EEEM071 Coursework Report

Deep Wilson Aricatt
da01075@surrey.ac.uk

Abstract

In this paper, we propose a solution to address the vehicle re-identification (ReID) task using the VeRi dataset as part of our coursework requirements. Our approach combines ResNeXt and Squeeze and Excitation (SE) networks to enhance feature representation, in conjunction with a combination of softmax and triplet loss. We prove that methods like feature fusion and attention can improve the model performance drastically as compared to vanilla feature extractors. Our best model (SE ResNeXt 50) achieves a mean average precision (mAP) score of 68.9% and a rank-1 score of 93.6%, surpassing the baseline results. Through experiments and analysis, we demonstrate the effectiveness of our proposed approach along with the effect of tuning different hyper-parameters. The code implementation is publicly available at https://github.com/deepwilson/vehicle_reid

1. Introduction

Vehicle Re-Identification (Re-ID) is a computer vision task that aims to identify the same vehicle across different cameras, given a query image. The goal is to match the query image with the images of the same vehicle captured by other cameras in a large-scale surveillance system. This problem is challenging due to variations in viewpoint, illumination, occlusion, and resolution. Vehicle Re-ID has many practical applications, such as traffic monitoring, vehicle tracking, and law enforcement.

2. Experiments with baseline model

2.1. Baseline Model

Following the guideline published we refer to the starter code [2]and select MobileNet V3 [1] as our standard baseline backbone network. Inspired by the general representation learning capabilities of the ResNet family [2] we perform extensive number of experiments using ResNet 18 as well.

During the training stage, we ensure that the pipeline is initialized as follows:

1. The MobileNet V3 model is initialized with pre-trained parameters from ImageNet, and the fully connected layer is modified to match the number of identities in the training dataset.

2. A training batch is created by randomly selecting N identities and K images per person, resulting in a batch size of $B = N \times K$. For our experiments, K is set to 4 and N depends on the batch size B. We ensure to set the PyTorch sampler to Random Identity Sampler to ensure hard negative triplets can be fed to the triplet loss function.

3. Input images are resized to 224x224 pixels and horizontally flipped with a probability of 0.5.

4. The images are decoded into range of 0 to 1. The RGB channels are then normalized by subtracting 0.485, 0.456, 0.406, and dividing by 0.229, 0.224, and 0.225, respectively [3].

5. The model outputs feature embeddings f (for triplet loss) and prediction logits p (for cross-entropy loss).

6. Feature embeddings f are used to calculate triplet loss [4], and prediction logits p are used to calculate cross-entropy loss. The margin m of triplet loss is set to 0.3.

7. Adam optimizer is used with an initial learning rate set to 0.0001 (we use a lower rate since we make use of Imagenet pre-trained models) and decreased by a factor of 0.1 at the 20th and 40th epochs. The total number of training epochs is 60.

We train the baseline models by tuning various hyper-parameters 5. In each stage of the experiments, hyperparameters such as batch size, choice of optimizer, image size, Embedding Size, custom head architecture and augmentations were modified to explore their impact on 2 metrics (mAP and r1). The best parameters were selected from each experiment and used as a starting point for the next stage. This iterative approach allowed us to systematically explore the hyper-parameter space and identify the optimal settings for the final model. The baseline model training results are summarized in 1 and the experiments are discussed in detail in 5

[2]https://github.com/Surrey-EEEM071-CVDL/CourseWork

Model	VeRi 776 r = 1	mAP
Baseline Model	79.1	49.5
+Batch Size 64	84.2	54.2
+Optimizer rmsprop	86.9	56.9
+Image Size 512,512	88.3	59.5
+Embedding Size 1024	90.8	61.6
+Multi-step scheduler with warmup	91.5	63.2
+Custom Head FC-BN-ReLU-Dropout-FC	91.7	**67.2**
+Augmentation random erase	**92.4**	66.7

Table 1. The performance of baseline model is evaluated on VeRi 776 dataset along with set of modifications explained in section 5. Baseline model is described in 2.1

3. CNN Architectures

To explore architecture design we chose resnet-50 over mobilenet. However, due to the computational cost associated with training them, we performed our experiments with a resnet-18 model.

Our main motivation was to study about 2 major modifications that can be made to existing models. Feature fusion (concat and addition) and Attention methods (spatial and channel based). Within attention blocks we used Squeeze and excite network blocks (channel attention) and CBAM blocks (spatial and channel attention).

We evaluate the different flavours of models by combining above mentioned techniques all of which are described in table 2:

3.1. Feature Fusion

Feature fusion refers to the process of combining or merging different features extracted from multiple layers/stages of the network to form a more robust and informative representation of the input data. We visualized the feature maps learnt by the conv blocks in the different stages of the resnet-18 backbone. Each row in the 1 corresponds to the feature maps learnt by the conv blocks by the 4 stages in the resnet.

As part of the experiment we would like to verify if fusing from different stages of any kind leads to an improvement in the score over using the vanilla models or not. For this purpose we decided to fuse outputs from the 3rd and 4th stages of the resnet-18 model.

The output from stage 3 has a shape of ([$batch\_size$, 256, 14, 14]) whereas stage 4 has a shape of ([$batch\_size$, 512, 7, 7]). In it's raw form they cannot be fused together hence, we ensure that the output from stage 3 is compatible with stage 4 output by using conv1x1 to change the output channels to 512 and additionally use a stride of 2 to downscale the spatial size from 14x14 to 7x7.

To fuse feature maps from different stages we use 2 widely used methods: - Concatenation (stacking the feature maps along the channel dimension) - Addition(adding the feature maps of two or more layers together element-wise)

3.1.1 Concatenation

Concatenation involves concatenating the features maps along the channel dimension to form a longer feature vector. This allows the network to capture both low-level and high-level features before being passed to the embedding module that is then passed to the triplet loss. On the downside concatenation can increase the dimensionality of the feature space.

3.1.2 Addition

Addition involves adding the feature maps together element-wise. This allows the network to selectively emphasize important features from different layers, while suppressing irrelevant or noisy features. However, it is possible that addition can result in a loss of information, since the resulting feature map will only contain the sum of the original feature maps.

3.2. Attention blocks

3.2.1 Squeeze-and-Excitation Networks

The central idea of Squeeze-and-Excitation Networks [5] is to improve the quality of representations produced by a CNN by explicitly modeling the interdependencies between the channels of its convolutional features. This is achieved through "Squeeze-and-Excitation" (SE) blocks, which adaptively recalibrates channel-wise feature responses by selectively emphasizing informative features and suppressing less useful ones.

3.2.2 CBAM (Convolutional Block Attention Module)

The central idea of CBAM (Convolutional Block Attention Module) is to adaptively refine intermediate feature maps in feed-forward convolutional neural networks by sequentially inferring attention maps along two separate dimensions, channel and spatial, and multiplying them to the input feature map for adaptive feature refinement.

3.3. ResNeXt

We additionally explored ResNeXt as well (we show this as a separate section in the table 2) The ResNeXt paper [6] proposed a new neural network architecture that improves upon the ResNet architecture by introducing a novel block design that uses a split-transform-merge strategy to combine the output of parallel convolutional pathways. The authors found that increasing the cardinality(number of convolutional pathways) of the network, while keeping other

factors such as depth and width constant, improved the network's performance on image classification benchmarks. This model gave us the highest accuracy overall. However, this could also be due to the larger number of parameters.

Model Architecture	r1	mAP
ResNet18	90.5	63
SE-ResNet18	90.3	63.5
SE-ResNet18 Concatenation	**90.9**	**64.3**
SE-ResNet18 Addition	89.6	62.4
ResNet CBAM	89.9	62.8
ResNet CBAM and Concatenation	89.5	63.3
SE-ResNet18 Global Concatenation	89.9	63.2
Model Architecture (ResNeXt)	**r1**	**mAP**
SE-ResNeXt50	**93.6**	**68.9**

Table 2. Model flavours

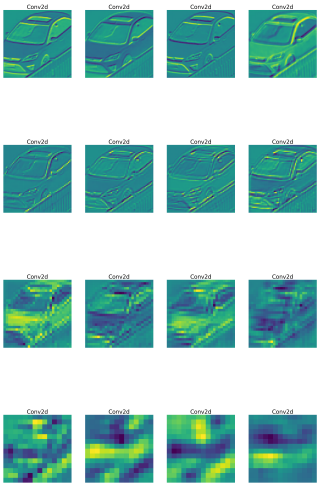

Figure 1. Feature Maps - Resnet 18

Each row in the figure corresponds to outputs from the respective stage in Resnet

3.4. Conclusions

In terms of modifications to the networks, SE-ResNet18 model with Concatenation fusion gave the best results in terms of r1 and mAP (refer table 2.). However, it can be observed that the vanilla ResNet18 also gave comparable performance. For this dataset, it can be said that fusing features using concatenation is a better option as compared to addition. Since we observed that there was a boost in accuracy we also added a new flavour that fused features from all 4 stages before passing to the embedding module. We call this model as SE-ResNet18 Global Concatenation. However, this model did not give any outstanding results. We conclude there should be a better was to fuse the low level features from initial stages and high level features from the later stages.

Additionally, squeeze and excitation blocks performed better that CBAM blocks. This could be due to the fact that as the network gets deeper and deeper in a typical CNN the spatial resolution of the feature maps decrease, while the channel dimensions increases. As a result channel attention could play a more important role than spatial attention. Also, the inductive biases of the CNN like locality bias, translation invariance, kernel sizes invariance (fixed size kernels are used in CNNs) take care of learning good spatial representations as well.

4. Data Augmentations

Data augmentation helps by increasing the variability of examples in datasets. Figure 2 shows the different augmentations applied to the input image

4.1. Random Erasing REA

Zhong et al. [7] proposed a data augmentation approach called Random Erasing Augmentation (REA) to address occlusion problems in person ReID. Inspired by this work we use REA to randomly select a rectanglur region in an image and erase its pixels with mean values. The hyper-parameters $0.02 < Se < 0.4, r1 = 0.3$ are set to control the aspect ratio and area of the erased region. We noticed that the default values in the starter code were taken from person-reid problems(aspect ratio 1.5 which is suitable for pedestrians) and hence changed the values to have more square/rectangular shaped regions to be erased.

4.2. Random 2D Translation

Randomly translates the input image in the x and y directions by up to 1/8 of the image size. The resulting image is then cropped to the original size. This helps to simulate the effect of objects appearing at different positions in the image.

4.3. Color Augmentation

Randomly alters the intensities of the RGB channels of an image.The augmentation uses a fixed set of eigenvectors and eigenvalues to transform the intensities of the RGB channels.

4.4. Color Jitter

Inbuilt PyTorch augmentation that randomly adjust brightness, contrast, saturation, and hue.

4.5. Center Crop

Crops the input image from the center to a specified size of 200x200 pixels. This helps to focus on the central part of the image and remove any unnecessary background. We chose this particular augmentation taking inspiration from Arcface [8] where the input images of faces are cropped and

aligned(rotated faces are straightened) in such a way that unnecessary background pixels are removed. This helped Arcface in achieving better accuracy overall.

4.6. Conclusion

We observe that REA used alone can improve the overall accuracy as shown in table 3. Choosing data augmentations is difficult because it is not always clear which augmentations will be most effective for a given dataset or problem. It can also be time-consuming to manually try out different combinations of augmentations and evaluate their impact on model performance. Reinforcement learning can be used for auto augmentation, which involves using a neural network to learn a policy for selecting augmentations that maximize model performance. The network is trained through trial-and-error, where it generates a sequence of augmentations for a given image, evaluates the resulting model performance, and updates its policy based on the feedback. However, this could require a lot of trial and errors and significantly increase training time.

Augmentation	r1	mAP
REA	**92.4**	**66.7**
REA + Jitter + Color Aug	91.6	66.5
REA + Jitter + Color Aug + Center Crop + 2D Trans	90.4	65.7

Table 3. Augmentation experiments

Figure 2. Data Augmentations

5. Exploration of Hyper-parameters

Hyperparameters in deep learning are parameters that are set before the learning process begins and directly affect how well a model trains. Examples of hyperparameters include the learning rate, the number of epochs, momentum, regularization constants etc. We additionally explore the effect of some other design choice as well such as batch_size, embedding size, image size, choice of optimizer etc. which we believe can affect the overall accuracy. Out findings are summarized in tables corresponding to each sub-section

5.1. Embedding Size

In triplet loss, the embedding size matters because it determines the dimensionality of the feature space in which the model maps the input images. In general, we observed that larger embedding sizes (refer table 4) can capture more information about the input resulting in better performance.

Embedding Size	r1	mAP
128	89.8	60.1
256	89.5	59.0
512	89.0	60.7
1024	**90.8**	**61.6**

Table 4. Effect of Embedding size

5.2. Batch Size

In the context of triplet loss, a larger batch size can help to ensure that the triplets selected for training are more diverse and representative of the overall distribution of samples. However, as per the experiments conducted (refer table 5), higher batch size proved to be detrimental in learning. This could be due to the fact that as the batch size gets larger, the gradients that flow backwards have less stochasticity.

Batch Size	r1	mAP
16	**86.5**	53.1
32	85.6	**54.6**
64	84.2	54.2
128	82.8	53.2
384	79.1	49.5

Table 5. Effect of Batch Size

5.3. Learning rate schedulers

Learning rate schedulers are used to adaptively adjust the learning rate (more apt to be called as step-size) during training, which could improve the model's performance.They help to avoid overfitting, improve convergence, and ensure the model is not stuck in a local minimum. Commonly used schedulers include multi-step, step decay, exponential decay, and cyclic learning rate, each with their own advantages and disadvantages.We try different lr schedulers as shown in table 6

5.4. Why warm-up works?

For example in our experiment we use RMSProp which computes a moving average of the squared gradients to get an estimate of the variance in the gradients for each parameter. For the first update, the estimated variance is just the square root of the sum of the squared gradients for the first batch. In general, this will not be a good estimate(the network is just starting to learn), and the first update could push the network in a wrong direction. To avoid this problem, we give the optimiser a few steps to estimate the variance while making as little changes as possible (warm-up phase) and only when the estimate is reasonable, we use the higher learning rate. We then continue with normal lr rate schedule

Scheduler	r1	mAP
Multi-step	88.3	59.5
Warmup Linear	**91.5**	**63.2**
Warmup Cosine	91.1	62.7

Table 6. Learning rate schedulers

5.5. Image Size (W,H)

Larger images can provide more information and thereby better performance (refer table 7), but can also result in longer training times. On the other hand, smaller images may lead to loss of important details. We also tries changing the aspect ratio to rectangular shape in order to evaluate if it helps in boosting accuracy as compared to square aspect ratio. However, both performed equally well.

Image Size	r1	mAP
128x128	82.8	53.5
224x224	86.3	57.9
512x512	**88.3**	59.5
512x288	88.2	**60.5**

Table 7. Effect of Image Size

5.6. Optimizer

RMSprop outperforms the other optimizers in terms of both r1 and mAP metrics (refer table 8), while SGD performs the worst. RMSprop has the best convergence which is clearly visible in the figure 3 SGD didn't converge for a learning rate of 0.0001 and as a result the experiment was performed with an lr of 0.001. SGD is known to get stuck at saddle points and hence this was expected.

An important observation to be noted is that when training the heavier models(resnet18 based models mentioned in 2) using RMSprop it did not converge properly. For these models we ended up using Adam optimizer. During the training process, it was observed that the cross-entropy loss oscillated 3, but it did not significantly affect the learning process. It is likely that this is a normal occurrence due to the triplet (htri) loss adjusting feature vectors in a way that may violate the classification metric. However, over time, the effects of this phenomenon were reduced.

Optimizer	r1	mAP
Adam	85.2	54.2
AdamW	84	52.90%
AMSgrad	83.9	53.6
RMSprop	**86.9**	**56.9**
SGD	80.5	48.9

Table 8. Effect of Optimizer

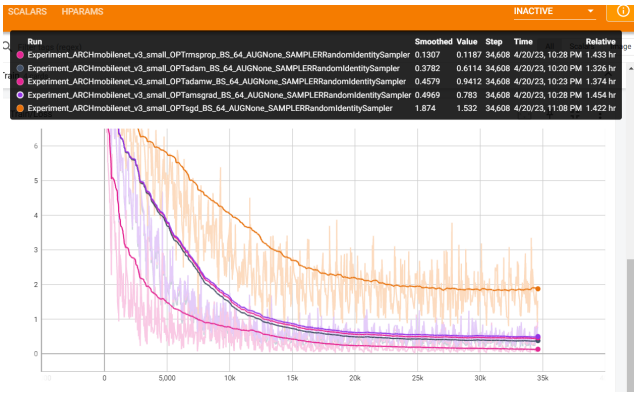

Figure 3. Optimizers: Training loss (TensorBoard plot)

6. Extra Credit

As we are using a combination of cross-entropy loss and triplet loss, we study both types of losses in detail. After studying more about contrastive losses like Arcface, Sphereface, Cosface etc compared with triplet loss, we came up with the following hypothesis: cross-entropy loss is more suitable with cosine distance whereas triplet loss is more suitable with Euclidean distance. As a result these two losses could have inconsistent targets in the embedding space. Our hypothesis is that we need to modify the head of the model after feature extraction before passing to classifier fc layers by making use of BN layers and ReLU activation functions and dropout. Additionally, taking inspiration from the [9] and [10], we can set the bias in the fc layers to zero. This will ensure that the classification hyper-planes will pass through the zero co-ordinate. This will help in bringing the embedding space within a Gaussian space which will help the cross-entropy loss to converge quickly [10] (as observed in figure 4). We refer to existing works and discussions online and experiment different modified head architectures. Different architectures were evaluated as mentioned in table 9:

ReLU is an activation function that introduces sparsity to the network. Placing ReLU before or after BN layer is a debatable topic and hence, we carry out extensive experiments to infer what is the right design choice for our problem statement. The dropout acts like a regularizer and helps in achieving better generalization. Using dropout is quivalent to using multiple "thinned/modified networks" during training. A combination of all of these are applied to the embedding received from the backbone network in the following order FC-BN-ReLU-Dropout-FC after which it is passed to the classifier FC layer for cross-entropy loss. As shown in table 9 this ordering gave a far better accuracy as compared to vanilla embedding.

Head Design	r1	mAP
Without BN - Vanilla	89.9	63.3
FC-ReLU-Dropout-BN-FC	89.9	63.4
FC-ReLU-BN-FC	90.0	63.4
FC-BN-ReLU-Dropout-FC	**91.7**	**67.2**
FC-BN-ReLU-FC	90.9	63.2
FC-BN-FC	89.9	62.6

Table 9. Head Designs

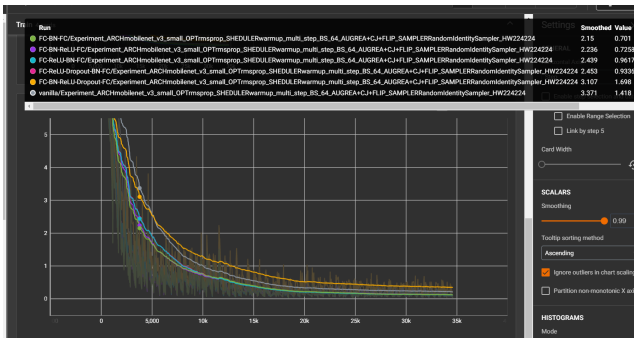

Figure 4. Training loss for Head designs (TensorBoard plot)

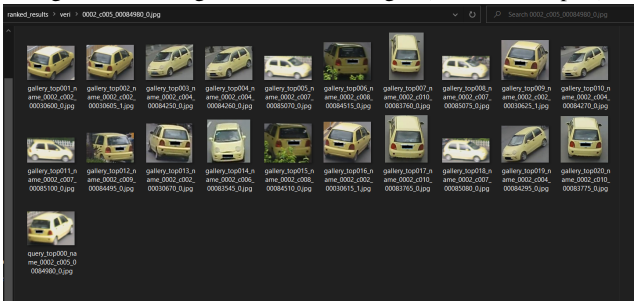

Figure 5. Visualize testing set - 1 sample

References

[1] Andrew Howard, Mark Sandler, Grace Chu, Liang-Chieh Chen, Bo Chen, Mingxing Tan, Weijun Wang, Yukun Zhu, Ruoming Pang, Vijay Vasudevan, Quoc V. Le, and Hartwig Adam. Searching for mobilenetv3, 2019. 1

[2] Kaiming He, Xiangyu Zhang, Shaoqing Ren, and Jian Sun. Deep residual learning for image recognition, 2015. 1

[3] Normalization in the mnist example. https://discuss.pytorch.org/t/normalization-in-the-mnist-example/457/7. 1

[4] Alexander Hermans, Lucas Beyer, and Bastian Leibe. In defense of the triplet loss for person re-identification, 2017. 1

[5] Jie Hu, Li Shen, Samuel Albanie, Gang Sun, and Enhua Wu. Squeeze-and-excitation networks, 2019. 2

[6] Saining Xie, Ross Girshick, Piotr Dollár, Zhuowen Tu, and Kaiming He. Aggregated residual transformations for deep neural networks, 2017. 2

[7] Zhun Zhong, Liang Zheng, Guoliang Kang, Shaozi Li, and Yi Yang. Random erasing data augmentation, 2017. 3

[8] Jiankang Deng, Jia Guo, Jing Yang, Niannan Xue, Irene Kotsia, and Stefanos Zafeiriou. ArcFace: Additive angular margin loss for deep face recognition. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 44(10):5962–5979, oct 2022. 3

[9] Weiyang Liu, Yandong Wen, Zhiding Yu, Ming Li, Bhiksha Raj, and Le Song. Sphereface: Deep hypersphere embedding for face recognition, 2018. 5

[10] Hao Luo, Youzhi Gu, Xingyu Liao, Shenqi Lai, and Wei Jiang. Bag of tricks and a strong baseline for deep person re-identification, 2019. 5