Detecting Human Body Parts and Building Skeleton Models using Deep Convolutional Neural Networks

Iftimie Florentin Alexandru

2017

Contents

1 Introduction

Recent advances in real-time object detection with FasterRCNN [16] enabled researchers to add new features to the model and make it flexible to new tasks. One task that was explored by researchers at Facebook was to add a segmentation branch that generated the contour of the detected object, creating a new model called FastMaskRCNN [12]. The goal of this project is to replicate the above results and add new tasks to the model by adding body parts segmentation and key point regression. The first task that was researched during this project was the key point regression. Key point regression means that the human body has multiple joints and we need to find the x,y coordinates of those points. The best model that was found was OpenPose [9, 17, 20] that was running at 2.3 fps on a Nvidia 960M GPU.

Later in this project, the body parts segmentation task was researched. Body parts segmentation consists in detecting every person in the image and for each person find the contour of each body part. We found that although there are papers [11, 19, 21, 14, 13] published about this task, there was no implementation available on the Internet.

Such tasks are important for multiple applications ranging from the automotive industry, medical industry, fashion applications and robotic rescuing operations. For this project the task that was mostly researched was the body parts segmentation task and the model that was selected for modification and training was MaskRCNN.

This work is organized in the following way. Chapter 2 contains state-of-the-art public models evaluations and public datasets description. Chapter 3 contains my work on this problem and the steps that I took in order to solve a part from it. Chapter 4 shows some outputs of the neural network. Chapter 5 shows my conclusions and some notes for improvement. Appendixes shows how to download the repository, configure the project in order to start running and see the results and how the project is structured.

2 Related Work

In this section is discussed the current state of the art models for key point regression, as there are no publicly available models for body parts segmentation.

There are two approaches to analyzing the human body for these kind of tasks. The first approach is the top-down architecture that consists in detecting all persons in an image and for each person detect their key points or segment their body parts. This approach is based on the Faster-RCNN model that includes additional task-specific branches.

The second approach consists in detecting all body parts in an image and create graphs with their connections or propose some heuristics for reconstructing the human pose. Most of these models are based on the Fully Convolutional Neural Network [15] approach and for each body part there is generated a heatmap that detects the positions of a certain part in the image.

2.1 Public models performances

The below table shows the performance of the publicly available implementation of different models. The below results are tested using a laptop with Nvidia 960M GPU and i7 processor. As it can be observed there is no model on body parts segmentation.

Table 1: Performance of publicly available models

Model	GPU	CPU	Results
OpenPose 6 stages	2.3 fps	Does not run on CPU	Very accurate keypoints
OpenPose 4 stages	3.1 fps	Does not run on CPU	Less acurate keypoints
DeepCut	Out of memory	200 s	Very accurate keypoints
Human part discovery	10 s	Not tested	Bad segmentation
Pose Singleperson	0.53 s	Not tested	Very accurate keypoints
Pose Multiperson	1.7 s	Not tested	Very accurate keypoints
DeepCut-cnn	Out of memory	2.4 s	Very accurate keypoints
Convolutional pose machines	Not available	10 minutes	Very acurate keypoints

2.2 Datasets

Over the Internet there are a lot of public datasets that have key point annotations, body parts segmentation or both of them, for images with single instances or multiple instances. The following table contains the list of the datasets and their specifications. Some of them are not yet public and their specifications remained unfilled.

Table 2: Databases

Name	Kp	Seg body	Seg parts	Size	Multiperson	Singleperson	Parts/Keypoints
COCO	Yes	Yes	No	45K	Yes	Yes	17 Kp
FLIC	Yes	No	No	25K	Yes	Yes	11 UpperBody Kp
Freiburg	No	No	Yes	215	No	Yes	14 P
LSP	Yes	No	No	10K	No	Yes	14 Kp
MPII	Yes	No	No	25K	Yes	Yes	14 Kp
Parse	Yes	No	No	305	No	Yes	14 Kp
Poseevaluator	Yes	No	No	6K	Yes	Yes	6 UpperBody Kp
Pose in the wild dataset	Yes	No	No	800	Yes	Yes	8 Kp
PASCAL VOC	No	Yes	Yes	20K	Yes	Yes	22 P
Chalearn	No	No	Yes	8K	Yes	Yes	14 P
MIT SceneParsing	No	Yes	No	22K	Yes	Yes	14 P
ADE20K	No	Yes	Yes	1.1K	Yes	Yes	¿6 P
PoseTrack	Yes	?	?	20K	Yes	Yes	?
J-HMDB	Yes	Yes	Yes	13K	No	Yes	15Kp/14P
Penn-Action	Yes	No	No	¿10K	No	Yes	13 Kp
HumanParsing	No	Yes	Yes	10k	No	Yes	18 P
Fashionista	Yes	Yes	Yes	685	No	Yes	14K 14 P
VideoPose	Yes	No	No	1K	No	Yes	6 Kp (upper and lower arms)
VGG (Youtube, BBC)	Yes	No	No	23K	No	Yes	7 P
FYDPP/UYDP	Yes	No	No	?	No	Yes	14 K

For the JHMDB database it was utilized a very convenient tool for annotating the images as depicted in Figure 1. Such a tool makes the annotation a lot faster while maintaining the quality. In the figure below the user is able to drag am modify naturally the mask of the body parts and of the key points.

The selected datasets for training the model of this project remained Pascal VOC, ADE20K, Chalearn and JHMDB. Pascal VOC contains 1K images with persons in it with very well annotated body parts. ADE20K contains 1K images and is poorly annotated and with a lot of mistakes. Chalearn contains 6K images with decent annotation, but not very precise. Also it does not have a lot of variation in the dataset as the samples are drawn from a few videos. JHMDB is very well annotated with 30K images but its downside is that the images contains a single instance of a person.

The links for these datasets are found in the references [2, 1, 6, 8, 7]. In order to download the Chalearn dataset, the user must register and then download the data [3].

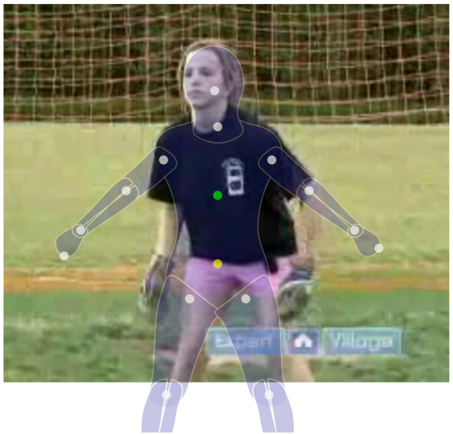
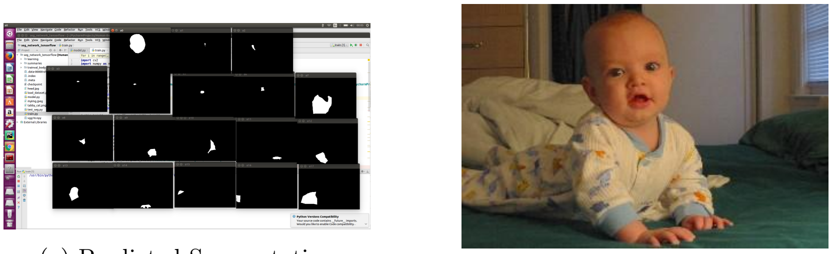


Figure 1: Tool for body parts annotation



(a) Predicted Segmentation

(b) Input Image

Figure 2: Fully Convolutional Neural Network Excercise

3 The steps and description of this solution

3.1 Fisrt attempts

Before I went to modify MaskRCNN my first approaches to this problem were in modular steps such as understanding how Fully Convolutional Neural Networks works and understand what the transpose convolution does and how to improve the segmentation precision. For this step I implemented my own Fully Convolutional Neural Network using VGG [18] as the backbone and overfitted on a single image as in figure 2.

The next thing relevant to my problem was to adapt MaskRCNN to detect all body parts in an image as in Figure 3. The problem with this approach is that the model does not know that those body parts belong to that person. If more persons are in the image, the problem becomes much bigger. I quickly discarded this approach as the problem of assembling each part to each person was a task too difficult to resolve it in useful time. Thus I needed my model to be able to detect all persons in the image and detect their body parts from a

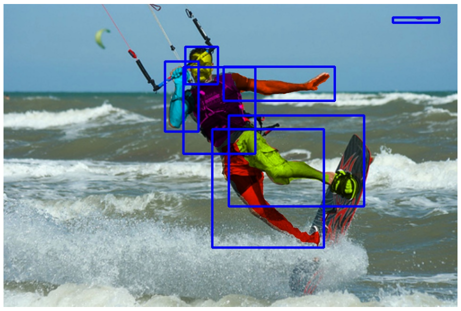


Figure 3: MaskRCNN applied on independent body parts detection and segmentation

single shot, without any other heuristics on top of the model.

3.2 Final approach

MaskRCNN is a model of a neural network that is able to do object detection, classification and segmentation. The ideea is to detect all persons in an image and segment their body parts. This is a real-time model running at 200 fps on a Titan X GPU.

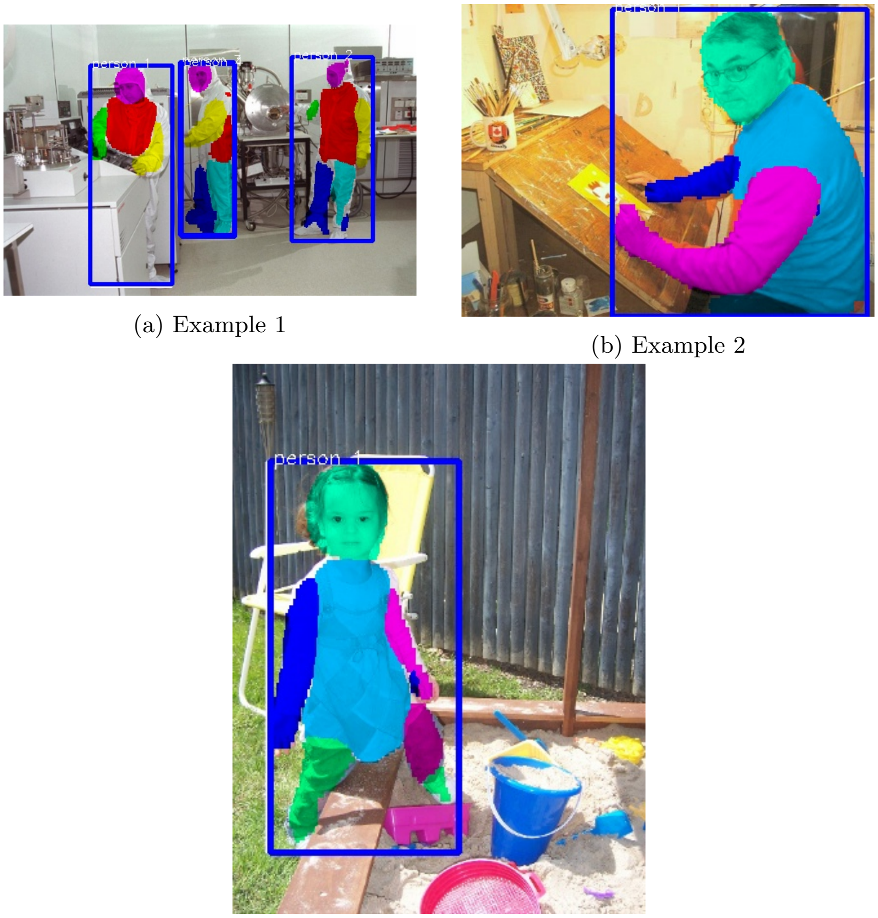
MaskRCNN is an extension of FasterCNN. The latter had 2 branches. One for region proposal and one for the classification. This extension has 3 branches. One is for for object detection another is for classification and bounding box regression and another one is for segmentation. The initial model was generating a mask for the entire object and my task during this project was to modify the network to detect only persons and for each person segment their body parts. Another task that is to be researched is the key point regression task. The following images in Figure 4 depict the results on the training set.

3.3 Personal Modifications

This section shows how I modified the original code from the GitHub repository by Charles Shang [4].

Starting from the root directory in the file libs/nets /pyramid_network.py line 287 and 349 I modified the pooled_height and pooled_width to 56 by 56 pixels.

At line 360 in the same file, the mask is upsampled by using transposed convolution [10, 5] to 112 by 112 pixels. The mask at that point has the shape [N,112,112,256] and a [1,1] convolution kernel is applied to the Tensor in order to get [N,112,112,7] where 7 is the number of parts in the image (6 body parts and a mask that contains the segmentation for the entire body). I also included the entire segmentation of the body because it is easier to segment and better annotated and the network is usually better at learning the segmentation of the whole body. When drawing each body parts is clipped using the mask of the



(a) Example 1

(b) Example 2

(c) Example 3

Figure 4: MaskRCNN on person detection and body parts segmentation

body in order to fit better inside the boundaries. The output of the mask is passed into a sigmoid layer that will help transform the values into probabilities.

Also, in libs/nets/pyramid_network.py line 566 the function mask_encoder also must be modified in order to generate the proper mask targets given the proposals. The mask targets are also reshaped to 112x112 pixels. The actual targets are generated inside libs/layers/mask.py at line 75.

Training the neural network was done in different ways. First I trained with the Pascal VOC dataset that contained about 1000 images but the model over fitted the dataset immediately. I also played around with the output of the mask in order to increase the precision of the segmentation. A mask of 112x112 pixels was better than a mask of 56x56 pixels, but one problem that appeared when the network was training was that at a certain point, the network was proposing 120 regions for a single image and that caused an Out-of-memory exception because for a single convolutional layer the tensor shape was [120,112,112,256] that occupied 1.4 Gb of GPU memory. My first attempt to overcome this problem was a multistage training approach. The idea was to train the network for person detection only then include the segmentation branch. I thought that training only for human detection would reduce the number of proposals and it did. One problem that I found here was to restore the parameters of the network as the checkpoint only contained the parameters for the bounding box regression and classification branches, and the restorer threw exceptions if the parameters for the newly included segmentation branch were not found in the checkpoint.

After I managed to restore the parameters and leave the segmentation branch parameters initialized with Gaussian distributions I resumed the training. But the network got itself into the same problem stated earlier. The gradients that contained the loss for the segmentation affected the parameters for the region proposal network and as the training was running, the RPN was proposing more regions in order to stabilize itself. After I found that there was a flag for the maximum number of Rois per image I changed that from 256 to 80 and the training continued without problems.

One thing to be mentioned is that I first started training this model by over fitting on a single image to test if it works. Initially I modified only for human body parts detection but in that form I did not know which part belonged to which person. So in order to get rid of reconstruction algorithms or branch and cut algorithms, I decided to transform the output of the mask branch as inspired after the Fully Convolutional Network where at the last layer for visualizing the results, the argmax for each class.

3.4 Understanding the implementation

The graph of the model generated in Tensorboard was not very clear and it contained a lot of variables that were not necessary to visualize. Thus, in order to study and understand how the model works and is trained I looked over the entire code and generated my own graph. The following parts describe how the model works. This scheme is found in mask_rcnn_final.xml and can be opened and edited in draw.io . Generating this scheme allowed me to better understand the code in order to be able to modify it.

The first part is the ResNet part visualized with the layers at different scales as in Figure 5. It has the role of feature extraction. It takes as input an image with any size and the C2, C3, C4, C5 layers are used for the Feature Pyramid Network construction detailed in the following pages.

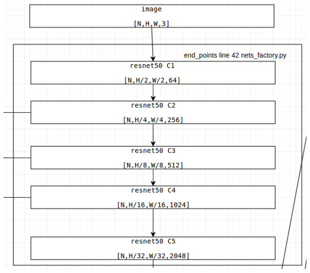


Figure 5: ResNet part

From the above figure, the C5 layer is upsampled step by step and fused with the intermediate layers of the ResNet as depicted in Figure 6.

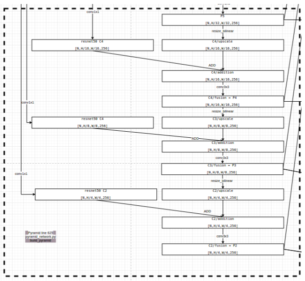


Figure 6: Upsampling part

For each step of the upsampling process such as P5,P4,P3 and P2, a region proposal network is attached in order to generate proposals for different scales as in Figure 7. The region proposal network generates the bounding box, the classification scores and the anchors.

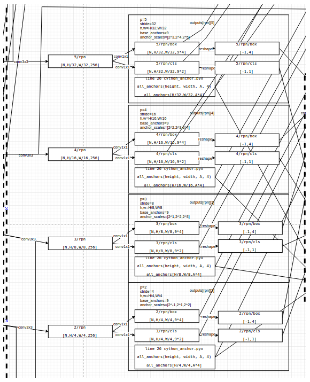


Figure 7: RPN part

For the region proposal part there is attached a loss function that takes the proposed regions and the objectness score of each region and the generated bounding box targets and score targets as in Figure 8.

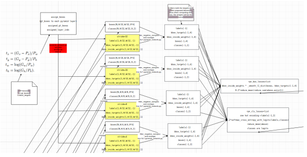


Figure 8: RPN targets part

The region proposals, classifications and anchors from each region proposal network (Figure 9) are all concatenated in their respective tensors as in Figure .

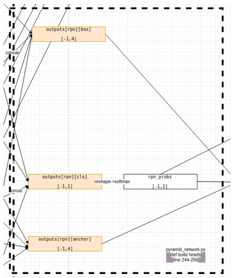


Figure 9: RPN targets part

After the concatenation there are applied the following 3 operations as in Figure 10. The first one is to decode the proposal from the normalized value into real coordinates. The next one is to sample the proposals with respect to the ground truth to reduce the redundant information and to keep the number of proposals per image to a certain number. After the region proposals have been sampled the next operation is to assign each proposal to the layer of the pyramid network that it came from. In the blue box there are some Tensors called mask_rois and mask_scores but there are not further used.

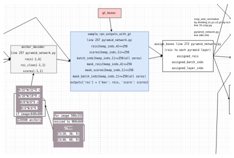

gt_boxes
anchor_decoder
line 257 pyramid_network.py
rois[-1,4]
roi_clses[-1,1]
scores[-1,1]
sample_rpn_outputs_with_gt
line 257 pyramid_network.py
rois[keep_inds,4]<=256
scores[keep_inds,1]<=256
batch_inds[keep_inds,1]<=256(all zeros)
mask_rois[keep_inds,4]<=256
mask_scores[keep_inds,1]<=256
mask_batch_inds[keep_inds,1]<=256(all zeros)
outputs['roi'] = {'box': rois, 'score': scores}
assign_boxes line 272 pyramid_network.py
(rois to each pyramid layer)
assigned_rois
assigned_batch_inds
assigned_layer_inds
crop_and_normalize
by dividing x1,y1,x2,y2 by w,h
line 70 crop.py
pyramid_network.py
line 280-291
H/32*W/32*9 +
H/16*W/16*9 +
H/8*W/8*9 +
H/4*W/4*9
if image=640x480
229500 anchors
for image 500x333
resized to 960x640
27000
C5(20, 30, 9)
C4(40, 60, 9)

Figure 10: Decode, Sample, Assign operations

After the boxes have been assigned to each layer in the pyramid network, the following operation consists in cropping and resizing each image given the proposal. The resize is done to a 14x14 image using bilinear interpolation. Then the regions are concatenated into a single Tensor as in Figure 11. From there

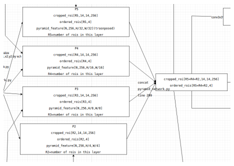

P5
cropped_roi[R5,14,14,256]
ordered_rois[R5,4]
pyramid_feature[N,256,H/32,W/32](trasnposed)
R5=number of rois in this layer
P4
cropped_roi[R4,14,14,256]
ordered_rois[R4,4]
pyramid_feature[N,256,H/16,W/16]
R4=number of rois in this layer
P3
cropped_roi[R3,14,14,256]
ordered_rois[R3,4]
pyramid_feature[N,256,H/8,W/8]
R3=number of rois in this layer
P2
cropped_roi[R2,14,14,256]
ordered_rois[R2,4]
pyramid_feature[N,256,H/4,W/4]
R2=number of rois in this layer
concat
pyramid_network.py
line 299
cropped_roi[R5+R4+R2,14,14,256]
ordered_rois[R5+R4+R2,4]
conv3x3

Figure 11: Cropping into 14x14 regions

3 branches are attached as in Figure 12. The refined bounding box regression head, the classification head and the mask head. And for the bounding box regression head it is also attached a roi decoder head that takes the normalized values and transforms them into real coordinates.

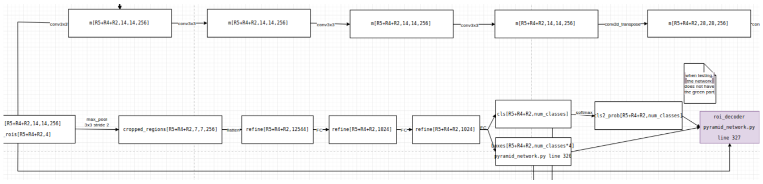


Figure 12: Classification, Refined regression and Segmentation branches

In this Figure 13 is also depicted the refined bounding box loss and the scheme for generating the refined bounding box targets. In the following Figure 14 is depicted the mask loss.

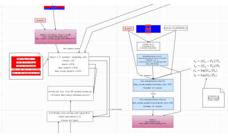


Figure 13: Bounding box loss and bounding box targets

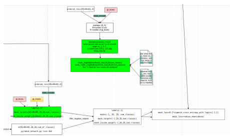
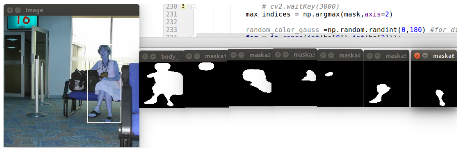


Figure 14: Mask loss

4 Results

The model takes the image into the rgb format. I later found that it has a redundant operation that flips the channels to bgr format. Operation that could be eliminated in order to reduce the computation. The following images represent the output of the neural network for an image taken from the training set. The masks in black and white from Figure 15 are the 112x112x7 mask, each channel being visualized separately.



```
    # cv2.waitKey(3000)
max_indices = np.argmax(mask,axis=2)

random_color_gauss =np.random.randint(0,180) #for di
```

Figure 15: Output masks

Another thing to mention in the results section is that I started writing proprietary LibNNF code for some of the layers that this model was utilizing.

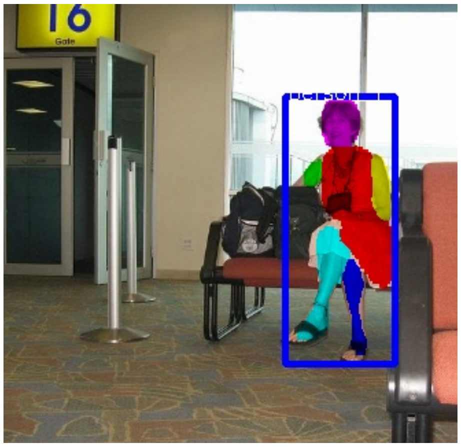


Figure 16: Drawn masks

5 Conclusions

MaskRCNN is a very heavy model of a neural network. Training it requires a lot of trial and error as it is the cutting edge algorithm in object detection and segmentation. It must be mentioned that the average time for training the network and obtaining decent results takes about 3 days. If the dataset is not large enough the network will improve its results for segmentation but will over fit the dataset as the model capacity is very large. The complete set combined with Pascal VOC body parts, ADE20K, Chalearn and JHMDB contains 33K images.

Referring to the datasets, some of them are badly annotated, other are too small or contain single instance images.

Another critical point is the size and the point where to attach the new task-specific branches. If precision is important, a big segmentation mask branch can generate memory exceptions and if keypoint regression is also included it remains a question about which layers contain the necessary information that can generate accurate key points. My approach was to add a keypoint regression branch that took the [N,112,112,256] Tensor from the mask branch.

References

[1] ADE20K Dataset. http://groups.csail.mit.edu/vision/datasets/ADE20K/. [Online; accessed August-2017].

[2] Chalearn Dataset. http://gesture.chalearn.org/mmdata#Track1. [Online; accessed August-2017].

[3] Chalearn Registration. https://competitions.codalab.org/competitions/2231#participate-get-data . [Online; accessed August-2017].

[4] CharleShang MaskRCNN Unoficial implementation. https://github.com/CharlesShang/FastMaskRCNN . [Online; accessed August-2017].

[5] Convolution arithmetic Theano. http://deeplearning.net/software/theano/tutorial/conv_arithmetic.html. [Online; accessed August-2017].

[6] JHMDB Dataset. http://jhmdb.is.tue.mpg.de/challenge/JHMDB/datasets. [Online; accessed August-2017].

[7] Pascal VOC Bodyparts Annotations Dataset. http://www.stat.ucla.edu/ xianjie.chen/pascal_part_dataset/pascal_part.html. [Online; accessed August-2017].

[8] Pascal VOC Dataset. http://host.robots.ox.ac.uk/pascal/VOC/voc2010/. [Online; accessed August-2017].

[9] Zhe Cao, Tomas Simon, Shih-En Wei, and Yaser Sheikh. Realtime multi-person 2d pose estimation using part affinity fields. In *CVPR*, 2017.

[10] Vincent Dumoulin and Francesco Visin. A guide to convolution arithmetic for deep learning, 2016. cite arxiv:1603.07285.

[11] Bharath Hariharan, Pablo Arbelaez, Ross Girshick, and Jitendra Malik. Object instance segmentation and fine-grained localization using hypercolumns. *IEEE Trans. Pattern Anal. Mach. Intell.*, 39(4):627–639, April 2017.

[12] Kaiming He, Georgia Gkioxari, Piotr Dollár, and Ross B. Girshick. Mask R-CNN. *CoRR*, abs/1703.06870, 2017.

[13] Xiaodan Liang, Si Liu, Xiaohui Shen, Jianchao Yang, Luoqi Liu, Jian Dong, Liang Lin, and Shuicheng Yan. Deep human parsing with active template regression. *CoRR*, abs/1503.02391, 2015.

[14] Xiaodan Liang, Chunyan Xu, Xiaohui Shen, Jianchao Yang, Jinhui Tang, Liang Lin, and Shuicheng Yan. Human parsing with contextualized convolutional neural network. *IEEE Trans. Pattern Anal. Mach. Intell.*, 39(1):115–127, January 2017.

[15] Jonathan Long, Evan Shelhamer, and Trevor Darrell. Fully convolutional networks for semantic segmentation. *CoRR*, abs/1411.4038, 2014.

[16] Shaoqing Ren, Kaiming He, Ross Girshick, and Jian Sun. Faster R-CNN: Towards real-time object detection with region proposal networks. In *Advances in Neural Information Processing Systems (NIPS)*, 2015.

[17] Tomas Simon, Hanbyul Joo, Iain Matthews, and Yaser Sheikh. Hand keypoint detection in single images using multiview bootstrapping. In *CVPR*, 2017.

[18] Karen Simonyan and Andrew Zisserman. Very deep convolutional networks for large-scale image recognition. *CoRR*, abs/1409.1556, 2014.

[19] Peng Wang, Xiaohui Shen, Zhe L. Lin, Scott Cohen, Brian L. Price, and Alan L. Yuille. Joint object and part segmentation using deep learned potentials. *CoRR*, abs/1505.00276, 2015.

[20] Shih-En Wei, Varun Ramakrishna, Takeo Kanade, and Yaser Sheikh. Convolutional pose machines. In *CVPR*, 2016.

[21] F. Xia, P. Wang, X. Chen, and A. Yuille. Joint Multi-Person Pose Estimation and Semantic Part Segmentation. *ArXiv e-prints*, August 2017.

A Installation and running

```
git clone https://github.com/Iftimie/MaskRCNN_body.git
cd MaskRCNN_body
mkdir data/coco
mkdir data/coco/pretrained_models
mkdir data/coco/records
mkdir output
mkdir output/mask_rcnn
wget http://download.tensorflow.org/models/resnet_v1_50_2016_08_28.tar.
    gz
tar -xvzf resnet_v1_50_2016_08_28.tar.gz
mv resnet_v1_50.ckpt data/pretrained_models/
cd libs
make
cd ..
cd data/coco/records
wget https://www.dropbox.com/s/tdcf1sslvoe0k4z/out_ade20k.tfrecord
wget https://www.dropbox.com/s/bce3n5ohk8m8c1s/out_chalearn.tfrecord
wget https://www.dropbox.com/s/upp8k6z0mn76sfe/out_JHMDB.tfrecord
wget https://www.dropbox.com/s/8q047rbfw1q19c5/out.tfrecord
cd ../../..
open libs/configs/config_v1.py and modify the absolute path of the
    following lines:
line 11,15,49
make sure NOT to add / at the end of each path
CUDA_VISIBLE_DEVICES=0 python train/train.py
#CUDA_VISIBLE_DEVICES=0 python train/train.py --draw #will draw images
    in output/

cd convert_data && bash download_and_convert_data.sh
```

B Project structure description

```
convert_data directory contains the scripts where the datasets are being
    converted into tfrecords.
        – ChalearnLAPEvaluation.py and ChalearnLAPSample.py is the API
            for Chalearn dataset
        – convert_ADE20k_human_body_parts.py is self explanatory. it
            contains comments inside for what it does
        – convert_CHALEARN_human_body_parts.py same here
        – convert_VOC_human_body_parts.py  same here
        – convert_jhmdb.py contains the keypoint annotations also
        – download_and_convert_data.sh will run the above scripts. but is
            VERY SLOW, so you might want to run separately each command
        – human_body_parts.m loads X.jpg, Y.png, Y.txt from ade20K and
            outputs X.img, Y.mat . then  convert_ADE20k\_human_body\
            _parts.py will load the X.img, Y.mat and output .tfrecord
        – read_my_data_keypoints.py is used for visualizing jhmdb images
            with keypoints
        – visualize_records_human_body_parts.py is used to visualize the
            rest of databased without keypoints

data directory contains the tfrecords and resnet pretrained weights

draw directory is used for offline drawing (initially it was faster to
    save the outputs of the network, download them from the server and
    run the drawing locally using draw/draw.py)
        – draw/data is where I downloaded the outputs of the network in
            .npy files
        – draw/metric.py is where I wrote the metrics. it is commented
        – draw/utils.py contains the drawing functions

libs directory contain the implementation of MaskRCNN
most important files here are
        –config/config_v1.py. This file contains some useful flags
        –nets/pyramid_network.py. This file contains the implementation of
             the network

media directory contains some result on the training set

output/mask_rcnn
    directory contains the tensorboard log files and the checkpoints

train/train.py is the starting point for training.

unit_tests is not important

crontab.sh contains the scripts that check if the training process is
    running or not. if it is not running then it will start training.
    this is necessary because the training throws some exceptions after
     some big number of iterations in order for this crontab to work,
    the user must enter the following commands:
        crontab -e
        */5 ****full/path/to/crontab.sh
        service cron reload

mask_rcnn_final.xml can be opened in draw.io in order to visualize the
    model
```

C Repository branches

```
While testing multiple experiments, I created more branches: CPU,
    keypoints, master, noshuffle, test
entering:
git branch #will list the branches
or by entering
git checkout [branch name]

-CPU contains the branch for running this model on CPU. It is VERY SLOW
     (about 60 per image)
        -this branch contains the model and the .so files compiled for
            running on CPU. nothing else special
-keypoints contain the keypoint regression and it is trained only on
    jhmdb
        -in libs/nets/pyramid_network.py from line 360 to 370 is the
            keypoints branch
        -in libs/nets/pyramid_network.py linr 607-608 is defined the loss
            for keypoints branch
        -gt_keypoints are allready normalized
-noshuffle contain some scripts when I was testing the metric because
    normally the tfrecords are shuffled by the thread coordinator.
-test contain some client-server application scripts for testing on other
     images. The ideea is to have a webcam locally, take pictures
    locally and send them for processing on servers with GPU, then
    return back the drawn picture
        - in order to test the model in real time have one clone of the
            repo on server and another locally
        - enter on server: CUDA_VISIBLE_DEVICES=0 pyhton
            train/test.py #this will start the server and it will await
            for incomming images
        - enter locally: python train/client.py # this will start the
            client, will take a picture, send it to server, receive the
             processed image, and show it on screen. Don't forget to
            modify the IP at line 32
```