MIDDLE EAST TECHNICAL UNIVERSITY
DEPARTMENT OF COMPUTER ENGINEERING

SUMMER PRACTICE REPORT

STUDENT NAME: Deniz Rasim Uluğ

ORGANIZATION NAME: Metu ImageLab

ADDRESS: Middle East Technical University, Computer Engineering Department

START DATE:

END DATE:

TOTAL WORKING DAYS:

STUDENT'S SIGNATURE	ORGANIZATION APPROVAL

Contents

1 INTRODUCTION

I have done my summer practice at Metu ImageLab research laboratory in Ankara. Metu ImageLab is an offical laboratory of Computer Engineering department, which does academic research on Computer Vision. There, I worked with two graduate students of Prof. Emre Akbaş on the topic human keypoint estimation.

The topic's main problem is that given a picture with any number of humans in it, to detect each person's pose. By pose we mean the geometric structure of their limbs. This problem of detecting pose is reducable to the problem of detecting the joints of the human, which in computer vision jargon called "keypoints".

My internship handlers, the graduate students, Muhammed Kocabas and Salih Karagöz, had recently published a new neural network model to do this task. The model, as any other, would produce a set of predictions for the coordinates of the joints, per human per picture.

My job then, in this internship, was to build another model named "refiner" which given a set of predictions of a human in a picture, would "correct" the given predictions and output a better set of predictions.

Because of my already existing relations with the ImageLab, I did not have any specific start and end date. My inter-ship started at the start of summer until the end of it, with some natural absences. Yet I made my intern-ship well beyond 30 day mark. On top of that as of writing of this report, i have taken a second project again in the ImageLab. The second project is again on the topic of human keypoint estimation, and I will briefly go over it in Section 2.2.

2 INFORMATION ABOUT PROJECT

2.1 Refiner

As was explained, the job of the "Refiner model" was to take and improve the accuracy of a single human's keypoint prediction. The main was to use this simple and small model to improve the accuracy of any other keypoint detection model.

We used the famous COCO dataset[6] to train and test our model. In COCO dataset, each human has 17 keypoints. So at its most basic from, our model would take as input 17 coordinates, or 34 numbers, and output again 34 numbers, 17 coordinates.

The phases listed below were in reality interleaved throughout the research.

2.1.1 Analysis Of The Problem

The goal was simple, given a set of predictions (of coordinates of joints of a single person), output better predictions. The work was to determine the format of the input (pre-processing), the format of the output (post-processing), possibly additional information the we could feed to network along input, and finally, only after all these are determined, model and hyper-parameter search.

As a code base, we used the github repo pytorch-pose[2] The repo had a nice structure and cleanly written code to easily do experiments, save and monitor the results.

We used pytorch [?] library, and the deperment's servers and GPUs for training. As mentioned we used COCO dataset's ground truths to train out network, and also the COCO predictions of an example model (we used hourglass model's inferences for that)[3].

The initial model we choose was a very simply multi layer perceptron (MLP), with a couple of hidden layers and 34 output neurons. We did all our experiments with this "baseline" model, since the model is very simple, and we wanted to see (1) can the model learn (albeit how much) and (2) which decisions improve the accuracy (albeit up to what).

2.1.2 Baseline Model

Before doing any fancy staff, we wanted to get results using a baseline model and without using any trickery.

We had 22246 training and 2958 testing data from COCO(again, each "data" is 17 coordinates). We normalized our data to mean 0 and std 1, per joint and per axis (so image a mean array of shape $(17, 2)$). So we normalized our to between 1 and -1.

We utilized a "curriculum learning" style idea to train out network.

We first trained our network 200 epoch. First 20 epoch we gave as an input all coordinates, and expected the same coordinates. So first 20 epochs the model worked as an identity function. For the next 20 epoch, we randomly deleted a joint and set its coordinates to $(0, 0)$(we call this masking). And again expected the full coordinates. At each 20 epoch, we increased the masked join count by 1.

Naturally, each time masked joint count is increased, the accuracy fell, and subsequently increased until the next increment to masked joint count. After the process, the model had the accuracy rates at Table 1. The curriculum learning aspect is actually we incrementing the mask count over time.

So far so good. Then we repeated the process, but instead at each mask count we run the full 200 epoch. The results of this "milking every drop of accuracy" is given in Table 1.

I also tried the same strategy with a different model of the type "variational encoder". It was a simple encoder, again with only MLP structure. Since our main motivation was that the model would "memorize possible and general poses", an auto-encoder by definition would have done this job better. The experiments with auto-encoder gave much worse accuracy, around %20.

2.1.3 Testing On Real Predictions

Next, we wanted to test how well would our model perform on not ground truths of COCO, but on a real model's predictions. For this, we used the predictions of hourglass model, processed by the mentioned github repo[2] .

Without any processing, our model reduced the accuracy of the predictions from 89.1 to 54.0. Yet this was not really surprising, since we neither fed any image features to our model, nor trained our model to actually "refine" the predictions. By our training, we actually asked our model to "guess the missing predictions", by inspecting the non-missing coordinates.

To account for this, we again fed our model with hourglass predictions, but now we "masked" "bad predictions". To determine which predictions was bad, we first used ground truth data of COCO. If the distance of a prediction, normalized by person's scale, was greater then a certain threshold; we masked it. Then we walked trough the same process, but this time instead of using ground truth data, we used hourglass's own confidence scores. As common with many pose-estimation model, hourglass too outputs along coordinate predictions, a confidence score for each of it's prediction. Again, we masked the predictions whose confidence score is below a certain threshold. Table 2 shows the result of these experiments and of the ones in this section.

Following this, we noticed that while refiner was in-fact good at making predictions for missing joints, he was not good at "refining" the non-masked joints. Because of our methodology, non-masked joints are assumed to be good predictions by them-selves, and the refiner spew out slightly worse predictions for these non-masked ones. For this reason, we changed our post-processing to combine the non-masked predictions or hourglass with masked predictions of our model. Doing this and using ground-truth data do decide which joints to mask, instead of confidence scores, gave us our first improvement, a whooping $+0.3$ accuracy.

At one point we also tried adding Gaussian noise to the predictions, both in addition and instead of masking strategy, to see if the model could also learn to actually "refine" the predictions, but this approach failed miserably.

Also at one point we tried feeding a "distance matrix" to the network. By this we mean, we fed the network the distance of each known joint to one another. This is not "more information", but an information subtracted from the input on by us on network's behalf. This model took longer time to learn, and converged to a similar accuracy, so we dropped this also.

After all these, we also tried further training the refiner, not on GT inputs and GT targets, but on hourglass predictions as inputs and GT targets. This training strategy, 20 epoch per masking count, increased the accuracy improvement from -2.5 to -2.3, or in general extra training with predictions only gave slight increase.

2.1.4 Future Ideas

After these results, we came together and came up with ideas about how we can proceed to further improve the accuracy. The following ideas were accepted to try:

1) Feeding image features in to the network: In general, all pose

estimation models at some point produce image features of the input image. In our example model, whose inferences we try to improve, hourglass, does this at the end of every stack. We will use the outputs of the last stack, before the classification layer, as image features. We tough about running these features trough a simple CNN for dimensionality reduction, and feed in to our own refiner network as an 64-array input.

2) Feeding confidence scores in to network: Remember that we used confidence scores of hourglass to mask bad predictions. We tough, instead of doing this by hand, we could feed along with x and y coordinates, also the c confidence score to the network per joint, and let network decide which joints to replace.

3) After all these, model search and hyper-parameter search.

My internship ended before i could try these techniques. Since the project which i will mention in next section failed, currently I have returned to the refiner idea and trying the mentioned strategies.

2.2 Edge Aware Instance Partition

In this work, we wanted to try an extension to the already published model and paper of Muhammed Kocabaş and Salih Karagöz[5] . The model is a regular multi-person pose-estimation model. Normally such models comes in two variations, top-down and bottom-up. MultiPoseNet belongs to bottom-up models family with a slight twist.

2.2.1 Background

One important problem with human pose-estimation task is connecting the joints found by the model. When the task is single-person pose-estimation, a good model, after post-processing, would give you at most one prediction per joint, which are obviously the joints of the single human in the picture. But when the task is extracting the poses of multiple person, in addition of finding the coordinates of the joints, you also need to somehow assign each joint to a person instance or equivalently group joints in to person instances.

The top-down family of models, to handle this problem, first finds the person instances in the picture and crops them, and then apply single-person pose-estimation models on these crops. Obviously the techniques may vary, but what happens is that you run a full pose-estimation model multiple times over the regions of the image. For this reason, top-down family of models are usually slower then bottom-up family, but has higher accuracies.

The bottom up family, on the other hand, finds all the joints in the picture at once and then try to group them, so usually they run their pose-estimation models only once. There are many approaches to this grouping problem.

My handlers' model, MultiPoseNet, in a way combines the two approach in a bottom-up manner. They both run a pose-estimation model and a human detection model on the image once. For each human crop the human detection model gives, they instead crop the joint predictions, and feed the resulting crop to a mlp model called Pose Residual Network(PRN). What PRN does is that given a crop of joints, it essentially deletes the joints which it thinks are not part of the joints of the human in the crop. So in a way, if there are multiple joints of the same type in the given crop, it deletes all but one of them.

2.2.2 Analysis Of The Problem

Recently my handlers had heard a paper which does the task of instance-part segmentation. Their model would partition each human to its classes, including left arm, right arm, hat, dress, t-shirt, shoes etc. To accomplish the task, in their pipeline they make use of an "edge image". Using learning methods, they output an image such that on a black background they highlight only the edges of the humans in the picture. And then they were partitioning this edge image in to human instances, since in the former case there is no information about which edge or closed-shape belonged to which instance. In the latter tough, we expected that the authors to a good degree partitioned the "blobs" or "closed-shape regions" in to human instances, which they called "instance-aware edge maps. They would then further combine some stuff to produce their own results.

Then, our idea was to directly use these instance-aware edge maps to group joint-predictions in to human-instances, without any learning.

2.2.3 Solution

Directly using their results failed miserably, giving only %2 accuracy.The reason was that, their partition algorithm output too many "blobs". Say there are 2 person in the picture. The resulting edge map would have 2 big blobs overlapping the humans, but also many other blobs again overlapping the persons. For their own purpose, this did not pose a problem. But for us obviously that meant, using a direct assignment to blobs, would fail to assign many joints to biggest blob.

First, we tried using heuristics to both merge the blobs, assign the joints, and then re-assign them; but we could only improve the accuracy up to %10. Following this we spend some time discussing how we could by-pass the problem. One idea was to propagating lines from joints to surrounding edges, in practice making our "coordinates larger"; so that we could solve our problem similarly how the paper's authors solved it, but this also failed.

Then we noticed that, the authors themselves did not grouped the mentioned object classes in to human instances! Their dataset, it turned out, only required for them to partition the classes in to object-instances; not to further group them in to human-instances.

After the failures and the mentioned discovery, we ceased working on the subject to not waste further time.

3 ORGANIZATION

Metu ImageLab is one of the research laboratories formed and operated under the computer engineering department in METU. Mainly run by Prof. Fatoş Vural and Prof. Emre Akbaş, lab's main focus is solving computer vision problems in academia. Majority of the researches focuses on machine learning solutions to such problems.

3.1 ORGANIZATION AND STRUCTURE

The lab's website [4] currently lists 5 professors and there are many graduate and doctorate students affiliated with the lab. During the summer, six other undergraduate students also did their internships in ImageLab, tough in different projects and under different grad students.

3.2 METHODOLOGIES AND STRATEGIES USED IN THE COMPANY

MetuLab has 8 GPU cards in it's disposal, which can be used by lab's affiliates freely for their research projects. There isn't any specific methodology in the lab; all grad and doctorate students works with projects guided by their respective professors. The aim of all affiliates is, at the end, publishing papers and contributing to man-kind's knowledge.

4 CONCLUSION

The first part of my intern-ship, Ada Portal, tough me to always be careful while writing production code; there are tenfold of best practices, company standards, and performance-maintainability trade off. It was my first time writing production code and trying to use boring but powerful company libraries in right way.

The second part was definitely the most fun and en-lighting. My guider, also the boss, and my cousin, simply ”started” the project as; ”We have lots of data, there is this thing called Neural Network which make predictions out of data, go figure”. Not that i am doing any criticism, it was a new and important know-how for the whole company, and my guider gave the best possible insight and help for the project, which i am proud that we at the end had a tangible, working prototype that actually have correct predictions. The part of the project that i invested all my time in research instead of coding gave me great insight about the current state and importance of Machine Learning on solving problems well beyond human mind’s capabilities, and also the fact that we are still way too far away from replicating any functionality of the brain as it is implemented in brain itself. At least we, as humanity, seem to kick started the phase where we try to write algorithms which themselves implement algorithms, instead of writing those algorithms directly.

5 APPENDICES

5.1 FIGURES

	Original Acc.	Improved Acc.	Improvement
Only GT training	89.1	52.7	-36.4
Masking bad predictions with GT (MBP)	58.5	86.6	-30.6
MBP with confidence scores	89.1	56.5	-32.6
MBP with GT + Reusing Good Predictions	89.1	89.4	+0.3
MBP with confidence scores + RGP	89.1	87.6	-2.5

Table 1: Inference with different strategies

	Val	Input	Mask Only
Full Pass (After six mask)	74.8	60.0	78.2
Zero Mask	97.0	100	0
One Mask	91.5	93.2	89.9
Two Mask	89.2	86.6	92.3
Three Mask	85.6	80.0	89.4
Four Mask	82.6	73.3	86.9
Five Mask	80.4	66.6	84.8
Six Mask	76.8	60.0	80.4
w/ Dist. Matrix	97.0	100	0
Autoencoder (average)	27.5	100	0

Table 2: First and last two rows are trained 20 epoch per mask count. Other rows are trained in succession, 200 epoch each.

5.2 REFERENCES

References

[1] cocodataset.org

[2] github.com/bearpaw/pytorch-pose

[3] Newell, A., Yang, K., & Deng, J. (2016, October). Stacked hourglass networks for human pose estimation. In European Conference on Computer Vision (pp. 483-499). Springer, Cham.

[4] image.ceng.metu.edu.tr

[5] Kocabas, Muhammed, Salih Karagoz, and Emre Akbas. "Multi-PoseNet: Fast Multi-Person Pose Estimation using Pose Residual Network." European Conference on Computer Vision. Springer, Cham, 2018.

[6] http://cocodataset.org