Vehicle re-identification (Re-ID) Using Computer Vision

Section 1. Familiarity with the provided code.

Section 1.1. Discussion on the training and evaluation process of the default code.

Before executing the code, I ensure the dataset is unzipped and set my name and ID as environment variables. Then, I execute the Python script *main.py*, which trains and evaluates a MobileNetV3 (small) model. Due to computational complexity, I set the maximum number of training epochs to 3 and keep other hyperparameters as it is. Finally, I specify the log directory to save the logs and checkpoint. The model achieves 8.1% mAP in the test data. Figure-1 shows the CMC curve.

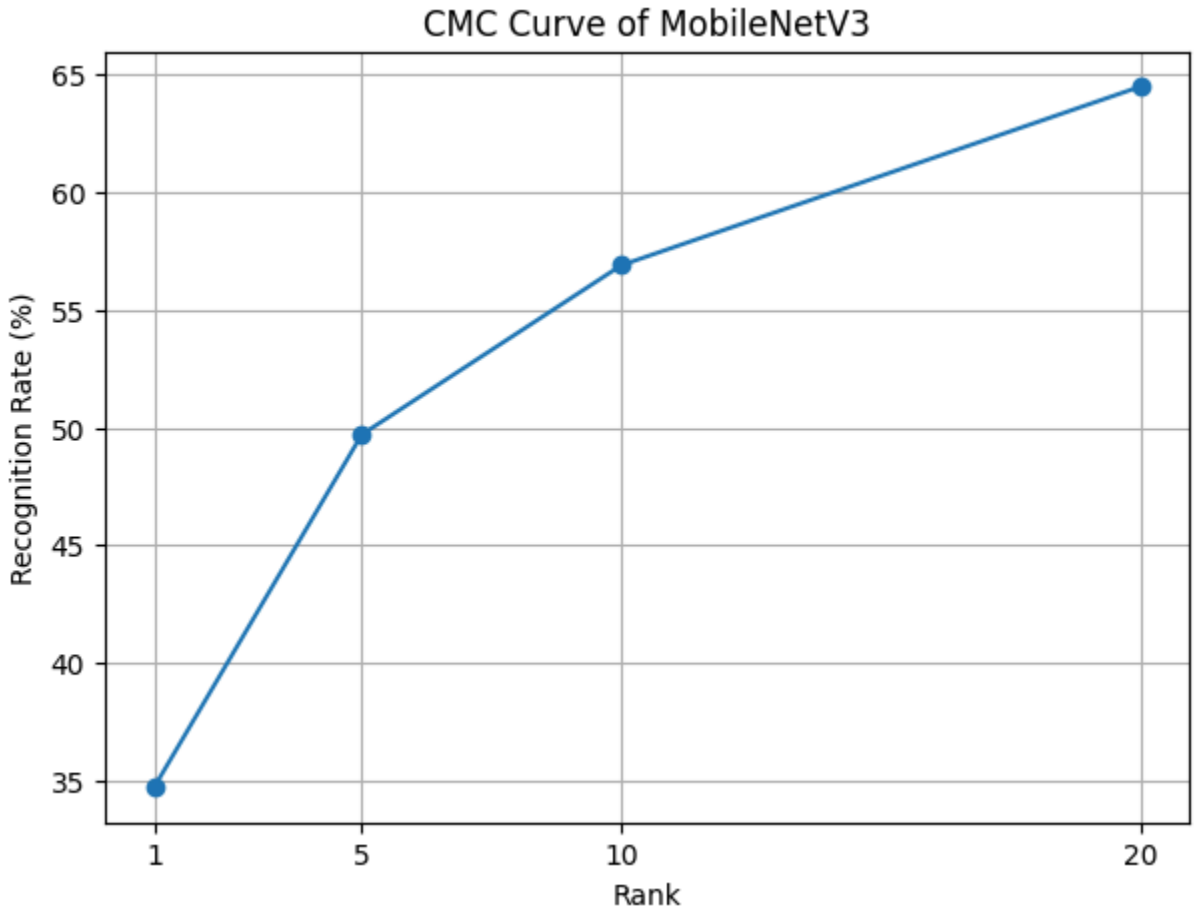

Figure 1. CMC curve of MobileNetV3 (Small)

Section 1.2. Critical discussion and contrast of ResNet with Section 1.1.

I implemented ResNet50 in this section. It achieves a lower mAP (38.3%) than the MobileNetV3 (39.3%). Also in the test data the ResNet model achieves lower mAP (7.7%) than MobileNetV3 (8.1%). Overall MobileNetV3 performs better than the ResNet50 model, the CMC curve of both models presented in Figure-2.

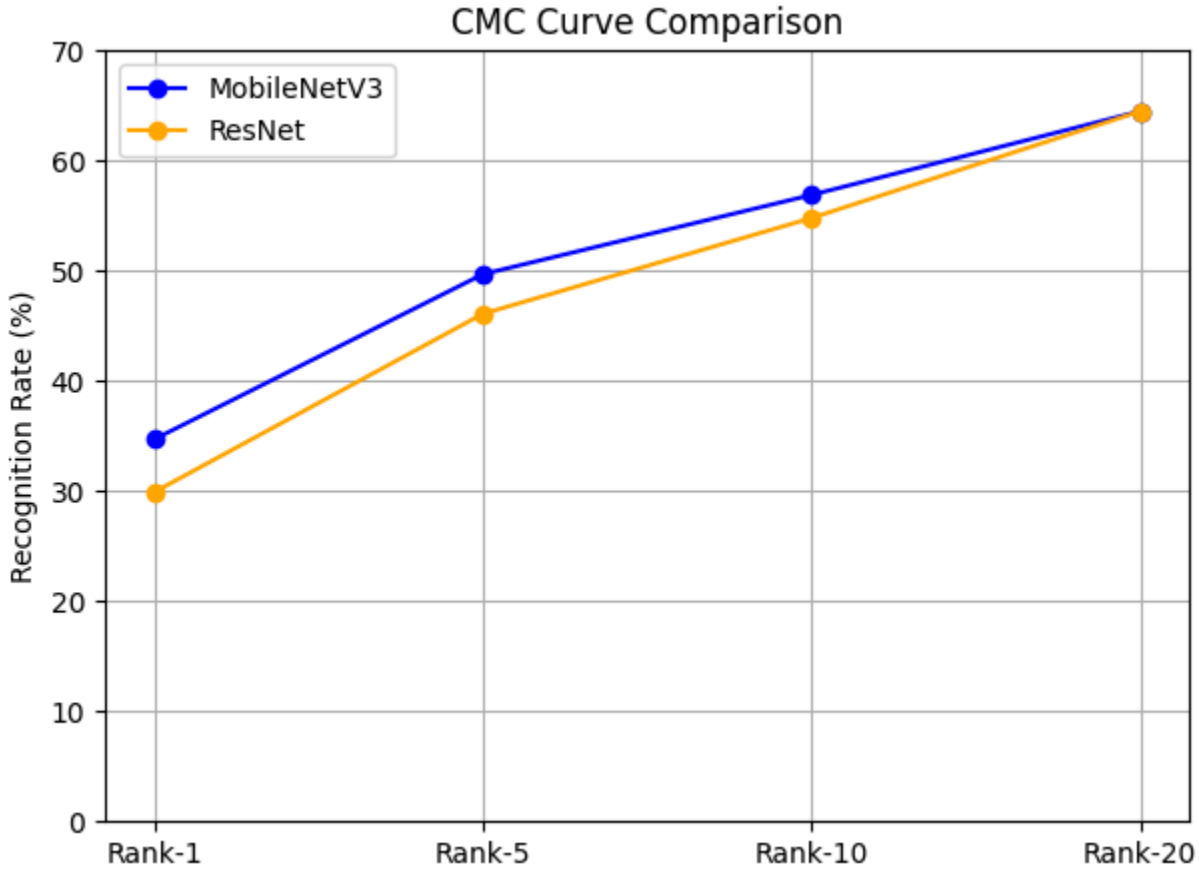

Figure 2. CMC curve comparison of MobileNetV3 and ResNet50

Section 1.3. Critical discussion and contrast of VisionTransformer with Section 1.1 and 1.2.

The VisionTransformer model obtains a better mAP score (8.7%) than the MobileNetV3 and ResNet50 in the training data. I keep the same configuration for the three models. The mAP scores comparison presented in Figure-3. VisionTransformer also achieves a lower CMC value than MobileNetV3 except for at rank-20. The CMC curve is presented in Figure-4.

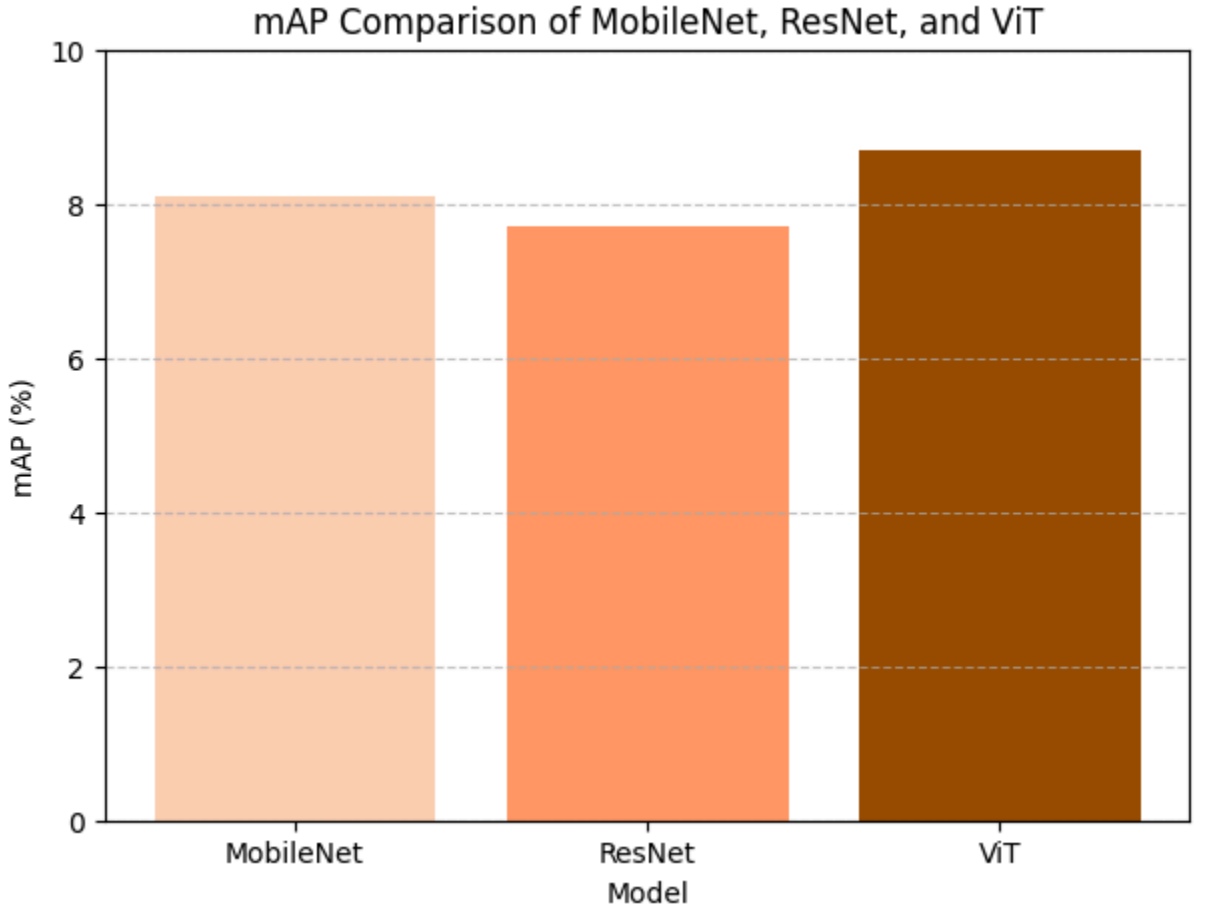

Figure 3. mAP score comparison of MobileNetV3, ResNet50, and VisionTransformer.

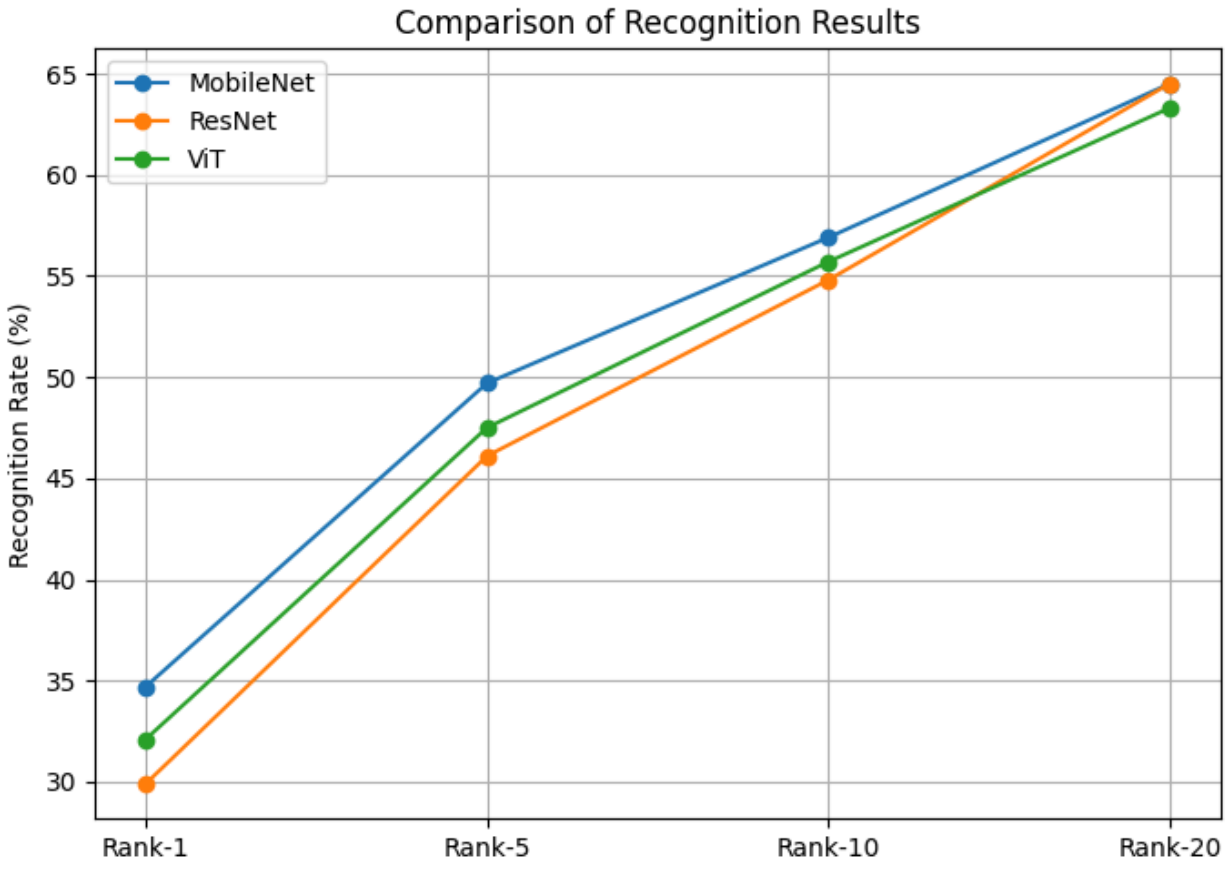

Figure 4. CMC curve comparison of MobileNetV3, ResNet50, and VisionTransformer.

Section 2. Dataset preparation and Augmentation experiments.

Section 2.1. Random crop data augmentation.

In this section, I applied random crop augmentation , modifying the *main.py.* The crop size was 100 x 100 pixels. Surprisingly, this augmentation technique does not change the result that much. After applying random crop augmentation, the MobileNetV3 achieves the same mAP score as previous which is 8.1%. Also the model achieved CMC values for rank-1 to rank-15 same as previous, in the rank-20 it achieved 1 more than the previous. The graphical representation is presented in Figure-5.

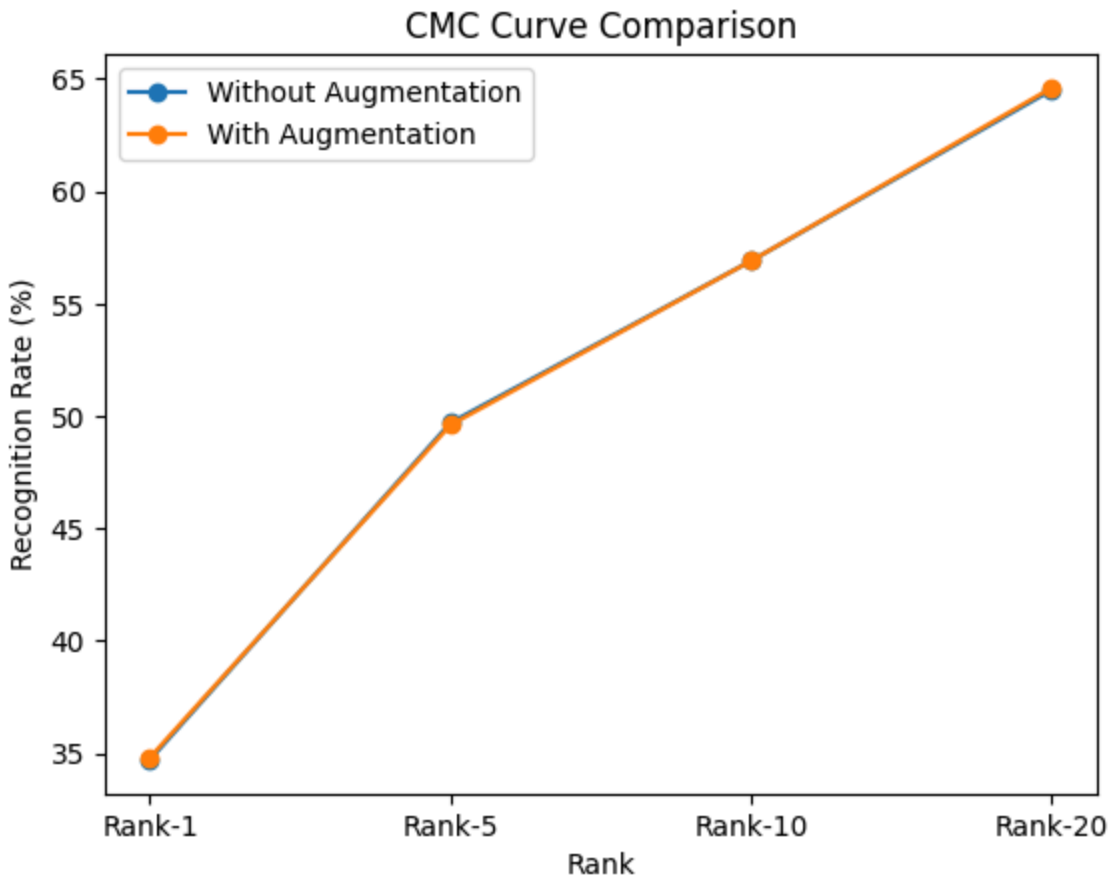

Figure 5. CMC curve comparison of MobileNetV3 after applying random crop augmentation.

Section 2.3. Blurring and horizontal flip data augmentation.

Both augmentation techniques were applied modifying the *main.py* code. In that code, blurring augmentation is implemented using Gaussian blur transformation from torchvision.transforms. It's applied to the input images during data loading, specifically using the *transforms.Compose* function to create a transformation pipeline which includes the GaussianBlur transformation with a kernel size of 3.Horizontal flip also applied following the technique. Again, horizontal flip does not affect the results, meaning the model achieves the same mAP score and CMC value. But blurring augmentation technique improves the model performance, by this technique the model achieved a 39.5% mAP score. It also achieved a higher CMC value, presented in Figure-6.

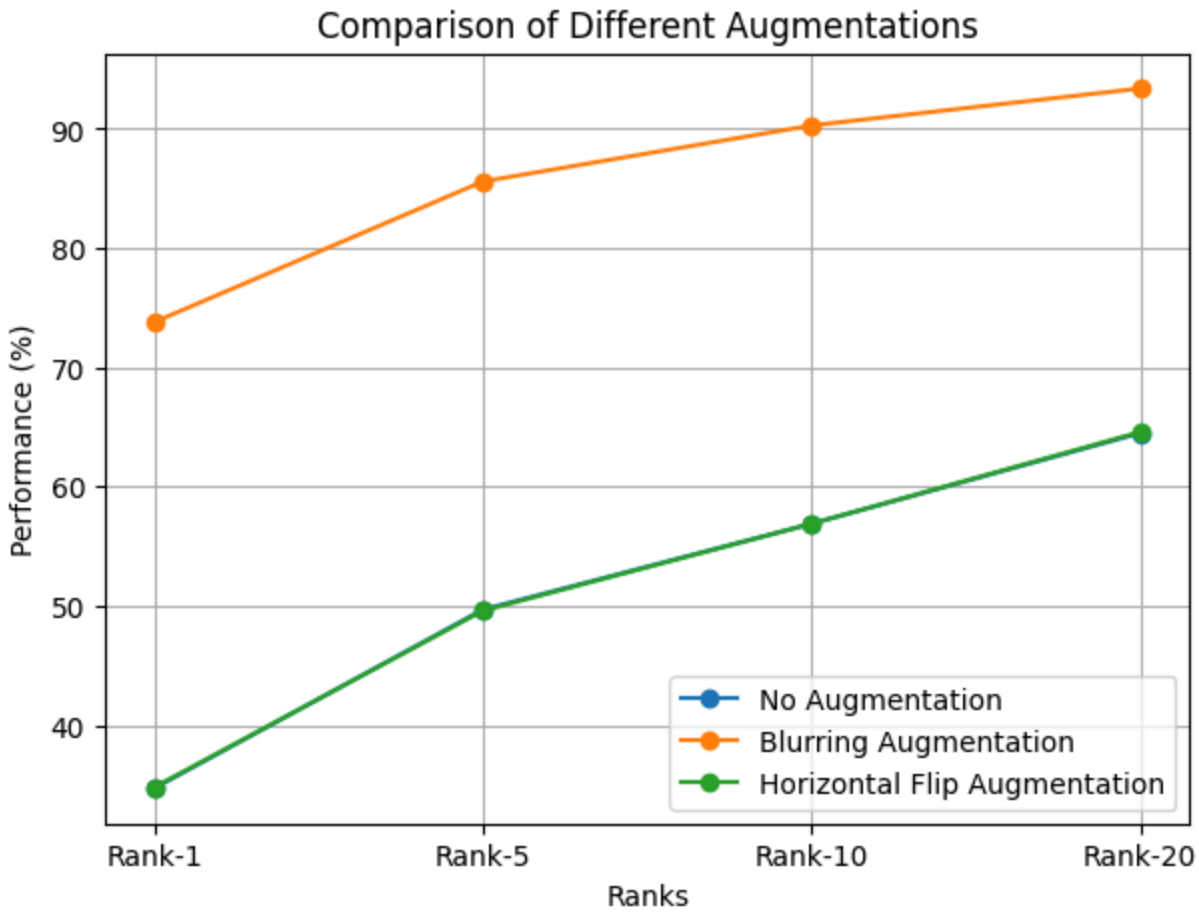

Figure 6. CMC curve comparison of MobileNetV3 after applying blurring and horizontal flip augmentation.

Section 2.3. Combined augmentation techniques employed in section 2.1 and section 2.2.

Again, I implemented a combined augmentation using *torchvision.transforms.Compose.* This allows multiple transformations to be applied sequentially to the input images during data loading. In this case, I included random horizontal flipping, random cropping, and Gaussian blurring with a kernel size of 3. Surprisingly, I did not notice any changes in the results. The model achieved the same mAP score as previous and CMC value expert the rank-20 for 1, presented in Figure-7.

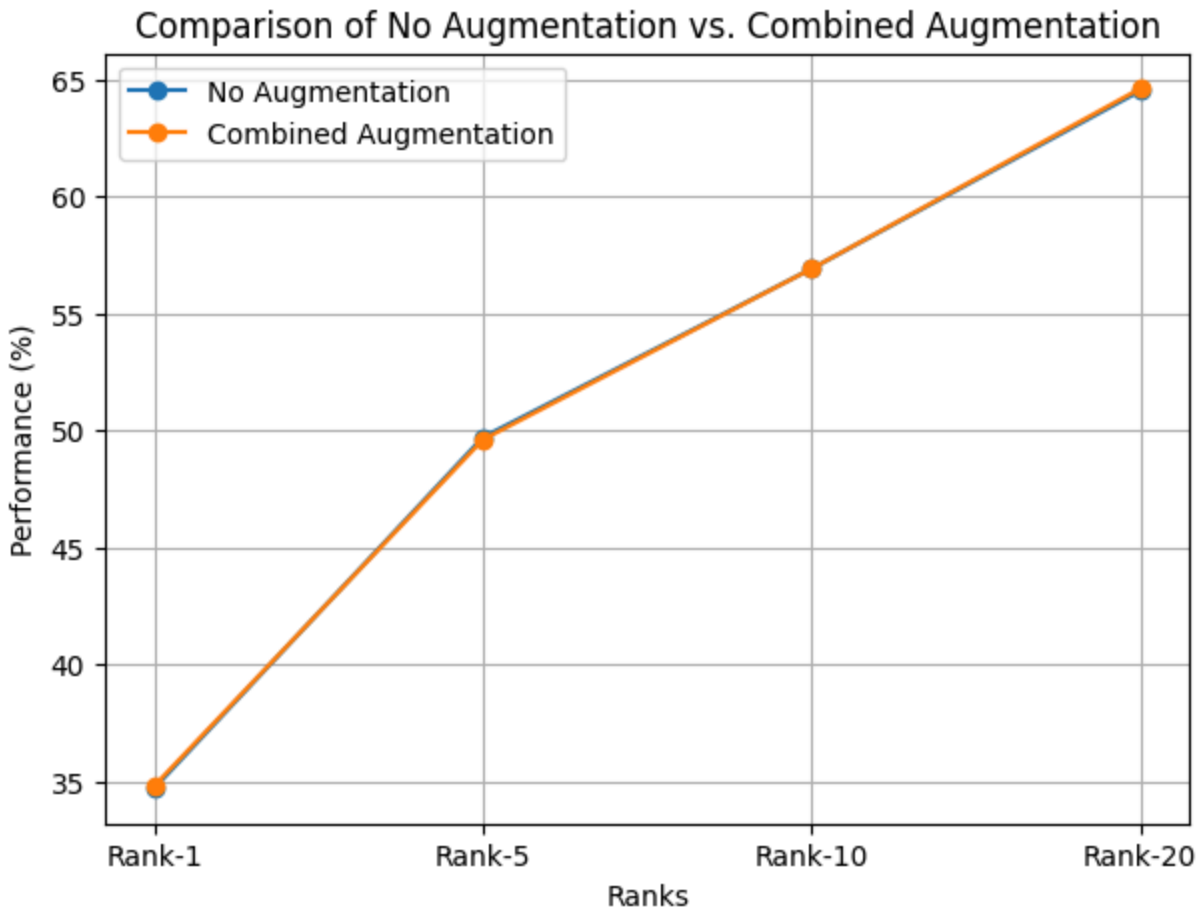

Figure 7. CMC curve comparison of MobileNetV3 after applying combined augmentation.

Section 3. Exploration of Hyperparameters.

Section 3.1. Exploration of learning rate (LR).

In this section, I experimented with 4 LR values (0.0001, 0.001, 0.01, 0.1) with the default one (0.0003). As I trained the model for 3 epochs (due to computational cost), I did not observe any great effect in the result. Again, mAP score did not change, for all these LRs models achieve the same score which is 8.1%. I noticed small changes in the CMC values which are presented in Figure-8. Out of all experimented LRs, *lr=0.001* performed slightly better, so I can say that it is the best lr value.

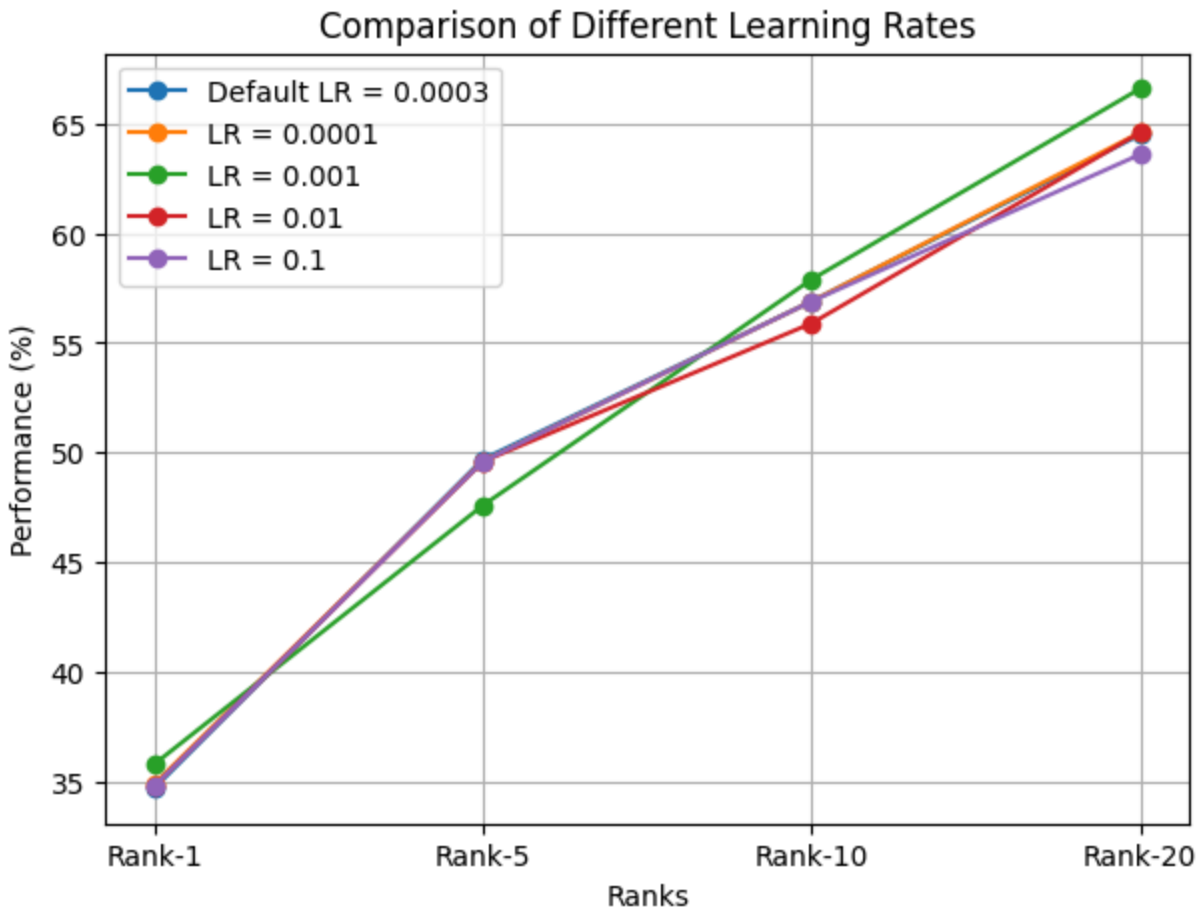

Figure 8. CMC curve comparison of MobileNetV3 after applying different LRs.

Section 3.2. Exploration batch sizes.

Here, I experimented with 4 different batch sizes (32, 128, 256, 512) employing the best LR (0.001). Again, I observed no difference in the mAP scores but CMC values. I applied the batch sizes and LR modifying the argument while running the training block of code. Out of all batch sizes, *batch size = 128* showed some improvement in CMC values, presented in Figure-9. By the experiments of section 3.1 and 3.2 I found that *best_lr = 0.001* and *best_batch_size=128*.

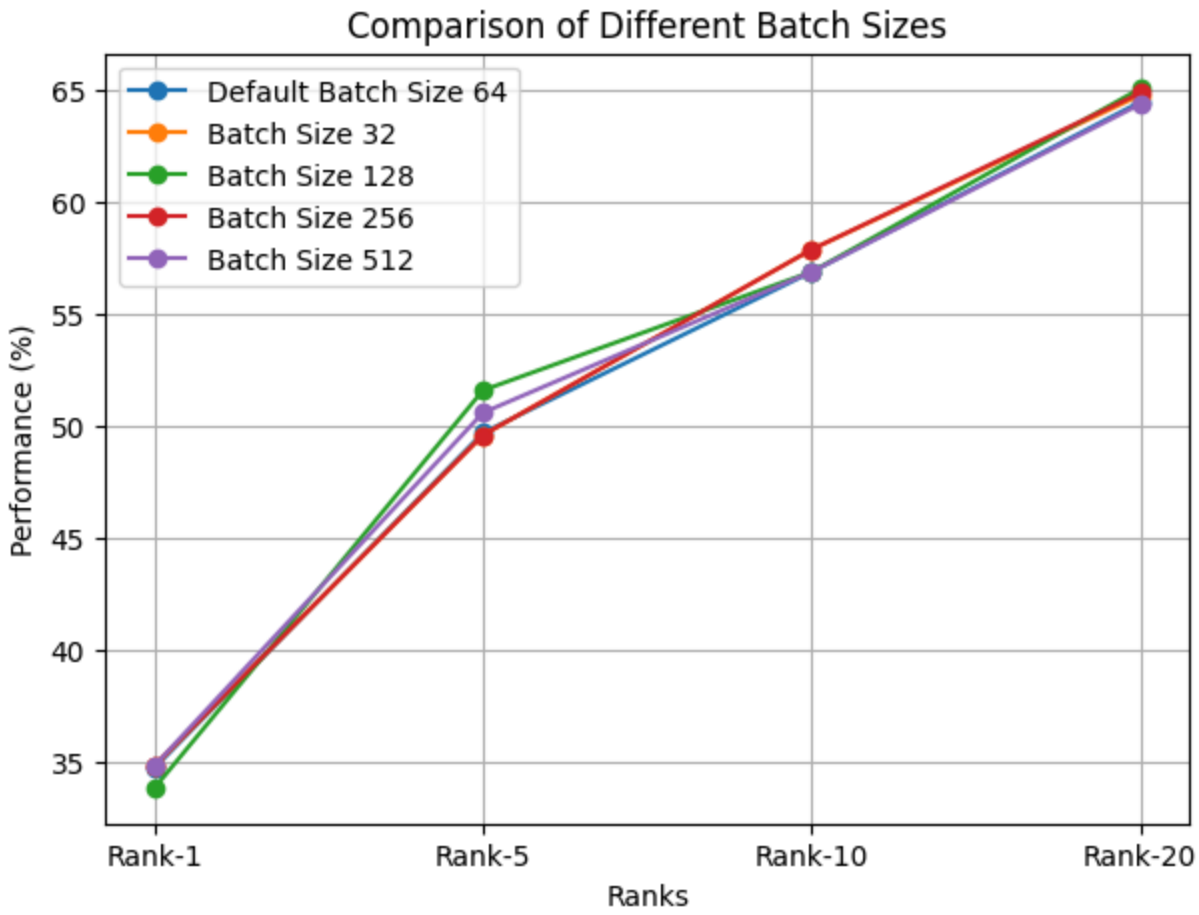

Figure 9. CMC curve comparison of MobileNetV3 after applying different batch sizes.

Section 3.3. Exploration of the optimizer with the best learning rate and batch size.

Again, I trained the MobileNetV3 model employing the *best_lr* and *best_batch_size*. In this section, I experimented with another optimizer: SDG. I applied this optimizer modifying the *main.py* internal class. This time the model showed some improvement in CMC values, but not in mAP score compared with the default one which was experimented in section 1.1, presented in Figure-10.

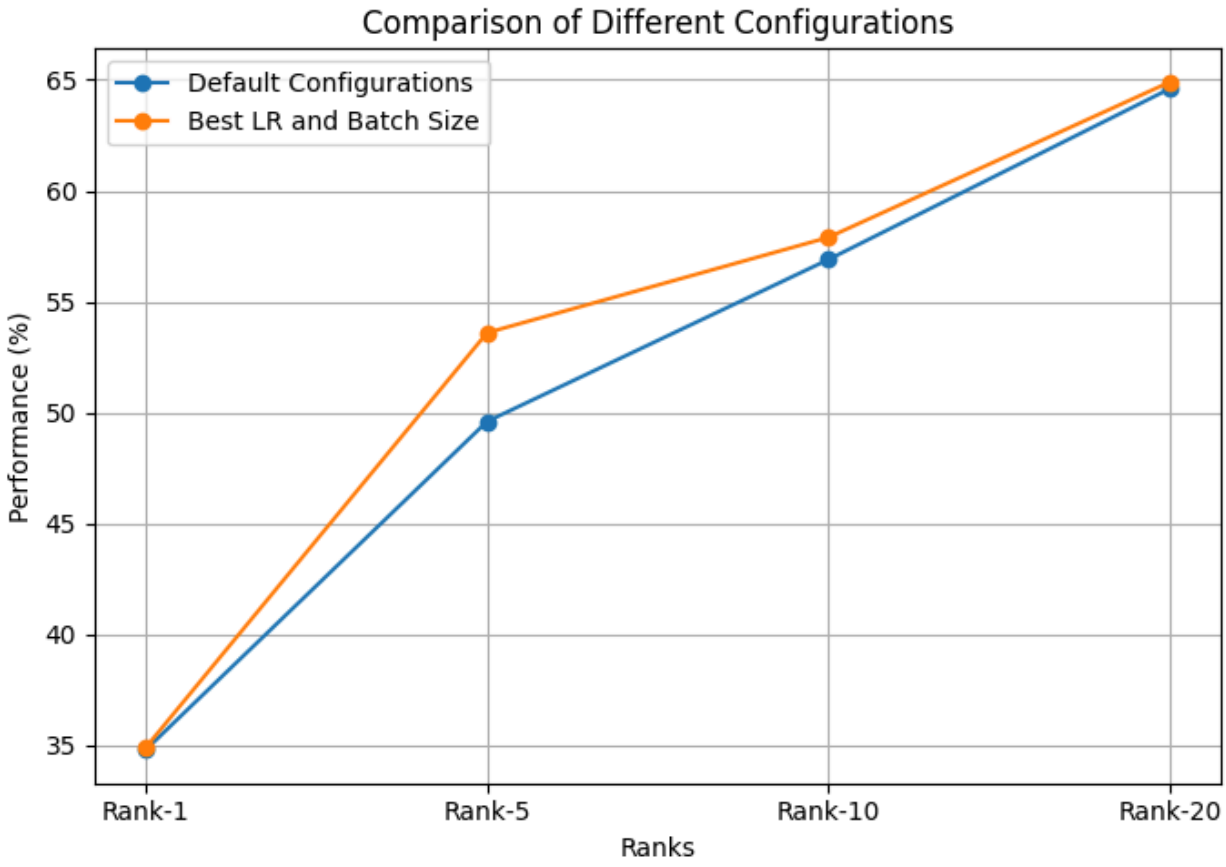

Figure 10. CMC curve comparison after applying best LR and batch sizes with the default one.