EEEM071 coursework report

Yash Mahesh Kulthe (6766477)

Abstract

The Vehicle Re-Identification is a crucial computer vision task that plays a vital role in matching images captured by various cameras and views in Intelligent Transport Systems. This task is challenging and requires a comprehensive approach to deal with different camera angles and lighting conditions. In this report, we have conducted an in-depth analysis of the Vehicle Re-Identification task on the VeRi dataset comprising over 50,000 images captured by 20 cameras placed at different locations.

Many experimental setups were explored by employing different augmentation techniques, model architectures, and optimizers. Our primary objective was to propose the best-performing model that can accurately identify vehicles in real-world scenarios. After conducting rigorous experimentation, a ResNet18 architecture is proposed, augmented with an additional convolutional block layer of dimension 1024 with ReLU activation in Basic Block of ResNet18, which has been trained on the VeRI dataset which achieved rank 1 accuracy of 88% and mAP of 60.1.

1. Introduction

The task of vehicle re-identification (Re-ID) [1] is evolving primarily for surveillance activities to identify the cars that may be captured by one camera. One camera won't be able to accurately follow and track the car because it will be moving on the road, but numerous cameras networked together on the road should be able to do so.

Reidentification is the process of determining whether the vehicle in the photos is the same as the one that was captured by the cameras.

The Vehicle ReId is a task in which there is a large gallery of images of vehicles, from which the features are extracted such that when an image is given to identify, its features are extracted and then the features from the gallery with the input image is measured and the one which has less loss or is close to the one in gallery is matched. This is a basic idea of Vehicle Re-identification.

But in machine learning, it is done in a different way, here we don’t have an image gallery to take all the images and keep matching with them which is time consuming, whereas, in here, there is a complex process of taking the cameras from the roads, which has images of the whole area, i.e. it will be having images of all the vehicles on the road. But a machine learning algorithm cannot be trained like that, so instead some algorithms are which will identify the vehicles in the image and build a bounding box around it and then crop those images. Here to make a model more generalizable only one type of vehicle can be taken for e.g., a car, or motorcycle. If there are many types of vehicles, it will be hard for the model to keep track of all the features.

Then the images which are created from this method are then annotated then put in the Vehicle Re-Identification models which can be models like CNNs, Siamese Networks, Triplet Networks, Attention based models, Transformer networks, and many more techniques. This model which is trained is then validated over the retrieval task such that an image from query set is taken and then it has a task of matching the similar images from a gallery set. This way the model is iterated, and it is being trained.

For the testing, the vehicle retrieval is done on a query and gallery set. The ranking list is also obtained for this by sorting the calculated query to gallery similarity.

1.1. Dataset

The dataset used in this study is the VeRi [2] dataset, which contains images of vehicles captured from multiple non-overlapping camera views in real-world scenarios. It comprises more than 40,000 bounding boxes of 619 vehicles captured by 20 cameras that cover a 1 km2 area in one day. The images were captured in various conditions and lighting, making it a practical dataset for vehicle re-identification. The dataset is labeled with different attributes and was collected in real-world unconstrained surveillance scenes.

The VeRi dataset is partitioned into three subsets: a training set with 37,778 images from 576 unique vehicle IDs captured by 20 cameras, a query set with 1,678 images from 200 vehicle IDs captured by 19 cameras, and a gallery set with 11,579 images from 200 vehicle IDs captured by 19 cameras. This partitioning allows for testing the performance of the model on unseen data during evaluation.

Subset	**#ids**	**#images**	**#cameras**
Train	576	37778	20
Query	200	1678	19
Gallery	200	11579	19

Table I: Image Dataset Statistics

1.2. Pipeline

The pipeline first initializes the available GPU or CPU and then proceeds to initialize the Image Data Manager.

This manager is responsible for loading the data from the VeRi [2] dataset and returning train and test dataloaders. The arguments provided through the argument parser are then passed to their respective functions.

The model is then initialized, taking the name of the model, number of classes, losses, and whether the model weights should be pretrained. If the model is to be pretrained, the pretrained weights are downloaded and assigned to the model. In addition to initializing the model, the losses for the pipeline are also defined. Specifically, cross-entropy loss and triplet loss are used.

Next, the optimizer and learning rate scheduler are initialized. If the model is set to evaluation mode, the test function is called, which runs the data on the model and computes the ranks and accuracies. On the other hand, if the model is set to train mode, the train method is called, which uses all of the functions defined above to train the model and evaluate its performance in terms of rank.

Throughout the training process, the pipeline evaluates the model at specified intervals using the test function. If the evaluation indicates that the model has improved, a checkpoint is saved containing the model state, the rank1 value, the epoch number, the model architecture, and the optimizer state.

Finally, a rank logger is used to track and store the performance of the pipeline. It shows the rank1 value for each dataset and each epoch, and a summary is displayed at the end of the pipeline. The elapsed time is also recorded and displayed at the end of the pipeline.

1.3. Loss Functions

This coursework has two loss functions used: cross entropy and triplet loss function.

Cross entropy loss is a widely used loss function in classification tasks. In the context of vehicle re-identification (VeRi), it is used to predict the probability of an image belonging to a specific identity (class). Given an input image, the model outputs a probability distribution over all the possible identities, and the cross-entropy loss is used to compare this distribution with the ground truth label. The objective of minimizing the cross-entropy loss is achieved when the predicted probabilities of the model correspond closely to the true labels of the data.

In contrast, triplet loss is used to learn an embedding space where images of the same identity are closer to each other, and images of different identities are farther apart. This type of loss function measures the dissimilarity between an anchor image, a positive image (which shares the same identity as the anchor), and a negative image (which belongs to a different identity). The distance between the anchor and the positive is minimized while the distance between the anchor and the negative is maximized in the formulation of the loss [3].

In the context of VeRi, a combination of these two losses is commonly utilized, where the triplet loss is used to develop a discriminative embedding space and the cross-entropy loss is used to categorize images into distinct identities. The weighted sum of the two losses makes up the final loss function. The model is tuned to minimize this total loss function during training. The model parameters must be updated using an optimizer, such as stochastic gradient descent (SGD) or Adam, and the gradients of the loss with respect to those parameters must be computed.

2. Experimentations

2.1. Data preprocessing and augmentation

Data augmentation and preprocessing is a technique which is used to increase the diversity and samples in training data. It involves transformation of images like cropping, rotating, flipping, etc.

The images from the query set can be seen in Figure 1. If these images are fed in the model, the model will be overfit on these images and not able to perform well on unseen data.

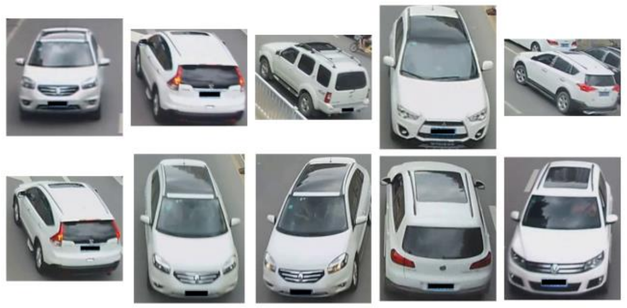


Figure 1- Images before augmentation

To perform augmentation, 5 things have been tried on MobileNetV3 small [6]. These things have been tried: RandomErase [4], ColorAugmentation [5], ColorJitter, GaussianBlur. Also, with this augmentation techniques by Jamali et al. [7] and Huynh et al. [8] which have been called as minimizing_aug and strong-baseline in this report. The results for the augmentation can be seen in Table II. Each step was run on 30 epoch and batch_size 256.

Type	mAP	Rank1	Rank5	Rank10	Rank20
MobileNetV3-small-Erase-jitter	52.4	82.7	93.1	95.6	97.7
MobileNetV3-small-Erase-jitter-coloraug	51.5	82.2	92.3	94.9	97.1
MobileNetV3-small-Erase-jitter-coloraug-GBlur	51.1	81.3	91.9	95.8	97.4
MobileNetV3-small-minimizing_aug	26.3	56.9	71.6	78.8	84.6
MobileNetV3-small-strong-baseline	30.3	63.2	77.7	82.9	88.3

Table II: Data augmentation

The best results were seen in just the erase and jitter, but for more generalization, the third augmentation was used as it has gaussian blur which will add noise to the data and generalize well on the dataset. The images when used the third augmentation can be seen in Figure 2.

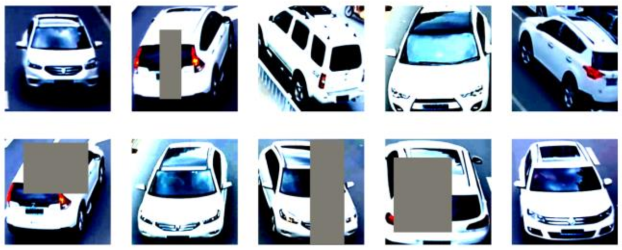


Figure 2 - Images after Augmentation

2.2. CNN Architecture

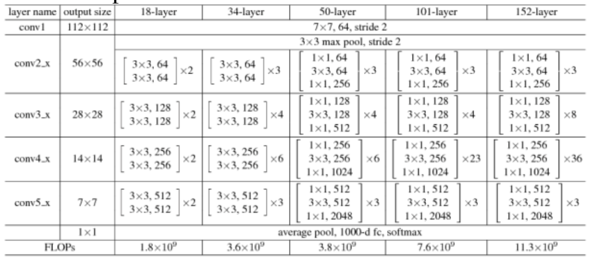
Four CNN architectures were tried on the dataset. The data augmentation was the one which was chosen in the last experiment. The four networks tried on were MobileNetV3 [6], EfficientNet [9], RegNet-y [10], and ResNet-18 [11].

All these models were trained on amsgrad [12] optimizer, 20 epochs and batch_size 128, with other default parameters. The results can be seen in Table III.

Model	mAP	Rank1	Rank5	Rank10	Rank20
Mob	51.1	81.3	91.9	95.8	97.4
Effi	60.5	88.5	95.2	97.4	98.6
Regnet	30.8	68.3	82.4	88.3	92.1
ResN18	55.9	85.6	94.3	96.5	98.1

Table III: CNN Architecture

Using this information, ResNet18 [11] architecture was chosen as it was providing similar ranks to EfficientNet [9], while EfficientNet had high mAP but ResNet18 took less time to compute.



layer name	output size	18-layer	34-layer	50-layer	101-layer	152-layer
conv1	112×112	7×7, 64, stride 2				
conv2_x	56×56	3×3 max pool, stride 2				
		$\begin{bmatrix}3\times3, 64\\3\times3, 64\end{bmatrix}\times2$	$\begin{bmatrix}3\times3, 64\\3\times3, 64\end{bmatrix}\times3$	$\begin{bmatrix}1\times1, 64\\3\times3, 64\\1\times1, 256\end{bmatrix}\times3$	$\begin{bmatrix}1\times1, 64\\3\times3, 64\\1\times1, 256\end{bmatrix}\times3$	$\begin{bmatrix}1\times1, 64\\3\times3, 64\\1\times1, 256\end{bmatrix}\times3$
conv3_x	28×28	$\begin{bmatrix}3\times3, 128\\3\times3, 128\end{bmatrix}\times2$	$\begin{bmatrix}3\times3, 128\\3\times3, 128\end{bmatrix}\times4$	$\begin{bmatrix}1\times1, 128\\3\times3, 128\\1\times1, 512\end{bmatrix}\times4$	$\begin{bmatrix}1\times1, 128\\3\times3, 128\\1\times1, 512\end{bmatrix}\times4$	$\begin{bmatrix}1\times1, 128\\3\times3, 128\\1\times1, 512\end{bmatrix}\times8$
conv4_x	14×14	$\begin{bmatrix}3\times3, 256\\3\times3, 256\end{bmatrix}\times2$	$\begin{bmatrix}3\times3, 256\\3\times3, 256\end{bmatrix}\times6$	$\begin{bmatrix}1\times1, 256\\3\times3, 256\\1\times1, 1024\end{bmatrix}\times6$	$\begin{bmatrix}1\times1, 256\\3\times3, 256\\1\times1, 1024\end{bmatrix}\times23$	$\begin{bmatrix}1\times1, 256\\3\times3, 256\\1\times1, 1024\end{bmatrix}\times36$
conv5_x	7×7	$\begin{bmatrix}3\times3, 512\\3\times3, 512\end{bmatrix}\times2$	$\begin{bmatrix}3\times3, 512\\3\times3, 512\end{bmatrix}\times3$	$\begin{bmatrix}1\times1, 512\\3\times3, 512\\1\times1, 2048\end{bmatrix}\times3$	$\begin{bmatrix}1\times1, 512\\3\times3, 512\\1\times1, 2048\end{bmatrix}\times3$	$\begin{bmatrix}1\times1, 512\\3\times3, 512\\1\times1, 2048\end{bmatrix}\times3$
	1×1	average pool, 1000-d fc, softmax				
FLOPs		1.8×10^9	3.6×10^9	3.8×10^9	7.6×10^9	11.3×10^9

Figure 3 - ResNet Architecture

To try out some changes in ResNet18, first thing was tried to interchange the BasicBlock with Bottleneck layer, secondly, instead of having 4 layers (see in Figure3) of convolution block in ResNet, one more was added having 1024 dimension to add more complexity to the model. Then a ResNet18_FC256 model was tried.

Hyungjun et al. [15] had used ReLU6 in their network and found out that it can help with the prevention of exploding gradients. ReLU6 sets the maximum output value to 6, while normal ReLU does not have any upper limit. This can help if the model is too deep i.e., having a large hidden layer. Another advantage of ReLU6 is that it can help prevent the problem of "dead neurons" that can occur with ReLU. Dead neurons are units in the network that always output zero, which can happen when the weights are initialized in a way that causes the unit to never activate. By limiting the maximum output of ReLU6, it can prevent this problem and lead to better model performance. Thus the ReLU activation was replaced with ReLU6 activation in the BasicBlock of the ResNet18 architecture .The results of these experiments can be seen in Table IV. All the models were run on batch_size 600 (due to large time for models to train), amsgrad optimizer and same default parameters.

ResNet18	mAP	Rank1	Rank5	Rank10	Rank20
ResNet18	56.4	84.7	93.3	96.4	97.8
BottleN	27.6	49.8	69.6	79.9	88.2
1024L	54	84.9	92.6	96.0	98.2
FC256	53.3	83.5	93.2	96.3	97.9
Relu6	52.5	81.6	92.2	95.2	97.1

Table IV: ResNet18 Architecture changes

With this as it can be seen, the normal ResNet18, the additional 1024 layered ResNet 18, ReLU6 ResNet18 and

FC256 ResNet all perform almost similar, but the least time consuming was the normal ResNet18, so it will be continued for next experimentation. But after hyperparameter tuning, these four models will be trained on the same parameters and then tested to see the better one of them. ReLU6 was seen to be working really well and also it could be used in other of these models so that they can be trained more, and the gradients will not explode.

2.3. Hyperparameter Tuning

Hyperparameter tuning is the most important process of any machine learning model. It is the process of selecting the optimal hyperparameters. This is important as if the parameters are set poorly, the model may not be able to learn effectively.

The first thing to be done in this is the selection of the best optimizer according to this problem. Four optimizers have been tried on the ResNet 18 model, those are, amsgrad [12], adam [13], Stochastic Gradient Descent [14], and RMSprop. These all optimizers were taken with their default values. The batch size is 600 while all the settings as previous on ResNet 18. The results of the best optimizer can be seen in Table V.

Optim	**mAP**	**R1**	**R5**	**R10**	**R20**
amsgrad	56.4	84.7	93.3	96.4	97.8
Adam	55.8	84.3	92.8	95.5	97.7
SGD	9.6	32.8	53.2	63.2	73.1
RMSprop	36	69	85	89.5	93.9

Table V: CNN Architecture

In this table, it can be seen that the best performance is given by amsgrad and adam. Other optimizers were too slow to converge and generalize on the data. For computation time amsgrad was faster than adam, also it gave better results throughout and converge faster.

The next thing experimented on was the learning rate of the amsgrad optimizer, different values such as 0.003, 0.00009, 0.0001 (weight decay 0.005), and 0.0003 was tried. The results of which are in Table VI.

LR	**mAP**	**R1**	**R5**	**R10**	**R20**
0.003	45.2	77.9	90.2	94.2	96.4
0.00009	22.5	49.2	68.6	77.2	84.2
0.0001(WD)	27.4	55.4	73.9	82.6	89.1
0.0003	56.4	84.7	93.3	96.4	97.8

Table VI: Learning Rate

Then the Learning rate Scheduler is used. It allows the learning rate to be adjusted during training which can help the model to converse faster and more effectively at the start. The purpose of the learning rate scheduler is to gradually reduce the learning rate during training, allowing the model to converge more steadily and prevent it from overshooting the optimal solution.

The learning rate schedulers tried in this coursework are, single step, multistep, and cosine. Which have been implemented such that the step size for single step is 10, for multistep 7,14, and for cosine the tmax is 20 while etamin is 0.00003. The results are as seen in Table VII. All these models were trained on 20 epochs, lr = 0.003, 600 batch size and gamma = 0.1.

LR-S	**mAP**	**R1**	**R5**	**R10**	**R20**
no lrs(0.0003)	56.4	84.7	93.4	96.4	97.8
Cosine	47.2	77.7	90.0	94.0	96.2
Singlestep	41.9	72.5	85.6	90.5	94.9
Multistep	38.1	65.6	87.7	87.4	92.0

Table VII: Learning Rate Scheduler

From the observation, it can be seen that without learning rate scheduler the model is performing better, which is due to ResNet18, as it has simple architecture also the amsgrad optimizer have built in adaptive learning rate scheme which performs better without any additional tuning.

Now, the final hyperparameter tuning is done on the triplet loss function i.e., the margin and the number of instances. In all the experiments the margin is set to 0.4 from 0.3 the default, and number of instances 6 from default 4. As larger margin will help in better separation of the instances which can result in better classification accuracy while providing a greater number of instances will provide more data to the model for learning.

This hyper parameter is tried on all the ResNet18 model variations that were seen previously. The ResNet18 with ReLU6 model is trained with 20 epochs while batch size is 600, it is also trained on batch size 128 to see the changes. Then the ResNet18 FC256 is ran on 20 epochs with 600 batch size, similar with ResNet18 normal. Then the ResNet18 with extra layer having dim 1024 is run on 1000 batch size (due to time complexity) and on 20 epochs, it is also trained using ReLU6. The results can be seen in Table VIII.

Model	**mAP**	**R1**	**R5**	**R10**	**R20**
ReLU6(600)	55.8	84.6	93.8	96.4	97.6
ReLU6(128)	57.5	85.1	94.5	96.4	97.9
FC256	54.4	83.4	93.9	96.5	98.4
ResNet18	55.2	83.5	93.1	96.0	97.7
1024-layer	55.5	85.5	94.5	97.0	98.3
1024-layer ReLU6	56.0	85.2	94.3	96.8	98.5

Table VIII: Margin and Number of Instances

Thus, it can be seen that the ResNet18 with ReLU6 and 128 batch size performs really good with 57.5 mAP as compared with ResNet 18 with extra layer which is 56. The ResNet 18 with extra layer is again tuned such that this time it is trained for 40 epochs with multistep learning rate scheduler having step size of 13, 25 which generalize really well on our data and provided mAP of 60.1 The figure 4 shows the comparison of performance of these three models. While Figure 5 shows the CMC curve of the ResNet18 with extra layer (40 epoch).

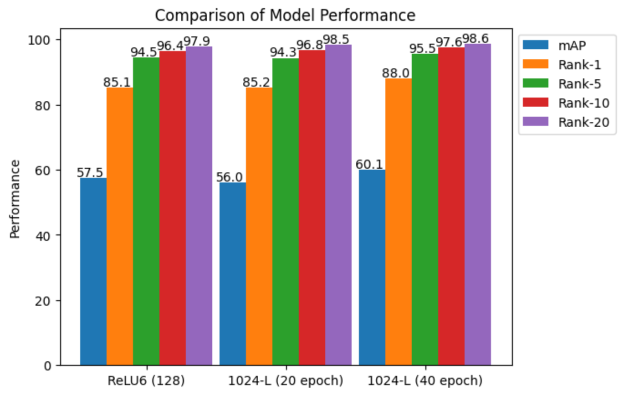


Figure 4 - Comparison of the models

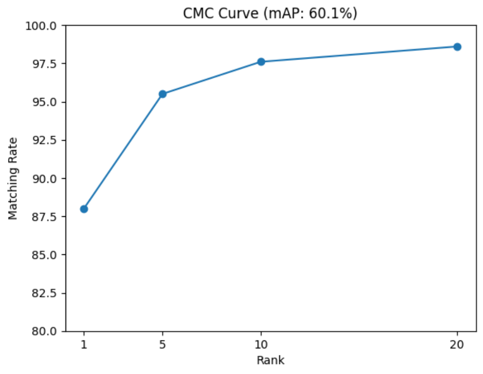
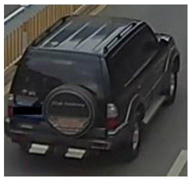


Figure 5 - CMC curve (ResNet 1024 L ReLU6)

2.4. Results and Observations

The given model was tested and the query image when passed to the model, gave us the retrieved results from gallery. The results can be seen below.



Query Image

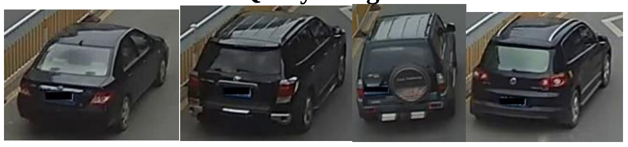


Figure 6 – Results

With this it can be seen that the ResNet 18 1024L is the best model with hyperparameters as shown in Table IX. The learning rate scheduler and ReLU6 activation function helped this model to generalize well through 40 epochs and not create vanishing gradient problem as well as overfitting. The ResNet 18 1024L (ReLU6) performed really well, looking at the batch size it was put with (1000). The model was trained on Nvidia A100 on Google Colab, which took about 40 minutes. The performance can be increased keeping the batch size less and high number of epochs with a good value of multistep learning rate scheduler. Due to computing constraints, this experiment was not tried.

HYPERPARAMETERS	VALUE
Optimizer	amsgrad
Learning Rate	0.0003
Lr- Scheduler	multi-step
Step-Size (For LRS)	13,25
Margin	0.4
Num Instances	6
Training Batch Size	1000
Epochs	40

Table IX: Hyperparameters for ResNet18 1024L (ReLU6)

3. Conclusion

In this coursework, the effectiveness of CNNs for vehicle re-identification tasks were explored. The performance of several state-of-the-art CNN architectures were evaluated including MobileNetV3, EfficientNet, RegNet-y, and ResNet18. Different optimizers such as amsgrad, adam, RMSprop, and SGD were used on ResNet18 as well as the changed architectures of ResNet18.
Through the experiments, it was found out that ResNet18 1024L using ReLU6 activation function (in BasicBlock of ResNet18) is the best performing model on the VeRI dataset, achieving a rank-1 accuracy of 88.0% and a mAP of 60.1.
This shows that CNNs are really effective for vehicle reID tasks and an optimum choice of hyperparameter can significantly improve performance. Additionally, data augmentation techniques were also experimented on and it showed an improvement on the model performance.

References

[1] Hsu, Hung-Min, et al. "Multi-camera tracking of vehicles based on deep features re-id and trajectory-based camera link models." *CVPR workshops.* 2019.
[2] Liu, Xinchen, et al. "Large-scale vehicle re-identification in urban surveillance videos." *2016 IEEE international conference on multimedia and expo (ICME).* IEEE, 2016.
[3] https://github.com/Cysu/open-reid/blob/master/reid/loss/triplet.py
[4] Zhong, Zhun, et al. "Random erasing data augmentation." *Proceedings of the AAAI conference on artificial intelligence.* Vol. 34. No. 07. 2020.
[5] Krizhevsky, Alex, Ilya Sutskever, and Geoffrey E. Hinton. "Imagenet classification with deep convolutional neural networks." *Communications of the ACM* 60.6 (2017): 84-90.

[6] Howard, Andrew, et al. "Searching for mobilenetv3." *Proceedings of the IEEE/CVF international conference on computer vision.* 2019.

[7] Jamali, Zakria, et al. "Minimizing vehicle re-identification dataset bias using effective data augmentation method." 2019 *15th International Conference on Semantics, Knowledge and Grids (SKG).* IEEE, 2019.

[8] Huynh, Su V. "A strong baseline for vehicle re-identification." *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*. 2021.

[9] Tan, Mingxing, and Quoc Le. "Efficientnet: Rethinking model scaling for convolutional neural networks." *International conference on machine learning. PMLR,* 2019.

[10] Radosavovic, Ilija, et al. "Designing network design spaces." *Proceedings of the IEEE/CVF conference on computer vision and pattern recognition.* 2020.

[11] He, Kaiming, et al. "Deep residual learning for image recognition." Proceedings of the IEEE conference on computer vision and pattern recognition. 2016.

[12] Reddi, Sashank J., Satyen Kale, and Sanjiv Kumar. "*On the convergence of adam and beyond.*" arXiv preprint arXiv:1904.09237 (2019).

[13] Kingma, Diederik P., and Jimmy Ba. "*Adam: A method for stochastic optimization*." arXiv preprint arXiv:1412.6980 (2014).

[14] Ruder, Sebastian. "An overview of gradient descent optimization algorithms." *arXiv preprint arXiv:*1609.04747 (2016).

[15] Kim, Hyungjun, et al. "Improving accuracy of binary neural networks using unbalanced activation distribution." *Proceedings of the IEEE/CVF conference on computer vision and pattern recognition*. 2021.