Efficient People Re-identification on Drones
Project Report, ELEC 515

Boning Li

Dec. 15, 2018

1 Introduction

Person re-identification (Figure 1) is a computer vision problem where one matches two images/videos of the same person taken by different cameras at different time and locations. It is typically accomplished by probing a query person against a set of gallery persons and estimating for pairwise similarity. The challenge is that query and gallery images may come from different domains (e.g. environment, luminance, resolution, etc.). In addition, from an applicational point of view, achieving such tasks on resource-limited drones in real-time also raises a great demand of algorithm complexity reduction.

Figure 1: Person re-ID problem defined.

The pipeline of this project can be treated as two parallel threads, one being the regular person re-ID problem, and the other being algorithm reduction and model compression. In this project, we first trained deep learning networks on public re-ID dataset. Upon successfully reaching the state-of-the-art re-ID accuracy, we compressed the networks to enable on-drone inference. Finally, the reduced networks were finetuned to restore their accuracy.

The network architectures we studied included ResNet(50) [1] and DenseNet(121) [2] based models. We ran experiments on two image-based datasets, namely the Market-1501 [3] dataset and the Duke-MTMC ReID [4] dataset. Algorithm reduction methods were channel pruning and weight quantization. The on-drone platform was a Raspberry Pi 3 model [5].

2 Literature Review

In this section we will review established work on person re-ID and model compression.

2.1 Person Re-identification

As a brief review of milestones in history, person re-ID was firstly proposed as a multi-camera tracking problem by Huang and Russell in 1997 when Bayesian posteriors on feature vectors were used [6]. In 2005, for the first time, it was defined as the "person re-identification" problem [7], although the methodology essentially unchanged. In 2006, person re-id became an image-based problem [8], and from that point, researchers started to view it as an independent computer vision task. In 2010, video data came into this field. It was proved that more frames could improve, but would eventually saturate the re-id performance [9]. In 2014, ImageNet and deep learning exploded in the community. The first re-ID system based on deep learning emerged [10]. From then on, most models become end-to-end, meaning that hand-engineered feature extraction is no longer necessary.

Siamese network was the most popular architecture for this specific task, because it allows for image pairs or triplets as the input. The bottleneck of the end-to-end networks was the lack of training data. Typically, one person only has two images available, one for query and one in the gallery.

2.2 Model Compression

Convolutional neural networks are both computationally and memory intensive. Thus, it is usually difficult to deploy them on embedded systems with limited hardware resources, such as mobile phones and drones. Various methods have been studied to compress the model, resulting in boost of the storage and speed.

Pruning refers to removing redundant surpasses and neurons in a network, proposed by Yann LeCunn in 1990 [11], and has regained popularity in recent years. Associated with weight sharing and Hoffman encoding, the scheme was proved to compress AlexNet and RNN by over 90% while the prediction performance was unharmed [12]. In fact, the concept quantization has been widely adapted into the model compression problem. Zhu *et. al.* designed a ternary net scheme that performed reasonably well with only three levels of weights [13]. Reinforcement learning is also rising in the field, for merits of automatically finding the optimal solution, as discussed in a recent paper [14]. However, pruning individual weights would result in irregular kernels, thus requiring specialized hardware accelerators, over which pruning by channel had a supreme advantage. He *et al* achieved 5x speed-up along with only 0.3% increase of error on VGG-16 [15]. Zhuang *et al* reported performance increase on ResNet-50 with 30% reduction of channels [16].

Nonetheless, the improvement in size and speed solely by pruning was usually not enough for real-time inference on embedded platforms. To achieve even higher reduction rate, fixed-point design was a powerful option. Anwar *et al* designed an fixedpoint optimization algorithm that dynamically estimates the quantization step size [17]. Besides, ternary and even binary networks were also proposed in [18] [19] [13]. These methods can drastically reduce the size of models and the complexity of computation, but the implementation of which usually calls for specialized software and hardware to fully utilize their strengths.

3 Methods

3.1 Re-ID Algorithm

In this project, our collaborators recommended a feature encoding - similarity scoring architecture for the person re-ID problem. The system consists of two parts, the first being a deep convolutional neural network as the feature extractor, and the second being an algorithm that matches the closest distance. Figure 2 illustrates the two stages of person re-ID in practice.

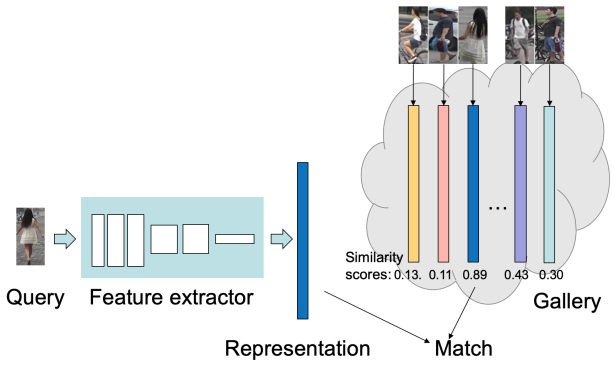


Figure 2: Image-based re-ID algorithm pipeline.

The feature extractor was taken from a trained ResNet or DenseNet, and the original model was trained in an object classification manner – to classify people's identity in the training set. Next, the fully connected layers was abandoned. The input to the fully connected classifier was redirected to be compared with the features of the query image. A score was computed for every identity in the gallery. Therefore, we could find the most likely identity as of the highest score.

In this system, the feature extractor was the most computation and storage intensive. Therefore, the goal was to cut down as much cost as possible in the feature extractor, to accomplish which, two methods were considered and implemented.

3.2 Reduction and Compression

Many methods have been proposed for reducing computation complexity. Here, we focus on two commonly used techniques, namely the channel pruning and fixed point quantization. This section will go through the theory of both methods.

3.2.1 Channel Pruning

Model Pruning refers to removing unimportant branches of the network, resulting in reduction of parameters and computations. Among all pruning techniques, channel pruning has an advantage over the others in terms of efficiency and portability. Individual weigh pruning results in irregular convolution kernels, and thus requiring specialized hardware or dataflow to accelerate or compress. On the contrary, as shown in Figure 3, channel pruning operates on entire channels, the output of which is still regular. Thus, no special manipulation is needed on the hardware side.

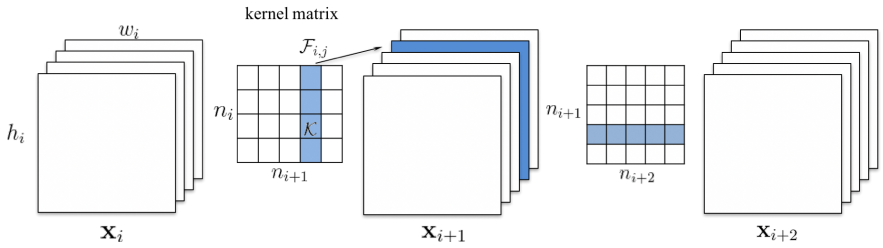


Figure 3: Channel pruning diagram.

In order to find out which channels can be removed with minimal performance loss, we apply the batch normalization parameters as the ranking of channel significance. The transform of batch normalization is shown in Figure 4. It projects a minibatch of activation to a less biased space with smaller internal covariate shift, and thus its weights can reflect the importance of each channel.

Input: Values of x over a mini-batch: $\mathcal{B} = \{x_{1...m}\}$;
Parameters to be learned: γ, β
Output: $\{y_i = \mathrm{BN}_{\gamma,\beta}(x_i)\}$

$$\mu_{\mathcal{B}} \leftarrow \frac{1}{m}\sum_{i=1}^{m} x_i \qquad \text{// mini-batch mean}$$

$$\sigma^2_{\mathcal{B}} \leftarrow \frac{1}{m}\sum_{i=1}^{m}(x_i - \mu_{\mathcal{B}})^2 \qquad \text{// mini-batch variance}$$

$$\widehat{x}_i \leftarrow \frac{x_i - \mu_{\mathcal{B}}}{\sqrt{\sigma^2_{\mathcal{B}} + \epsilon}} \qquad \text{// normalize}$$

$$y_i \leftarrow \gamma\widehat{x}_i + \beta \equiv \mathrm{BN}_{\gamma,\beta}(x_i) \qquad \text{// scale and shift}$$

Figure 4: Batch normalizaing transform on activation x_i. [20]

3.2.2 Fixed-Point Quantization

The representation of parameters and activations in PyTorch models is 32-bit floating-point number. As shown in Figure 5, the 32-bit floating-point representation consists of three parts, 1 bit for sign, 8 bits for exponent, and 23 bits for mantissa. The value of the number is given by

$$(-1)^S * (1 + M) * 2^{(E-bias)}, \text{ where } bias = 127$$

Figure 5: IEEE 754 floating-point data representation specification.

The represented values are unequally spaced between extremes (min $\approx 1.2 * 10^{-38}$, max $\approx 3.4 * 10^{38}$), such that the gap between adjacent numbers is much smaller for small values and much larger for large numbers. This notation employs 8 bits to become the exponent, which gives the extensive dynamic range. As the trade-off, it loses the 8 bits' resolution for all values.

Values with fixed-point representation (Figure 6), on the contrary, are equally spaced across the whole range, whose value is given by

$$(-1)^S * I.F, \text{ where } I.F = \sum_{-n}^{m-1} b_i 2^i$$

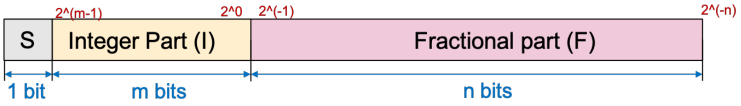


Figure 6: Fixed-point data specification (with scale).

Hence, we can clearly see that the dynamic range and precision of floating-point data come with a price of expensive implementation. On resource constrained platforms where energy becomes the bottleneck, we would rather sacrifice some precision in parameters representation to accelerate the computation and save energy.

In this project, we perform quantization via a self-defined layer named `quant`. The `fig:quant` layer follows the convolution, batch normalization and linear layers, takes their floating values as the input, and output a quantized version. The operations in `quant` layer are shown in Figure 7 (with min-max quantification method, more details in Section 4). We kept it in such way because the original models were in PyTorch which does not support low precision networks.

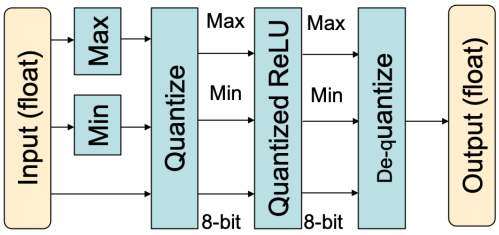


Figure 7: Quantification layer with minmax.

3.2.3 Evaluation

We train, validate and test on separate sets of data, and report the re-ID accuracy on the test set given by the model with the highest validation accuracy. To measure the success of this project, we will show in the following sections that higher targeted sparsity would result in smaller models and faster inference yet cause more difficulty in maintaining accuracy. More specifically, we try to identify the threshold that balances between accuracy and sparsity. We will also show the limits of untrained quantification. Discussion on the challenges and progresses will also be included. More visualization and discussion of the results are available in Sections **??** and 6.

4 Experiments

The experiments were implemented in Python on the deep learning research platform PyTorch. The training and compression was completed on multiple GEFORCE GTX 1080 Ti GPU's. The code was adapted from the

code provided by collaborators at `https://github.com/yyvettey/DroneReId`. Pruning and quantization functions were added on the base. For conceptual explanations and verification of ideations, we also used the VGG architecture and the CIFAR-10 Dataset for short-term tests.

4.1 Dataset

The Market-1501 dataset, whose samples are shown in Figure 8, was collected in front of a supermarket in Tsinghua University. A total of six cameras were used. Overall, this dataset contains 32,668 annotated bounding boxes of 1,501 identities.

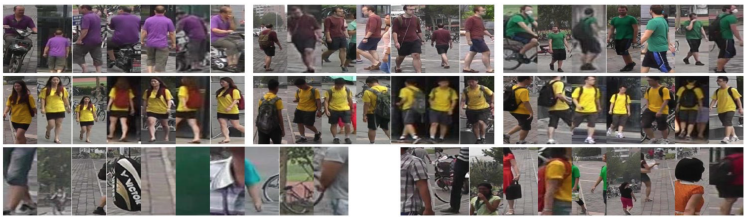


Figure 8: Market-1501 dataset sample bounding boxes.

In a similar format, we also used the DukeMTMC-reID dataset. DukeMTMC aims to accelerate advances in multi-target multi-camera tracking with the re-ID mission as a subset. It provides in total 36,411 bounding boxes with 1,404 unique identities.

Even though these are well developed datasets open to the public, they may not fit the ultimate goal of the extension of this project. We aim for using drone-carried cameras as the probing device, meaning that the shadow, angle and context will be vastly different from the ones captured by static cameras. While a team of senior undergraduate students from Rice University have been working on collecting data in our manner and labeling the collected video, we can start testing the algorithms on public datasets. Eventually, we could leverage transfer learning to train our own models faster and better.

4.2 Channel Pruning

On implementing channel pruning, the function should take the original model and targeted sparsity, and output the pruned model. We present the channel pruning by sparsity function in Algorithm 1. The implementation details varied from architecture to architecture. Generally speaking, the less internal links are there in the network, the more channels we could actually delete. For example, the pruning completeness of a simple VGG-style model can reach 100%, meaning that all channels to be removed can be removed. However, in the ResNet architecture, due to the addition across layers in the residual blocks, the last convolutional layer's output must be of the same shape as the first layer. This can be satisfied by adding a `selection` layer following every batch normalization layer whose shape needs to be maintained. The `selection` layer basically chooses channels from its preceding layer and thus reducing the dimension.

Algorithm 1 Channel-pruning algorithm

function PRUNE BY SPARSITY(original model, sparsity)
    a) Get pruning scheme:
    for Each layer l_i in the original model **do**
        Compute the number of channels n_i to prune by rounding the product of sparsity and the number of channels in l_i;
        Rank the channels and get a list of indices of top n_i channels;
        Append layer index l_i and channel index list $\{c_i\}$ to the scheme;
    b) Execute pruning:
    for Each layer l_i in the original model **do**
        Remove channels $\{c_i\}$ from l_i;
        Change the shape of following layer(s) if necessary;
    c) Restore accuracy:
    Finetune the pruned model.

The code for the `selection` layers is listed below. It filters the input data with a binary index mask.

```python
class channel_selection(nn.Module):
    def __init__(self, num_channels):
        super().__init__()
        self.indexes = nn.Parameter(torch.ones(num_channels))

    def forward(self, input_tensor):
        selected_index = np.squeeze(np.argwhere(self.indexes.data.cpu().numpy()))
        if selected_index.size == 1:
            selected_index = np.resize(selected_index, (1,))
        output = input_tensor[:, selected_index, :, :]
        return output
```

Listing 1: Channel selection

4.3 Quantization

We tried different quantization methods including linear, log, and min-max algorithms. In this section we implement them with PyTorch code.

With linear quantization, the entire data range is divided into L equal intervals of length Q

$$Q = (f_{max} - f_{min})/L$$

. Given a value f, first find its index

$$Q_i(f) = \lfloor (f - f_{min})/Q \rfloor$$

The quantization value is given by

$$Q(f) = Q_i(f)Q + Q/2 + f_{min}$$

The implementation is listed in the code block below.

```python
def linear_quantize(input, sf, bits):
    if bits == 1:
        return torch.sign(input) - 1
    delta = math.pow(2.0, -sf)
    bound = math.pow(2.0, bits-1)
    min_val = -bound
    max_val = bound - 1
    rounded = torch.floor(input / delta + 0.5)
    v = torch.clamp(rounded, min_val, max_val) * delta
    return v
```

Listing 2: Linear quantization

With linear quantization, the signal to noise ratio is large for high level but small for low level signals. Alternatively, we could first project the input to the log space, and complete quantization using either min-max or linear quantization methods, and then transform back to linear space. Operating in the well-scaled log space could potentially result in fine-grained approximation.

5 Results

5.1 Baseline Performance

In Figure 9 we present the baseline accuracies for person re-ID on Maket and Duke datasets, with ResNet-50 and DenseNet-121. The initial models were pretrained on ImageNet. The training took place in 60 epochs with SGD optimazation, learning rate being 0.1 for the first 40 epochs and 0.01 for the rest 20.

	ResNet-50	DenseNet-121
Market	Rank@1:0.872328 Rank@5:0.951010 Rank@10:0.969418 mAP:0.697656	Rank@1:0.880641 Rank@5:0.956057 Rank@10:0.975059 mAP:0.705519
Duke	Rank@1:0.745063 Rank@5:0.863106 Rank@10:0.901257 mAP:0.566456	Rank@1:0.799820 Rank@5:0.894075 Rank@10:0.922801 mAP:0.619564

Figure 9: Baseline performance of re-ID algorithms.

5.2 Channel Pruning

In Figure 10 we visualize layer-wise sparsity distribution. Each bar represents a convolutional layer in the network, and its height denoted its number of channels, green for ResNet-50 and blue for DenseNet-121. For each pruning pass with sparsity s from 0.05 to 0.5, we overlay the plot of number of channels with some transparency. Therefore, after all s's are finished, the color intensity of a bar can reflect the amount of critical channels in this layer.

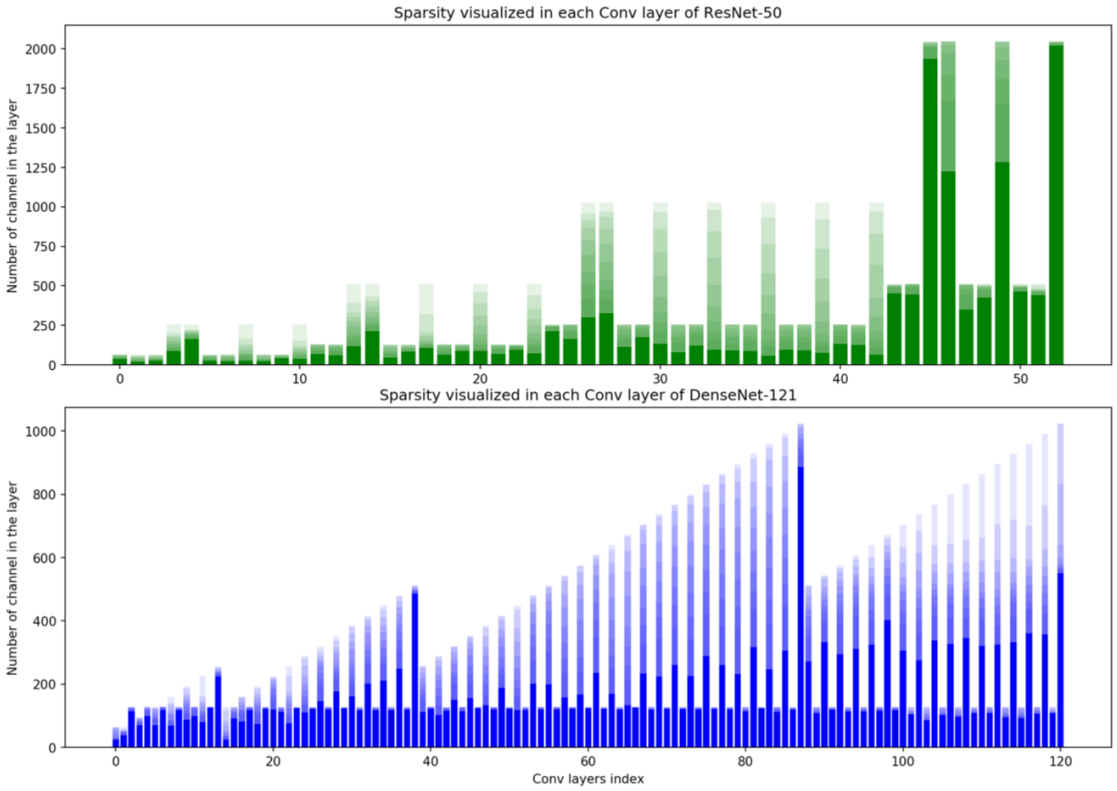


Figure 10: Visualization of sparsity distribution within ResNet and DenseNet models.

For ResNet, it seems that the layers in the middle contained more sparsity. The first and last layers were hardly touched. For DenseNet, things appear to be different, because the internal connections changed. Densely connected layers allows activations to skip multiple layers or blocks. Actually, if we view each dense block as an independent subsystem, the tendency will look familar. Again, the wrapping layers are significantly less pruned than the insiders.

Next, we move on to analyze the model performance after pruning. It is critical to measure the energy performance of our method. After all, the goal of any algorithmic methods mentioned in this project aims for enabling inference with lower disk storage, memory usage, power consumption and less time. We used

two criteria to evaluate the pruned models, the first being the model size, and the second being the inference time.

In Figure 11, highest validation accuracy was taken from all epochs; model size denoted how many Bytes the models took on the disk; Inference time was for predicting 10,000 images in the validation set. To generate these results, we pruned VGG-11, 16, and 19 models trained and validated on CIFAR-10.

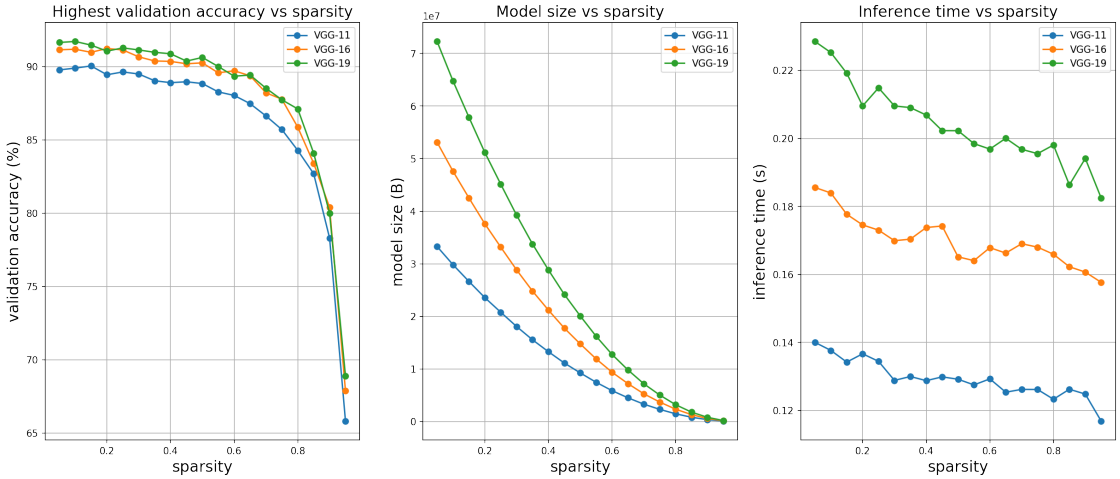


Figure 11: Validation accuracy, model size, and inference time vs sparsity

Obviously, the models benefited more on the storage side. Up to around 0.6 sparsity, the prediction accuracy was not severly harmed. However, the space that the model took on disk was reduced by $\sim 7\times$. Another observation came from the inference time. Although the general tendancy went down as the sparsity increased, the inference time of VGG-19 never got close to VGG-11. Even with 0.9 sparsity, its speed just merely became comparable with a full sized VGG-16.

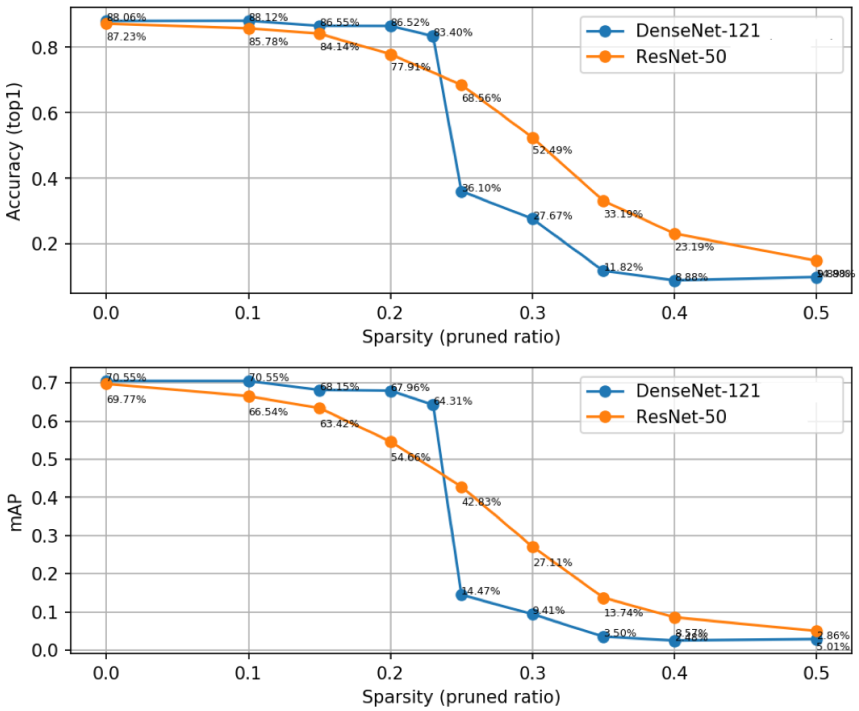
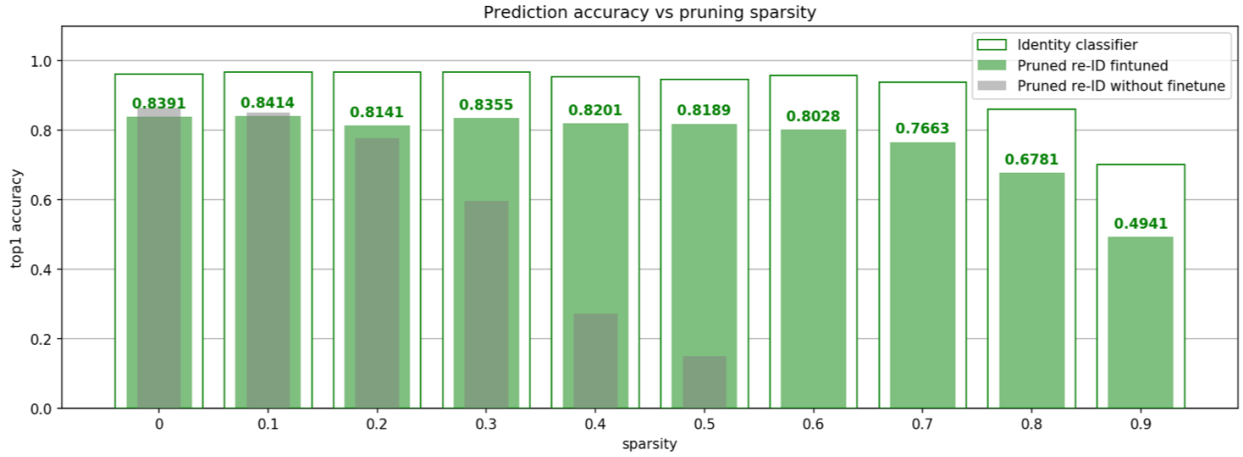


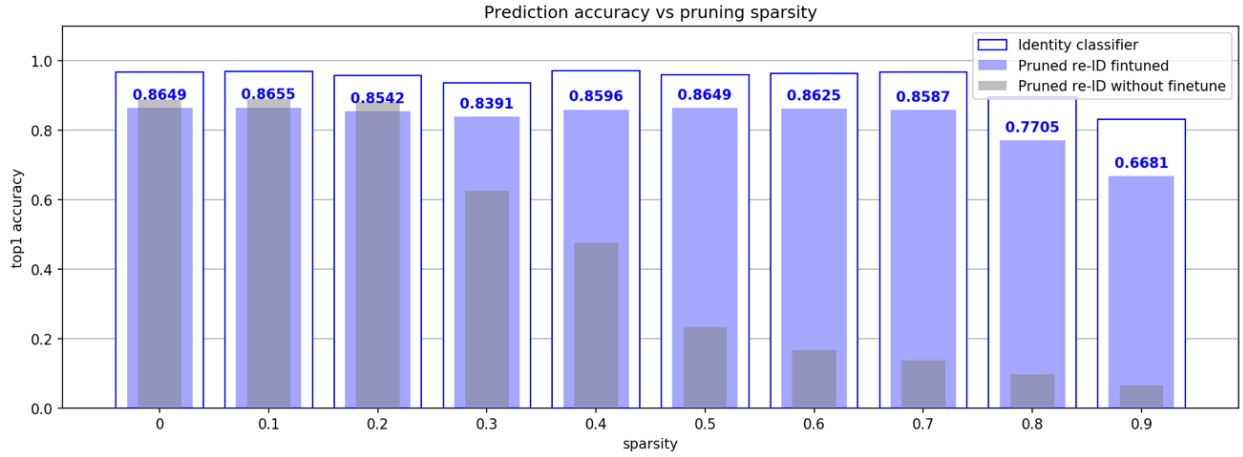
Figure 12: Channel pruning without retraining, up to 0.5 sparsity.

Figure 12 shows the re-ID accuracy and the mean average precision (mAP) of ResNet-50 and DenseNet-121. The reported results were produced by pruned model without finetuning. It can be observed that ResNet-50 had a more smooth drop than DenseNet-121. The underlying reason could be that with more convolutional layers, the DenseNet-121 was more robust to pruning at the beginning. However, loss of precision were more easily accumulated across the densely connected layers, causing the sheer drop at some point.

Obviously, being able to prune only 20% channel was not enough. To regain prediction accuracy, we performed finetuning on pruned models, and the results were satisfactory, as we can see in Figure 13.



(a) ResNet-50 based models.



(b) DenseNet-121 based models.

Figure 13: Predction accuracy vs pruning sparsity of ResNet-50

In this bar plot, the outer bar denoted the highest validation accuracy during 20-epoch finetuning. Recall that the system was composed of two modules, one being the trainable feature extractor. Here, we retrained the feature extractor to recognize people in the training set. This was defined as a classical classification task, and the correspondent accuracies were the classification accuracies, slightly different from the re-ID evaluation metrics. We show it as a guarantee that the model was retaining lost knowledge. The gray bars are the initial results of pruned yet unfinetuned models, whose values are same with those reported in Figure 11. We included them in this plot to verify the effectiveness of finetuning. Finally, the green bars are the validation accuracies on re-ID after fintuning. We can see enormous improvement of performance especially at high sparsity.

One more interesting observation Figure 13 was that in the small sparsity cases, namely when sparsity ≤ 0.1 in Figure 13a and ≤ 0.2 in Figure 13b, the finetuned accuracy were slightly lower than the unfinetuned. Our hypothesis was that with small sparsity, the pruned information was truly of little use, yet finetuning

took the model away from its previous optimum. Since the evaluation during training and re-ID was different, reaching maximum validation accuracy in training did not necessarily mean maximum accuracy in re-ID. It did not seem a sever problem at the current stage, but if it was undesirable, we could modify the loss function to bring the two evaluation metrics closer.

Overall, finetuning helped models regain accuracy particularly when they were pruned with high sparsity. Deeper models showed more robustness after finetuning. With post-prune finetuning, we can easily reach 0.5 sparsity for ResNet-50 and 0.7 sparsity for DenseNet-121.

5.3 Quantization

Figure 14 presents the top1 re-ID accuracy against the number of remained bits in ResNet-50. Since the changes only happened in the tailing stage, we log-scaled the axis for better visualization. We observed almost no accuracy loss with 8 or more bits in the forward parameters and activations. At 6 bits, the drop in accuracy became visible. Then the model performance collapsed drastically.

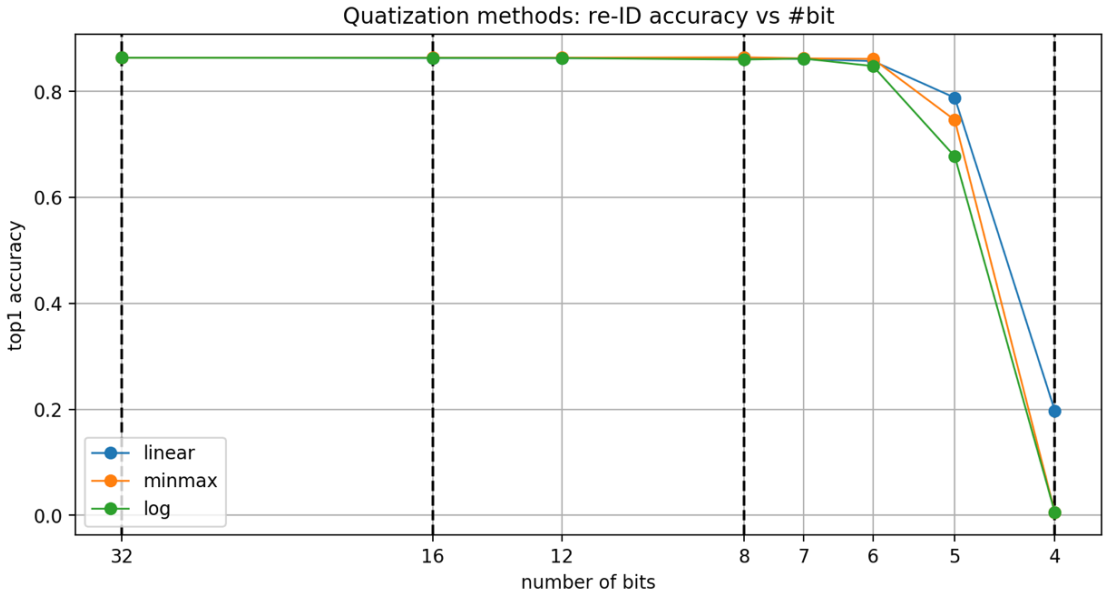


Figure 14: Quantization accuracy results without finetuning.

It is also shown that all of three quantization approaches applied gave reasonable performance until $5 \sim 4$ bits. Then linear accuracy went above the other two. One hypothesis could be that the sparse layers in the model did not provide much information to inference. With the majority of parameters near zero being less useful than the others, the strength of precisely quantize them became a wasted effort. Therefore, higher resolution in larger values was more critical to the prediction performance, and thus the linear quantizer slightly outperform the others.

6 Discussion

In this section, we will conclude on the progresses and challenges of this project, especially the challenges we are still facing to launch the person re-ID on drones. We will also review interesting observations from experiments and any inspiration it may have brought.

Intuitively, there are two directions to reduce the size of a deep convolutional model. We could either make the model slimmer as we did in this project, or we could make it shallower. Although we did not design an experiment on the latter aspect, The different original configurations and performance of VGG networks could roughly reveal some of the ideas. From depth 11 to 19, the accuracy was 91.93% and 92.45%, model size 36M and 77M, respectively. It could be open to debate whether the 0.5% growth in accuracy was worthy 114% growth in model size, likely dependent on the application and purpose. In our domain, however, the answer is most likely an assertive "no". Thus, besides slimming an existent model, it might also add more efficiency by carefully choosing a suitable architecture as the base model.

On the quantization experiments, it should be noted that because PyTorch does not support low precision representation of deep models, the quantized models here were just "acting" as they were quantized. The actual data representation of parameters and activation in the model were still 32-bit float numbers, with less decimal bits being nonzero though. Unfortunately, the quantization method cannot help with compressing the model at the current stage. However, given the long-term goal of this project, if necessary, we can switch to the TensorFlow framework whose latest release added support to quantized models, or directly implement it with hardware.

When quantizing a network, a common practice is to keep the precision in the layers immediately follow or precede the inputs and outputs. Our observation supports this practice, since it shows that the leading and tailing layers tended to be the last ones to prune. In other words, they may contain more important information than the layers in the middle. This agrees with common practice when training a ternary or binary model.

Most importantly, from this project, it has been learned that it is very difficult to compress models or reduce computations to a sufficient extent from a pure algorithmic aspect. Although the computational cost has been reduced via pruning, there is still a long way to go before the Raspberry Pi can afford it. The implementation of algorithms deeply relies on the hardware architecture. Thus, manipulation with hardware and dataflow should be critical to accelerate deep learning algorithms. For example, we could only prune channels because we had to maintain the regular shapes of kernels. Otherwise, the bonus of less parameters would be a total waste. In addition, the fixed-point data representation can only come into effect if the software supports it. In all, lower level architecture and circuits are the foundation of deep learning compression and acceleration. It can be foreseen without doubt that if the re-ID algorithm is successfully running on the drones, there must have been creative modifications done from the hardware side.

7 Future Work

In this project, we dived into two algorithmic approaches to try reducing the complexity of person re-ID models. To develop and utilize these models in our particular problem domain, we need to collect data. Even though we may not compete with the other dataset on the data amount, we can prevail due to the novelty of being the first dataset of human targets collected by multiple on-drone cameras.

Within our current framework, the most obvious first step would be implementing the "real" quantized models. We can investigate other deep learning framework such as TensorFlow. Another potential improvement was to design more intelligent policy for pruning. Currently, we manually define the policy to be pruning channels by batch normalization factors. To take one step further, we could let the system learn for itself which channels to prune via reinforcement learning process.

Speaking beyond the algorithmic methods, we will be actively considering hardware modifications if combining pruning and quantization still could not take the re-ID models to drones.

To sum up, this topic has a number of exciting extensions to explore. Other applications running on resource constrained platforms would also benefit from the new opportunities enabled by the efficient deep learning techniques developed in this project.

References

[1] K. He, X. Zhang, S. Ren, and J. Sun, "Deep residual learning for image recognition," in *Proceedings of the IEEE conference on computer vision and pattern recognition*, pp. 770–778, 2016.

[2] G. Huang, Z. Liu, L. Van Der Maaten, and K. Q. Weinberger, "Densely connected convolutional networks.," in *CVPR*, vol. 1, p. 3, 2017.

[3] L. Zheng, L. Shen, L. Tian, S. Wang, J. Wang, and Q. Tian, "Scalable person re-identification: A benchmark," in *Computer Vision, IEEE International Conference on*, 2015.

[4] Z. Zheng, L. Zheng, and Y. Yang, "Unlabeled samples generated by gan improve the person re-identification baseline in vitro," *arXiv preprint arXiv:1701.07717*, vol. 3, 2017.

[5] R. Pi, "model b," *Raspberrypi. org. Saatavissa: https://www. raspberrypi. org/products/raspberry-pi-3-model-b/. Hakupäivä*, vol. 6, p. 2018, 3.

[6] T. Huang and S. Russell, "Object identification: A bayesian analysis with application to traffic surveillance," *Artificial Intelligence*, vol. 103, no. 1-2, pp. 77–93, 1998.

[7] W. Zajdel, Z. Zivkovic, and B. Krose, "Keeping track of humans: Have i seen this person before?," in *Robotics and Automation, 2005. ICRA 2005. Proceedings of the 2005 IEEE International Conference on*, pp. 2081–2086, IEEE, 2005.

[8] N. Gheissari, T. B. Sebastian, and R. Hartley, "Person reidentification using spatiotemporal appearance," in *null*, pp. 1528–1535, IEEE, 2006.

[9] M. Farenzena, L. Bazzani, A. Perina, V. Murino, and M. Cristani, "Person re-identification by symmetry-driven accumulation of local features," in *Computer Vision and Pattern Recognition (CVPR), 2010 IEEE Conference on*, pp. 2360–2367, IEEE, 2010.

[10] W. Li, R. Zhao, T. Xiao, and X. Wang, "Deepreid: Deep filter pairing neural network for person re-identification," in *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, pp. 152–159, 2014.

[11] Y. LeCun, J. S. Denker, and S. A. Solla, "Optimal brain damage," in *Advances in neural information processing systems*, pp. 598–605, 1990.

[12] S. Han, H. Mao, and W. J. Dally, "Deep compression: Compressing deep neural networks with pruning, trained quantization and huffman coding," *arXiv preprint arXiv:1510.00149*, 2015.

[13] C. Zhu, S. Han, H. Mao, and W. J. Dally, "Trained ternary quantization," *arXiv preprint arXiv:1612.01064*, 2016.

[14] Y. He and S. Han, "Adc: Automated deep compression and acceleration with reinforcement learning," *arXiv preprint arXiv:1802.03494*, 2018.

[15] Y. He, X. Zhang, and J. Sun, "Channel pruning for accelerating very deep neural networks," in *The IEEE International Conference on Computer Vision (ICCV)*, Oct 2017.

[16] Z. Zhuang, M. Tan, B. Zhuang, J. Liu, Y. Guo, Q. Wu, J. Huang, and J. Zhu, "Discrimination-aware channel pruning for deep neural networks," in *Advances in Neural Information Processing Systems*, pp. 881–892, 2018.

[17] S. Anwar, K. Hwang, and W. Sung, "Fixed point optimization of deep convolutional neural networks for object recognition," in *Acoustics, Speech and Signal Processing (ICASSP), 2015 IEEE International Conference on*, pp. 1131–1135, IEEE, 2015.

[18] M. Courbariaux, I. Hubara, D. Soudry, R. El-Yaniv, and Y. Bengio, "Binarized neural networks: Training deep neural networks with weights and activations constrained to+ 1 or-1," *arXiv preprint arXiv:1602.02830*, 2016.

[19] M. Rastegari, V. Ordonez, J. Redmon, and A. Farhadi, "Xnor-net: Imagenet classification using binary convolutional neural networks," in *European Conference on Computer Vision*, pp. 525–542, Springer, 2016.

[20] S. Ioffe and C. Szegedy, "Batch normalization: Accelerating deep network training by reducing internal covariate shift," *arXiv preprint arXiv:1502.03167*, 2015.