25))
        self.conv10 = nn.Conv2d(192, 192, 5, 1, 2)

        self.conv11 = nn.Conv2d(210, 192, 5, 1, 2)
        self.upsample2 = nn.Upsample(size=(75, 50))
        self.conv12 = nn.Conv2d(192, 192, 3, 1, 1)

        self.conv13 = nn.Conv2d(210, 192, 5, 1, 2)
        self.upsample3 = nn.Upsample(size=(150, 100))
        self.conv14 = nn.Conv2d(192, 192, 5, 1, 2)

        self.conv15 = nn.Conv2d(210, 192, 5, 1, 2)
        self.conv16 = nn.Conv2d(3, 192, 3, 1, 1)

        self.conv17 = nn.Conv2d(192, 256, 3, 1, 1)
        self.conv18 = nn.Conv2d(256, 18, 1, 1, 0)
        self.conv19 = nn.Conv2d(18, 18, 1, 1, 0)
```

```

def forward(self, x):

    '''Local-to-Global'''
    x1 = F.relu(self.conv1(x))
    x1 = F.relu(self.conv2(x1))

    x2 = self.pool(x1)
    x2 = F.relu(self.conv3(x2))
    x2 = F.relu(self.conv4(x2))

    x3 = self.pool(x2)
    x3 = F.relu(self.conv5(x3))
    x3 = F.relu(self.conv6(x3))

    x4 = self.pool(x3)
    x4 = F.relu(self.conv7(x4))
    x4 = F.relu(self.conv8(x4))

    ''' Image-Level label prediction'''
    y = F.relu(self.conv9(x4))
    y = y.view(-1, 18 * 12 * 96)
    y = F.relu(self.fc1(y))
    y = self.dropout(y)
    y = F.relu(self.fc2(y))

    y0 = y.unsqueeze(2)
    y0 = y0.unsqueeze(3)
    y1 = y0.repeat(1, 1, 37, 25)
    y2 = y0.repeat(1, 1, 75, 50)
    y3 = y0.repeat(1, 1, 150, 100)

    ''' Global-to-Local '''
    x5 = self.upsample1(x4)
    x5 = F.relu(self.conv10(x5))
    x5 = x5.add(x3)
    x5 = torch.cat((x5,y1),1)

    x6 = F.relu(self.conv11(x5))
    x6 = self.upsample2(x6)
    x6 = F.relu(self.conv12(x6))
    x6 = x6.add(x2)
    x6 = torch.cat((x6,y2),1)

    x7 = F.relu(self.conv13(x6))
    x7 = self.upsample3(x7)
    x7 = F.relu(self.conv14(x7))
    x7 = x7.add(x1)
    x7 = torch.cat((x7,y3),1)

    x8 = F.relu(self.conv15(x7))
    x9 = F.relu(self.conv16(x))
    x8 = x8.add(x9)
    x8 = F.relu(self.conv17(x8))

    x9 = self.conv18(x8)
    x9 = x9.add(y3)
    x9 = self.conv19(x9)

    return (x9, y)

```

## Super-Pixel Segmentation

```
files = dir('../JPEGImages');
in_dir = '../JPEGImages/';
out_dir = '../SP_Data/';

ans = 0;
for file = files'
    if (file.name ~= ".") && (file.name ~= "..")
        ans = ans + 1
        if ans==3
            break
        end
        out_name = split(file.name, '.');
        out_name = out_name(1);
        out_name = out_name{1};
        name = strcat(in_dir, file.name);
        img = imread(name);

        no_segments = 500;
        [labels] = uint16(mex_ers(double(img),no_segments));
        imwrite(labels, strcat(out_dir, strcat(out_name, '.png')));
        disp(strcat(out_dir, strcat(out_name, '.png')));
    end
end
```

To get visualization of the super-pixels :

```
img = imread('./JPEGImages/2500_100.jpg');
grey_img = double(rgb2gray(img));

nC = 500;

lambda_prime = 0.5; sigma = 5.0;
conn8 = 1; % flag for using 8 connected grid graph (default setting).
[labels] = mex_ers(double(img),nC);

[height width] = size(grey_img);

[bmap] = seg2bmap(labels,width,height);
bmapOnImg = img;
idx = find(bmap>0);
timg = grey_img;
timg(idx) = 255;
bmapOnImg(:,:,2) = timg;
bmapOnImg(:,:,1) = grey_img;
bmapOnImg(:,:,3) = grey_img;

%// Randomly color the superpixels
[out] = random_color( double(img) ,labels,nC);

gcf = figure(1);
subplot(1,3,1);
imshow(img,[]);
title('input image. ');
subplot(1,3,2);
imshow(bmapOnImg,[]);
title('superpixel boundary map');
subplot(1,3,3);
imshow(out,[]);
title('randomly-colored superpixels');
```



## Code for computing Bag of Words and Cross-Neighbourhood Voting

```
import matplotlib.image as mpimg
from skimage.feature import hog
import cv2
import numpy as np
import matplotlib.pyplot as plt

def get_HOG(img):
    fd, hog_image = hog(img, orientations=9, pixels_per_cell=(16, 16),
                        cells_per_block=(1, 1), visualize=True, multichannel=True)
    return hog_image

def get_LAB(img):
    LAB_image = cv2.cvtColor(img, cv2.COLOR_BGR2LAB)
    return LAB_image

def cross_neighbourhood_voting(BoW, x):
    for i in superpixels:
        sum = 0
        for k in neighbourhood(i):
            sum += exp(-(BoW[i] - BoW[k])^2)
        neigh = 0
        for k in neighbourhood(i):
            neigh += (exp(-(BoW[i] - BoW[k])^2)/sum)
        x[superpixel(i)] = (1-alpha) * x + alpha * (neigh)

img = mpimg.imread('../JPEGImages/2500_1.jpg')
img = cv2.resize(img, (100, 150))

LAB_img = get_LAB(img)
hog_img = get_HOG(img)

sp_indices = np.array(mpimg.imread('2500_1.png'))

# To rescale the input between values 0 to 499
sp_indices = np.ceil((sp_indices/np.max(sp_indices)) * 499)
sp_indices = cv2.resize(sp_indices, (150, 100))

BoW = np.array([None] * 500)
```

```
for i in range(500):
    req_indices = (np.isin(sp_indices,[i])).ravel()

    RGB_fts = np.array([])
    LAB_fts = np.array([])

    for j in range(3):
        curr = img[:, :, j].ravel()
        RGB_fts = np.append(RGB_fts, curr[req_indices])
    for j in range(3):
        curr = LAB_img[:, :, j].ravel()
        LAB_fts = np.append(LAB_fts, curr[req_indices])

    hog_img = hog_img.ravel()
    HOG_fts = hog_img[req_indices]

    BoW[i] = np.append(np.append(RGB_fts, LAB_fts), HOG_fts)

    f = open((str(i) + ".txt"), "w")

    for j in range(BoW[i].shape[0]):
        print(BoW[i].shape)
        f.write(str(BoW[i][j]) + " ")
    f.close()

x = cross_neighbourhood_voting(x)
```

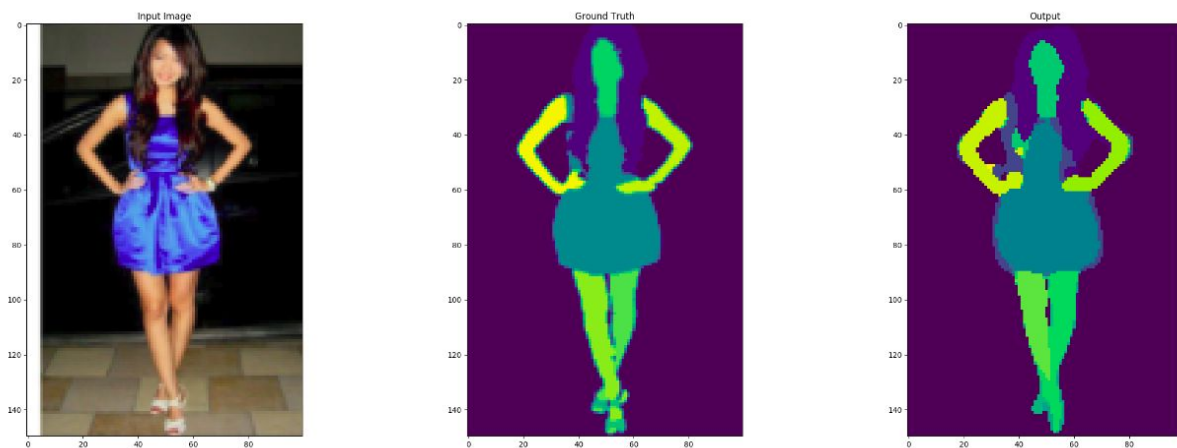


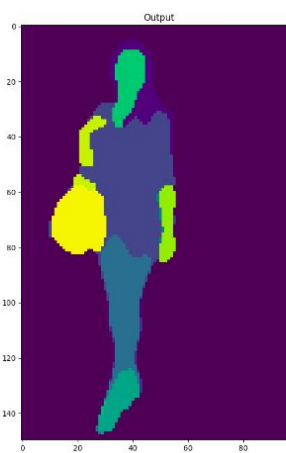
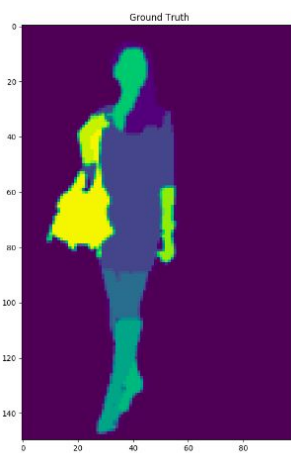
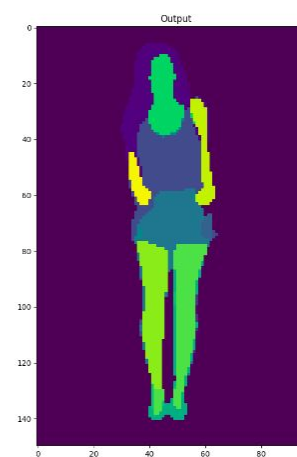
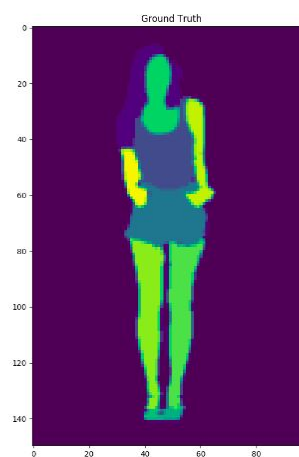
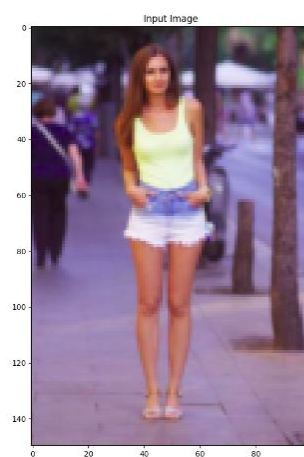
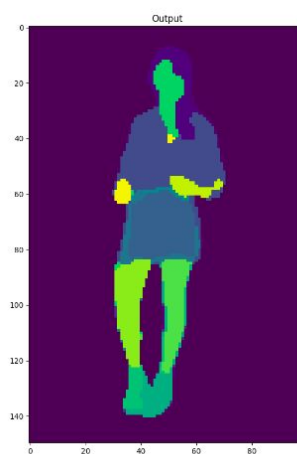
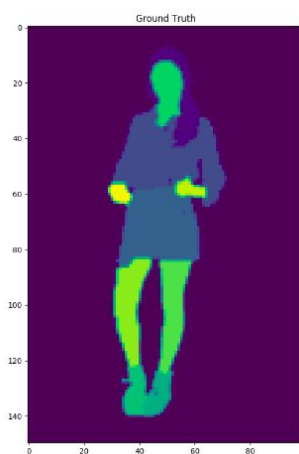
## Experiments performed :

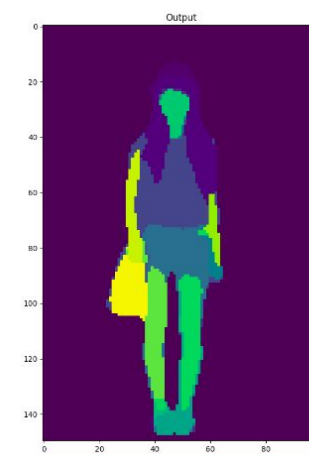
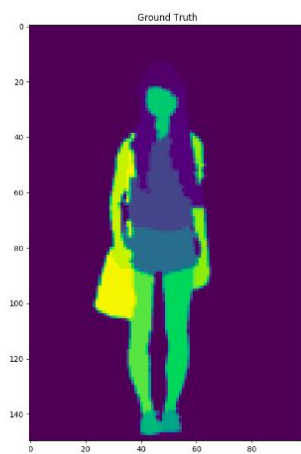
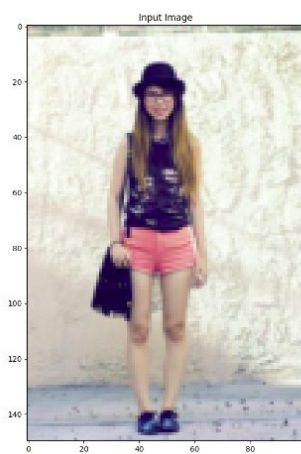
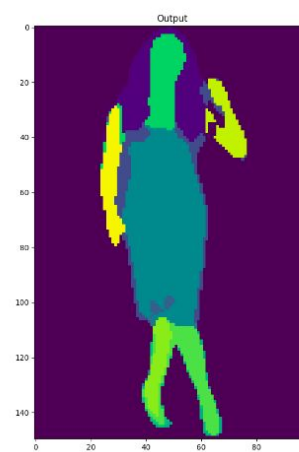
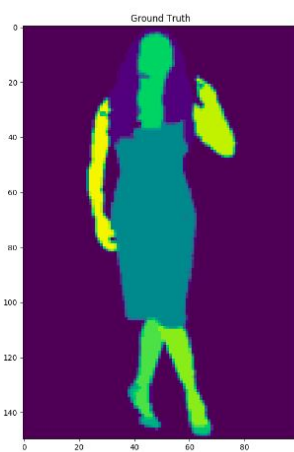
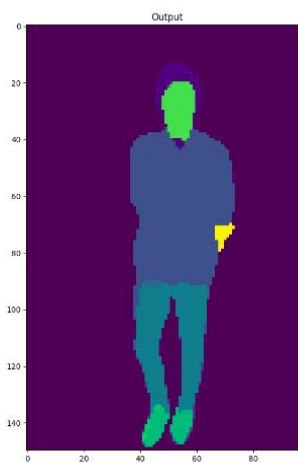
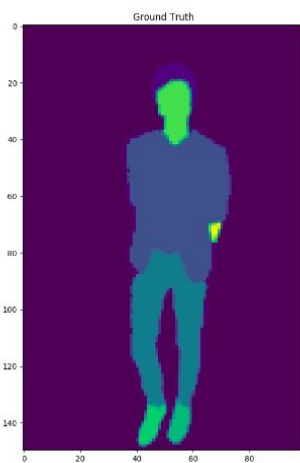
1. We have tried two types of super-pixel segmentation algorithms as explained above , Entropy rate based segmentation ( ERS ) and SLIC.
2. We also tried to improve the CNN's accuracy by changing the global context as follows: Instead of using the concept that a label is present (or) not we try to see how many pixels of each label are present in the image. This will significantly improve accuracy in the case of occlusions.

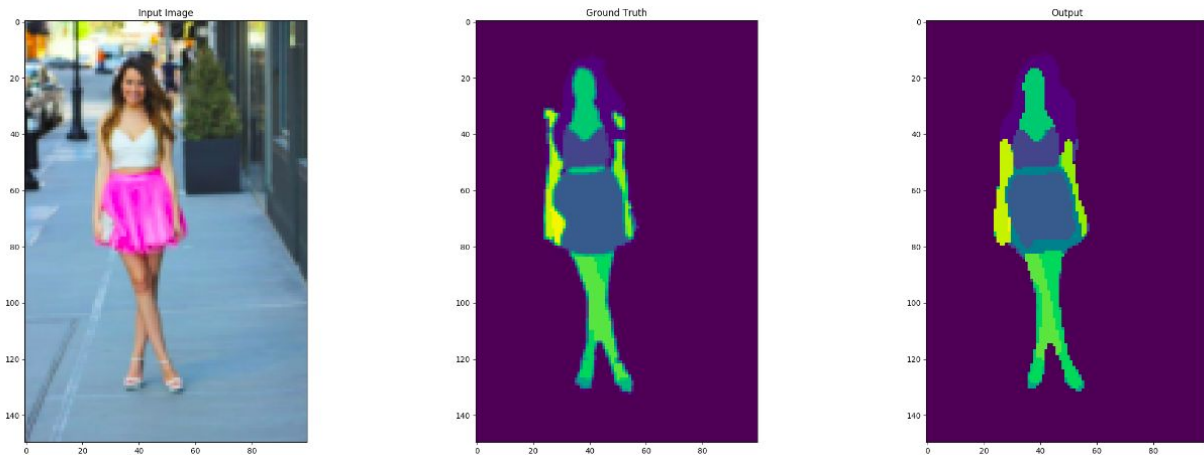
## Results :

### ATR Dataset : (Full network)

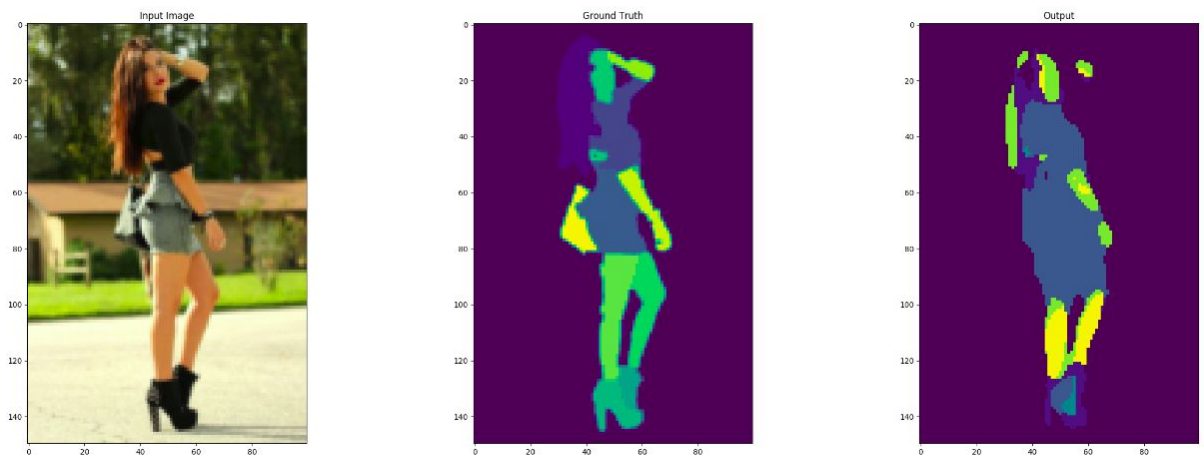






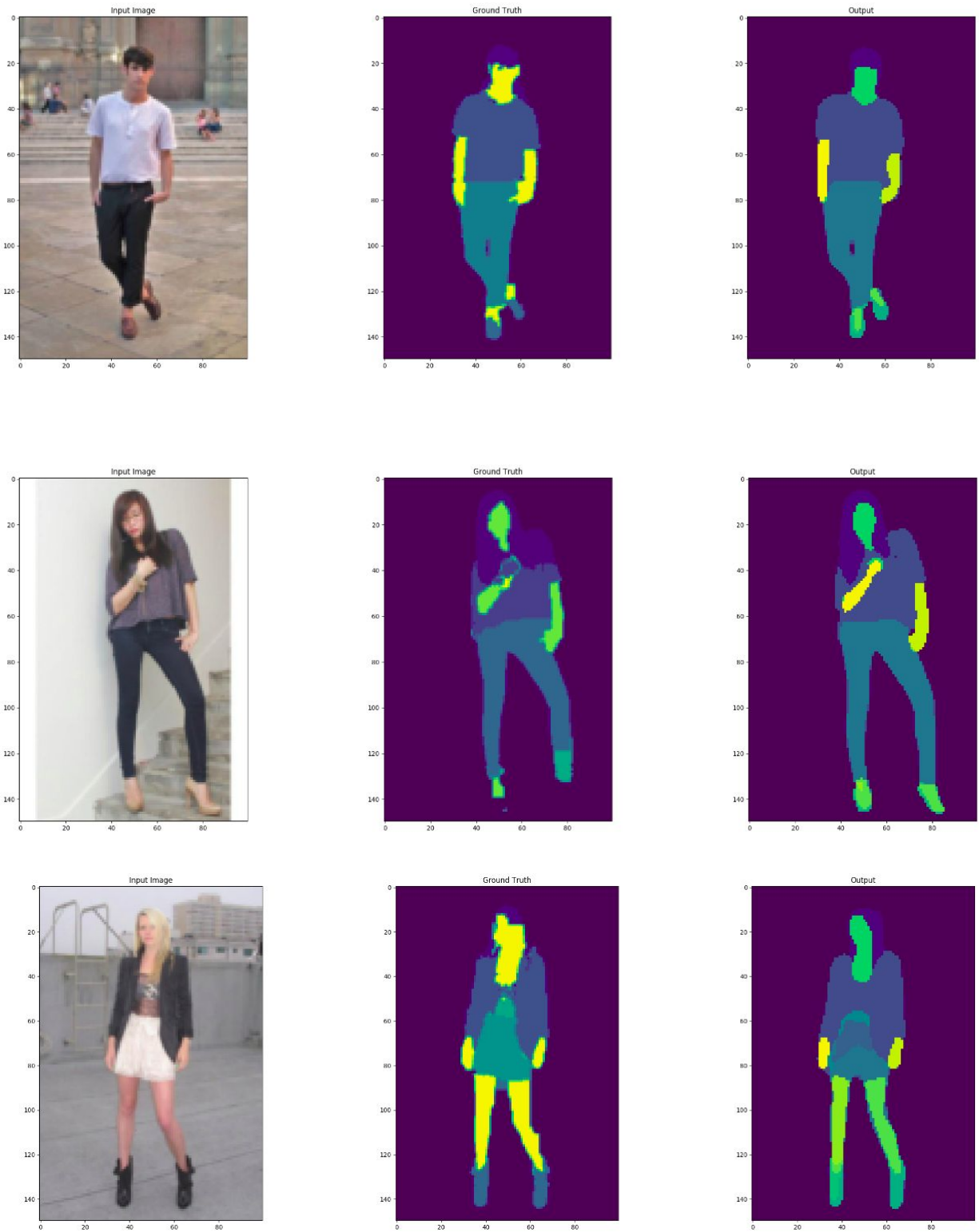


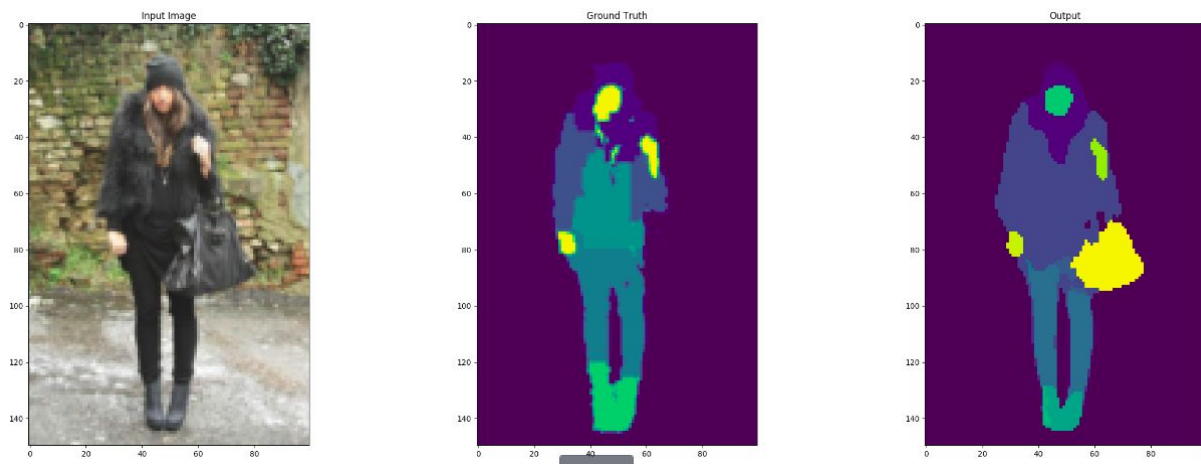
Without global layer context :



- This is the classic example of why we need to use global level context . The network has considered the **bag to be a part of the jeans due to color similarity**.
- When global context is used this confusion will be removed as the network knows that there is bag in the image and can label the pixels correctly.

## Results on Fashionista Dataset :



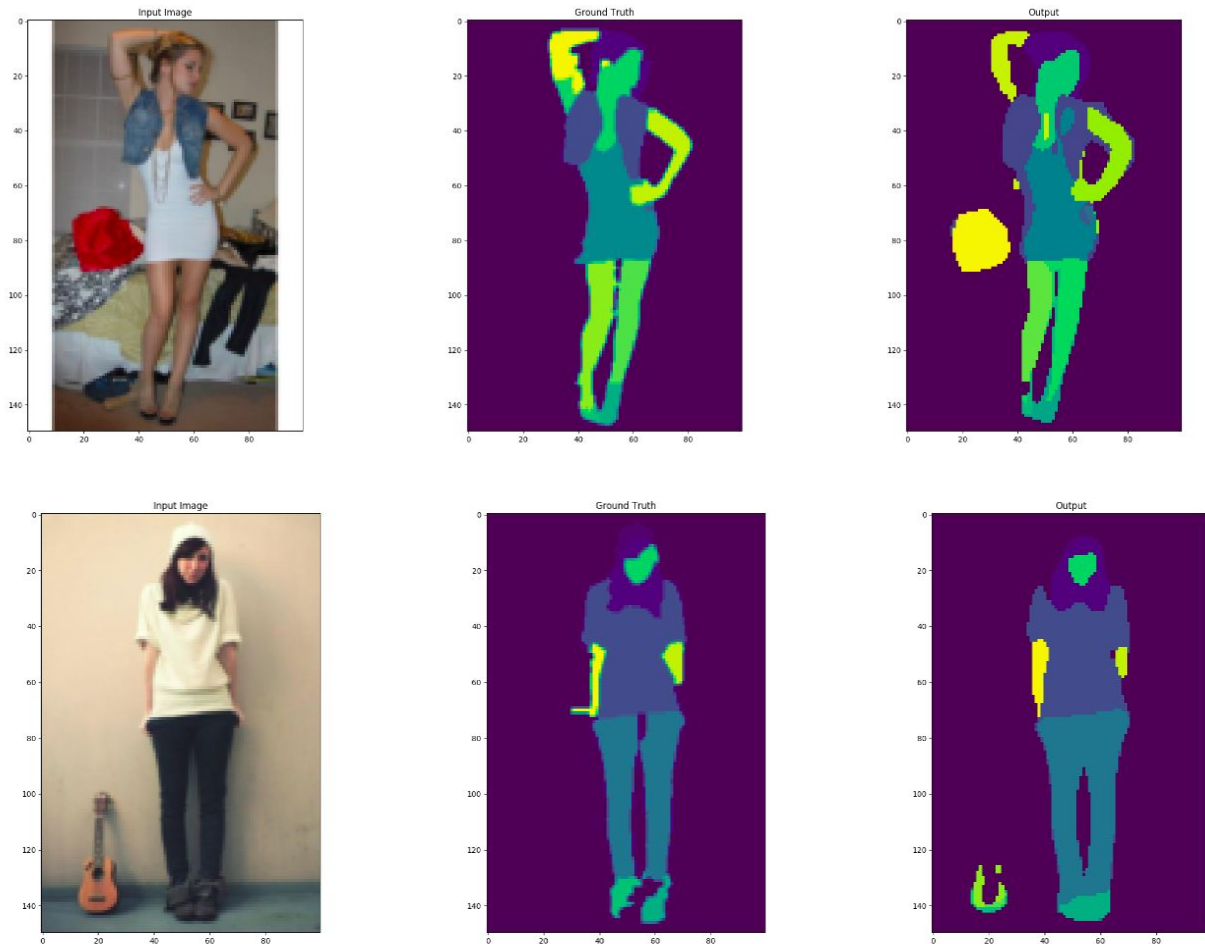


NOTE :

- The fashionista dataset has 56 labels in total , but our network predicts 18 defined labels
- Hence we have mapped each of the 56 labels to one of the 18 defined labels , for example all the upper body garments like shirt , coat , T-shirt are mapped to shirt itself.
- Hence we are getting a bit different prediction with respect to ground truth.

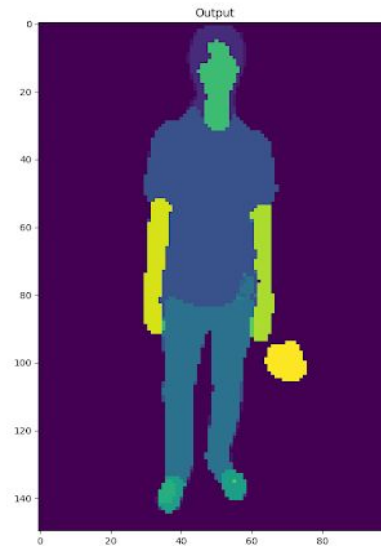


## Failure Cases :



- In this example , our network got confused with the pillow being a bag. Rest of the prediction is fine .
- This is because the fully connected layer predicted that a bag is present in the image.
- We can improve on this by using an approach similar to GrabCut that is the user can give certain inputs in such a way that ambiguities are resolved.
- In this case the information that pillow is background can be given.


### Custom Images Outputs:



### Per Part Metrics: Atr Dataset:

In this part we have analyzed the network's accuracy in predicting each of the 18 labels.

Label	Accuracy Score
Hair	70.65
Sun glass	10.34
Upper Cloth	71.42
Skirt	70.35
Pants	72.69



Dress	75.29
Belt	15.43
Shoe	45.34
Face	64.81
Leg	60.45
Left arm	48.37
Right arm	50.26
Bag	40.39