Triplet Loss for Person Re-Identification on RTX2080 and EdgeTPU

Cüneyt Erem

Universität Bonn
s6cuerem@uni-bonn.de

Abstract. For many recent years, computer vision by applying deep neural networks has been advanced with different aspects that person re-identification is one of this area. This kind of the systems have been using in different areas especially for the surveillance and security fields. Before the recent years, triplet loss function has been seen that it was not effective for the re-identification. In some researches, it was found that triplet loss is very useful and has competitive results in this area and it can be applied for different data-set samples. Our work is to examine whether it is efficient on the standard GPU RTX 2080 and also for the Google Coral edgeTPU devices.

1 Introduction

Many different researches have been published on the person re-identification system by using deep learning methods. Before the paper 'In Defence of the Triplet Loss for Person Re-identification' has been published, common view for the performance of using the triplet loss on the person re-identification was not good enough because calculating losses by the training and embedding, they have some problems [2]. But this paper showed that actually it can compute the embedding by calculating Euclidean distances and gives good results that can compete with known state-of-the-art approaches [1]. Another study from the GitHub repository showed that they can successfully re-implement the code from the paper and achieve the same results as paper mentioned even if it is mentioned that in some parts, there are some version differences that causes different scores on the results [3]. In this study, we examined that applying the triplet loss can yield the similar results as paper and repository mentioned and we applied their methods on the standard GPU RTX 2080 and Google Coral edgeTPU device, then compare their performance results.

Hardware and Software Architecture

Coral EdgeTPU is a USB accelerator that provides co-processor with high speed machine learning inference that performs 4 trillion operations per second such as it can execute mobilenet v2 at most 400 fps. It is compatible with tensorflow-lite [5].

Fig. 1. Source [5]

Fig. 2. Source [6]

Nvidia RTX 2080 GPU is one of the powerful GPU that can execute machine learning models with very fast duration and compatible with tensorflow-gpu [6].

Used software components are Ubuntu 18.04, Anaconda 3, Python 3.7.9, cuDNN major:7/minor:6/patch level:0, Tensorflow-gpu 1.15.0. Installed software should be compatible with each other, otherwise it gives error. This project is mainly based on tensorflow-gpu 1.15.0 and other compatible components that it gives us powerful computer vision and deep learning resources, for the edgeTPU we also use edgeTPU components and tensorflow converter components which will be discussed detail in the later parts.

2 Triplet Loss

In the triplet loss paper, they proposed that using triplet loss with a specific and smart manners of the CNN, they can create competitive results with the previous known methods for re-identification of the person [2]. When the data is transmitted to the CNN with three CNN components with shared weights, they are computed and send to the triplet loss state, then they will be recalculated with loss propagation or converged to the ending state. These three components are 'anchor' that we select an image from the data-set, 'positive' that belongs to the anchor class and 'negative' that does not belong to this class. The triplet loss function is calculated by using distance metric with calculating squared euclidean distance, however in some situations, it reduces the performance in the initial steps so that sometimes non-squared euclidean distance is also considered [1].

In triplet loss function, the data-set grows by n whereas number of total triplets of anchor, positive and negative values increase by n^3. This situation seems inefficient however instead of calculating the whole data-set, only specific ones with

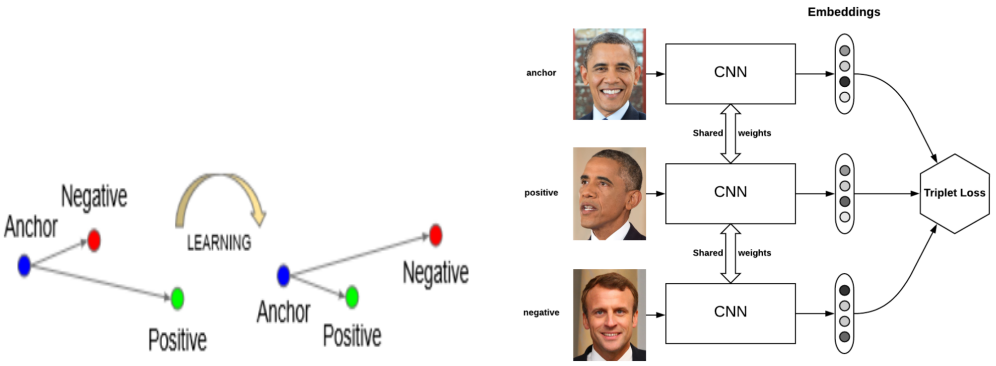


Fig. 3. Source [2]

Fig. 4. Source [7]

three different loss strategies will be taken into account. In classical approach, when we choose set of B triplets. Their images are converged into 3B embedding that results in B term for the loss. Yet, it there are $6B^2 - 4B$ possible combinations. In the paper, authors have proposed another way to define the batches that they computer triplet losses as batch hard, batch all and lifted embedding loss [1].

Batch Hard

1. Choose P classes (person identities)
2. Choose K images per class (person)

This will result in PK images per batch. Now for every anchor **'a'** in the PK images, we will find the **hardest positive** and **hardest negative** sample for that sample 'a' across the batch.

$$\mathcal{L}_{\mathrm{BH}}(\theta; X) = \overbrace{\sum_{i=1}^{P}\sum_{a=1}^{K}}^{\text{all anchors}} \Big[m + \overbrace{\max_{p=1\ldots K} D\left(f_\theta(x_a^i), f_\theta(x_p^i)\right)}^{\text{hardest positive}} - \underbrace{\min_{\substack{j=1\ldots P \\ n=1\ldots K \\ j\neq i}} D\left(f_\theta(x_a^i), f_\theta(x_n^j)\right)}_{\text{hardest negative}} \Big]_+, \quad (5)$$

Batch Hard We randomly select P classes (person id) and K images for each person in PK image batch that for each anchor, we compute the triplet loss function with hardest positive that have maximum distance for the same person and hardest negative samples that have minimum distance to another person

class. Margin is for deciding for maximum distance for positive and minimum distance for negative sample [1].

Batch All

This is an extension of the above strategy to all possible combinations of triplets from a batch of **PK** images: PK*(PK-K)*(K-1) triplets will contribute to the loss term.

$$\mathcal{L}_{\mathrm{BA}}(\theta; X) = \overbrace{\sum_{i=1}^{P}\sum_{a=1}^{K}}^{\text{all anchors}} \overbrace{\sum_{\substack{p=1 \\ p\neq a}}^{K}}^{\text{all pos.}} \overbrace{\sum_{j=1}^{P}\sum_{\substack{n=1 \\ j\neq i}}^{K}}^{\text{all negatives}} \left[m + d^{i,a,p}_{j,a,n}\right]_+, \quad (6)$$

$$d^{i,a,p}_{j,a,n} = D\left(f_\theta(x^i_a), f_\theta(x^i_p)\right) - D\left(f_\theta(x^i_a), f_\theta(x^j_n)\right).$$

Batch All In total, we have PK * (PK – K) * (K – 1) triplets for all batches that we go through all PK images for each sample. For each anchor, we compute the distance of all positives and all negatives instead of calculating specific distances. Batch hard and Batch all are commonly used in deep learning methods like max-pooling and RELU non-linearity [1].

Lifted Embedding Loss

- It is similar to **batch all** but the only difference is the logarithmic of summation of the distances. It is based on the lifted structured loss mentioned in this paper.

$$\mathcal{L}_{\mathrm{LG}}(\theta; X) = \overbrace{\sum_{i=1}^{P}\sum_{a=1}^{K}}^{\text{all anchors}} \Bigg[\log \overbrace{\sum_{\substack{p=1 \\ p\neq a}}^{K} e^{D\left(f_\theta(x^i_a), f_\theta(x^i_p)\right)}}^{\text{all positives}} + \log \underbrace{\sum_{\substack{j=1 \\ j\neq i}}^{P}\sum_{n=1}^{K} e^{m - D\left(f_\theta(x^i_a), f_\theta(x^j_n)\right)}}_{\text{all negatives}} \Bigg]_+ . \quad (7)$$

Lifted Embedding Loss For each anchor, it also calculates logarithmic of the all positives and all negatives. It is optimized by considering traditional 3B batches. Normally, distance metric is calculated by squared euclidean distance

$D(f_\theta(x_i), f_\theta(x_j)) = \|f_\theta(x_i) - f_\theta(x_j)\|_2^2$. But for the initial experiment performances, it has some side effects since there is no need for computation so that using non-squared distance is more efficient for the loss formula [1].

In the paper and repository, they have shown that by applying different parameters and changing network, they can achieve good and competitive results for the person re-identification. Trinet using resnet-50 architecture and Lunet using resnet-v2 gives good results especially for the embedding [1].

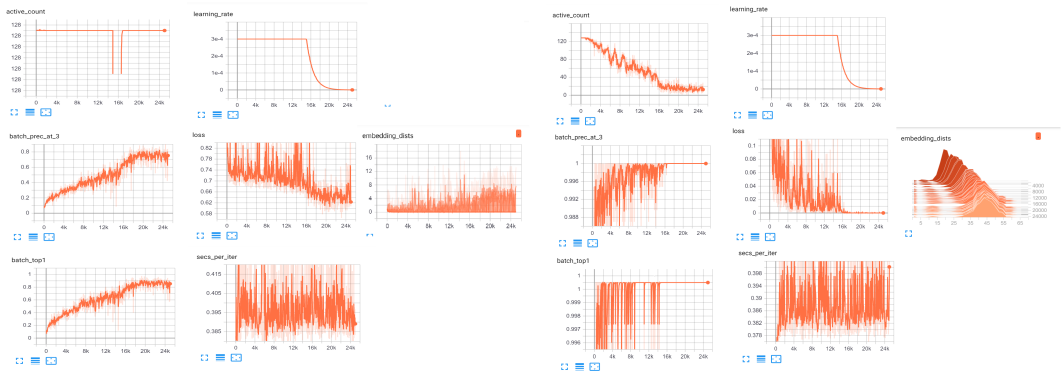
For the market data-set MQ, mAP (mean average precision) gets 76.42, rank-1 gets 90.53 and rank-5 gets 96.29 which shows that using triplet loss on market data-set, it can get better result than other loss functions just because changing the parameters accordingly [1]. For that reason, we examined this loss function and apply it as repository mentioned, then modified it on edgeTPU device.

3 Experiment

The experiment is based on three parts; data-set, training and computing embedding.

Data-set: We used 2 different data-sets for the experiment. First one is Market-1501 in which contains 12937 training images for 1501 people, 19733 test images and 3368 query images, and custom data-set consists of 9 test images and 9 query images for three different people.

Training: Training is done with same frameworks as paper and repository mentioned []. We used Adam optimizer, learning rare is 3e-4 and number of iterations is 25000. For the network architecture, initial checkpoints can be random which will discover from zero and yield less results and pre-defined that increases the accuracy by resnet v1 50 which has inception pre-processing and has input image size 299, mobilenet v1 1 224 which is efficient CNN for mobile networks compared to others.



Training graphs give results for random checkpoint and resnet v1 50 respectively. We can say that whenever random checkpoint is initialized, loss rate is

very high when we compare the paper results. However, initial checkpoint resnet v1 50 gives significantly low loss value and training becomes much better which is close to the paper results.

Computing Embedding: It computes the embedding between set of pictures for test and query data-sets. Batch size has $1/8 * H * W$ with $H = 256$ and $W = 128$ on Market-1501 data-set and batch size for the RTX 2080 is 32 whereas for edgeTPU is 1 to eliminate out of memory error. For the RTX 2080, after embedding is calculated, it is output into HDF5 file. For the edgeTPU device, there is a conversion steps that we need to take tensorflow model and make train and quantize then export it. After creating frozen graph, we apply post quantization, then send it to the tensorflow lite. After compiling it, edgeTPU model is created and ready to deploy on edgeTPU hardware [4]. So during this step, it is exported to tflite file, then it is output into HDF5 which is a bit differently applied than standard GPU implementation.

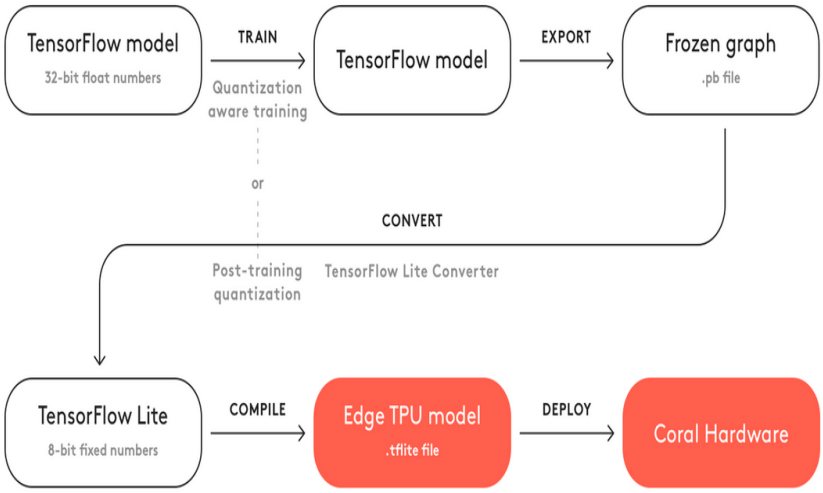


Fig. 5. Source [4]

4 Evaluation

By using h5 files, we can compute metric euclidean distance of linear norm between test and query images. To see accuracy level between data-sets, we compute mAP (mean average precision) and rank 1 to 10 levels. For the custom data-set example. We have three people with ID 1491. 1497 and 1501.

For both test data-set and query data-set, we have 9 images per each and two of people have similar look such as yellow t-shirt and shirt, 1 person has different look than others. Whenever we define distances of L-2 norm after CNN with triplet loss is done, distances are; $Person1491 - 1491 : 13.404276$, $Person1491 - 1497 : 27.014582$, $Person1491 - 1501 : 57.73232$. So, whenever we compare two images that consists of same person anchor and positive sample, it gives smaller distance value, if it is anchor and negative sample but similar look, then distance increases. If anchor and negative farthest sample are compared, then it gives very high value. It shows that triplet loss successfully gives shared weights correctly even if similar people exist, and it can re-identify them.

Evaluation	rtx2080						
dataset	market1501				custom_dataset		
checkpoint	default_checkpoint	resnet_v1_50	Market1501_25000	mobilenet_v1_1_224	default_checkpoint	resnet_v1_50	Market1501_25000
mAP	2.00%	66.53%	66.88%	64.80%	100.00%	100.00%	100.00%
top-1	4.39%	82.90%	83.40%	81.83%	100.00%	100.00%	100.00%
top-2	7.01%	88.39%	88.51%	87.65%	100.00%	100.00%	100.00%
top-5	11.97%	93.59%	93.76%	92.34%	100.00%	100.00%	100.00%
top-10	18.71%	95.78%	96.02%	94.92%	100.00%	100.00%	100.00%
	edgetpu						
dataset	market1501				custom_dataset		
checkpoint	default_checkpoint	resnet_v1_50	Market1501_25000	mobilenet_v1_1_224	default_checkpoint	resnet_v1_50	Market1501_25000
mAP	0.22%	66.15%	63.21%	64.00%	100.00%	100.00%	100.00%
top-1	0.24%	82.36%	79.07%	81.15%	100.00%	100.00%	100.00%
top-2	0.45%	88.72%	85.90%	87.14%	100.00%	100.00%	100.00%
top-5	0.80%	93.53%	91.27%	91.78%	100.00%	100.00%	100.00%
top-10	1.43%	95.72%	94.66%	94.86%	100.00%	100.00%	100.00%

We evaluate the models for the Market-1501 data-set and custom data-set for random checkpoint, resnet v1 50, market1501 25000 provided by author and lastly mobilenet v1 1 224. In figure X, Trinet, most successful result, with data-set market-1501 gets 76.42, rank-1 gets 90.53 and rank-5 gets 96.29 accuracy results respectively as mentioned previous part [1]. For the random checkpoint it gets percentages of 2.00, 4.39, 7.01, 11.97, 18.71. For the edgeTPU it gets much

lower than these values as 0.22, 0.24, 0.45, 0.80 respectively. However these values are not exactly same for each iteration because it is based on random values. So it can give higher results for initialize again for RTX 2080, it gives 12.60, 24.32, 34.56 and 49.55. Still we can assume that edgeTPU is not good enough for random checkpoint. resnet v1 50 gives the highest results for both RTX 2080 and edgeTPU, where edgeTPU has slightly lower values. For the market-25000 and mobilenet v1 1 224, they still gives very good results as paper and repository mentioned, where RTX 2080 is slightly worse than paper mentioned and edgeTPU is slightly worse than RTX 2080, but overall difference is only between 0.1-2.0 on average. For the custom data-set, each RTX 2080 and edgeTPU gives 100 accuracy result for all levels because training consists on thousands of images whereas test and query data-sets consist of only 9 images per each.

runtime	rtx2080	edgetpu
market1501		
training	~180 min	~180 min
embedding	~3 min	~22 hours
evaluation	~1 min	~1 min

Training part and evaluation part are almost same for RTX 2080 and edgeTPU. Main difference in implementation was embedding parts. Whenever we run the code, RTX embedding lasts only 3 minutes whereas edgeTPU lasts 22 hours for standard speed with 250 MHz, 11 hours for maximum speed with 500 MHz but this yields the device becoming warm and it is not recommended by us for such long hours.

RTX 2080 performance is not exactly as good as paper and repository mentioned, and edgeTPU performance is not as good as RTX 2080, there are several reasons for this issue. Tensorflow is sometimes inconsistent and have slow performance whereas scikit-learn version causes different scores than normal output [3]. Person re-identification process requires high memory and GPU usage, to eliminate out of memory error, we reduced the batch sizes as 32 for RTX 2080 and 1 for edgeTPU so that this results in lowering the performances. EdgeTPU is not powerful exactly as RTX 2080 and quantization and converting steps also can slow down the performance for the edgeTPU [6].

5 Conclusion

In conclusion, we examined whether paper and repository results can be implemented and give the similar results on Nvidia RTX 2080 GPU and EdgeTPU devices in this study. RTX 2080 is very powerful and it can nearly gives the similar results as previous works however memory issues can diminish the performance. For the edgeTPU, even if it is not good enough for the random checkpoint, for

many pre-defined checkpoints, it is also powerful enough to get similar results as RTX 2080. It is important because for the Nvidia, many different software products need to be installed and compatible each other. However, edgeTPU only needs to be inserted into computer and make changes for the tensorflow. Only main problem is run-time duration that embedding step takes too long to be executed in which 3 minutes versus 22 hours. If there is not time issues need to be considered, then it is useful, otherwise it is a big problem. Yet, edgeTPU has pretty impressive accuracy results for the data-sets.

6 Future work:

In the future, quantization steps and converting parts compute the CNN and triplet loss methods successfully [4], however, run-time duration is a big problem, if these steps can be optimized enough and it can calculate the process fast enough, then we can say that edgeTPU is very good device to apply identification problems or other computer vision studies to be executed. Because whenever applying many Nvidia software products, on needs to be careful to install correct ones, otherwise there will be problems occur. Also not everyone can buy powerful GPU as simple. However, USB edgeTPU device is easy to use and has accessible price to implement such hard GPU-required studies [5]. Elimination of the memory errors also can increase the performance slightly so that it can give almost best results as we wanted.

References

1. A. Hermans, L.Beyer, B. Leibe: In defense of triplet loss for Person Re-Identification. In CVPR (2017)
2. F. Schroff, D. Kalenichenko, J. Philbin. FaceNet: AUnified Embedding for Face Recognition and Clustering. InCVPR (2015)
3. Triplet-based Person Re-Identification, github.com/VisualComputingInstitute/triplet-reid. Last accessed Oct 2020
4. TensorFlow models on the Edge TPU, https://coral.ai/docs/edgetpu/models-intro/. Last accessed Oct 2020
5. USB Accelerator, https://coral.ai/products/accelerator/. Last accessed Oct 2020
6. Nvidia RTX 2080 GPU, https://www.nvidia.com/tr-tr/geforce/graphics-cards/rtx-2080/. Last accessed Oct 2020
7. Olivier Moindrot blog, omoindrot.github.io/triplet-loss triplet-mining. Last accessed Oct 2020