# Use Block Casper to Improve Transfer Learning in VehicleX data

Hao Shi

Research School of Computer Science, Australian National University Canberra Australia
U7077338@anu.edu.au

**Abstract.** This paper proposes an improved version of Casper, we call it BlockCasper. The BlockCasper net consists of step-by-step added two-layer partial residual "Blocks" and benefits from a three-level learning rate mechanism. We applied Block Casper to image classification tasks in VehicleX dataset. Block Casper was introduced to replace the final fully connected layer in Resnet, Densenet and Googlenet. The CNNs was transfer learned from ImageNet pretrained weights. VehicleX with a 1362 class label was utilized to evaluate the performance of the proposed architecture. Evaluation upon test set shows that BlockCasper helped to improve the test accuracy of 1.59% in Densenet, 2.78% in Resnet and 1.66% in Googlenet.

**Keywords:** Cascade, Casper, VehicleX, Finegrained Classification, Image Classification, Transfer Learning

## 1 Introduction

Transfer learning has been a well-used technique in many areas, image classification is one of these. Many CNN architectures have been proposed in recent years, like ResNet, DenseNet, GoogleNet, EfficientNet and so on. Finetuning pretrained model in particular dataset can decrease the training time cost and make model converging easily.

Network structure needs hardly exploration to be determined nowadays. As a self-determine network technology, an advanced neural network based on Cascor(Adams and Waugh, 1995), Casper (Treadgold and Gedeon, 1997) has the characteristic that the output layer is connected to all input and hidden neurons. Resnet style skip connection (He, Zhang and Ren, 2016 ) structure has a similar topology and has been widely used in various areas. Casper has the potential to be explored to improve the performance in modern datasets.

Casper is neuron based architecture and is hard to be used in CNN layers. However, All CNN models adapt fully connected layer as classifier in the final layer. This gives the potential to improve the CNN models with Casper. In this essay we will use improved Casper to replace the final fully connected layer in CNNs in transfer learning, aim to improve the performance of transferred CNNs.

### 1.1 Casper

Casper (Tom,1993) is a Cascade-style network based on Cascor. Casper has only connections between inputs and outputs at the beginning. After certain number of iterations, a new hidden neuron is added repeatedly until loss converges. All inputs and previous hidden neurons are connected to the new one, the output of the neuron is connected to all output neurons of the network.

A distinctive characteristic of Casper is the three-level different learning rates. When a new neuron is added, all weights connecting to the new neuron from the previous input and hidden neurons adapt L1 as the learning rate; the weights between the new neuron and the outputs use L2; all remaining adapt L3. L1>>L2>L3 (Treadgold and Gedeon, 1997). Outputs are connected to all hidden and input neurons in Casper, this enables Casper to avoid gradient vanish and explosion problem.

### 1.2 VehicleX Dataset

VehicleX dataset was proposed by ANU and NVIDIA in 2019(Yue, 2019). It contains synthetic vehicle pictures Created in Unity from 1362 vehicles (vehicleID) of various 3D models. Pictures are generated with 11 editable attributes, including light intensive, camera height, orientation, etc. You can see details in table 1.

| Label | Type | Range/Classes | Description |
|---|---|---|---|
| camDis | Float | 10.1-22.1 | Distance between the camera and the vehicle. |
| camHei | Float | 4.0-12.0 | Height of the camera |
| lightDir | Float | 45-120 | The direction of light, from 45° to 120°. |
| linghtInt | Float | 0-2.7 | Intensive of light |
| orientation | Float | 15.1-345.9 | The direction of the head of the vehicle. |
| cameraID | Category | 20 | Each distinctive number represents a certain position of the camera |
| colorID | Category | 12 | Paint color of the vehicle. |
| typeID | Category | 11 | A rough classification of vehicles. |
| vehicleID | Category | 1362 | A Finegrained classification of vehicles. |

**Table 1.** Generating attributes of each picture in VehicleX

The V2 version of VehicleX dataset provides the original images. The whole dataset contains 75516 observations, and is split to train set, validation set, and test set with 45438, 14936, 15142 observations respectively. In this paper, we will evaluate the 1362 classes label VehicleID only.

### 1.3 Task Description

VehicleX is chosen because of its close connection with the modern computer vision area. This dataset has 5 regression labels and 4 classification labels to be capable to test a technique. Also, predicting picture attributes is meaningful in the modern computer vision area, this may be applied in violation detection and automatic drive. So we construct a task of predicting the vehicle classification label of 1362 classes.

The original paper for VehicleX aimed at object detection(Yue, 2019) which is different from our task. Our task is to improve the performance of classification of VehicleID label in VehicleX dataset of CNNs including DenseNet, ResNet and GoogleNet. So the comparing benchmark of this task will be the unimproved version of these three CNNs.

## 2 Method

Experiments were executed in a computer with Intel Xen X5650 CPU, 48GM RAM, NVIDIA 3090 GPU. The operating system is Ubuntu 20.04.2. NVIDIA CUDA version is 11.2. We used Pytorch 1.8.1 under Python 3.8.5 to construct our models.

### 2.1 Data Preparing

The original data we used was the pictures of different vehicles with fine-grained label of 1362 classes. We used a popular method to process the images, that is, transferring images into tensors with 3 channels, each channel stand for Red, Green and Blue. 64 * 64 matrixes stand for the pixels in each channel. Pixels were normalized with the distribution of ImageNet, which was means of 0.485, 0.456 and 0.406; standard deviation of 0.229, 0.224 and 0.225. Labels was clearly given, the only thing we did in the label was subtracting 1 so that the labels was range from 0 to 1361, this made the label suitable for pytorch.

### 2.2 Block Casper

Casper was proposed in 1997. Which is far from now. A lot of new techniques have been proposed, like Adam optimizer(Kingma and Ba, 2014), batch normalization(Loffe and Szagedy, 2015), ReLU activation(Glorot, Bordes and Bengjo, 2011), and dropout(Srivastave, Hinton and Krizhevsky, 2014), etc. Casper has the potential to be improved by these techniques.

Modern datasets have been more and more complicated. Take VehicleX for instance, there are 3*64*64 inputs and 1362 outputs in vehicleID label. Hundreds even thousands of hidden neurons could be needed. Each time only one neuron is added in Casper, the whole network needs to be trained in increasing epochs (Treadgold and Gedeon, 1997). This costs a very long time to achieve abundant hidden units. The original task of Casper when proposed had only 2 inputs and 2 classes. Facing the task with thousands of input and output, new added neurons may struggle to make improvement in loss given there are already thousands of cross-connections in Casper.

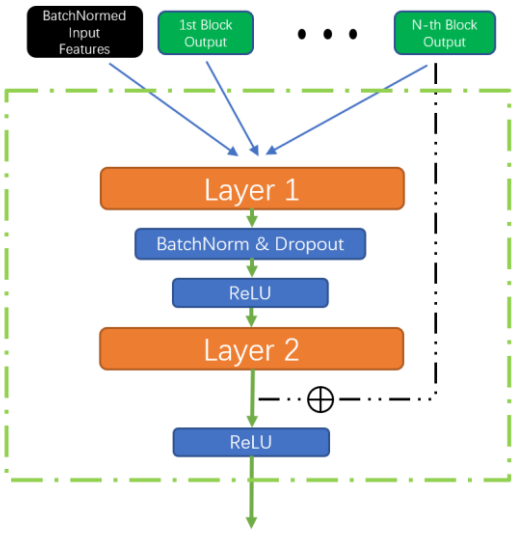

**Fig. 1.** Sketch map of the N+1 th "Block" in BlockCasper. There are bias terms in the two hidden layers.

Above all and after a lot of experiments, we propose an improved version of Casper: BlockCasper. Instead of adding a neuron a time, Block Casper adds a "Block" each time. A Block consists of two connected linear layers. A hyperparameter Block-Size determines the neuron number of the two hidden layers. Inputs after batch normalization and all previous hidden Block outputs are connected to layer 1. Batch normalization, dropout and ReLU activation follow behind layer1. Layer 2 follows behind but has no dropout and batch normalization. If the new Block is not the first one, a partial residual connection(black dotted line in Fig1) is added before the activation of Layer 2, the added part is the output of the previous block.

It is worth noting that we did not abandon the direct connection between input and output, which is different from previous work, the reason why we do this is that the task is to improve the CNN, and the direct connection could inherit the weights of the last fully connected layer of CNN. We use Adam optimizer instead of Progressive RProp in Casper(Treadgold and Gedeon, 1997). Three-level learning rates mechanism is retained, the parameters inside the new added Block utilizes L1, weights connecting the new Block and outputs use L2 and all others adapt L3, L1>>L2>L3. Given that there are no direct connections, the initial state of a BlockCasper contains 1 hidden block without a partial residual connection. In the initial state, BlockCasper utilizes L1 learning rate for all parameters.

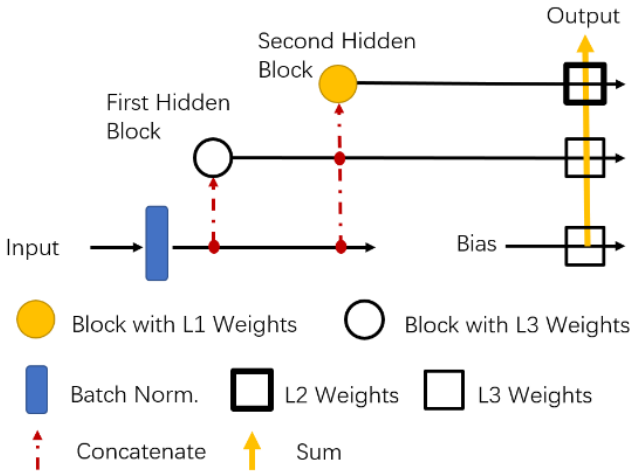


**Fig. 2.** Sketch map of BlockCasper. Note: Partial residual connection of the second hidden Block is not showed in this picture.

Another improvement is the iteration determination mechanism, Instead of iterating 15+P*N epochs after each neuron added in Casper, Block Casper uses early stopping() to determine iterations automatically. Early stopping is used in two parts of BlockCasper, ES1 and ES2. ES1 is reset each time a new block is added, when the validation metric does not decrease(increase for accuracy) in certain number of epochs, stop training the current Block and go back to the weights that perform best. ES2 evaluates whether a new block improves the validation performances, each time ES1 gives a stop training signal, ES2 evaluates the validation metric. When a certain number of new Blocks cannot improve the validation performance, trace back to the best performance architecture and weights and the training process stops. The patience of ES1 and ES2 are 20 and 3.

BlockCasper take the advantage of Cascade-style networks, each Block is connected with the output layer(Treadgold and Gedeon, 1997). Similar to Resnet skip-connection architecture(He, Zhang and Ren, 2016 ), Cascade-style structure can properly avoid gradient explosion and vanish, partial residual connection in hidden Blocks may contribute more to avoid these problems. Early stopping mechanisms can prevent the model from over-fitting and keep the most suitable structure and weights for the task(Prechelt, 1998). Moreover, a "Block" with 2 hidden layers and partial residual connection instead of a single neuron added each time can avoid the problem that "Single neuron challenges thousands of initial neurons" mentioned before in large input and output dimension dataset.

In this paper, L1 is set to be the same as the learning rate in transfer training CNNs and L1=40*L2=200*L3, the ratios are consistent with Casper. Dropout rate is 0.2, ReLU activation is used throughout the net.

### 2.3 Experiment Design

We used ImageNet pretrained CNNs including DenseNet121, ResNet18 and GoogleNet to be trained with our dataset. There was an replacement of the final fully connected layer with a new fully connected layer of 1362 output neurons because the VehicleID label in VehicleX has 1362 classes which is different from ImageNet. At this period, learning rate was set to be 0.0005 which was determined with many attempts. The number of epochs was determined by early stopping with patience of 5. Batch size of 64 were adapted because of the limitation of GPU memories.

After transfer learning in the first period, BlockCasper was introduced to replace the final fully connected layer of CNNs, the direct connection in BlockCasper inherited the weights of the final fully connected layer in CNNs. When training BlockCasper, all previous layers of CNNs were frozen and no longer trainable. In practice, in order to reduce duplicated calculating, we defined an extra BlockCasper net and exported the output before the fc layer of CNNs to be the input of the BlockCasper net. After converging, BlockCasper and the backbone of CNNs were combined to be the final model. The learning rate and batch size remained the same in the second period and the block size of BlockCasper was 200, it is worth noting that there is a dropout rate of 0.2 in BlockCasper.

Then the performance was evaluated in both transferred CNNs and the BlockCapser-CNNs.

### 2.4 Difference from Previous Work

This time we use BlockCasper to function as the last fully connected layer in a trained CNN, abandoning the direct connection may waste the pretrained weights, so we did not abandon the direct connection in BlockCasper.
In the Codes, we aggregated all util py files to a single file to make codes easier to be read. Previous work handled all labels of VehicleX dataset and this paper processed only VehicleID, previous work was done with the 2048-dimension embedded features while this paper adapted the original picture, the corresponding model architecture part was modified to be adapted to these changes.

## 3 Result and Discussions

Table 2 shows the training cost and performances in training set of transferred CNNs and BlockCasper-improved CNNs. Training BlockCasper costs additional time and epochs, but we could see the cost is worth to pay, the training loss and accuracy have been significantly improved. Almost all accuracies have been closed or equal to 100 percent, and we will see this is not overfitting later in table 3.

| | **Transferred CNNs** | | | | | **BlockCasper CNNs** | | | | | |
|---|---|---|---|---|---|---|---|---|---|---|---|
| | Loss | Acc.% | Top-5 Acc.% | Epochs | Time Cost(s) | Loss | Acc.% | Top-5 Acc.% | Epochs | Time Cost(s) | Block Num |
| DenseNet121 | $3.97\times10^{-2}$ | 98.94 | 99.95 | 18 | 3959 | $\mathbf{1.21\times10^{-3}}$ | **99.98** | **100** | 184 | 687 | 3 |
| ResNet18 | $3.83\times10^{-2}$ | 99.10 | 99.97 | 20 | 1687 | $\mathbf{1.23\times10^{-4}}$ | **100** | **100** | 379 | 2575 | 9 |
| GoogleNet | $2.36\times10^{-2}$ | 99.32 | 99.60 | 19 | 1890 | $\mathbf{1.17\times10^{-4}}$ | **100** | **100** | 292 | 1094 | 3 |

**Table 2.** Training cost and performances.

Table 3 shows the performance of unimproved and improved version of CNNs in validation and test set, embedding BlockCasper to CNN has decreased the losses and increased the accuracies in all three CNNs. Concentrating on test set, ResNet18 benefit most from this inception, error rate was decreased from 5.7% to 2.92%. CNN models gained an average of 2.09% increase of accuracy. The average error rate has been decreased from 4.66% to 2.57%. Top 5 error was decreased from 0.477% to 0.313%. The average loss was optimized from 0.167 to 0.122.

| | | **Transferred CNNs** | | | | | **BlockCasper CNNs** | | | | |
|---|---|---|---|---|---|---|---|---|---|---|---|
| | | Loss | Acc.% | Top-5 Acc.% | Err. % | Top-5 Err.% | Loss | Acc.% | Top-5 Acc.% | Err. % | Top-5 Err.% |
| Val Set | DenseNet121 | $1.32\times10^{-1}$ | 96.40 | 99.60 | 3.60 | 0.40 | $\mathbf{7.70\times10^{-1}}$ | **98.23** | **99.74** | **1.77** | **0.26** |
| | ResNet18 | $2.11\times10^{-1}$ | 94.48 | 99.31 | 5.52 | 0.69 | $\mathbf{1.21\times10^{-1}}$ | **97.14** | **99.55** | **2.86** | **0.45** |
| | GoogleNet | $1.48\times10^{-1}$ | 95.70 | 99.65 | 4.30 | 0.35 | $\mathbf{1.20\times10^{-2}}$ | **97.44** | **99.70** | **2.56** | **0.30** |
| Test Set | DenseNet121 | $1.31\times10^{-1}$ | 96.26 | 99.70 | 3.74 | 0.30 | $\mathbf{8.63\times10^{-2}}$ | **97.85** | **99.79** | **2.15** | **0.21** |
| | ResNet18 | $2.15\times10^{-1}$ | 94.30 | 99.27 | 5.70 | 0.73 | $\mathbf{1.56\times10^{-1}}$ | **97.08** | **99.62** | **2.92** | **0.38** |
| | GoogleNet | $1.56\times10^{-1}$ | 95.47 | 99.60 | 4.53 | 0.40 | $\mathbf{1.25\times10^{-1}}$ | **97.36** | **99.65** | **2.64** | **0.35** |

**Table3.** Performance comparison in validation and test sets.

## 4 Conclusion and Future Work

This paper proposed an improved version of Casper, BlockCasper. And used BlockCasper to replace the final fully connected layer in three CNN models, this inception has significantly improved the performance of transfer learned CNNs in VehicleX dataset.

BlockCasper has better performance than MLP(Hao, 2021), the replacement of the classifier(last fully connected layer) of CNNs with BlockCasper help improved the performance. However, BlockCasper has its shortcomings, the architecture that each block is connected to previous blocks costs a lot time to train and inference. Those who would like to improve CNNs with BlockCasper should balance the cost and the profit.

This paper also contains shortcomings. CNNs was trained with steady learning rate did not explore the entire potential of the models. Nowadays graduate decreasing learning rate mechanism like cosine decrease are used in almost all CNNs. Whether BlockCasper can improve the completely trained CNNs is worth exploring in the future.

## 5 References


Treadgold N K, Gedeon T D. A cascade network algorithm employing progressive RPROP[C]//International Work-Conference on Artificial Neural Networks. Springer, Berlin, Heidelberg, 1997: 733-742.

Huang G, Liu Z, Van Der Maaten L, et al. Densely connected convolutional networks[C]//Proceedings of the IEEE conference on computer vision and pattern recognition. 2017: 4700-4708.

Szegedy C, Liu W, Jia Y, et al. Going deeper with convolutions[C]//Proceedings of the IEEE conference on computer vision and pattern recognition. 2015: 1-9.

He K, Zhang X, Ren S, et al. Deep residual learning for image recognition[C]//Proceedings of the IEEE conference on computer vision and pattern recognition. 2016: 770-778.

Adams A, Waugh S. Function evaluation and the cascade-correlation architecture[C]//ICNN. 1995, 2: 942-946.

Yao Y, Zheng L, Yang X, et al. Simulating content consistent vehicle datasets with attribute descent[J]. arXiv preprint arXiv:1912.08855, 2019.

Ioffe S, Szegedy C. Batch normalization: Accelerating deep network training by reducing internal covariate shift[C]//International conference on machine learning. PMLR, 2015: 448-456.

Kingma D P, Ba J. Adam: A method for stochastic optimization[J]. arXiv preprint arXiv:1412.6980, 2014.

Glorot X, Bordes A, Bengio Y. Deep sparse rectifier neural networks[C]//Proceedings of the fourteenth international conference on artificial intelligence and statistics. JMLR Workshop and Conference Proceedings, 2011: 315-323.

Srivastava N, Hinton G, Krizhevsky A, et al. Dropout: a simple way to prevent neural networks from overfitting[J]. The journal of machine learning research, 2014, 15(1): 1929-1958.

Prechelt L. Early stopping-but when?[M]//Neural Networks: Tricks of the trade. Springer, Berlin, Heidelberg, 1998: 55-69.