UNIVERSITY OF SCIENCE AND TECHNOLOGY OF HANOI
UNDERGRADUATE SCHOOL

Bachelor Thesis

by
Nguyen Nhu Hieu
USTHBI9-103
Information and Communications Technology
Academic years: 2018 - 2021

Title:

Human pose estimation using Nvidia Jetson Nano

Supervisors:

Than Cao Cuong - Assilla Vietnam
Dr. Nguyen Hoang Ha –
University of Science and Technology of Hanoi

Declaration

I, Nguyen Nhu Hieu, hereby declare that the work in this thesis is entirely my own under the guidance of my external supervisor Than Cao Cuong and my internal supervisor, Doctor Nguyem Hoang Ha. I certify that the work and results are totally honest and have not been published in the same or any similar form. All methods, comments, and statistics referenced from other authors and organizations' work have been acknowledged and indicated clearly. If any fraud is found, I will take full responsibility for the content of my thesis and will accept any penalty from the thesis defense committee and my university.

Hanoi, July 2021

Nguyen Nhu Hieu

Acknowledgments

The creation of this thesis wouls not be possible without the help and guidance of many people to whom I wish to express my most sincere gratitude.
Firstly, I would like to express my sincere gratitude toward my external supervisor Than Cao Cuong for guiding my throughout my 3 months internship and to suggest me the idea of this thesis in the first place. I would also like to thanks Asilla Inc. for offering me an internship position in their company.
Secondly, my thanks also goes to Dr. Nguyen Hoang Ha for accepting me as my internal supervisor and to aid me thoughout my internship. My appreciation also extends to Dr. Tran Giang Son, for granting me the permission to utilize USTH ICT's servers and for listening for my requests.
Thirdly, I would like to express my gratitude toward my colleagues at the R&D room of Asilla Inc. for providing me with such an amazing environment for my internship and for providing me with aid whenever I needed.
Last but not least, I would like to thank all my family members, friends and the professors, doctors and staff at USTH for supporting me during the past academic years.

Abstract

As machine learning techniques improve, many things that have been impossible before have become possible now. One of such is to be able to detect human pose. Application of such detection are many, some of those included action detection or for the pose detected to be used in creating a virtual avatar along with many things else. In those application, there is a need for the model to run on edge computing. Even so, many current pose estimation models aim toward accuracy. By focusing on accuracy, pose estimation models sacrifice their speed and consume a lot of resource that would be impossible to provided on edge computing. This thesis aim to developed and deploy a model with improved speed and usable accuracy for those application, the model must have a satisfied performance on limited resources of an edge computing device. One of the such possible platform that can perform edge computing are Jetson Nano devices developed by Nvidia. Model which are converted into TensorRT are best optimize for running on running on this type of device and this thesis choose the github trt-pose[1] `https://github.com/NVIDIA-AI-IOT/trt_pose` as the framework for implementation on the device. trt pose use Pytorch-based model and its model, training evaluation and live demo code all use Python language, however, the rest of the program use C++ language for its fast calculation .The aim of this thesis to experiment with the model provided by the github and to find ways to improved on it.

Contents

Chapter 1

Introduction

1.1 Context and motivation

1.1.1 About human pose estimation

Human pose estimation is a section in the field of computer vision that aims to detect and predict a human body skeleton made up of a set of human body parts like wrist, shoulder, hip, knee, ... Such body part can be called keypoint for abbreviation. The number of keypoint is determined beforehand for the machine learning model.
There's currently two main approaches for the task, a top-down approach and a bottom-up approach. The top-down approach uses object detection models like Mask R-CNN[2] or YOLO[3] to first detect the bounding box of a person then use another model to detect one human pose in each bounding box. On the other hand, the bottom-up approach directly detects all keypoint from the image then matches the keypoint with each other to form a human pose.
Each approach has its own up and down sides, top-down approach can be more accurate than bottom-up approach regardless of the number of people in the image as long as those people are not occluded .For the bottom-up approach, the keypoint matching might have lower accuracy if there is too many people in the image but can be more accurate than top-down approach if the people in the image are occluded. This is because when people are occluded, their bounding boxes heavily overlapped which lead to a false keypoint detection when running the second model

of the top-down approach. The running time of the top-down approach is linearly related to the amount of people in the image and so generally the bottom-up approach will be faster if there are a large number of people in the image. If the image only contain a small number of people, the top-down approach is generally faster than the bottom-up approach. This thesis choose to use the bottom-up approach because it is suitable for the limited calculation that Jetson Nano have and its accuracy are guaranteed when the system has to work in crowded place where people are usually occluded with each others.

	image has a large number of people	image has a small number of people
the people in the image are occluded	Bottom-up approach are faster and more accurate than top-down approach	Top-down approach are faster than bottom-up approach but bottom-up approach are more accurate
the people in the image are not occluded	Bottom-up approach are faster than top-down approach but top-down approach are more accurate	Top-down approach are faster and more accurate than bottom-up approach

Figure 1.1: comparison between bottom-up approach and top-down approach

1.1.2 About the Jetson Nano

Jetson Nano is a small, powerful computer developed by Nvidia that allow to run machine learning model with low power cost. Jetson Nano can be deployed easily in many place like drone, robot or public camera. Jetson Nano is builted with a Quad-core ARM A57 CPU, a 128-core NVIDIA Maxwell GPU and a 4 GB 64-bit LPDDR4 RAM. It contain 4 USB 3.0 Type A port, an USB 2.0 Micro-B port, 2 HDMI/DisplayPort and a Gigabit Ethernet connection. It can be connected to any type of camera and run computer vision model in real-time. Jetson is powered by a DC 5V 4A

adapter, its USB 2.0 Micro-B port can also powered with 5V 2A power, as such it has a very low power consumption and can be deploy with just a battery. All of its spec make it possible for the device to be deployed on the field to be used for edge computing thus improved the response times and save bandwidth.

Figure 1.2: Jetson Nano

The github trt-pose[1] `https://github.com/NVIDIA-AI-IOT/trt_pose` is one the available github for deploying a machine learning model on Jetson Nano and in this case a bottom-up human estimation model. trt pose[1] utilizes the same overall pipeline as Openpose[4] but with a much

simpler architect thus allowing it to run faster at the cost of its accuracy.

1.2 Objective

The objective of our thesis are as follow:
- To experiment with the github model by either replace the backbone model and to replace the loss function for training and to train such model
- To compare the speed and the accuracy of such model to the pretrained models provided by the github
- To deploy the model on Jetson Nano and to try to estimate human pose on various image and video.

1.3 Expected Outcome

With such objective, we aim to have the resulting model with change backbone and loss is expected to have an advantage compare to the pretrained models, the advantage can include have higher speed without having lower accuracy, higher accuracy without having lower speed or having both higher accuracy and higher speed. The model should be able to be deployed easily on any Jetson Nano device and the device should be able to run the resulting model in real-time.

Chapter 2

Methodology

2.1 Training and validation dataset

The training and validation both use the COCO dataset[5] which contain up to 110.000 images for training and 5000 images for validation. There are also many others humanpose dataset like MPII human pose dataset[6], PoseTrack[7] which is a dataset that aims for training the human pose track process, OCHuman[8] which only included heavily occluded human pose, we choose to use the COCO dataset because its data are generalize enough for used. The number of people keypoint annotations in the COCO dataset at more than double of that at 260.000 people in training images and 11.000 people in validation images. Image sizes can range from 59x51 to 640x640 although most of them have size larger than 300x300, a person object with size smaller than 32x32 is not annotated and have their bounding box noted. COCO annotation[5] originally contain 17 keypoints: nose, left eye, right eye , left ear, right ear, left shoulder, right shoulder, left elbow, right elbow, left wrist, right wrist, left hip, right hip, left knee, right knee, left ankle and right ankle but trt pose[1] modified it to add on 1 more keypoint: neck which is set in the middle between right shoulder and left shoulder.
The limb structure in the annotation of COCO[5] originally include 19 different limb: left ankle to left knee, left knee to left hip, right ankle to right knee, right knee to right hip, left hip to right hip, left shoulder to left hip, right shoulder to right hip, left shoulder to right shoulder, left shoulder

to left elbow, right shoulder to right elbow, left elbow to left wrist, right elbow to right wrist, left eye to right eye, nose to left eye, nose to right eye, left eye to left ear, right eye to right ear, left ear to left shoulder and right ear to right shoulder. trt pose[1] remove 3 limbs: left shoulder to left hip , right shoulder to right hip and left shoulder to right shoulder limbs to add 5 limbs: neck to nose, neck to left shoulder, neck to right shoulder, neck to left hip and neck to right hip.
Ground truth keypoints have the form $[x_1,y_1,v_1,...,x_k,y_k,v_k]$, where x,y are the keypoint locations and v is a visibility flag defined as v=0: not labeled, v=1: labeled but not visible, and v=2: labeled and visible. Each ground truth object also has a scale s which we define as the square root of the object segment area.

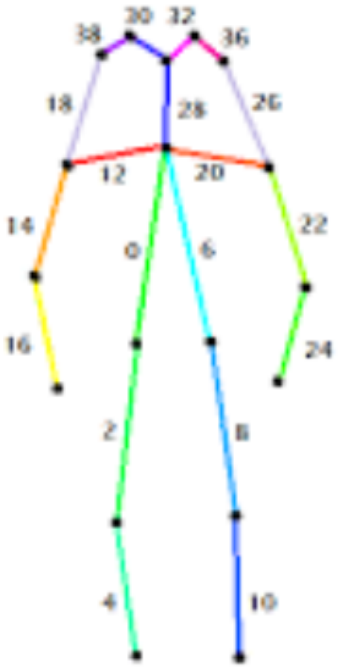

Figure 2.1: modified human pose

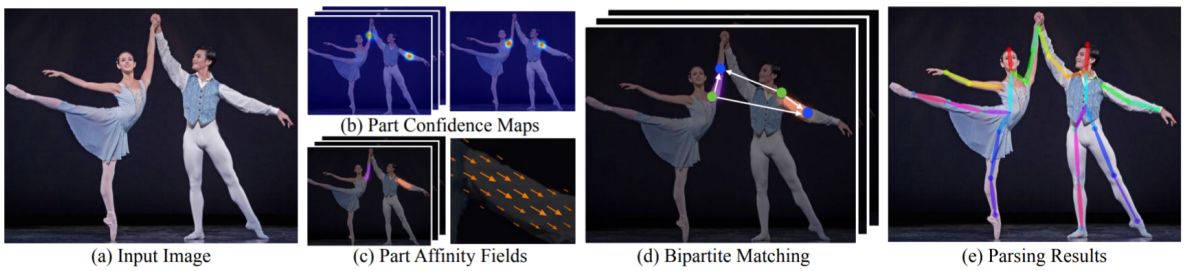

Figure 2.2: Overall pipeline

2.2 Overall pipeline

The below figure illustrates the overall pipeline of Openpose[4] and is also the pipeline that trt pose[1] uses for its human pose estimation. The input image is first passed through a machine learning model to predict a set of confidence map each indicated the position of each keypoint and a set of part affinity field which indicated the connection between a pair of keypoint. We call such connection a limb (even between body part like eye and ear) to abbreviate. Using the set of confidence map, we can predict the position of each type of keypoint on the image, the predicted keypoints then form a set of bipartite graph by connect all keypoints of one type to another. Each bipartite graph which represent one type of limb uses the part affinity field to measure the association between each keypoint, the association scores generated is then used by a greedy algorithm to matches the keypoints that belongs to the same person, the final result is parsed onto the input image for the output result.

2.2.1 Model architecture

Unlike the model used by Openpose[4], trt pose[1] uses a very simplified version of it. In such model, the input image is first passed through a backbone model to extract feature from the image. The backbone is made up of the first layers of a pretrained object classification model like resnet[9], densenet[10], vgg[11]. Because the features extracted have low resolution, it is passes through a number of transposed convolutional layers with stride 2 to increase the the resolution by 2 or 4 times. Such features

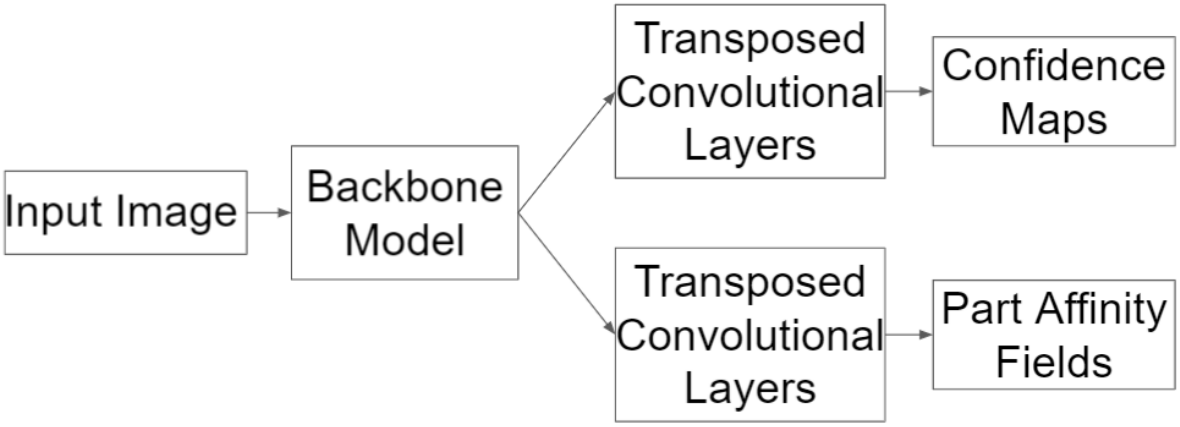

Figure 2.3: Model architect

is then used to predict confidence map on one branch and to predict part affinity field on the other. The result fields and maps can also be pass though an upsampling layer to further increase its resolution for higher accuracy.

Backbone model

Usually, the backbone model of a human pose estimation model is an object classification model. This is because the backbone model of a machine learning model has the task of feature extraction, as such, a previously use model architect along with its pretrain parameters would leads to a faster convergence of the transposed convolutional layers. For trt pose[1], the github uses resnet[9] and densenet[10] as its backbone for benchmark, specifically, it uses densenet-121[10] for higher accuracy and resnet-18[9] for higher fps, this thesis will uses the Deep Layer Aggregation (DLA)-34[12] model as the backbone.

Introduced by Fisher Yu Et Al. in the paper , Deep Layer Aggregation[12] presents a new architect of Iterative Deep Aggregation and Hierarchical Deep Aggregation, aggregation in the paper is define as the combination of different layers throughout a network. As one of the basic component of the model, aggregation node used by the model has the purpose of combining and compressing the inputs and it learns to select and project important information to maintain the same dimension between the out-

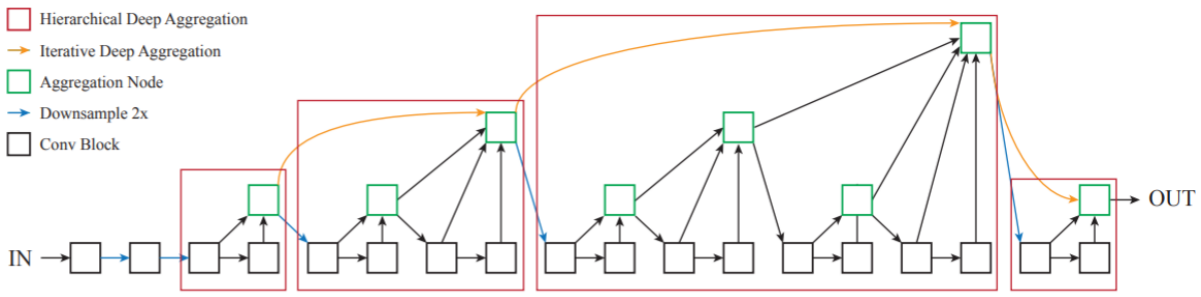

Figure 2.4: dla architecture

put and the input.

Iterative Deep Aggregation uses the concept of skip connection intro-

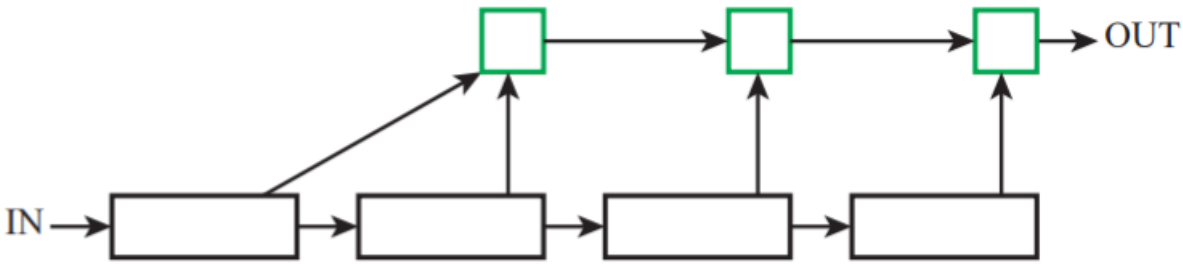

Figure 2.5: Iterative Deep Aggregation

duced by the paper *Deep Residual Learning for Image Recognition* [9] but in a different way. Instead of using it as a shortcut between layer, DLA aggregates from the shallowest layer to the deepest which allow us to refine the coarse feature from the shallowest lever as they propagated through different stages. This structure is shown in the above figure.

Hierarchical Deep Aggregation combines feature channel from all layers in a tree-like structure, unlike Iterative Deep Aggregation which only combine layers sequentially. With the tree-like structure, HDA allows the model to learn richer combinations that span more of the feature hierarchy, the output of each sub-tree is routed back to the model which allow us to propagate the aggregation of all previous blocks instead of the preceding block alone to better preserve features. This structure is shown in the above figure.

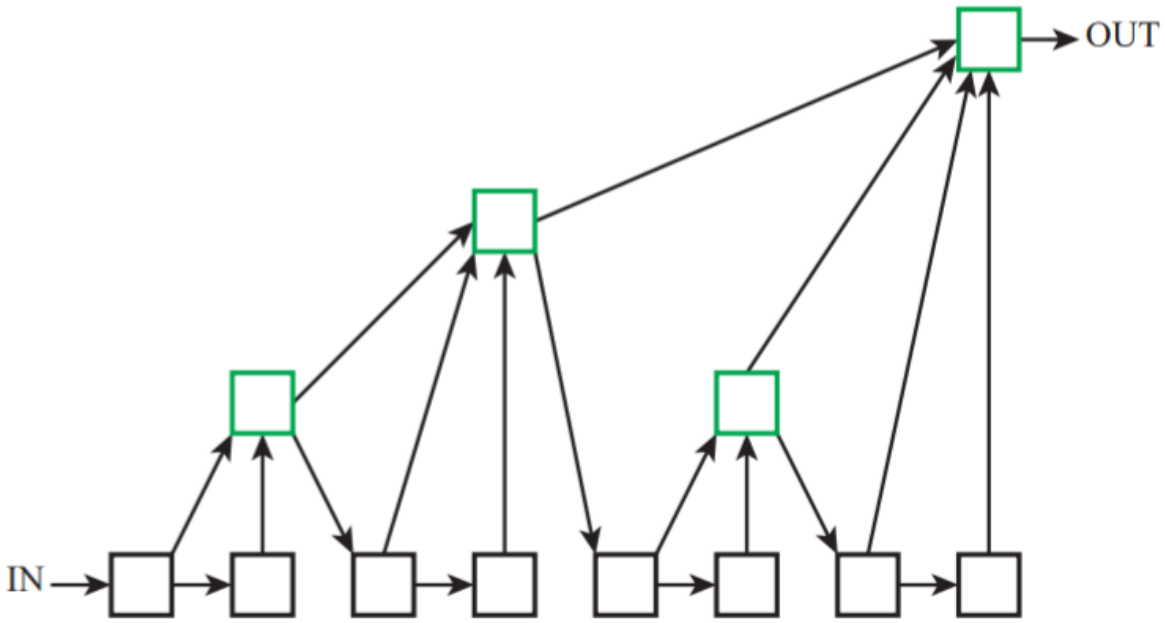

Figure 2.6: Hierarchical Deep Aggregation

DLA-34[12] that we use for backbone is an augmented model from the resnet-34[9] model, DLA[12] uses the same basic block as resnet[9] but instead of using residual connection dla[12] connects across stages with IDA and within and across stages by HDA. This results in a model with similar number of layers but with fewer parameters and better performance. DLA-34[12] has about 30% fewer parameters and about 1 point of improvement in top-1 error rate. Compared to ResNet-50[9] and ResNet-101[9], DLA[12] networks can still outperform the baselines significantly with fewer parameters.

2.2.2 Confidence map

As stated above, in bottom-up approach, we first detect all keypoints in an image, and Openpose achieved this by using confidence maps. In machine learning, for the task of numerical coordinate regression which keypoint detection belongs to, there are two main approaches, the first approach uses the fully connected layer to directly return to the coordinate point, such as yolo-v1[3], the second approach uses predictive Gaussian heat map,

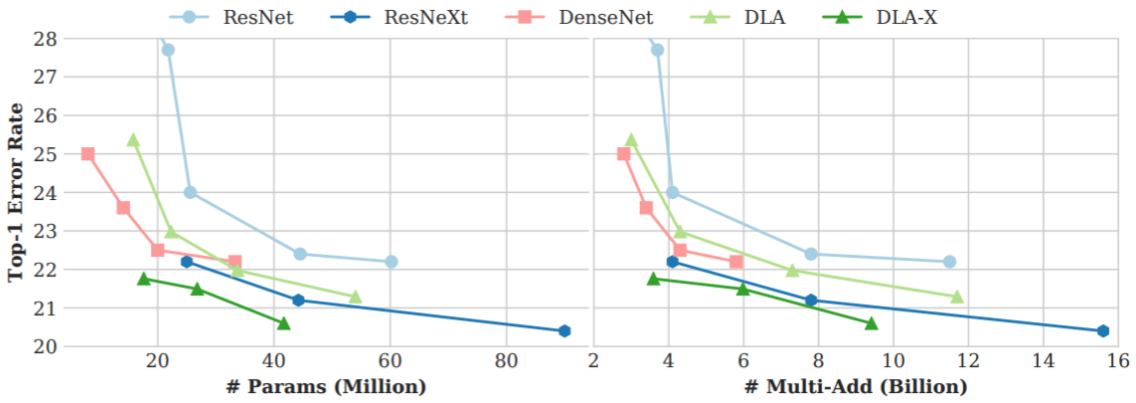

Figure 2.7: Evaluation of DLA on ImageNet Large Scale Visual Recognition Challenge. DLA/DLA-X have ResNet/ResNeXt backbones respectively. DLA achieves the highest accuracies with fewer parameters and fewer computation

then finds out the index corresponding to the peak value as the coordinate point by argmax and is what Openpose[4] uses. Each approach has its up and down sides, the advantage of fully connected layer type of approach is that the output is the coordinate point and so the training and forward speed can be fast, and it is end-to-end full differential training, however, the disadvantage of it is the lack of spatial generalization ability, that is, the loss of spatial information on the feature maps. For Gaussian heat map method, the advantage of this type of approach is that the accuracy is usually higher than that of the first method because of its spatial awareness on the image, its disadvantages is that because the required output feature map is very large, the training and forward speed is very slow, and the memory consumption is large and the output from input to coordinate point is not a fully differential model, because from heatmap to coordinate point, it is obtained through argmax. As keypoints are generally connected through the people they are from, it is better for the model to have the spatial awareness for better human pose estimation, as such, Openpose[4] chooses to use the confidence maps which is a Gaussian heat map method.

Each confidence map is a 2D representation of the confidence score or the possibility that a particular body part can be located in any given pixel, each map only represents one single type of body part. Ideally, if a single person appears in the image, a single peak should exist in each confidence map if the corresponding part is visible; if multiple people are

Figure 2.8: confidence map

in the image, there should be a peak corresponding to each visible part for each person. The peak here in confidence map ground truth are generate by the two-dimensional Gaussian function for each keypoint as follow

$$f(x, y) = exp(-((x - x_0)^2 + (y - y_0)^2)) \tag{2.1}$$

Where x,y is the coordinate of the confidence map and x_0,y_0 is position of a keypoint, for overlap keypoint peak, the ground truth takes the maximum value out of the the values generate by keypoints in order for the precision of nearby peaks remains distinct. At test time, we can predict keypoint by performing non-maximum suppression to locate peaks on the map.

2.2.3 Part affinity field

There are multiple possible ways to matches keypoint with each other and Openpose[4] along with trt pose[1] which uses the same pipeline utilizes part affinity field to solve the problem.

Part affinity field (paf) is a 2D vector field that encode the orientation

Figure 2.9: part affinity field

information of a limb (a pair of keypoint), the number of limb and consequentially, the number part affinity fields are determined beforehand. For each pixel in the area belonging to a particular limb, a 2D vector encodes the direction that points from one keypoint to the other. Each type of limb has a corresponding PAF joining its two associated body parts.

To came up with the values for the paf ground truth, trt pose[1] utilizes

Figure 2.10: part affinity field vector representation

the same Gaussian function used by confidence map to encode the vector field.
For any point C on the field, with a limb made up of two keypoints A and B, the scalar projection of vector $\overrightarrow{AC}$ onto vector $\overrightarrow{AB}$ is calculated and if this dot product is smaller than the length of limb AB and larger than 0, point C is considered to lie between A and B. For such cases, the vector values are scaled exponentially by the 2D distance d from C to line AB using a simple Gaussian function of

$$paf(C) = exp(-d^2) * \overrightarrow{AB_u} \tag{2.2}$$

Where paf(C) is the part affinity field vector values at point C and $\overrightarrow{AB_u}$ is the unit vector of limb AB. For point which lie outside of the above boundary, the Gaussian function is modified to include the distance d_t from from such point C to a line perpendicular with line AB and go through the nearer keypoint

$$paf(C) = exp(-(d^2 + d_t^2)) * \overrightarrow{AB_u} \tag{2.3}$$

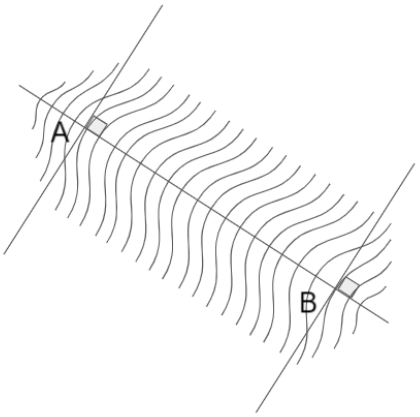

Figure 2.11: a 3D represent of the vector absoulte value

For paf with multiple limb, unlike in confidence map which takes the maximum value, paf averages the affinity fields of all people in the image.
During testing, trt pose[1] measures association between candidate part detections by computing the line integral over the corresponding PAF

along the line segment connecting the candidate part locations. Specifically, the association E is calculated as follow

$$E = \int_{C=A}^{C=B} paf(C)\, dC/||AB|| \tag{2.4}$$

where AB is the keypoint pair that formed a limb. In practice, E is approximated by uniformly sampling and summing vector value on limb AB.

2.2.4 Bipartite matching

After performing non-maximum suppression on the confidence maps, we achieve a discrete set of human part part candidate locations on the image, each confidence map show one type of keypoint. These part candidates define a large set of possible limbs where each can be given a score by the part affinity field that represent that type of limb using the line integral computation on the PAF, equation (2.4). Using all the limb that belongs to one type we could form a bipartite graph which connect one keypoint to another. As one keypoint can only belongs to one person, the problem now becomes a bipartite matching problem where each weight of an edge is the score calculated from a limb represent that edge. This problem can be solve using the Hungarian algorithm which is a greedy algorithm and is what Openpose[4](and by extension, trt pose[1]) uses to connect all human part with each others. The result connection can then be parsed onto the image.

Hungarian algorithm

The Hungarian algorithm is a greedy algorithm that solve the assignment problem, It was developed and published in 1955 by Harold Kuhn, who gave the name "Hungarian method". James Munkres reviewed the algorithm in 1957 and since then the algorithm has been known also as the Kuhn–Munkres algorithm or Munkres assignment algorithm. The time complexity of the original algorithm was $O(n^4)$ however it is minuscule

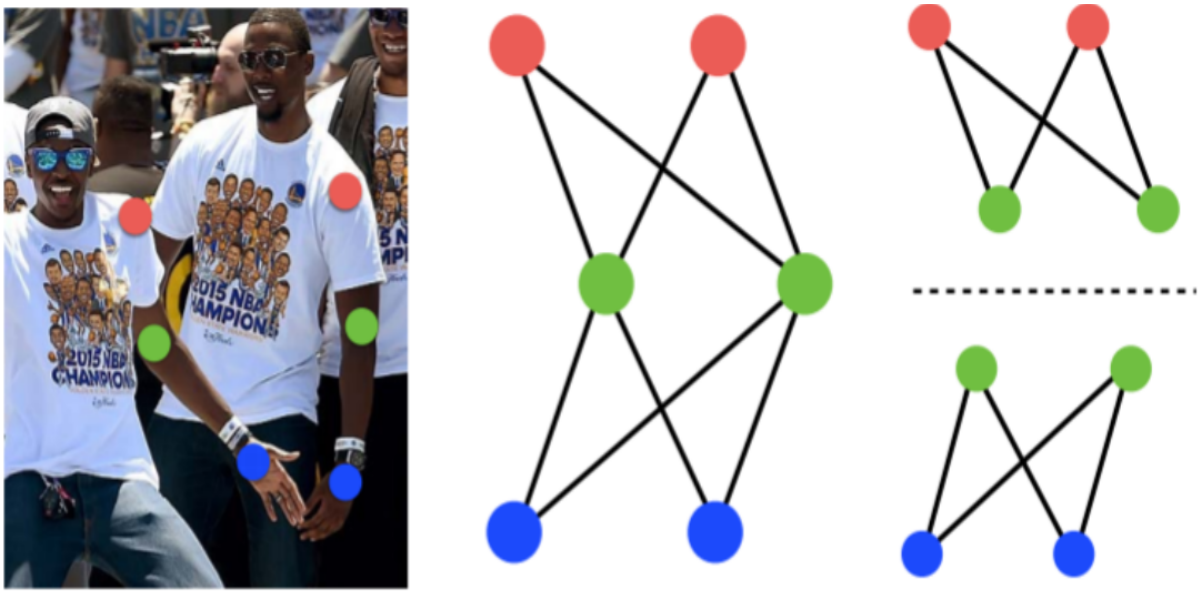

Figure 2.12: Bipartite matching

compare to the running time of the machine learning model itself. The assignment problem of bipartite matching that we are trying to solve here can be understand as finding the optimal assignment of a mxn limb score matrix for a maximum total score, the value of the matrix range from 0 to 1. An example of the problem can be as follow:
Assuming a 4x5 matrix score where a,b,c,d,e is the same type of keypoint and are connected to another keypoint which consist of 1,2,3,4. a2 here represent the score of the limb which connect keypoint a with keypoint 2. The Hungarian algorithm solve this problem by using multiple steps, the steps are as follow:

Table 2.1: a limb score matrix

a1	a2	a3	a4
b1	b2	b3	b4
c1	c2	c3	c4
d1	d2	d3	d4
e1	e2	e3	e4

- Step 1: We turn the problem into a find minimum total score problem by taking each element and have 1 minus it, the result matrix is as follow where a1' = 1 - a1 which makes all element of the matrix to be between 0 and 1

a1'	a2'	a3'	a4'
b1'	b2'	b3'	b4'
c1'	c2'	c3'	c4'
d1'	d2'	d3'	d4'
e1'	e2'	e3'	e4'

- Step 2: We turn the matrix into a square matrix by adding in the matrix rows of 0s or columns of 0s, in this case we add a dummy column of 0s to make a 5x5 matrix

a1'	a2'	a3'	a4'	0
b1'	b2'	b3'	b4'	0
c1'	c2'	c3'	c4'	0
d1'	d2'	d3'	d4'	0
e1'	e2'	e3'	e4'	0

- Step 3: We then perform row operations on the matrix for each row by taking the smallest element and subtracting the whole row to it, because the matrix contains a column of 0s which is the minimum, the operation have no effect on the matrix

a1'	a2'	a3'	a4'	0
b1'	b2'	b3'	b4'	0
c1'	c2'	c3'	c4'	0
d1'	d2'	d3'	d4'	0
e1'	e2'	e3'	e4'	0

- Step 4: We perform column operations on the matrix similar to the row operations of step 3, taking the smallest element of each column and subtracting the whole column to it.

a1”	a2”	a3”	a4”	0
b1”	0	b3”	b4”	0
c1”	c2”	0	0	0
0	d2”	d3”	d4”	0
e1”	e2”	e3”	e4”	0

- Step 5: We try to cover all the zeros within the matrix with a minimum number of lines, in this case, we can cover the all the zeros within the matrix with a minimum number of 4 lines. If the minimum number of lines required to cover all zeros is equal to the number of row of a nxn square matrix then an assignment can be done by finding n elements from the matrix each with the value of 0 and no element share the same row or column.

					x
	a1”	a2”	a3”	a4”	0
x	b1”	0	b3”	b4”	0
x	c1”	c2”	0	0	0
x	0	d2”	d3”	d4”	0
	e1”	e2”	e3”	e4”	0

- Step 6: If the minimum number of lines required to cover all zeros is not equal to number of row of a square matrix then we would find the smallest

element from all the element that are cover by the lines. We subtract this from all the element that are not covered and add it to all elements that are covered twice by the line. we then repeat step 5 to see if we can find an optimal assignment, if we still can't find it, we repeat step 6 and then step 5 until it is possible, in this exemplary case, an optimal assignment can be found.

a1"'	a2"'	a3"'	a4"'	0
b1"	0	b3"	b4"	e3"
c1"	c2"	0	0	e3"
0	d2"	d3"	d4"	e3"
e1"'	e2"'	0	e4"'	0

2.3 Loss function

We use two different loss function for training: mean square error and focal L2 loss[13], compare to trt pose[1] which only uses mean square error for both the confidence maps and part affinity fields, our paper replaces mean square error for confidence maps into focal L2 loss[13] while keeping the part affinity fields loss as mean square error.

2.3.1 Mean square error

Mean square error is the standard loss method use for training model, the loss takes the different between the predict value by the model and the ground truth value then square it up to be divided by the number of values in ground truth. The formula is as follow

$$MSE = \frac{1}{N}\sum_{i=1}^{N} M(i)(p_i - t_i)^2 \tag{2.5}$$

Where N is the number of values to predicted, M(i) is the binary mask covering the image taken from the dataset where area with mask=0 is the

area contain a person object without a human pose annotation because it is smaller than 32x32, p_i is the predicted value for pixel i and t_i is the ground truth value. We use the binary mask in order to avoid penalizing the true positive predictions during training.

2.3.2 Focal L2 loss

Mean square error works by minimizing the difference between real and predicted key-point positions in a pixel level regression framework. The issue of this approach is that the predictions only contain the positions of pixels, while their semantic information is lost. For example, imagine a sparse binary image where six pixels are labeled to 1 (blue) and all other pixels labeled to 0 (white).

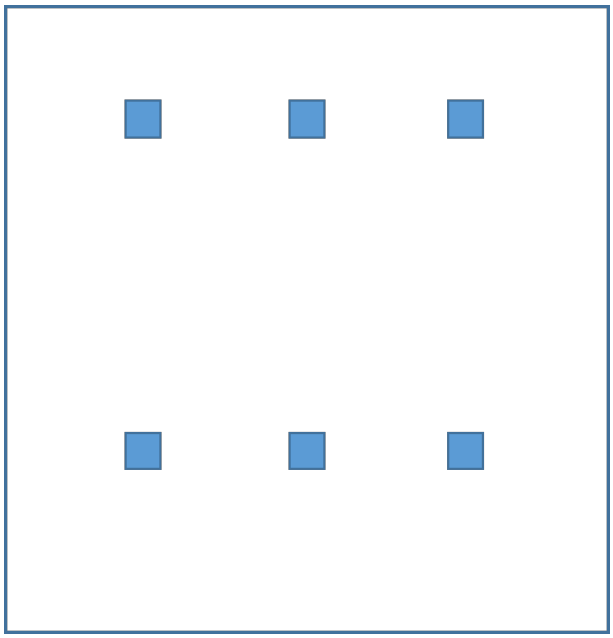

Figure 2.13: sparse binary image

As most the of image here is labeled 0 (white) while only a few is labeled 1 (blue), the MSE value calculate here will mostly take 0 as the ground truth and so will over representing it. This is a case of imbalanced label in which the confidence map that we use also suffer from. As confidence map has most of its area equal to zero (background) and only a small portion of

it corresponds to the Gaussian distribution (foreground), it is in a similar situation with the sparse binary image above. The Gaussian function that generate the peak for confidence map can be modified to increase the size of the peak, however, this will also lead to a decrease in the performance of the non-max suppression algorithm. Following the solution in the paper Simple Pose: Rethinking and Improving a Bottom-up Approach for Multi-Person Pose Estimation[13] by Jia Li et al, we decided to utilize focal L2 loss[13] function as the solution to the problem.
Focal l2 loss[13] here is simply a l2 loss modified to be a focal loss[14], where l2 loss is the square different between ground truth and the prediction made by the model. On the other hand, focal loss[14], implemented in Focal Loss for Dense Object Detection[14] paper by He et al, is a modified version of the binary cross entropy loss where a multiplicative value is added in order to decrease the loss as the ground truth value gets higher.
The probability of ground truth class here (the X-axis) is the value predict

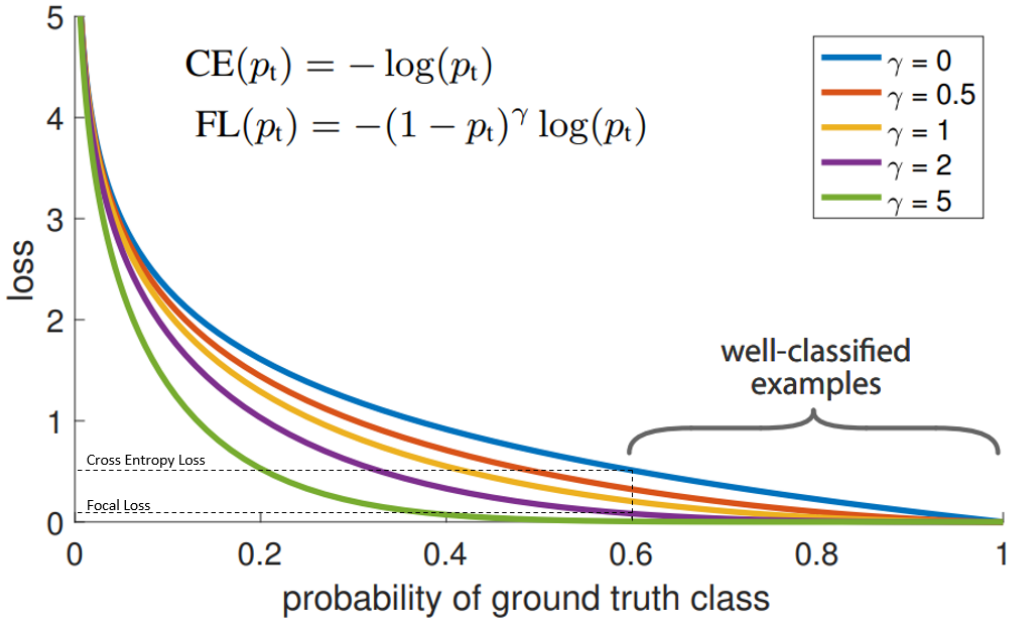

Figure 2.14: Focal Loss vs Cross Entropy Loss

by the model compare the ground truth class. For example, if the ground truth here is 1 and the value predicted by the model is 0.6, the probability

of ground truth class (called p_t for short) would be 0.6, but if the ground truth is 0 and the value predicted by the model is 0.6, p_t would be 0.4 as it is equal to (1-0.6). As can see from the graph, when p_t is calculate to be 0.6, cross entropy loss still reach up to 0.5. This mean cross entropy loss required the the model to predict with a much higher probability. In other word, cross entropy loss required the the model to be very confident about its prediction. This can negatively impact the performance of the model as it force the model to be overconfident and so cant generalize really well. Focal loss[14] solve this problem by lowering the loss as p_t gets higher, effectively allow the model to predict with a lower probability score which turns the models attention towards the rare class in case of class imbalance. The Focal loss[14] formula is mathematically defined as:

$$FL(p_t) = -\alpha_t(1-p_t)^{\gamma}log(p_t) \tag{2.6}$$

Where γ controls the shape of the curve. The higher the value of γ, the lower the loss for well-classified examples, so we could turn the attention of the model more towards 'hard-to-classify examples. Applying focal loss[14] to the l2 loss, the paper Simple Pose: Rethinking and Improving a Bottom-up Approach for Multi-Person Pose Estimation[13] comes up with the following formula:

$$FL_2L(i) = \frac{1}{N}\sum_{i=1}^{N} M(i)(p_i - t_i)^2(1-sdt)^{\gamma} \tag{2.7}$$

where

$$sdt = \begin{cases} p_i - \alpha, if\ t_i < thre \\ 1 - p_i - \beta, else \end{cases} \tag{2.8}$$

In training, we set γ=2, α=0.1, β=0.02 and $thre$=0.01.

2.4 Model configuration

The model backbone utilizes the dla 34[12] architect with only the two last layers (the global pool layer and a fully connected layer) removed. This set the stride of the backbone to be 32 times (for example, a 320x320

size image input would receive 10x10 size features) and as such the backbone output up to 512 features with stride 32 from the input image. The features is then put through two branch of transpose convolution layers, one branch output confidence map and the other output part affinity field. The two branch is similar to each others in that each branch contain two transpose convolution layer each with stride 2, thus each branch have a stride of 4. The result confidence map and part affinity field is then pass through an upsampling layer which have a stride of 2. This result in the model having a stride of 4, where an input image of 320x320 would output multiple confidence maps and part affinity fields of 80x80. The number of confidence map is set to be 18 following the 18 keypoints of the modified COCO dataset[5] and the number of part affinity fields is set to be 42 which represent 21 limbs in the dataset.

2.5 Training and testing configuration

The model is trained for 249 epochs, with benchmark of the model every 3 epochs, compare to the two pretrained model provided by trt pose[1] where the model with resnet[9] backbone is also trained for 249 epochs and the other one with densenet[10] backbone is trained for 160 epochs. The training is done with the Adam optimizer[15] with the learning rate change overtime, the learning stay at 0.001 from epoch 1 to epoch 74, from epoch 75 it is change in a 0.0001 until epoch 150 where it is reduce again to 0.00001. It is trained with a batch size of 32 and with image size 256x256, in training the model doesn't use the upsampling layer and smaller image size for faster training time, as such the model output 32x32 confidence maps and part affinity fields. In testing, the model is evaluated with image size 320x320 which result in 80x80 confidence maps and part affinity fields.

2.6 TensorRT model converting process

In order for the model to run optimally on Jetson Nano device, there is a need to convert the model to TensorRT[16]. This process is done using the library torch2trt[17] `https://github.com/NVIDIA-AI-IOT/torch2trt` which

is provided by NVidia themselves. TensorRT is built on CUDA, NVIDIA's parallel programming model, and enable fast performance on many type of device. TensorRT optimizes the network by combining layers and optimizing kernel selection for improved latency, throughput, power efficiency, and memory consumption. If the application specifies, it optimizes the network to run in lower precision, further increasing performance and reducing memory requirements. To be more specific, TensorRT perform 5 different optimization on the model, the optimization are as follow:
- Precision calibration: maximizes throughput by quantize models to INT8 while preserving accuracy. As most deep learning frameworks train neural networks in full 32-bit precision(FP32), this quantization increase its throughput significantly. To quantize full-precision information into INT8 while minimizing accuracy loss, TensorRT must perform a process called calibration to determined how best represent the parameters as 8-bit intergers
- Layer and tensor fusion: optimizes the use of GPU memory and bandwidth by fusing nodes in a kernel. TensorRT do this by parsing the network computational graph and looks for opportunities to perform graph optimizations, the concatenation layer in the model can also be eliminate by preallocating output buffers and writing them in a strided fashion. These optimization do not change the underlying computation in the graph, they instead look to restructure the graph to perform the operations much faster and more efficiently.
- Kernel auto-tuning: selects best data layers and algorithms based on the target GPU platform. As an example, there are several different algorithms for convolution layers, TensorRT will pick the implementation from a library of kernels that delivers the best performance for the target GPU, input data size, filter size, tensor layout, batch size and other parameters. This ensure that the deployed model is performance tuned for the specific deployment platform as well as for specific neural network being deployed.
- Dynamic tensor Memory: minimizes memory footprint and reuses memory for tensors efficiently. This is done by allocating memory for each tensor only for the duration of its usage, avoiding memory allocation overhead for fast and efficient execution
- Multi-Stream execution: scales to multiple input streams, by processing

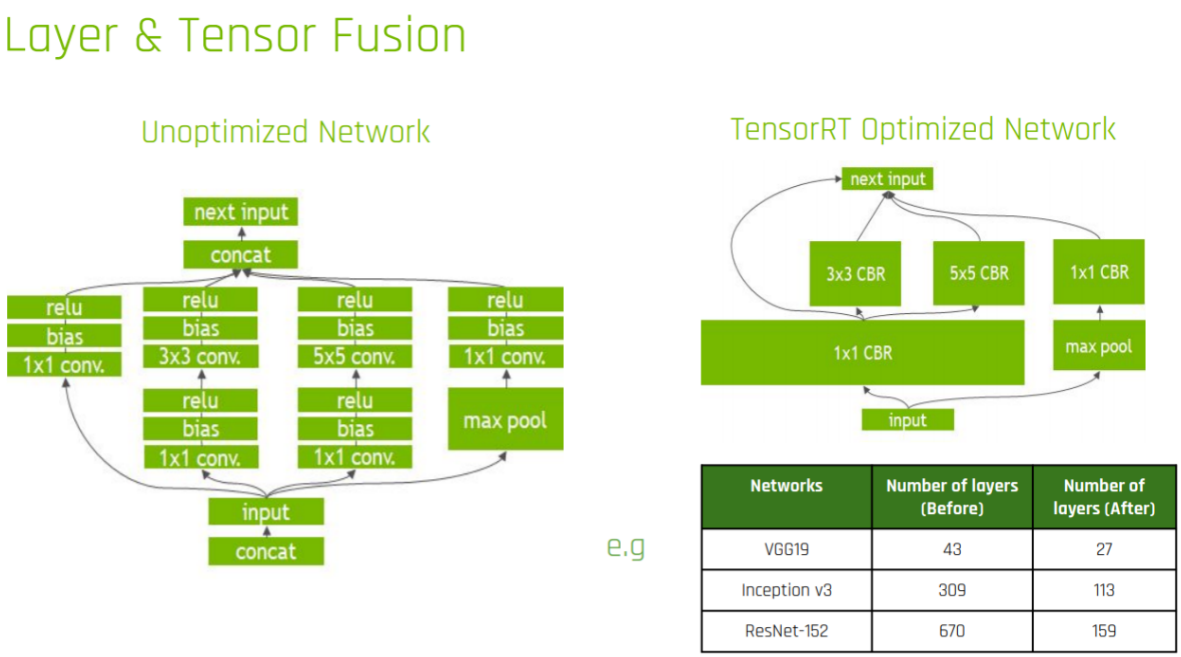

Figure 2.15: Layer and tensor fusion

them in parallel using the same model and weights.

The result TensorRT model can have several times the FPS of its Pytorch counterpart.

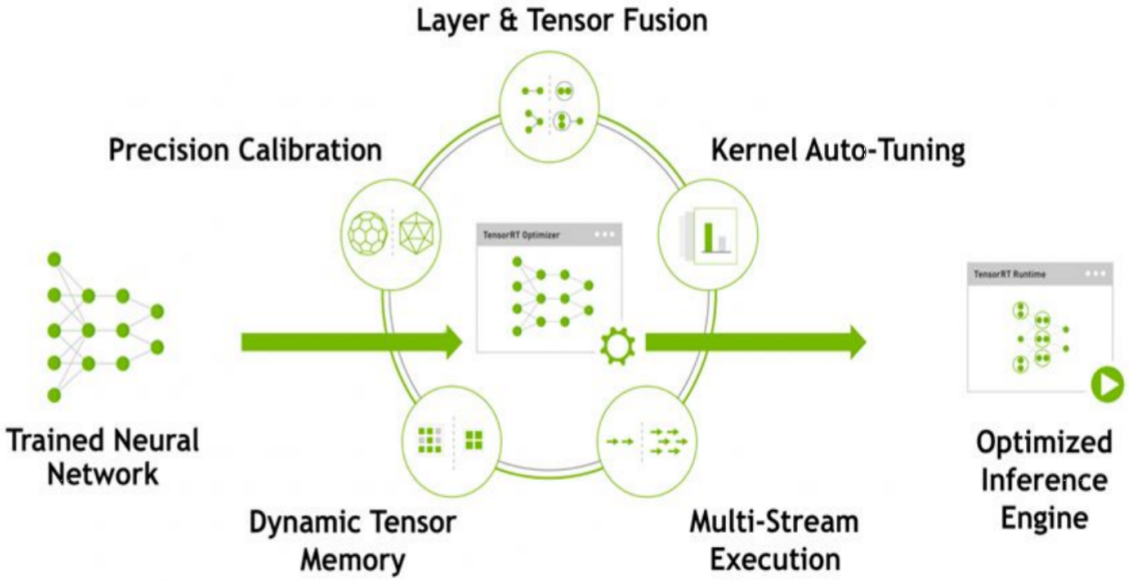

Figure 2.16: TensorRT optimization

Table 2.2: Fps comparison of models running on Jetson Nano

Model	Pytorch	TensorRT
alexnet	46.4	69.9
squeezenet1_0	44	137
squeezenet1_1	76.6	248
resnet18	29.4	90.2
resnet34	15.5	50.7
resnet50	12.4	34.2
resnet101	7.18	19.9
resnet152	4.96	14.1
densenet121	11.5	41.9
densenet169	8.25	33.2
densenet201	6.84	25.4
densenet161	4.71	15.6
vgg11	8.9	18.3
vgg13	6.53	14.7
vgg16	5.09	11.9

Chapter 3

Result and discussion

3.1 Evaluation metrics

Human pose estimation utilizes the same evaluation metrics as object detection, namely average precision (AP) and average recall (AR) and their variants. At the heart of these metrics is a similarity measure between ground truth objects and predicted objects. Object detection use IoU as this similarity measure. For Human pose estimation, COCO dataset[5] propose an analogous similarity measure called object keypoint similarity (OKS).

3.1.1 Object keypoint similarity

In object detection, IoU or Intersection over Union is used to separate good detection from bad detection. It is calculated by taking the overlap area between the predicted area and the ground truth and divided it by the union area of the two, the higher the IoU, the better a prediction is. OKS is the same in that the higher the OKS, the better a prediction is. The formula to calculate OKS is as follow:

$$OKS = \frac{\sum_{i=1}^{N}[exp(\frac{-d_i^2}{2s^2k_i^2})\delta(v_i > 0)]}{\sum_{i=1}^{N}[\delta(v_i > 0)]} \tag{3.1}$$

Where d_i is the Euclidean distances between each corresponding ground truth and detected keypoint and the v_i are the visibility flags of the ground

truth (the detector's predicted v_i are not used). To compute OKS, we pass the d_i through an unnormalized Guassian with standard deviation sk_i, where s is the object scale which is defined as the square root of the object segment area for each ground truth object and k_i is a per-keypoint constant that controls falloff. For each keypoint this yields a keypoint similarity that ranges between 0 and 1. These similarities are averaged over all labeled keypoints (keypoints for which vi¿0). Predicted keypoints that are not labeled (v_i=0) do not affect the OKS. Perfect predictions will have OKS=1 and predictions for which all keypoints are off by more than a few standard deviations sk_i will have OKS 0.
COCO dataset[5] set $k_i = 2\sigma_i$ where σ_i is the per-keypoint standard deviation with respect to object scale s. It is therefore obtained by calculating the standard deviation of all keypoints of one type in the data set, reflecting the standard deviation of the current type of keypoint. σ_i varies substantially for different keypoints: keypoints on a person's body (shoulders, knees, hips, etc.) tend to have a σ value much larger than on a person's head (eyes, nose, ears). For people, the values of σ_i are 0.026, 0.025, 0.035, 0.079, 0.072, 0.062, 0.107, 0.087 and 0.089 for the nose, eyes, ears, shoulders, elbows, wrists, hips, knees and ankles, respectively.

3.2 Metrics

The following 10 metrics are used for characterizing the performance of a keypoint detector on COCO[5] and is what we use to evaluate our model.
- Average Precision (AP):
+ AP at OKS=0.50:0.05:0.95 (Average of the the value of AP at OKS=0.50, OKS=0.05 and OKS=0.95)
+ AP at OKS=0.50
+ AP at OKS=0.75
+ AP for medium objects (Calculate AP at OKS=0.50:0.05:0.95 for objects with $32^2 < area < 96^2$)
+ AP for large objects (Calculate AP at OKS=0.50:0.05:0.95 for objects with $area > 96^2$)
- Average Recall (AR):
+ AR at OKS=0.50:0.05:0.95 (Average of the the value of AP at OKS=0.50,

OKS=0.05 and OKS=0.95)
+ AR at OKS=0.50
+ AR at OKS=0.75
+ AR for medium objects (Calculate AP at OKS=0.50:0.05:0.95 for objects with $32^2 < area < 96^2$)
+ AR for large objects (Calculate AP at OKS=0.50:0.05:0.95 for objects with $area > 96^2$)
- Frame per second(FPS) : The number of frame the NVidia Jetson Nano system can process and output in real time

3.3 Evaluation device configuration

All evaluation are performed with a Nvidia Jetson Nano device, we will evaluate 3 different models, 2 of them are pretrain model using resnet18[9] and densenet121[10] backbone, the final model are the one with dla34[12] backbone we trained by ourselves. The resnet18[9] backbone model have a 224x224 as the input while the densenet121[10] backbone model have a 256x256 as the input, our model, the dla34[12] backbone model is trained with image size 256x256 but is evaluated with input size 320x320. As such the resnet18[9] backbone model will output confidence maps and part affinity fileds with size 56x56, the densenet121[10] backbone model will output with size 64x64 and the dla34[12] backbone model will output with ize 80x80. All models utilized the PyTorch library which allow for any input size image, but in order to fully optimized the model to running on the Jetson Nano device, NVidia provide the library torch2trt[17] which convert pytorch model to TensorRT[16], all 3 model are thus evaluated using their TensorRT[16] version performed by Jetson Nano. All training, evaluation and model code are python language, however, several part of the program use C++ language for the tasks, such task included: finding peaks in confidence maps, refining such peaks, calculating limb score using the part affinity fields, assigning connection using limb scores and the hungarian algorithm and using said connection to form a human pose object. All of those tasks allow many configuration values in the system, those a set as follow:
- The threshold to find peaks within confidence map is set to 0.05

- The threshold to except a limb connection within part affinity fields is set to 0.1
- The size of the window use to confirm a peak in a confidence map is set to 11
- The number of samples takes to give a score to a limb using integral is 7
- The maximum number of peaks taken in 1 confidence map is set to 100
- The maximum number of human pose objects taken is set to 100

This configuration values are only optimal for evaluating resnet18[9] and densenet121[10] backbone model and are changed when evaluating our dla34[12] backbone model, we only make two changes to the two thresholds as follow:

- The threshold to find peaks within confidence map is change to 0.2
- The threshold to except a limb connection within part affinity fields is change to 0.2

3.4 Result

The evaluation results using the above configuration are as follow
As can be seen from the table, our dla34[12] backbone model has higher

Table 3.1: Comparison between 3 models

	resnet18 trt	densenet121 trt	dla34 trt
AP at OKS=0.50:0.05:0.95	0.185	0.245	0.280
AP at OKS=0.50	0.365	0.453	0.524
AP at OKS=0.75	0.164	0.236	0.260
AP for medium objects	0.079	0.136	0.173
AP for large objects	0.329	0.401	0.434
AR at OKS=0.50:0.05:0.95	0.245	0.314	0.347
AR at OKS=0.50	0.410	0.501	0.564
AR at OKS=0.75	0.241	0.320	0.345
AR for medium objects	0.083	0.144	0.181
AR for large objects	0.467	0.546	0.574
FPS	16	9.5	13.5

evaluation result(higher AP and AR values) despite having higher FPS compare to densenet121[10] backbone model, it is slower than resnet18[9] backbone but have much higher evaluation caompare to it. It is because while dla34[12] backbone model have smaller number of parameters compare to densenet121[10] backbone model, the dla[12] architect allow the model to extract features more efficient than densenet[10], this results in both higher FPS value and higher AP and AR values. The resnet18[9] backbone has smaller number of parameters compare to dla34[12], thus it is able to have higher FPS at the cost of its AP and AR.
Using configuration values that not optimal to the model would lead lower evaluation, for example, when testing the dla34[12] backbone model using different threshold values like the standard threshold use by resnet18[9] and densenet121[10] backbone model, the results is as follow

Table 3.2: evaluation results of dla34 backbone model at different confidence map threshold and part affinity field threshold

	using standard threshold	using higher threshold	
confidence map threshold	0.05	0.25	0.3
part affinity field threshold	0.1	0.25	0.3
AP at OKS=0.50:0.05:0.95	0.260	0.268	0.252
AP at OKS=0.50	0.497	0.506	0.487
AP at OKS=0.75	0.235	0.245	0.225
AP for medium objects	0.151	0.163	0.154
AP for large objects	0.420	0.416	0.388
AR at OKS=0.50:0.05:0.95	0.335	0.330	0.305
AR at OKS=0.50	0.547	0.544	0.521
AR at OKS=0.75	0.332	0.322	0.293
AR for medium objects	0.163	0.170	0.157
AR for large objects	0.571	0.549	0.508

As stated above, dla[12] architect extracts feature from the image more efficiently than the two others model and thus increase the peaks confidence scores in its confidence maps and give higher score to the limbs calculated from its part affinity fields. And so both of its threshold value are increase

to 0.2 in order to remove unnecessary keypoint peak and limb, thus improve its performance. Increasing or decreasing one of the two threshold from 0.2 result in a lower performance as increasing it leads to the model missing positive keypoint and decreasing it leads to the model detecting more negative keypoint. Likewise Increasing or decreasing the threshold for evaluating resnet18[9] and densenet121[10] backbone model also lead to a decrease in its performance.

3.5 Sample result images

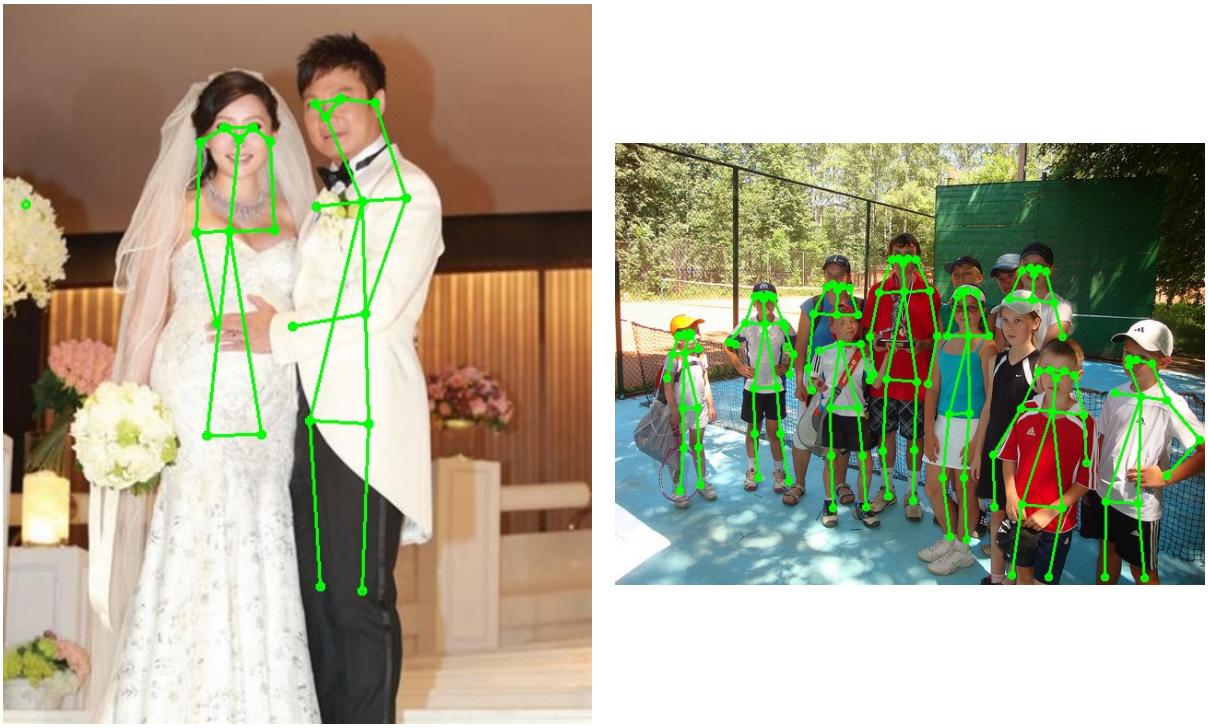

Figure 3.1: resnet18 backbone model predictions

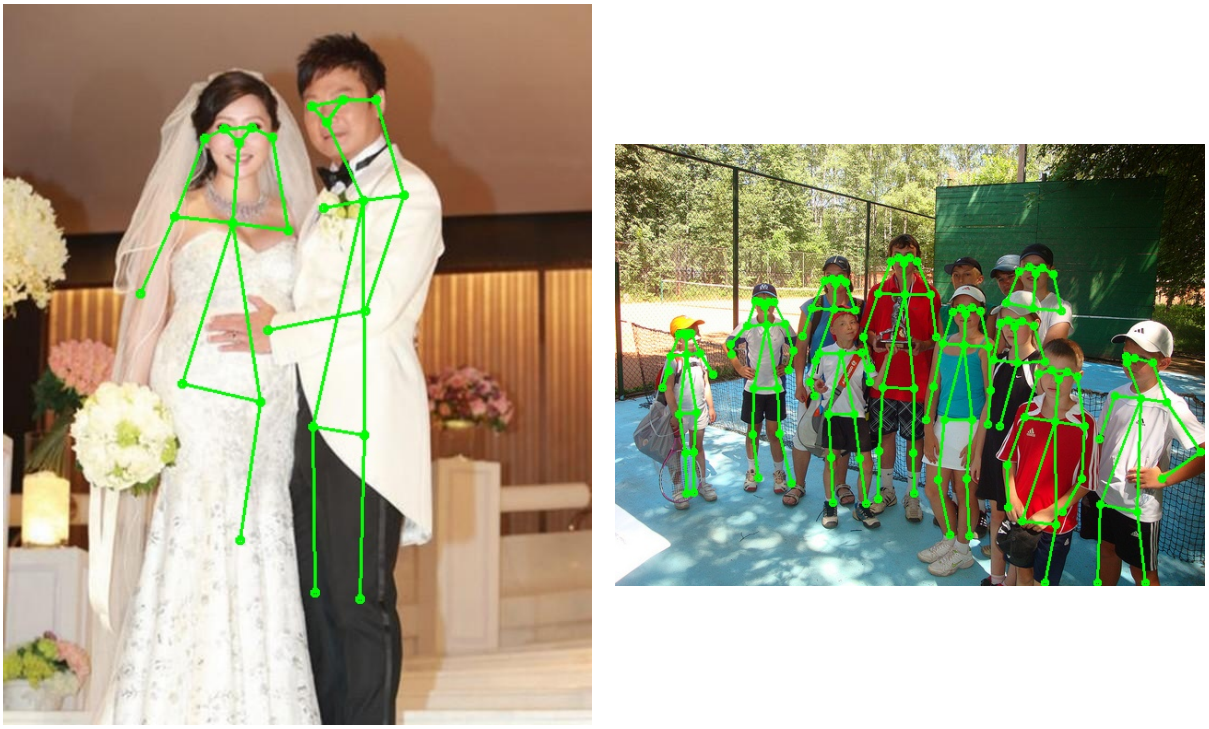

Figure 3.2: densenet121 backbone model predictions

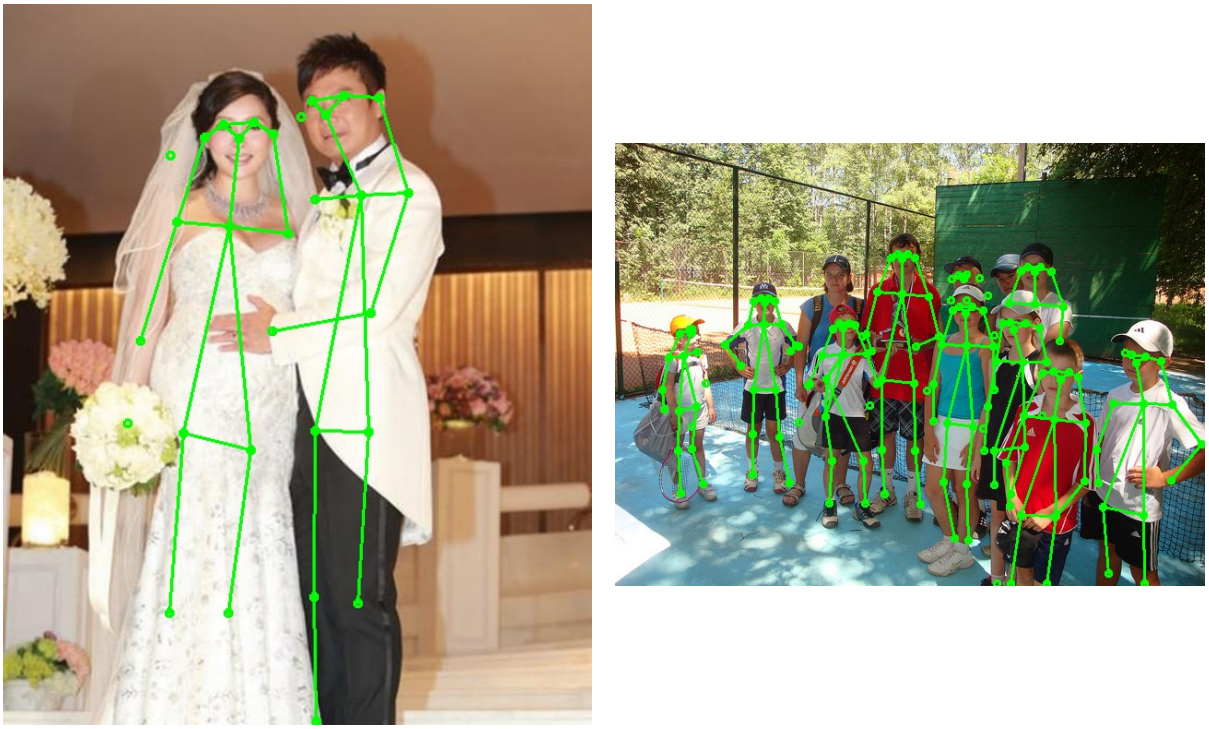

Figure 3.3: dla34 backbone model predictions

Chapter 4

Conclusion and future work

To summarize, this thesis has been able to improve the performance of the human pose estimation model use by trt pose[1] running on the Jetson Nano device by replace its backbone model and utilized a different, improved loss for the training process. The resulting model using the dla34[12] backbone was able to achieved higher AP, higher AR and higher FPS compare to a pretrained model that uses a densenet121[10] backbone. Our model,the dla34[12] backbone model has a lower FPS value compare to the other pretrained model which uses a resnet18[9] backbone but as the architect of such model is much simpler, our model was able to achieved much higher AP and AR.

For future works, the backbone model could be replace with a new model like Efficientnet[18] for more experimentation to improve the performance even further. A new type of loss like Wingloss[19] or a new type of optimizer like the ranger optimizer[20] can be used to further experimenting with the training process. Or a technique like knowledge distillation[21] can used to increase the model speed without compromising on its Precision or Recall.

In conclusion, the project has allow us to improve our researching and coding skills while helping us to familiarize ourselves with this special field of deep learning and computervision knowledge called human pose estimation.

Bibliography

[1] Real-time pose estimation accelerated with nvidia tensorrt, https://github.com/nvidia-ai-iot/trt_pose.

[2] Kaiming He, Georgia Gkioxari, Piotr Dollár, and Ross Girshick. Mask r-cnn. 2017.

[3] Joseph Redmon, Santosh Divvala, Ross Girshick, and Ali Farhadi. You only look once: Unified, real-time object detection. 2015.

[4] Zhe Cao, Gines Hidalgo, Tomas Simon, Shih-En Wei, and Yaser Sheikh. Openpose: Realtime multi-person 2d pose estimation using part affinity fields. 2018.

[5] Coco dataset, https://cocodataset.org/home.

[6] Mpii human pose dataset, http://human-pose.mpi-inf.mpg.de/.

[7] posetrack dataset, https://posetrack.net/.

[8] Ochuman dataset, https://github.com/liruilong940607/ochumanapi.

[9] Kaiming He, Xiangyu Zhang, Shaoqing Ren, and Jian Sun. Deep residual learning for image recognition. 2015.

[10] Gao Huang, Zhuang Liu, Laurens van der Maaten, and Kilian Q. Weinberger. Densely connected convolutional networks. 2016.

[11] Karen Simonyan and Andrew Zisserman. Very deep convolutional networks for large-scale image recognition. 2014.

[12] Fisher Yu, Dequan Wang, Evan Shelhamer, and Trevor Darrell. Deep layer aggregation. 2017.

[13] Jia Li, Wen Su, and Zengfu Wang. Simple pose: Rethinking and improving a bottom-up approach for multi-person pose estimation. 2019.

[14] Tsung-Yi Lin, Priya Goyal, Ross Girshick, Kaiming He, and Piotr Dollár. Focal loss for dense object detection. 2017.

[15] Diederik P. Kingmaand and Jimmy Ba. Adam: A method for stochastic optimization. 2014.

[16] Nvidia tensorrt, https://developer.nvidia.com/tensorrt.

[17] An easy to use pytorch to tensorrt converter, https://github.com/nvidia-ai-iot/torch2trt.

[18] Mingxing Tan and Quoc V. Le. Efficientnet: Rethinking model scaling for convolutional neural networks. 2019.

[19] Zhen-Hua Feng, Josef Kittler, Muhammad Awais, Patrik Huber, and Xiao-Jun Wu. Wing loss for robust facial landmark localisation with convolutional neural networks. 2017.

[20] Zitao Chen, Guanpeng Li, and Karthik Pattabiraman. A low-cost fault corrector for deep neural networks through range restriction. 2020.

[21] Geoffrey Hinton, Oriol Vinyals, and Jeff Dean. Distilling the knowledge in a neural network. 2015.