Improved CasPer algorithm: CasPer with Tower-Cascade Architecture

Fangxu Zhao,
Research School of Computer Science
Australian National University, Canberra, Australia
u6822201@anu.edu.au

Abstract. Constructive neural networks with cascade architechture are powerful feedforward neural networks. However, there still some problems facing in current state of art. This paper aims to explore the improvement of applying Tower Architecture for depth limiting to CasPer algorithm. In this method, series of cascade towers with same maximum size are built in training process, which control the maximum depth of network and maintain the advantages brings by constructive cascade architecture. The experiments show that applying Tower-Cascade architecture have an obvious improvement on the model performance of Loss and Accuracy.

Keywords: CasPer algorithm, image-manipulation, eye gaze.

1 Introduction

CasPer algorithm builds Constructive Cascade Neural Network, which is a feedforward neural network in which the network architecture is built during the learning process, similar to Cascade Correlation (CasCor) [2]. Building the neural network during the learning process allows it to obtain a good match between network complexity and the complexity of the problem to solve [1].

CasPer inserts the hidden neurons one at a time, the new hidden neuron takes previous hidden neurons' outputs as part of its input, which allows the new hidden neuron to learn the remaining network error, so that less hidden neurons will be needed than original neural network.

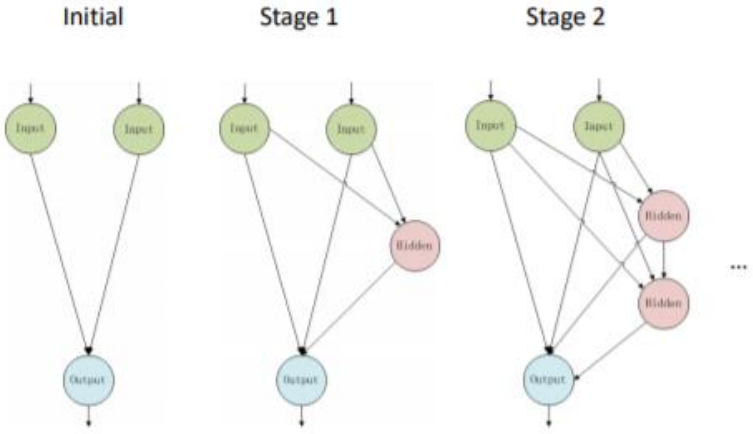

Fig. 1. Cascading topology of Cascade Architecture

As you can see, the input size of the new hidden neuron keeps increasing with the construction of the network, which leads to the exponential increasing of the total weights of the network.

To avoid this, the limitation of the network depth is introduced. The maximum depth limit is set before the construction of the network, when the limit is reached, the next hidden neuron begins a new cascade [7]. In this case, the total weights of the network will increase linearly, which ensures that the complexity of the network will not exceed the complexity of the problem, keep the structure simple and efficient.

Meanwhile, the goal of this research is that the reduction of complexity should not hurt the performance, and have better training performance.

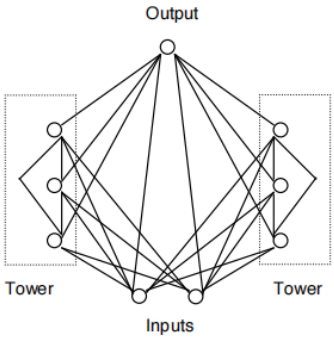

Fig. 2 A cascade tower architecture with a tower size of 3 [7]

The improvement in structure is multifaceted, such as faster convergence during training, better prediction accuracy at the end of training, and so on. A detailed comparison and analysis will be presented in the following sections. The study of the improvements brought by structural improvements can assess the effectiveness of the efforts made in this direction and explore possible directions for future improvements.

2 Dataset and Method Implementation

2.1 Dataset Inspection and Preprocessing

The dataset I use is that contained in the paper presented by Sabrina Caldwell, Tamás Gedeon, Richard Jones, Leana Copeland [4]. Which contains the participants' use of eye gaze, whether the observed image has been manipulated and their judgement as to whether the image has been manipulated.

The objectives of this dataset were based on how we use our eye gaze when looking at an image to find out whether it is possible to determine whether the image is manipulated; secondly, to find out whether it is possible to determine what the participants vote (their judgment on the image that they are looking at. Manipulated, Unmanipulated, or Do not know).

Table 1. The data that dataset contains.

Headings of data	Type	Meanings
participant	Integer	Id number of the participants (integer between 1 and 80)
num_fixs	Integer	Total number of fixations by the participant when looking at the image
fixs_duration	Float	Total amount of time (in seconds) that the participant spent looking at the image
num_man_fixs	Integer	Total number of fixations by the participant when looking within the target area
man_fixs_dur	Float	Total amount of time (in seconds) that the participant spent looking within the target area
image	Integer	The image id (integer in [10, 11, 12, 13,14, 15])
image_manipulated	Integer	Whether the image the participant views is the manipulated or unmanipulated version (0 = unmanipulated, 1 = manipulated)
votes	Integer	The verbal opinion of the participant as to whether the image is manipulated or unmanipulated (0 = voted unmanipulated, 1 = voted manipulated, 2 = don't know)

```
mean num_fixs:  81.84677419354838
mean man_fixs:  16.06451612903226
mean fixs_dur:  18.94441129032258
mean man_dur:  3.7771505376344088
```

Fig. 3 mean value of each feature.

Among all these data, the “image id” and the “participants” are redundant features. After dropping these columns, the total number of features for training is 4, with the last two features as labels for prediction, which is not much for training. So feature extraction or feature selection will not be considered in this case.

The dataset contains 5 images, with its manipulated version. Which presented as follow:

Fig. 4. Images provided (in order 10, 11, 12, 13, 14).

In terms of the objectives of the dataset, we do not need to consider the images and the participants themselves, but only the way in which the participants used their gaze, in relation to both whether the images were processed and their votes.

2.2 Model building for improved CasPer algorithms

Constructive algorithms such as CasPer and Cascade Correlation (CasCor) start with the minimal size NN architecture often with no hidden neurons as in Fig. 1. Initially, all input neurons are fully connected to the output neurons. The number of connections in the initial network would be very large if there are too many hidden neurons added [3].

In building the model, I prefer to call and insert hidden neurons in an object-oriented way, which makes the whole process of building and training the model more intuitive. Based on this, I created a class called Hidden Neuron, which contains the output value and the weight it carries, defines as follow:

```
class hidden_neuron(object):
    def __init(self, in_dim):
        self.weight = torch.randn(in_dim, 1, device='cpu', dtype=torch.float,
requires_grad=True)
        self.out_val = 0

    def calculate_output(self, input_data):
        self.out_val = torch.tanh(torch.mm(input_data, self.weight))
        return self.out_val
```

The whole process of building the network during training can be presented like this:

```
1. Add new hidden neuron to the current tower
2. Training the network by RPROP, records the loss and accuracy during training
3. Check the size of the current tower, if reaches the upper bound, start a new
empty tower
3. Check training loss to decide whether keep adding hidden neurons
4. If True, back to step 1, stop completely otherwise.
```

The out layers are needed for matching dimension the outputs with the out dimensions required. The out layers is added one dimension a time, the newly added dimensions are recorded inside the tower, and will be recorded in the network class after the size reaches the maximum limit.

In this process there two things which need to be mentioned: the first is when to stop training the network after new hidden neurons have been added, and the second is how to decide whether to add new hidden neurons after training is complete. The first part will be discussed in detail in 2.3, so we will focus on the second part here.

We can easily know by the structure of the network that the key of avoiding overfitting and get good performance is the strategy of deciding when to stop adding hidden neurons. The strategy that I used is comparing the precious loss with the loss with hidden neuron added. Since if the loss stops decreasing or decrease very less during the training of RPROP, than it means the network training meets its limit, overfitting will take place if keep training from now.

The definition of stop strategy here is as follow:

$$\boldsymbol{Stop} = ((\boldsymbol{Loss_{i-1}} > \boldsymbol{Loss_i})\boldsymbol{and}\ (\boldsymbol{Loss_{i-1}} - \boldsymbol{Loss_i} < \boldsymbol{0.001} * \boldsymbol{Loss_{i-1}}))\ \boldsymbol{or}\ (\boldsymbol{Loss_i} == \boldsymbol{0}) \qquad (1)$$

$$(the\ i\ here\ represented\ the\ number\ of\ hidden\ neurons)$$

For the training process, I used the same training process as CasPer algorithm: RPROP to train the whole network, I used `torch.optim.Rprop` as the optimiser to train the whole network automatically.

2.3 Training Methodology

I use the same training strategy as CasPer, and CasPer uses RPROP to train the whole network. In new neuron addition, initial RPROP learning rates set by weight location.

There are 3 regions in network, each with own initial learning rate (Fig. 2). The reason for setting initial learning rate like this (L1 >> L2 > L3, in Fig. 2) is that with extreme larger initial learning rate(L1) could make the new hidden neuron learn the remaining network error fast, and L2 > L3 can make the network to reduce the error without too much interference from older weights.

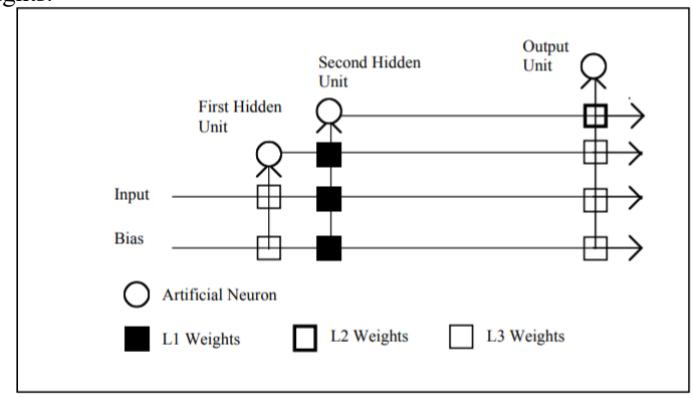

Fig. 5 Initial learning rate by weight location when new neuron added (L1 >> L2 > L3).

In my implementation after introducing Tower-Cascade structure, the only difference is that I initial the learning rate of the weights of hidden neurons in current tower to be L1, the weights of the corresponding output layer of hidden neurons in current tower to be L2, all the other weights to be L3.

$$L1 = [weights\ of\ hidden\ neuron\ in\ Tower_{current}]$$
$$L2 = [weights\ of\ output\ layers\ of\ the\ same\ hidden\ neurons]$$
$$L3 = [all\ other\ weights]$$

The weight update of RPROP is as follows:

$$\boldsymbol{W_{ij}} = \begin{cases} -\Delta_{\boldsymbol{ij}}^{(\boldsymbol{t})}, \boldsymbol{if}\ \frac{\boldsymbol{\partial E^{(t)}}}{\boldsymbol{\partial W_{ij}}} > \boldsymbol{0} \\ +\Delta_{\boldsymbol{ij}}^{(\boldsymbol{t})}, \boldsymbol{if}\ \frac{\boldsymbol{\partial E^{(t)}}}{\boldsymbol{\partial W_{ij}}} < \boldsymbol{0} \\ \boldsymbol{0}, \boldsymbol{otherwise} \end{cases} \qquad (2)$$

the Δ_{ij} here is the step size, which has the update rule as follows:

$$\Delta_{\boldsymbol{ij}}^{\boldsymbol{t}} = \begin{cases} \boldsymbol{\eta^+} \cdot \Delta_{\boldsymbol{ij}}^{\boldsymbol{t-1}}, \boldsymbol{if}\ \frac{\boldsymbol{\partial E^{(t-1)}}}{\boldsymbol{\partial W_{ij}}} \cdot \frac{\boldsymbol{\partial E^{(t)}}}{\boldsymbol{\partial W_{ij}}} > \boldsymbol{0} \\ \boldsymbol{\eta^-} \cdot \Delta_{\boldsymbol{ij}}^{\boldsymbol{t-1}}, \boldsymbol{if}\ \frac{\boldsymbol{\partial E^{(t-1)}}}{\boldsymbol{\partial W_{ij}}} \cdot \frac{\boldsymbol{\partial E^{(t)}}}{\boldsymbol{\partial W_{ij}}} > \boldsymbol{0} \\ \Delta_{\boldsymbol{ij}}^{\boldsymbol{t-1}}, \boldsymbol{otherwise} \end{cases} \qquad (3)$$

here initial step sized of 0.1, $\eta^+ = 1.2, \eta^- = 0.5, \Delta_{max} = 50, \Delta_{min} = 10^{-6}$. And CasPer uses weight decay to improve generalization, so error gradient used in Casper is:

$$\frac{\boldsymbol{\partial E^t}}{\boldsymbol{\partial W_{ij}}} = \frac{\boldsymbol{\partial E^{t-1}}}{\boldsymbol{\partial W_{ij}}} - \boldsymbol{D} \cdot \boldsymbol{sign}(\boldsymbol{W_{ij}}) \cdot \boldsymbol{w_{ij}^2} \cdot \boldsymbol{2^{-T \cdot Hepoch}} \qquad (4)$$

$$Hepoch\ =\ epochs\ since\ addition\ of\ last\ hidden\ neuron, D\ =\ decay\ parameter.$$

And, for the stopping strategy of training, I used training loss recorded along the training process and the number of epochs to define convergence. If the loss did not decease more than 1% in the specific number of epochs, then the training process will be considered as convergence. The limit of epoch can be represented as follow:

$$\boldsymbol{Limit_{epoch}} = \boldsymbol{P} * \mathbf{Num_{hidden}} + \mathbf{15} \qquad (5)$$

$$P \text{ is an hyperparameter, which is setted to } 1e10 \text{ in my implementation}$$

$$\boldsymbol{Stop} = (\boldsymbol{Loss_i} < \boldsymbol{Loss_{i-1}}) \ \boldsymbol{and} \ (\boldsymbol{Loss_{i-1}} - \boldsymbol{Loss_i} < \boldsymbol{Loss_{i-1}} * \mathbf{0.001}) \ \boldsymbol{while} \ \boldsymbol{i} < \boldsymbol{limit_{epoch}} \qquad (6)$$

$$i \text{ here represents the number of epochs}$$

By define the stopping strategy like this, the network can have best convergence properties, thus making the structure of the network more concise [5].

In addition, I used Cross Entropy Loss as loss function, and hyperbolic tangent as Activation Function. According to [3], hyperbolic tangent functions are symmetric functions, which are believed to be able to yield faster convergence than non-symmetric functions. Since the weight among the labels are not the same, I add the weight of labels as a parameter into the Cross Entropy Loss function for weighting the different classes differently for imbalanced classes.

2.4 Hyperparameter Tuning

2.4.1 Tuning for P

Since this parameter is used to bound the maximum epoch, what we want is for the model to converge in training, I set P to 1e10 (1^10) for trying to make the model entering converge condition every time of training.

2.4.2 Tuning for maximum depth limit (max tower size)

We can easily see that the weights the whole network is almost the same if the number of hidden neurons is the same and larger than the maximum depth limit.

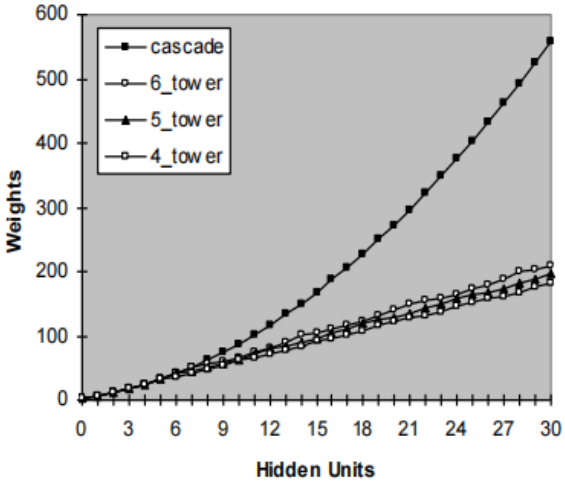

Fig. 6 network weights vs hidden weights [7]

So, no matter which maximum depth limit I choose, it has little effect on the final structure of the whole neural network. In this case, I chose 6 as my final maximum depth limit.

3 Results and Evaluation

The results of the experiment are as follows:

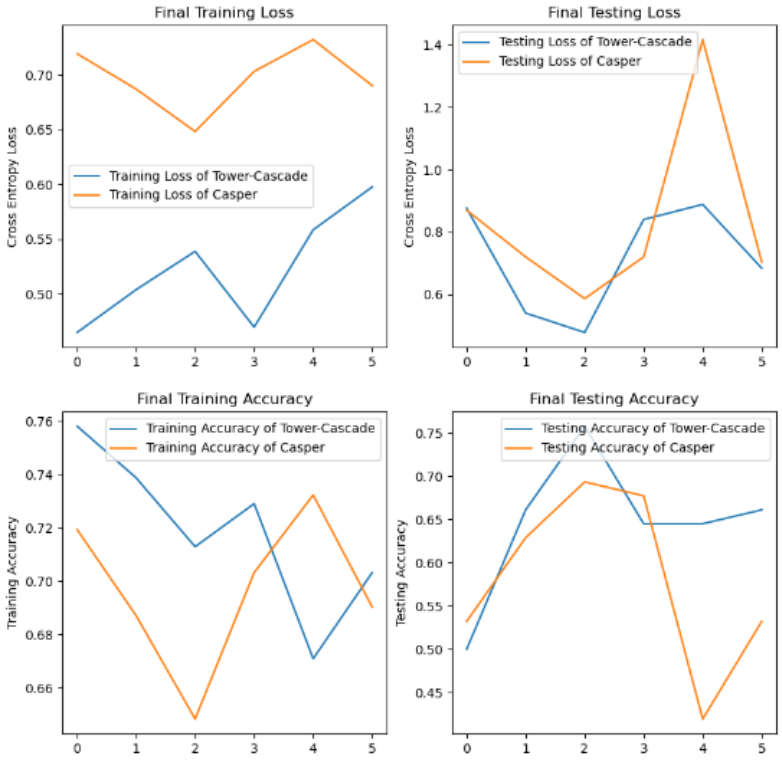

Fig. 7 result of K-fold testing.

The experimental setup is done by the K-fold algorithm. Specifically, the whole dataset is divided into 5 parts, and the others are used as the test set and the training set in turn, and the information is collected after the final formation of the model. It is clear from the graph that the results from the new model are better than those from the old model, both in terms of loss value and accuracy value.

From this, we can directly deduce that, the model obtained by the CasPer algorithm with the maximum depth limit, has a better performance in terms of the final predicting accuracy and loss.

Fold id	Tower-Cascade: Number of hidden neurons	Tower-Cascade: total epochs	Tower-Cascade: total weight (approximate by Num_hidden*4)	CasPer: Number of hidden neurons	CasPer: total epochs	CasPer: total weight (approximate by Num_hidden^2)
0	77	479	308	20	294	400
1	67	512	268	23	251	529
2	69	506	276	19	294	361
3	70	501	280	15	164	225
4	14	254	56	21	250	441
5	21	250	84	29	422	841

Table 2. result of K-fold testing.

The results shows that the Tower-Cascade structure needs more epochs and more hidden neurons to from up the network.

Since the total weights are mostly the weights of the hidden neurons, the total weights of Tower-Cascade can be approximate by number of hidden neurons. We can find that although Tower-Cascade uses more hidden neurons, the total weights of the network are smaller than or close to those of CasPer.

Thus, by adding the maximum depth limit, the complexity of the network can be effectively reduced, which is equivalent to "reducing the total weight" of the network while achieving the same or even better performance. More importantly, Tower-Cascade adds more hidden neurons in the approaching epoch, indicating that Tower-Cascade converges earlier in the training process using RPROP, which means that Tower-Cascade can achieve better results with less training time.

From these results we can see that after using the new structure, not only the complexity of the network is decreasing, but the training time of network is also shortening, and the performance of the network is increasing. All these three points show that this structural progress simplifies the network without losing the original advantages of CasPer in generalization.

In the whole result, the only shortcoming is that Tower-Cascade uses more total epochs and hidden neurons, which means that although the overall structure of the network is simplified, it still needs a lot of hidden neurons to build the body of the network, and during this period, although the training epochs of each hidden neuron are not too many. In this period, although the training epoch of each hidden neuron is not much, but in general, it still needs many training sessions to make the model achieve the expected effect. This may be a direction for future improvement.

Here is comparison in details of fold 5:

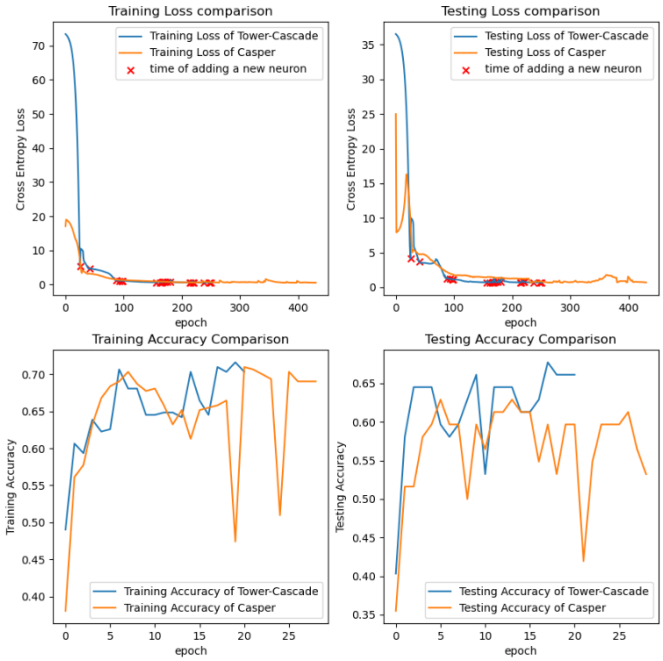

Fig. 8 Comparison of Tower-Cascade with CasPer

The comparison shows that the accuracy of Tower-Cascade has larger loss in the beginning, decrease very fast in the staring epochs. Which proves that it maintains the advantage of Cascade Correlation. And Tower-Cascade have more stable increasement on accuracy, which may cause by the simpler and stabler structure of network.

4 Discussion

The improvements of introducing Tower-Cascade architecture are various. In terms of the performance of network, the higher Accuracy and lower Loss took place on multiple fold of experiments, which proves that it has a better performance. And more hidden neurons added in same total epochs as Casper, proves that it will converge sooner that Casper while training.

But there are still problems. One problem is that too many total epochs and hidden neurons. The limitation of this structure is that average weights of hidden neurons are much lower than Casper, which leads to more hidden neurons to solve the same problem or achieve same performance. And it is the reason of taking more epochs and more hidden neurons than Casper.

One possible fix towards this problem is that may be adding some connections among the towers instead of completely independent. Adding these connections may leads to learning the remaining error of the previous towers, so that adding new towers may be more affective. As for how to add the connections with adapting the advantages of Casper, that may be a subject of future work.

References

1. Kwok, T.-Y., Yeung, D.-Y.: Constructive Algorithms for Structure Learning in Feedforward Neural Networks for Regression Problems. IEEE Trans. on Neural Networks 8, 630--645 (1997)
2. Fahlman, S.E. and Lebiere, C. "The cascade-correlation learning architecture," Advances in Neural Information Processing, vol.2, D.S. Touretzky, (Ed.) San Mateo, CA: Morgan Kauffman, 1990, pp. 524--532.
3. Khoo S., Gedeon T. (2009) Generalisation Performance vs. Architecture Variations in Constructive Cascade Networks. In: Köppen M., Kasabov N., Coghill G. (eds) Advances in Neuro-Information Processing. ICONIP 2008. pp. 236--243. Lecture Notes in Computer Science, vol 5507. Springer, Berlin, Heidelberg. https://doi.org/10.1007/978-3-642-03040-6_29.
4. Veasey S.C. et al. (2009) O. In: Binder M.D., Hirokawa N., Windhorst U. (eds) Encyclopedia of Neuroscience. Springer, Berlin, Heidelberg. https://doi.org/10.1007/978-3-540-29678-2_15
5. Treadgold N.K., Gedeon T.D. (1997) A cascade network algorithm employing Progressive RPROP. In: Mira J., Moreno-Díaz R., Cabestany J. (eds) Biological and Artificial Computation: From Neuroscience to Technology. IWANN 1997. Lecture Notes in Computer Science, vol 1240. Springer, Berlin, Heidelberg. https://doi.org/10.1007/BFb0032532
6. Śmieja, F.J. Neural network constructive algorithms: Trading generalization for learning efficiency?. *Circuits Systems and Signal Process* **12,** 331–374 (1993). https://doi.org/10.1007/BF01189880
7. N. K. Treadgold and T. D. Gedeon, "Exploring architecture variations in constructive cascade networks," *1998 IEEE International Joint Conference on Neural Networks Proceedings. IEEE World Congress on Computational Intelligence (Cat. No.98CH36227)*, 1998, pp. 343-348 vol.1, doi: 10.1109/IJCNN.1998.682289.

Appendix

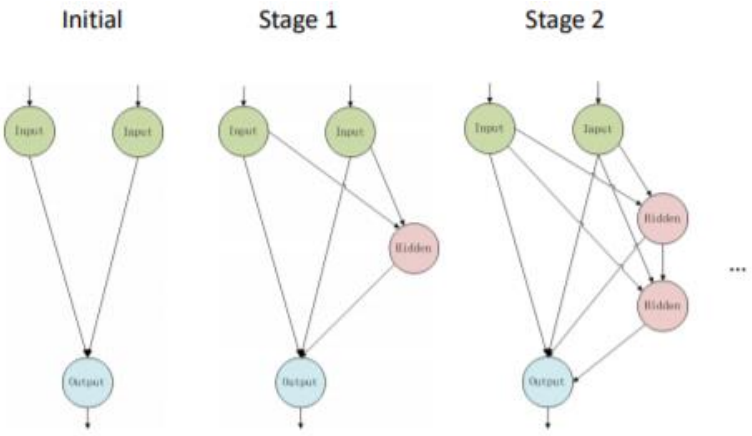

Fig. 9. Cascading topology of Cascade Architecture

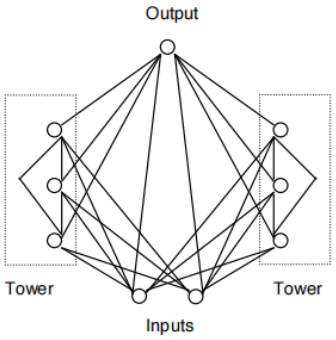

Fig. 10 A cascade tower architecture with a tower size of 3 [7]

```
mean num_fixs:  81.84677419354838
mean man_fixs:  16.06451612903226
mean fixs_dur:  18.94441129032258
mean man_dur:  3.7771505376344088
```

Fig. 11 mean value of each feature.

Fig. 12. Images provided (in order 10, 11, 12, 13, 14).

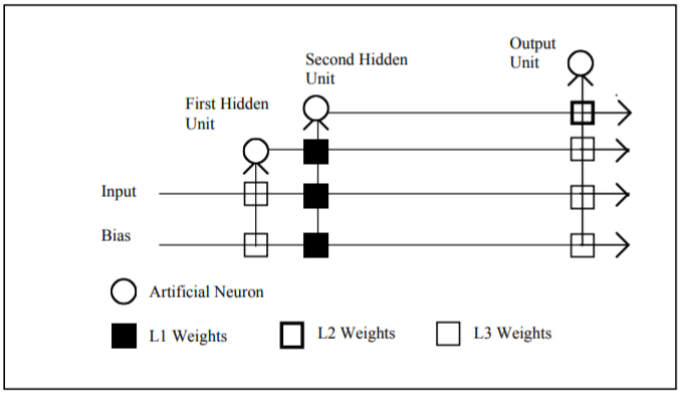

Fig. 13 Initial learning rate by weight location when new neuron added (L1 >> L2 > L3).

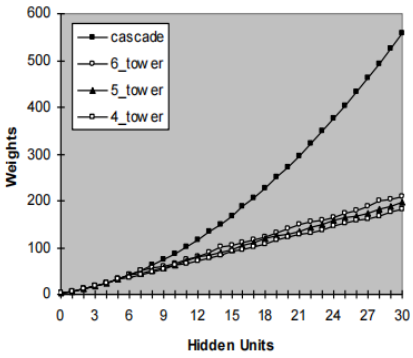

Fig. 14 network weights vs hidden weights [7]

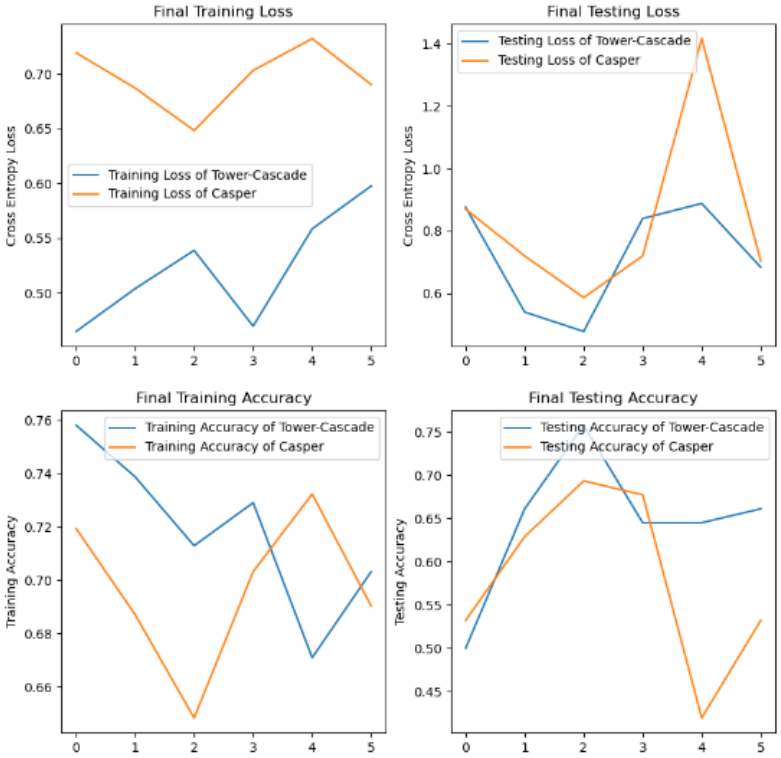

Fig. 15 result of K-fold testing.

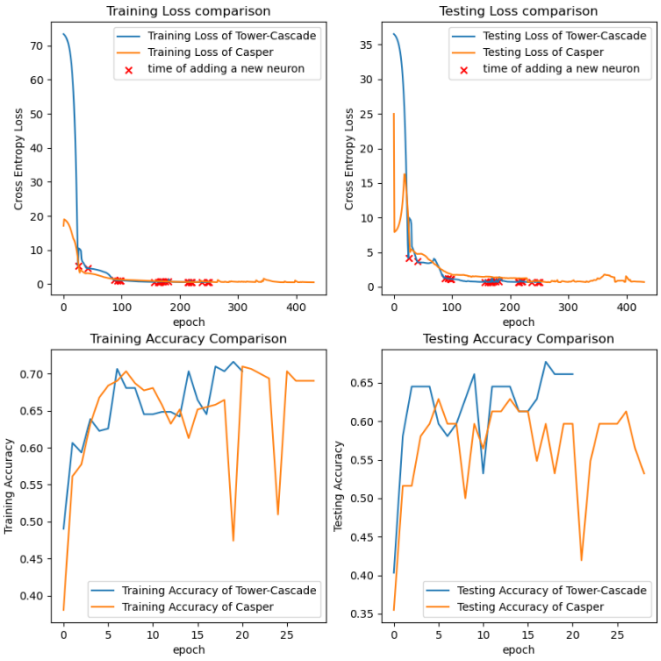

Fig. 16 Comparison of Tower-Cascade with CasPer

Table 3. The data that dataset contains.

Headings of data	Type	Meanings
participant	Integer	Id number of the participants (integer between 1 and 80)
num_fixs	Integer	Total number of fixations by the participant when looking at the image
fixs_duration	Float	Total amount of time (in seconds) that the participant spent looking at the image
num_man_fixs	Integer	Total number of fixations by the participant when looking within the target area
man_fixs_dur	Float	Total amount of time (in seconds) that the participant spent looking within the target area
image	Integer	The image id (integer in [10, 11, 12, 13,14, 15])
image_manipulated	Integer	Whether the image the participant views is the manipulated or unmanipulated version (0 = unmanipulated, 1 = manipulated)
votes	Integer	The verbal opinion of the participant as to whether the image is manipulated or unmanipulated (0 = voted unmanipulated, 1 = voted manipulated, 2 = don't know)

Fold id	Tower-Cascade: Number of hidden neurons	Tower-Cascade: total epochs	Tower-Cascade: total weight (approximate by Num_hidden*4)	CasPer: Number of hidden neurons	CasPer: total epochs	CasPer: total weight (approximate by Num_hidden^2)
0	77	479	308	20	294	400
1	67	512	268	23	251	529
2	69	506	276	19	294	361
3	70	501	280	15	164	225
4	14	254	56	21	250	441
5	21	250	84	29	422	841

Table 4. result of K-fold testing.