Pose Estimation using Stacked Hourglass Networks

Nishant Bhansali
IIT Roorkee
9558428844 || nishantbhansali80@gmail.com

Pose Estimation is a very important task in modern day computer vision as it can further be used to improve our understanding of humans in images and videos.It can be used in the sports industry,or as a military application,to identify threats etc.

An important part of pose estimation is keypoint estimation that is retrieving the coordinates of various joints such as elbows,knees,wrists and shoulders. We use 14 key points to estimate the pose of a subject.The best model created till now is Cascade Feature Aggregation which has an accuracy of 93.9% (according to the metric PCK which is explained later).I have studied and implemented the research paper Stacked Hourglass Network For Human Pose Estimation which has an accuracy 90.9%.

Why Stacked Hourglass Networks?

If we think about how a human brain recognises or 'learns' about pose estimation ,I would say that we need to look at the image on two scales.I would first zoom in to the picture to precisely locate the position of joints on the human body.Secondly,I would look at the bigger picture ,to understand the precise relation between the joints I have located.This is exactly what a Stacked Hourglass Network does.It captures and consolidates information about the image across all scales of the image.A persons orientation,arrangement of limbs,relationship between adjascent joints are amongst many cues that are best recognised at different scales. Multiple Stacked Hourglass modules provide repeated bottom-up(high resolution to low res.) top down(low resolution to high res.) inferences.Subsequent hourglasses have the opportunity to evaluate higher order spatial relationship

Model Summary

1) Aim of this model is to obtain precise pixel location of 14 key points on the subject.For this we focus on a single person from an RGB image .If an image has multiple persons ,the one in the centre is annotated as our subject(by centering along the x-axis).

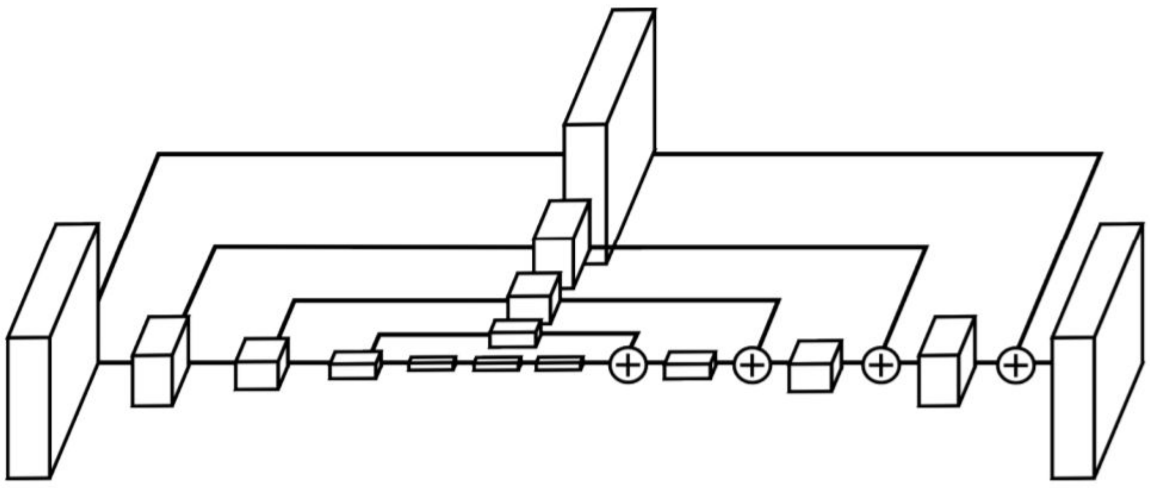
2)FLIC and MPII datasets are used.In the FLIC dataset we have a total of 5003 images out of 3987 are used for training and 1016 are used for testing.The images are cropped around a target person and used resized to 256 x 256.For data augmentation rotation (+,-,30 degrees) and scaling (0.75-1.25) were used.200,000 iterations were needed to fully train the model.

3)Optimization was done using RMSprop,Learning rate uses as specified in the model is 2.5 x 10^-4 but in we use L.R of 10^-3,with a batch size of 16
.
4) MSE Loss was used as explained in the model architecture section to generate a heatmap,where each pixel simply represents the probability of a key-point to be present on that pixel.

5) An 8 Hourglass model is capable of giving an accuracy of 0.901 while a 2 hourglass network can give an accuracy of 0.885

6)PCK : percentage of correct keypoints : percentage of detections that fall within the normalised distance from the ground truth.For FLIC ,it is normalised by torso size and for MP2,by the size of the head.

Model Architecture



1) This is a single hourglass module.Multiple modules like this one one are stacked end to end to perform repeated bottom up-top down inference.What it basically does is pools down to very low resolution and then upsamples and combines features across multiple resolutions.These modules have a symmetric distribution ,i.e. Bottom up-top down have similar capacity.Each box in this diagram corresponds to a single residual module,which is explained later

2)We do not use unpooling or deconv layers ,instead simple nearest neighbour upsampling and skip connections are used for top-down processing.

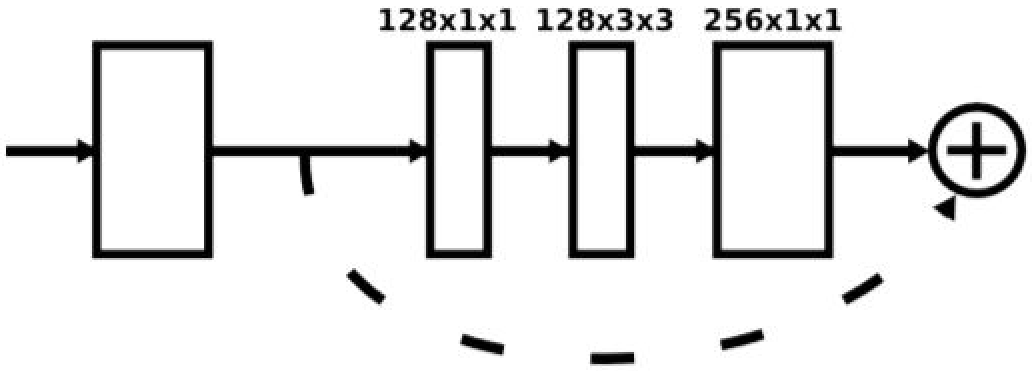
3)This is a single pipeline structure with skip layers to preserve information at each resolution [smallest res. = 4x4].Convolutional and max-pooling layers are used to process features down to a very low resolution. At each Maxpooling step the network branches off and applies more convolutiona at the original pre-pooled resolution .After reaching the lowest resolution we begin top down sequence of upsampling and combining of features across scales.To bring together information across two adjacent resolutions, we follow do nearest neighbor upsampling of the lower resolution followed by an elementwise addition of the two sets of features. do nearest neighbor upsampling of the lower resolution followed by an elementwise addition of the two sets of features.

4)To bring together information across two adjascent resolutions, we use nearest neighbour upsampling which is low resolution + element wise addition of the two sets of features.

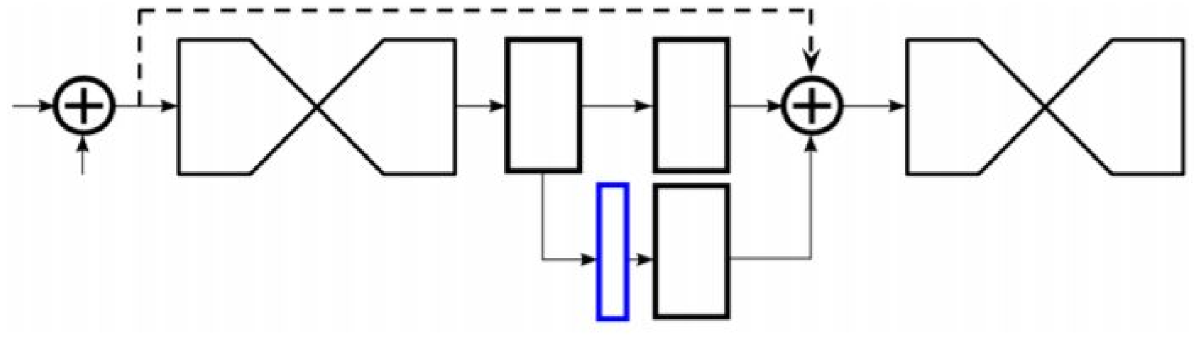
5)After reaching the output of one hourglass ,1x1 convolution is applied to produce final network prediction ,which is heatmap as discussed previously.

6) The network splits in between to generate a heatmap on which loss can be applied.This is intermediate supervision.Then a 1x1 convolution is used to 'remap' this heatmap to match the number of channels of the intermediate features .These are added together along with features from the preceding hourglass .

7) This is a residual module that we have used throughout the network.



8)The network splits and produces a set of heatmaps (outlined in blue) where a loss can be applied. A 1x1 convolution remaps the heatmaps to match the number of channels of the intermediate features. These are added together along with the features from the preceding hourglass.



Integration of our Model with Android

Firstly,it is important to note that it is not possible to train the model on android device,therfore we have to use our pre-trained model and integrate it with our android app.Various attempts to achieve this have already been done ,for example PoseNet with TensorFlow Lite .But do not have full control over application as no images are stored as a server is not provided with TF Lite.

If we need to host servers,I think that the best option would be to with the help of AWS(Amazon Web services).It is also compatible with various deeplearning frameworks,and we should be able to train bigger and better models as well.To develop cloud powered mobile apps,we can use Amplify framework,which also has allows integrating machine learning into the application through Amplify Predictions

Code analysis

I have analysed the code published by princeton university which uses PyTorch framework
https://github.com/princeton-vl/pytorch_stacked_hourglass

My code analysis for model architecture and training can be found here
https://github.com/nixx14/Pose-Estimation-with-Stacked-Hourglass