Use Neural Network to Classify and Compress Images

Sicheng Hao

Research School of Computer Science, Australian National University
{Sicheng Hao, u6919094@anu.edu.au }

Abstract. Training image classification model can be very time consuming. A very good way to maintain good classification accuracy while significantly reducing computational complexity is to use Principle Components. However, in theory, because CNN use original images to train the model, it should have much higher accuracy than PCA method. But because given datasets have too much noise information, the CNN accuracy is actually lower. Aside from above, I also validated that we can compress images to a desired quality by pruning the hidden layer neurons.

Keywords: Neural networks; Classification; Optimize methods; Progressive image compression; Google Net; CNN

1 Introduction

Within the traditional business research, classification has been regarded as an important domain (Hongkyu Jo & Ingoo Han, 1996). The image classification is widely used in military, industrial application and our daily life. Example containing missile guidance, merchandise recognition, search engine, social software and smartphones. There are tons of machine learning method to classify data as long as there is appropriate type of data. Those algorithms include but not limited to e.g. Logistic regression, KNN, k-means, SVM, Neural network and Random forest. Among all the methods, neural network has many advantages, e.g. auto-learning, easy to use. The key feature for neural network is "Black Box". It is both pros and cons. The advantage is that users can have almost zero comprehension to the internal neural network structure. All they need to do is to cram all the images and labels into the model and wait for it to process a little while. Usually the results are pretty promising. However, the neural network model has to consume tons of data, in order to get a reasonably good result and the computational complexity is high due to the Back-propagation algorithm. This is fetal for those who does not have enough data and computational resources. But, because it simulates bio-neurons, we can expect great prospect of neural network.

In reality, images usually have a lot of pixels, which dramatically increase their eigen vector numbers. Consequently, the computational complexity is increased aggressively. In CNN, we use PCA to reduce dimensions. However, because the reason above, we couldn't use all the principle components to train a neural network if there are not enough computational resources. Thus, we can choose the first several components that contain the most information. This is a very good way to sacrifice accuracy and reduce computational complexity, and is what we are going to conduct in the first section of this paper. I am going to train a basic image classification neural network using Principle Components rather than the whole image. The given training materials are 5 LPQ features, 5 PHOG features and corresponding labels.

In the second part, we use features to predict the same features, and prune hidden neurons as many as possible. This can reduce the neural network complexity while retaining much of the information of original data, which is very important when the problem becomes more and more complex.

Although using PCA to reduce training complexity is very efficient and can retain much of the accuracy, it can never reach the high accuracy by fully functional CNN that takes original images and generate their labels. However, as mentioned above, training CNN is very time consuming for weak hardwares. So, some preprocessing must be conduct to settle this problem. In the third section, I'm going to use an advanced CNN called Google Net to do image classification on raw images. Also, computational-complexity-reduce methods and related tuning methods will also be covered in this section.

2 Method

In this section, I will cover the methods I used and the considerations of choosing a certain method. Including, neural network implementation, optimizing the accuracy, progressive image compression, hyperparameter tuning and google net classification. This section will be divided into 3 subparts: Classification by Principle Components, Compression and Classification by images

2.1 Classification by Principle Components

Problem Analysis. I have a dataset of total 675 entries 90% for training, 10% for test, each entry has 10 features. Each of the entry has 10 features and a label, indicating the emotion of that picture. My job is to use 5 LPQ features and 5 PHOG features to predict the facial emotions and optimize the accuracy as much as possible.

Model Design. The neural network model is implemented as a class in python. It is a fully connected 3-layer neural network with an input layer, one hidden layer and an output layer. The activation function in hidden layer is sigmoid. The loss function is cross entropy. The optimizer is Adam. The input is those principle components features and the output is 7 labels that indicates different emotions.

In order to get best results, some measures are taken: normalizing the data, conducting training with different hidden neuron numbers to find the best number and use 3 groups of features to find which one have the best performance. All 3 measures will be covered in the following context of this section.

I have 2 indicators to evaluate the performance: Accuracy (number predict correct / total number of entries) and Loss (Cross Entropy Loss). The higher the Accuracy, the lower the Loss, the better the model.

Train & Find Optimal Hyperparameter. The basic idea of training this model is divided into 3 parts:

Step 1. Read, clean and normalize the data. The data may have a lot of "nan" values or error type of values. I need to first find them and clean them. In the process of cleaning "nan" values, I find one entry' 5 PHOG features are all "nan", so I dropped that entry, reducing the total entries to 674. I want to use 5 LPQ features, 5 PHOG features and all 10 features to train the model separately, so I group the input into 3 groups. However, all the data are pretty small, which may affect the accuracy. In order to improve the accuracy without introducing too much computational complexity, I normalized the data of each group. Whether this operation will improve the accuracy will be discussed in the "Results and Discussion" section.

Step 2. Build a basic neural network using Pytorch. This neural network includes: input, hidden and output layers, forward propagation method, loss function, gradient function and the backward propagation method. It should be noted that this is a fully connected neural network. Input and output layers have 5 or 10 neurons depending on the choose of feature groups, while the hidden layer can have arbitrary number of neurons. In order to find a neuron number that can reduce the loss while won't increase too much computational complexity, I compared results with different number of hidden neurons, and find that 5 hidden neurons are very likely to be the best choice.

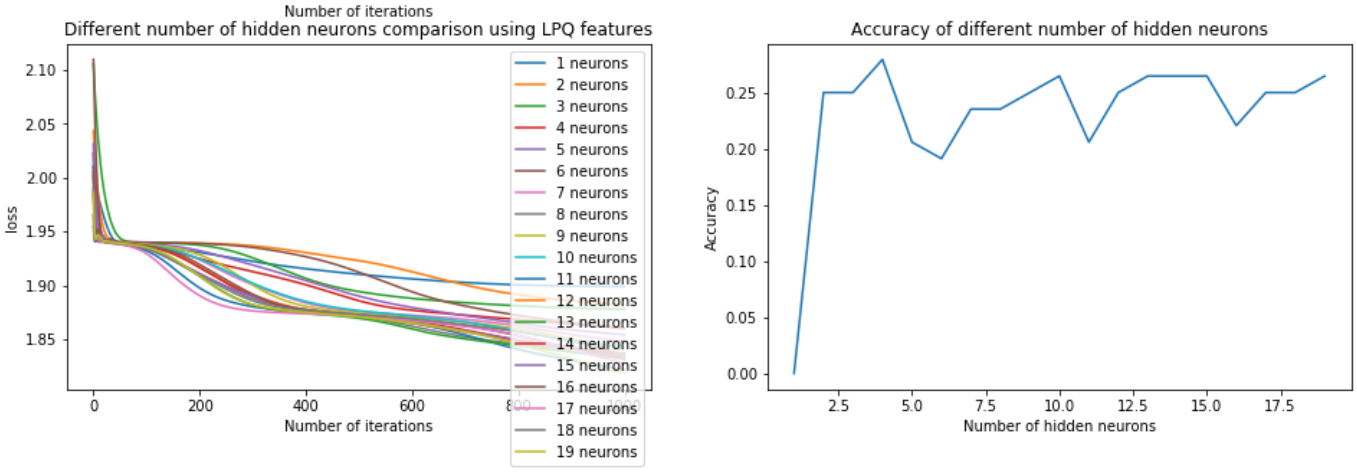


Figure 1. Aside from 1 to 5 hidden neurons, the loss does not reduce too much from 5 to 19 neurons. Besides, the accuracy from 5 to 19 is basically fluctuating randomly. So, I think setting the hidden neurons more than 5 will not affect accuracy too much but can increase the computational complexity and overfit the data.

After settling the hidden neuron number, the forward, backward propagation and loss, gradient function can be easily implemented by create a class and call Pytorch functions. Code will be provided in the "Code" section.

Step 3. I use Adam as optimizer and Cross Entropy as loss function. Firstly, define a neural network instance with 5 input neurons, 5 hidden neurons and 7 output neurons (because there are 7 labels). Then starts to train the model. Predict results using one group features (5 LPQ, 5 PHOG or all the features). Then calculate loss using cross entropy, calculate the gradient and do backpropagation [5]. Repeat above procedure 1000 times (can change, I choose 1000). Finally get the results and calculate accuracy.

However, since there are 2 different type features, I want to find which combination performs the best. In the above context, I've already divided those features into 3 groups: 5 LPQ, 5 PHOG and all 10 features. Then I did the comparison among these three groups Loss results. And do this several times because the results are not consistent. Although sometimes loss of LPQ features is higher than loss of PHOG features and sometimes on the contrary, the loss of LPQ + PHOG is always the minimum. However, when it comes to accuracy, most of the time LPQ features have the highest accuracy and LPQ + PHOG always has the lowest accuracy.

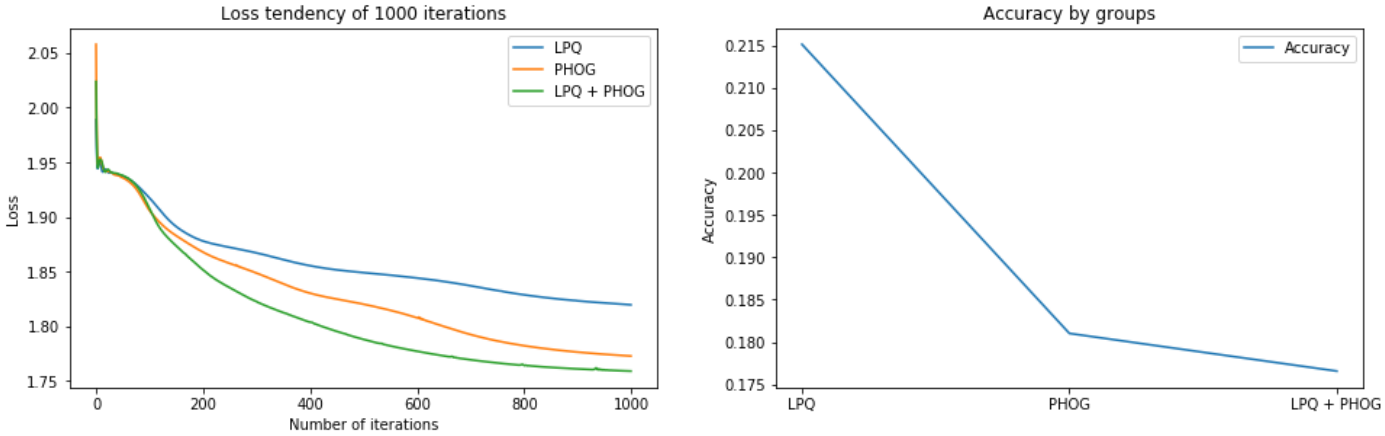


Figure 2. LPQ+PHOG always has the lowest loss. Most of the time LPQ has the highest accuracy.

Conclusion of 3 Measures. This is the conclusion of above 3 measures to optimize results. One is that I first normalize the data. Second is that after investigation, I choose to use 5 neurons in hidden layer to optimize both loss and computational complexity. Finally, I choose to use 5 LPQ features because it has the highest accuracy most of the time.

Conclusion Model. My model is 3-layer fully connected neural network with sigmoid activation function. The loss function is Cross Entropy and the optimizer is Adam. The model has 5 input neurons, 5 hidden neurons and 7 output neurons. The training algorithm is forward and backward propagation algorithm. The iteration is 1000 times.

2.2 Progressive Image Compression

Problem Analysis. In progressive, we should use less hidden neuron than input and output neurons, while input and output neurons should be the same to input, compress and output the image [1]. The job is to input the image, set the exact same image as the output, and train the neural network. After training the network, prune some unnecessary (or one) hidden neurons and train the model again [2, 6]. Repeat this process multiple times until the loss meets desired standard. To achieve this, I need to find a method to prune the hidden neuron while maintaining a relatively low loss.

Technique in the Paper. My idea came from the given paper. In given paper, authors discussed how to compress image by pruning hidden neurons while retain much of the original information in original images. The basic idea in the paper is to pruning hidden neurons with similar functionality. The main technique is to calculate angular separations between hidden neuron vectors. Vectors that have 15 or less, 165 or more (maximum 180) degrees are considered very similar. However, in reality, there are situations that angles between hidden neuron vectors are always under 15 degrees and always above 165 degrees. Methods to solve this problem will be covered below.

Model Design. Similar to Task 1 "Classification", I implemented a neural network class, but with one more function. I defined a function to return the hidden layer of the network, which can be used to prune the hidden neurons. My neural network is an input-hidden-output 3 layer fully connected neural network, with 5 or 10 input and output neurons. The input layer should have the same amount of neurons as output layer. And the hidden neurons should be at first more or equal to input and output neurons then pruned to be less than input and output neurons. The loss function is MSELoss() and the optimizer is Adam.

The critical part is how to prune and retrain the network. The basic idea is to use Angular Separations to show the similarity of every pair of hidden layer output vectors. The hidden layer output vectors, which is vectors from pattern presentation set, is the output of that hidden neurons [3]. If two vectors are really similar, which means the angular separation between them are small, their outputs are very similar. Thus, their functions (or weights) in the neural

network are basically the same. In this case, we can prune one of those neurons and retrain the network. Repeat this process multiple times until the loss meets desired standard, a progressive image compression neural network is finished.

Train & Find Optimal Feature. Since Task 1 "Classification" has already find the optimal hidden neuron numbers, which is 5. So, in this task, I only need to figure out which group of features performs the best. The 3 feature groups are 5 LPQ features, 5 PHOG features and all 10 features. Also, I split the dataset into training and testing parts. The loss is the test loss. The pruning procedures are below.

Step 1. Declare a neural network instance. If it's the first loop of pruning, remain the network weights and biases being the initial states. Else, assign last loop's pruned neurons' weights and biases.

Step 2. After certain training iteration (not pruning loop), record last iteration hidden layer outputs (this is the pattern presentation set). And define a matrix whose position (i, j) value is the angular separations of ith and jth vector in the pattern presentation set. Then find the pair that has the least angle. Delete one of the neurons from that pair randomly and store the weights and biases after pruning that neuron. Judge if loss satisfies my condition. If yes, enter next pruning loop, if not, break.

Conclusion of Optimal Features. After running the code several times. I find that using all 10 LPQ+PHOG features to train a progressive image compression neural network can pruning more neurons than PHOG and LPQ while maintaining a relatively low loss. I ignored data before 5 hidden neurons because only when hidden neurons are less than 5, will the model indeed compressing the image.

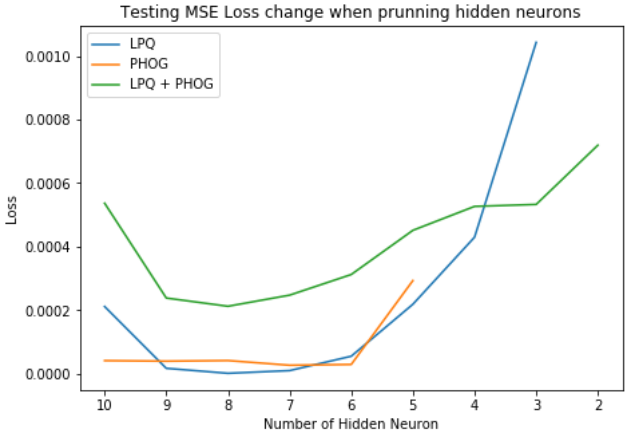


Figure 3. This is one time of loos trend graph according left hidden neuron numbers. The threshold I set is 0.001, meaning if the loss is below 0.001, I deem it as acceptable. We can see the network using LPQ+PHOG features can prune to 2 hidden neurons, while LPQ 3 and PHOG 5.

2.3 Classification by Images

Problem Analysis. Using Principle Components can reduce computational complexity, however, in theory, its accuracy cannot compete with CNN that trained with original images. To verify this and take a look at how powerful CNN is. I managed to classify images by original image, not Principle Components. I split the training and test sets. 80% for training and 20% for test.

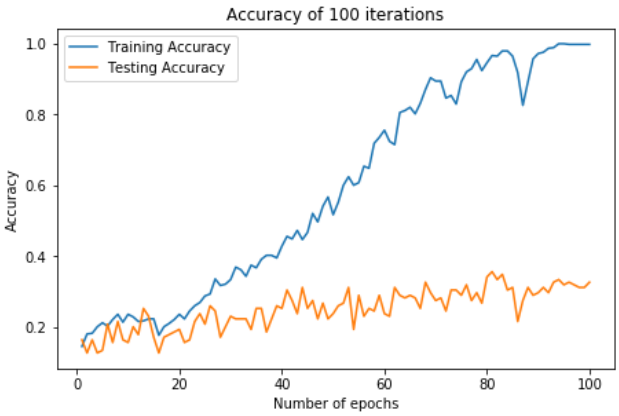
Model Design. In previous sessions, I've already drew conclusion about some hyperparameter tuning. So in this session I will use above Cross Entropy loss, Adam optimizer and the exact same training procedure. Just shifting simple neural networks to advanced CNN – Google Net, such that my model can take raw images and outputs labels.

Google net is currently state of the art CNN model. Its basic computation unit is no longer convolutional layer or pooling layer, but a combination of convolutional, batch norm and pooling layers. In google net, training is conducting by batches. However, each batch may have different normalization distributions. It may affect the final results dramatically. And batch norm layer above is used to solve this problem. The google net code is open source.

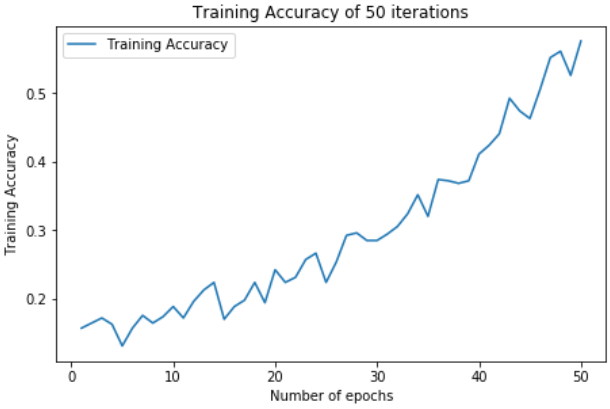
Preprocess & Train & Avoid Overfitting. With the exact same training procedure as above. I'll lay emphasis on preprocessing and how to avoid overfitting.

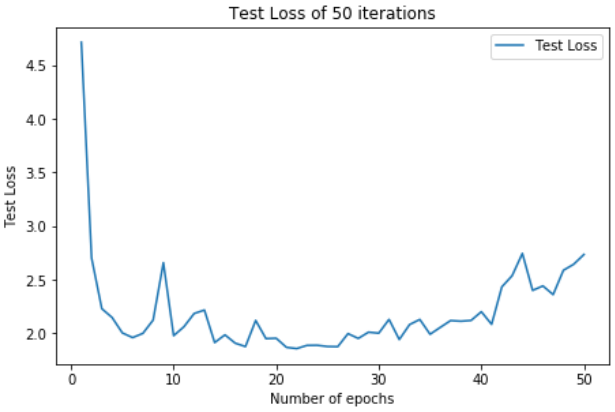
Preprocessing: The preprocessing procedure should be the similar as above "Classification by Principle Components". So first I will normalize the images. But because the problem that CNN requires tons of computational resources, we should also resize the images to a smaller scale.

Avoid Overfitting: I first trained my model with 100 epochs and get below graphs.



We can see clearly that after 40 epochs, my model starts to overfitting. And when the epoch reaches 100, the training accuracy reaches shockingly almost 100%. This means my model is indeed overfitted. To avoid overfitting and generate more accurate accuracy, I decide to run the model by 50 epochs. And below is the graph.





We can see that between 20 to 30 epochs, the test loss is the smallest. I assume that the average of test accuracy in this interval is the most accurate result for my CNN model.

3 Results and Discussion

3.1 Classification by Principle Components. I use two metrics to evaluate the performance of my model. One is entropy loss, another is accuracy (predict correct ones / all entries). Also, in order to make sure the accuracy is indeed reflecting the general accuracy rather than overfitting results, I split the inputs into training and testing sets. 90% of original dataset is for training and 10% for testing.

Results. After training 1000 times. I output the average loss and accuracy of the test set, which are 1.8723 and 0.2032 respectively. Loss result is within the expectation since in task 1, I want to reduce loss as much as possible while keeping the computational complexity low. So, I chose 5 hidden neurons. The 1.9245 loss is within the expectation perfectly. The accuracy is also within the expectation. Model using LPQ features indeed usually has higher accuracy than models with PHOG and LPQ+PHOG features, whose accuracy are 0.1029 and 0.1942 respectively. In conclusion, choosing 5 hidden neurons can indeed keeping the loss low while maintaining low computational complexity and model with LPQ features indeed has higher accuracy than whom with PHOG features and LPQ+PHOG features.

Discussion. However, this loss is still too high and the accuracy is still too low. The reason might be the hidden neurons and training epochs are still not enough. If the device is powerful enough, I train the model even more times and use large number of hidden neurons. To some extent, this method is feasible, because both training and testing accuracy are pretty low and similar, meaning the model hasn't overfit yet. If we increase the hidden neuron numbers and train even more times, we can still improve the training, testing accuracy and reduce loss.

3.2 Progressive Image Compression. In this task, I use one metric loss to evaluate the performance of my model. During training, I want the loss of the model, which represents how much information has lost during compression, below 0.001 and prune the hidden neuron based on this standard. Also, I split the training and testing dataset. 90% of original dataset is for training and 10% for testing

Results. Model with LPQ+PHOG feature usually can maintain much of the information while pruning the hidden neurons. And the loss is pretty low both in training and testing set.

Discussion. Aside from the conclusion from given paper "PROGRESSIVE IMAGE COMPRESSION" that by pruning the hidden neuron, we can indeed compress the image, I can conclude that model using LPQ+PHOG features have lower loss when pruning the same amount of hidden neurons [4]. This means if we want to compress the image harder and lose more information, we should use PHOG features. If we want to compress the image while keeping most of the information, we should use LPQ+PHOG features.

3.3 Classification by Images. I use 80% of original dataset to train and 20% to validate. Before training I applied normalization and resize to original images.

Results. To better display the real results, I calculate mean of loss and accuracy between 20-30 epochs. Because in this interval, the test loss is the smallest. The result is 1.9209 and 18.07% respectively. Detailed graph is below. However, there is a very interesting point in my results – although after 40 epochs, my model starts to overfit, but the mean of accuracy is also rising.

Accuracy of 20-30 epochs	Angry	Disgust	Fear	Happpy	Neutral	Sad	Surprise	Total
Test Accuracy	12.50%	46.67%	20.83%	12.50%	5.88%	23.53%	9.52%	18.07%

epochs	20-30	40-50	60-70	80-90
Mean Test Accuracy	18.07%	25.23%	31.96%	35.05%

Discussion. The results are far away from my assumption that CNN can generate better accuracy. Previous PCA method has accuracy of 20.32% while CNN only has 18.07% accuracy. I looked through the dataset and find that this dataset is very bad material to directly apply CNN to do classification. Faces are only small part of the image and most of them are not in the same position of the image. It is not possible to generate better results unless we first extract human faces from each image. As far as that interesting point, I believe this happened because the validation set is too small and too similar to training set. If the test set is large enough or different enough, we can see that after 40 epochs, test accuracy will drop just like the test loss.

4 Conclusion and Future Work

Conclusion. Training epochs, feature selection and hidden neurons number can all affect the result accuracy and loss in Principle Components based classification. And this method can indeed retain relatively good accuracy while significantly reducing computational complexity. When it comes to Image Compression, by pruning the hidden neuron, we can indeed compress the image. And the model using LPQ+PHOG features, when pruning the same amounts of hidden neurons, can have lower loss. In theory, CNN can classify human faces better but I failed to validate this because the provided images have too much noisy information that we simply could not generate better results.

Future Work. In classification part, if we can increase training iterations, hidden neurons numbers and have more PCA features, the testing accuracy can improve dramatically. However, even when we are lack of information and device have less power to run more iterations, we can still improve the results by choosing different group of features and add more but not too many hidden neurons. Knowing how to improve the results using limited resources is very important, since in reality, we can even have more constraints than right now.

In compression part, although the work is done, we can still see many flaws in the process. E.g. What will happen if we crank up the starting hidden neurons numbers to even more neurons? And indeed, in the end the model using LPQ+PHOG features has the least loss, but most of the time its loss is bigger than other two models. Also, for LPQ+PHOG features, when pruning less than 10 meaning it starts to compress the image while other two start to compress after pruning to 5 neurons. Is it fair to compare them based on left hidden neurons? Maybe we can improve the experiment by making the x-axis of the graph to be (input_feature_dimension – hidden_neuron_number_left) or (hidden_neuron_number_left / input_feature_dimension).

In CNN part. The accuracy has very high potential to go up as long as a method of recognizing and extracting human faces is applied before training CNN. If provided with a dataset with pure human faces, CNN can certainly do a better job than Principle Components based classification.

5 The References and Citations

1. Dhall, A., Goecke, R., Lucey, S., & Gedeon, T. (2011, November). Static facial expressions in tough conditions: Data, evaluation protocol and benchmark. In 1st IEEE International Workshop on Benchmarking Facial Image Analysis Technologies BeFIT, ICCV2011.
2. Carreira-Perpinán, Miguel A., and Yerlan Idelbayev. "“Learning-Compression” Algorithms for Neural Net Pruning." Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition. 2018.
3. Gedeon, TD, Harris, D, “Network Reduction Techniques,” Proc. Int. Conf. on Neural Networks Methodologies and Applications, AMSE, San Diego, vol. 2, pp. 25-34, 1991.
4. Gedeon, T. D., and D. Harris. "Progressive image compression." [Proceedings 1992] IJCNN International Joint Conference on Neural Networks. Vol. 4. IEEE, 1992.
5. Rumelhart, David E., Geoffrey E. Hinton, and Ronald J. Williams. Learning internal representations by error propagation. No. ICS-8506. California Univ San Diego La Jolla Inst for Cognitive Science, 1985.
6. Namphol, Aran, Steven H. Chin, and Mohammed Arozullah. "Image compression with a hierarchical neural network." IEEE transactions on Aerospace and Electronic Systems 32.1 (1996): 326-338.