How Casper can Improve the Residual Deep Network?

Boran Yang

Research School of Computing Science,
Australian National University

Abstract. With the development of technology, the quality of people's daily life is also improving. In recent years, the emergence of the concept of "smart cities" has had a positive impact on people's daily life [1]. One of the most important parts of a "smart city" is an intelligent transportation system, and the successful identification of different types of vehicles through images captured by urban traffic cameras is the basis of an intelligent transportation system. In this paper, two approaches are chosen to achieve the classification task of various vehicles, the first one is the residual network based on deep CNN, especially we use ResNet50. And the second one is based on ResNet50 with an additional hidden layer constructed by Casper between the final fully connected layer and the output layer of the model. Both methods are tested separately on the vehicle-X dataset, which contains 1362 different vehicle classes. The experimental results show that the ResNet-50 with Casper could slightly improve the performance. And SGD (with momentum) is actually more suitable than RMSprop to deal with the images.

Keywords: Vehicle multi-classification, Residual Deep Network, ResNet-50, Vehicle-X dataset, Casper Algorithm

1 Introduction

With the popularity of smart cities and their related concepts, attention is gradually being focused on their subsystems intelligent transportation systems and autonomous driving. For intelligent transportation systems, its sub-tasks include but are not limited to reasonable monitoring of road conditions, identifying vehicles violating traffic laws, and planning traffic flow. And most of the sub-tasks related to intelligent transportation systems have high requirements for vehicle monitoring and classification.

The cornerstone of machine learning is data, and the quality of the data will determine how well the algorithm performs. A suitable dataset is crucial for training models. Our goal is to apply the vehicle multi-classification model to real-world scenarios. Real data will be the best choice, since the data collected in the real world is not always taken in a well-lit environment, and it may be affected by the viewing angle and shooting distance. However, it is often expensive to collect and label real-world data. The dataset chosen for this paper, Vehicle-x, is a large-scale dataset consisting of synthetic data constructed by the graphics engine Unity, which contains 1362 vehicles in various 3D models with fully editable attributes. In theory, it can generate an infinite number of images of different vehicles by modifying the relevant construction parameters. These editable parameters include many factors, such as light intensity, light direction, camera distance, etc., which can simulate the real data to a certain extent. It has been validated that pre-training on Vehicle-X or combining it with real-world data can improve the performance of the relevant models on real test data sets [2].

The task proposed in this paper is based on vehicle-X dataset and classifies 1362 3D-vehicles among them. By learning the image features of different vehicles, the deep learning model will perform the prediction of vehicle types based on the unseen images. For this multi-classification task, two CNN models are used. The first one is residual network based on deep CNN [3], which has had a landmark impact on deep learning for image recognition. Specifically, we chose ResNet50, a detailed discussion can be found in Section 2.1. The second one is based on ResNet50 with an additional hidden layer between the fully connected layer and the output layer, which is constructed by Cascade Network with Progressive RPROP (Casper) [4], an algorithm that automatically tries to add hidden neurons between the input and output layers. We test both models on Vehicle-X dataset and perform a reasonable hyperparameter tuning. Based on the final results, we can conclude that the ResNet-50 with Casper (with SGD) is slightly better than the ResNet-50. And accidentally, we find that the SGD (with momentum) is more suitable than RMSprop to deal with images.

2 Methods

In this section, we will introduce two methods we used separately in detail. The first one is residual deep network, specifically, we choose ResNet-50. Since ResNet-50 is a typical residual deep network that uses the least number of layers and consists of bottleneck residual blocks, which could significantly reduce the number of parameters. [3]. The other one is a combination of ResNet-50 and a hidden layer that constructed by Casper algorithm.

2.1 Residual Deep Network (RseNet-50)

The depth of the network is crucial for the model. Generally, deeper networks can extract more complex features [11]. However, deeper networks may not perform better. The experimental results show that the deep network may have a Degradation problem: when the network depth increases, the accuracy of the network becomes saturated or even decreases [3]. The proposal of residual deep network (ResNet) solves this problem to a large extent. ResNet is a deep learning network composed of a series of residual blocks. A residual block can be mathematically expressed as Eqn.1, where x_{l+1} denotes the vectors that considered by layer $l+1$ and $F(x_l, W_l)$ can denote multiple convolutional layers. Fig.1 illustrates the architecture of a standard residual block.

$$x_{l+1} = x_l + F(x_l, W_l) \quad (1)$$

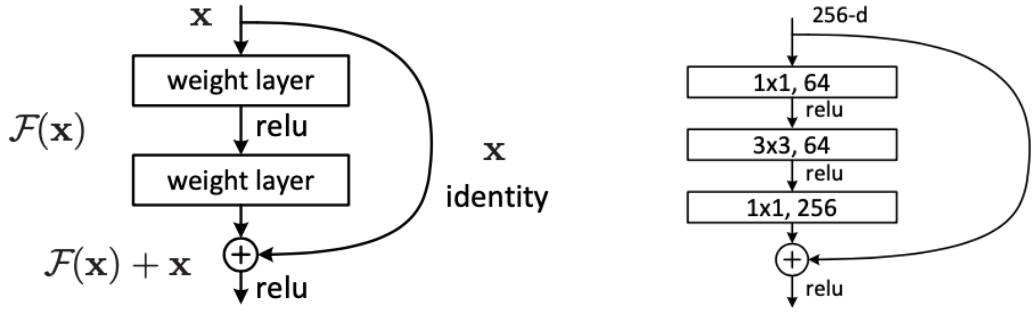

Fig. 1. A standard residual block (left) and a "bottleneck" residual block (right) [3].

As shown in Fig.1, residual block consists of two main parts, the identity mapping x (a.k.a. shortcut connection) and the normal convolutional mapping F(x). The F(x) is generally composed of two to three convolution layers, which is denoted as "weight layer" in the Fig.1. There are only two convolution layers are shown in the Fig.1 left. For the ResNet-50 we use, the normal convolutional part of each Residual block contains three convolution layers, which is shown in Fig.1 right. The residual block used in ResNet-50 is called a "bottleneck" residual block. At the end of each residual block, there is an element-wise addition operation, which means adding the normal convolution part and the the identity mapping part. However, in a convolutional neural network, the number of channels of x_l and x_{l+1} may be different. Therefore, we need to upgrade or reduce the dimensionality of the identity mapping part before the addition operation. Generally, 1×1 convolution kernel is used to change the number of dimensions. Note that the last ReLU activation function of the residual block is after the addition operation. In addition, the Batch Norm is used to control the problem of gradient explosion/gradient vanishing. The more detailed version of architecture of "bottleneck" residual block can be found in the Fig.2. And the "bottleneck" residual block with 1×1 convolution shortcut can be mathematically denoted as Eqn.2, where $h(x_l)$ is the vector with 1×1 convolutional operation.

$$x_{l+1} = h(x_l) + F(x_l, W_l) \quad (2)$$

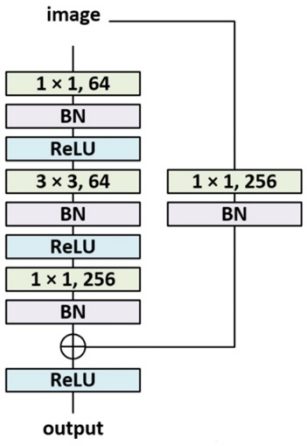

Fig. 2. Architecture of a "bottleneck" residual block.

The standard residual block and "bottleneck" residual block are discussed above. And the entire ResNet-50 consists of a series of "bottleneck" residual blocks. The architecture of entire ResNet-50 can be found in Fig.3. Each block with annotation "×m" in Fig.3 can be considered as "m" connected "bottleneck" residual blocks. There are some classical residual networks such as ResNet-18, ResNet-34, ResNet-50, ResNet-101 and ResNet-153. The detail can be found in Table.1. First two models use the standard residual blocks, and rest models use the "bottleneck" residual block since it actually requires significant less parameters, which require less memory space and the computing resources. ResNet-50 is the classical model that consists of "bottleneck" residual block but with least total layers, and that is why we choose ResNet-50 as our first model.

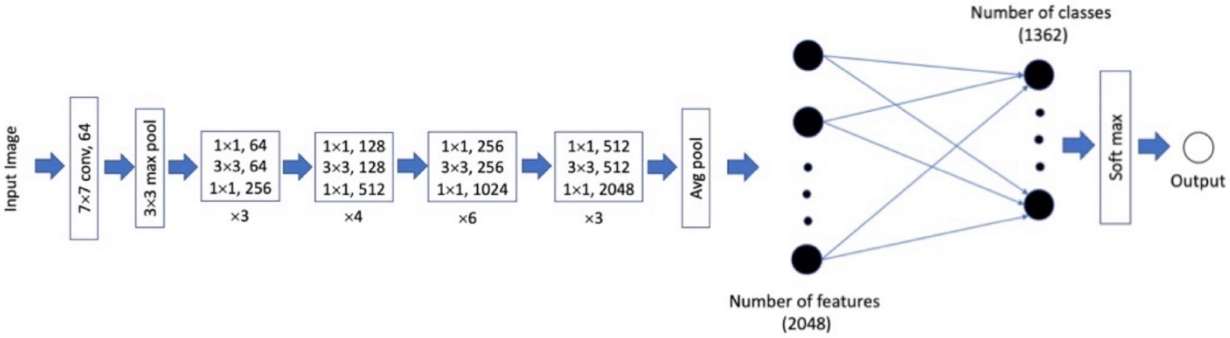

Fig. 3. Architecture of the ResNet-50.

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

Table 1. Architectures for different Residual Networks [3].

2.2 Residual Deep Network with Casper (ResNet50-Casper)

The second model is based on ResNet50 with an additional hidden layer between the fully connected layer and the output layer, which is constructed by Cascade Network with Progressive RPROP (Casper) [4]. The architecture for the first part of the ResNet50-Casper is same as the previous mentioned ResNet-50, and this section will focus on the Casper algorithm. The full architecture of ResNet50-Casper can be found in Fig.4.

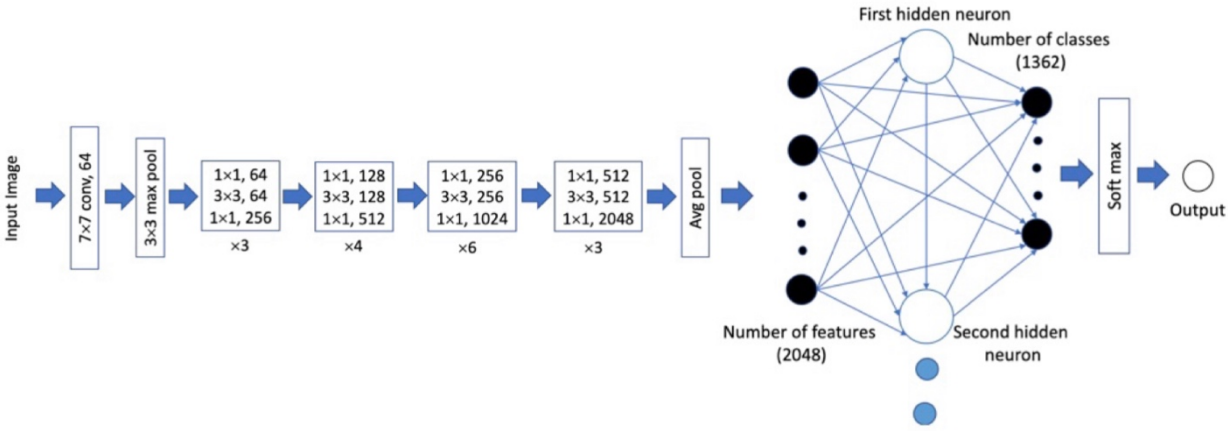

Fig. 4. Architecture of the ResNet50-Casper. And for Casper part, it demonstrates the topology that after adding second hidden neuron.

The Cascade Correlation (Cascor) algorithm was proposed by Fahlman and Lebiere in 1990. It can automatically construct a network structure [5]. The algorithm contains the following steps:

1. The network is initialized by directly full connecting all the input features to the output layer, which means there is no hidden neurons at the first step.
2. Then it will add one single hidden neuron while freezing the weights of all previous neurons.
3. The network will train the new hidden neuron with the output layer.
4. Repeat the step 2-3 until it converges or meets some specific conditions.

The Cascor algorithm is very efficient since every hidden neuron only needs to be calculated once [4]. However, the algorithm could lead the entire network to be extremely large since it freezes the weight of neurons [6].

To overcome the shortcomings of the Cascor algorithm, a new algorithm called Casper was proposed by Treadgold and Gedeon in 1997 [4]. The Casper algorithm uses the similar idea of the Cascor, but unlike Cascor, Casper takes a different strategy to deal with the weights of existing hidden neurons. The whole network can be divided into three different regions, and each region has its own learning rate. The definition of each region and the corresponding learning rate can be found as followed:

1. Region 1: weights that connected from previous hidden neurons and input features to new hidden neuron. The corresponding learning rate can be denoted as L1.
2. Region 2: weights that connected from new hidden neuron to the output layer. The corresponding learning rate can be denoted as L2.
3. Region 3: weights that for all the other connections. The corresponding learning rate can be denoted as L3.

And the relationship between L1, L2 and L3 can be denoted as L1>>L2>L3. In addition, Casper also uses weight decay to improve the generalization ability. Specifically, Casper uses the modified version of Rprop algorithm [7]. We choose to use the RMSprop algorithm [8] to replace the original Rprop algorithm since Rprop algorithm is not suitable for mini-batch training. And we use mini-batch training since the dataset is relatively large. Therefore, the RMSprop algorithm is more suitable for our situation.

As for the termination condition, the network will terminate if the training process fails to reduce a certain amount of loss in the next 15+P×N epochs, where P is a self-defined hyperparameter and N is the number of all the neurons in the network.

3 Experiments and Discussion

In this section, we make experiments for both ResNet-50 model and ResNet50-Casper model on vehicle-X dataset and evaluate their performance. We first introduce the dataset, the data preprocessing and the evaluation metrics we used, and then we describe the hyperparameter tuning process. Finally, the two models are compared and discussed based on the relatively optimal combination of hyperparameters that we derived. Note that this experiment utilizes cloud-based GPU computing resources, GeForce RTX 2080 Ti and Tesla V100-SXM2-16GB.

3.1 Dataset

Since the cost of collecting and labeling a large amount of real-world data is very expensive, the data used in this experiment are all derived from the vehicle-X dataset. The data in the dataset is synthesized by the image engine Unity, which provides convenience and flexibility for the research of computer vision field. It contains a total of 1362 vehicles in various 3D models with fully editable attributes. And by modifying the relevant parameters, theoretically unlimited images can be generated. However, this article only uses the original images of 1362 different vehicles in the dataset.

3.2 Data Pre-processing

The entire vehicle-X dataset is divided into training set, validation set and test set, which contain 45438, 14936 and 15142 images respectively, and the ratio of the number of images they contain is about 3:1:1. Before starting the training, the size of each image is resized from 256*256*3 to 64*64*3. Since 64 is a relative appropriate size, it can improve the training speed, reduce the demand on computing resources, and it will not lose too much information of the original images.

During the training of a neural network, especially when training on a large dataset, the training speed can be improved by normalization. For unnormalized input features, the gradient descent algorithm may require more iterations to find a minimum, and it usually uses a small learning rate. However, for normalized input features, the gradient descent algorithm is able to find the minimum more directly, regardless of the position where it starts. Therefore, a larger learning rate can be used in the gradient descent algorithm to speed up the training process. The formula of normalization can be found in Eqn.3, where it normalizes the input image channel by channel. We choose the mean = [0.485,0.456,0.406] and std = [0.229, 0.224, 0.225] for RGB channel. The value of mean and std are coming from random sampling from the ImageNet.

$$output[channel] = \frac{input[channel] - mean[channel]}{std[channel]} \quad (3)$$

3.3 Evaluation Metrics

We use two different metrics to evaluate the performance of network. The first one is accuracy. Accuracy is the ratio between correct prediction samples and the total samples. The second one is mAP (mean average precision) [9], which can balance the relationship between precision and recall.

3.4 Hyperparameter tuning for ResNet-50

The hyperparameters for ResNet-50 are batch size, number of epochs and learning rate. We initially set the number of epochs to be 70, size of the mini batch to be 64 and the learning rate to be 0.1 base on our empirical experience. And we also set an adjusting strategy that changes the learning rate with the training process. Given a metric (loss), when the process stops to optimize, the learning rate will be adjusted by becoming the one tenth of the current learning rate. Specifically, if the metric (loss) is greater than the value of best metric (lowest loss) × (1-1e-4) for five consecutive epochs, then the current learning rate will be updated by Eqn.4. In addition, the minimal learning rate is set to 1e-5.

$$lr_{new} = lr_{old} \times \lambda, \text{where } \lambda = 0.1 \quad (4)$$

As you can see in the Fig.5, the loss of both of training set and validation set are becoming very low before the 50th epoch. That indicates the training process converge before the 50th epoch. Therefore, we adjust the number of epochs to 50. And base on that, we test and compare different size of the mini-batch and different initial learning rate. The results of comparison can be found in Table 2 and Table 3. Therefore, we finally choose the number of epochs to be 50, size of the mini batch to be 64 and the learning rate to be 0.1.

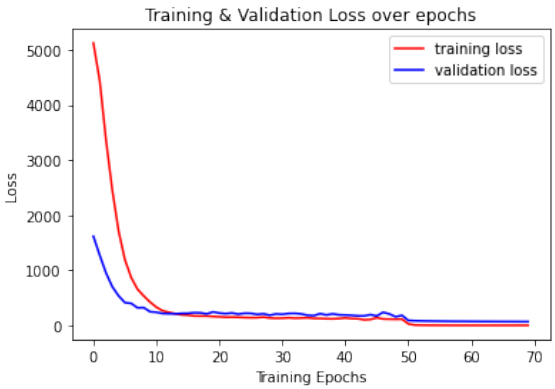

Fig. 5. The visualization of training loss and validation loss over 70 epochs. Size of mini batch is 64, initial learning rate is 0.1, optimizer is SGD.

Size of mini batch	mAP on test set (%)
32	82.24
64	**93.09**
128	92.02
256	90.30

Table 2. Comparison of different size of mini batch over 50 epochs.

Initial learning rate	mAP on test set (%)
0.1	**93.09**
0.05	92.88
0.01	92.20

Table 3. Comparison of different initial learning rate over 50 epochs.

3.5 Hyperparameter tuning for ResNetCasper-50

Firstly, we try to use the same adjusting strategy for learning rate as we used in ResNet-50. The optimizer for CNN is SGD and the optimizer for Casper is RMSprop. Ans we also choose the size of mini batch to be 64 and the initial learning rate to be 0.01. To make sure the network can converge, we initially set the number of epochs to be 200. In this case, the network converges at the 99th epoch with three hidden neurons in the Casper layer. The loss of training set and validation set are visualized in the Fig.6. We found that the loss will rocket to a very high level every time the new hidden neuron was added. And the validation set loss at the final convergence (with three hidden neurons) is actually higher than the loss when there is only one hidden neuron, which indicates the fact that the training process actually meet negative optimization.

After careful observation and analysis, we propose a hypothesis that the previous learning rate update strategy is not suitable in this case. After the first hidden neuron is added, learning rate starts to change with the update strategy, but this update strategy will make learning rate monotonically decrease. Therefore, after the second hidden neuron is added, the learning rate will always be trained with the updated value (the value of learning rate is usually the lower bound in the update policy, in this case it is 1e-5). As shown in the Fig.6, each addition of a neuron results in a very high loss. In this case, training with a very small learning rate can lead to a very slow training process and it is very easy to fall into a local optimum.

Based on the proposed hypothesis, we try to change the learning rate update strategy and increase the lower bound of learning rate in order to train the model with a not too small learning rate even after the addition of hidden neurons, so that it would not easily fall into a local optimum solution. We tried several different combinations of learning rates,

which contain different initial learning rates and different lower bounds. The results can be found in Table 4. We can notice that the higher lower bound of learning rate could lead a better performance. However, there is still a significant difference between the ResNet-50 and the ResNet50-Casper.

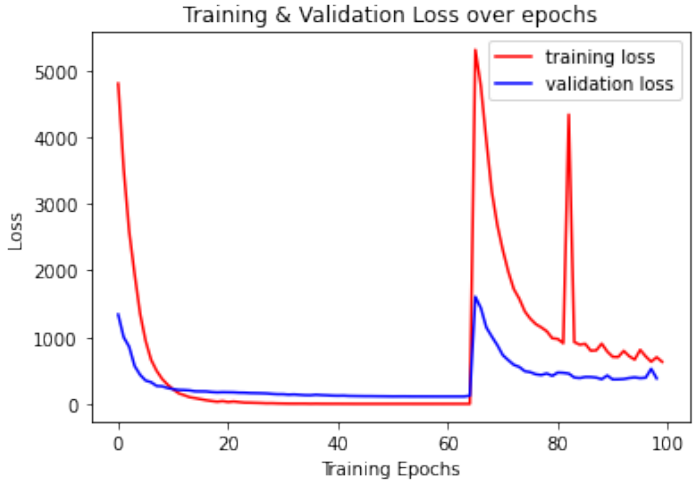

Fig. 6. The visualization of training loss and validation loss over 99 epochs. Size of mini batch is 64, initial learning rate is 0.01, the CNN optimizer is SGD, and the Casper optimizer is RMSprop.

Initial learning rate	Lower bound of learning rate	mAP on test set (%)
0.01	0.001	**77.91**
0.01	0.0001	71.83
0.01	0.00001	72.81
0.1	0.001	74.05
0.1	0.0001	74.17
0.1	0.01	77.22

Table 4. Comparison of different combinations of learning rates. For the update strategy of learning rate, each combination contains different initial learning rate and different lower bound. ResNet50-Casper with RMSprop.

Since there is still a significant difference between the performance of ResNet-50 and ResNet50-Casper, we try to make some further adjustments on the ReNet50-Casper. Originally, we use SGD as optimizer for the CNN part of the ResNet50-Casper, and the layer that constructed by Casper uses the RMSprop as the optimizer, since the paper that proposed Casper suggested to use Rprop. We use RMSprop since the training process use the mini batch and Rprop cannot handle the mini batch training process. Now we try to replace the RMSprop in the layer that constructed by Casper with SGD optimizer. By comparing different combinations of the learning rate in the updates strategy, we obtain that the current best combination is initial learning rate with 0.1 and the lower bound is 0.001. The results of comparison can be found in Table 5.

Initial learning rate	Lower bound of learning rate	mAP on test set (%)
0.1	0.001	**93.66**
0.01	0.001	91.49

Table 5. Comparison of different combinations of learning rates. For the update strategy of learning rate, each combination contains different initial learning rate and different lower bound. ResNet50-Casper with SGD.

3.6 Results and Discussion

Base on the previous discussion of the tuning hyperparameters, we compare the ResNet-50, the ResNet50-Casper (with RMSprop) and the ResNet50-Casper (with SGD). The final results can be found in the Table 6. And the results indicate that the performance of ResNet50-Casper (with SGD) is significantly better than that of the ResNet50-Casper (with RMSprop), which indicates that the SGD is actually more suitable to deal with images. RMSprop is an optimizer with adaptive learning rate, it is not good at finding the flat minima, which is an important factor to the generalization. Therefore, the training loss of RMSprop could be relatively low, but the test performance is worse than the SGD.

In addition, the results indicate that the performance of ResNet50-Casper (SGD) is just slightly better than that of ResNet-50. The layer that constructed by Casper is between the final fully connected layer and the output layer. The input features for final fully connected layer is 2048 and the neurons for output layer is 1362. The number of weights between them could be very large, and the layer that constructed by Casper usually contains two or three hidden neurons. The two or three hidden neurons could only cause a slight influence on the entire deep network. Therefore, the performance of ResNet50-Casper is mainly based on the performance of the residual deep network.

Model	mAP on test set
ResNet-50	**93.09**
ResNet50-Casper (RMSprop)	77.91
ResNet50-Casper (SGD)	**93.66**

Table 6. Comparison of different models. Note the RMSprop and SGD are only for the layer that constructed by Casper.

4 Conclusion and Future Work

We make experiments and test ResNet-50, the ResNet50-Casper (with SGD) and ResNet50-Casper (with RMSprop) on the vehicle-X dataset. Our goal is to do vehicle multi-classification base on the images of 1362 different vehicles in the dataset. We find that the performance of ResNet50-Casper (with SGD) is significantly better than that of ResNet50-Casper (with RMSprop). We believe the SGD with momentum is more suitable than RMSprop to deal with images tasks. Since the RMSprop is an optimizer with adaptive learning rate, it can be easier to find the sharp minima, but it is not good at finding the flat minima. And flat minima are important to be considered that it has the generalization. In other words, the RMSprop could have lower training loss but usually end up with a worse test performance in image tasks. Besides, the performance of ResNet50-Casper (with SGD) has slightly improvement on that of ResNet-50. Since the hidden layer constructed by Casper end up with only two or three hidden neurons, and a few hidden neurons cannot have a lot influence on the whole deep network. However, training a model with Casper usually costs significant more time.

For the future work, the Transformer model [10] that used the self-attention mechanism perform a better performance than the normal CNN. It is a direction that we can try to optimize the Casper and try to apply the idea of it on the Transformer architecture.

References

1. Z. Ozcelik, C. Tastimur, M. Karakose, and E. Akin, “A vision based traffic light detection and recognition approach for intelligent vehicles,” in International Conference on Computer Science and Engineering (UBMK’17), pp. 424–429, 2017
2. Yao, Y., Zheng, L., Yang, X., Naphade, M., Gedeon, T.: Simulating content consistent vehicle datasets with attribute descent. In: ECCV (2020)
3. He, K., Zhang, X., Ren, S., Sun, J.: Deep residual learning for image recognition. In: CVPR. (2016)
4. N. K. Treadgold and T. D. Gedeon, A cascade network algorithm employing progressive rprop," in Proc. of the Int. Work-Conf. on Articial and Natural Neural Systems, Lanzarote, June 1997, pp. 723-732.
5. Fahlman, S. E., and Lebiere, C. The cascade-correlation learning architecture. In Advances in Neural Information Processing Systems 2 (1990), Morgan Kaufmann, pp. 524–532.
6. T.-Y. Kwok and D.-Y. Yeung, “Experimental analysis of input weight freezing in constructive neural networks," in Proc. of the IEEE Int. Conf. on Neural Networks, San Francisco, Mar. 1993, pp. 511-516
7. Riedmiller, M. and Braun, H. (1993). A direct adaptive method for faster backpropagation learning: The Rprop algorithm. In Proc. IJCNN, pages 586–591. IEEE Press.
8. Graves, Alex. Generating sequences with recurrent neural networks. arXiv preprint arXiv:1308.0850, 2013.
9. Y. Yue, T. Finley, F. Radlinski, and T. Joachims. A support vector method for optimizing average precision. In 30th Annual International ACM SIGIR Conference on Research and Development in Information Retrieval, pages 271–278, 2007.
10. Parmar, N., Vaswani, A., Uszkoreit, J., Kaiser, Ł., Shazeer, N., and Ku, A. (2018). Image Transformer. ArXiv e-prints.
11. Peng, D., et al., A novel deeper one-dimensional CNN with residual learning for fault diagnosis of wheelset bearings in high-speed Trains. IEEE Access, 2019. 1(7): p. 10278-10293.