Compressed Images for Fine-grained Classification

Zetao Zhang

u6575462@anu.edu.au
Research School of Computer Science, Australian National University

Abstract. Image compression is useful in areas like image storing and transferring, so it was a very important research area in the past. However, it would cause the information loss among images. I choose to use a shallow neuron network for compression, which prune the neurons based on their distinctiveness and use the compressed features to perform a fine-grained classification. By comparing results of fine-grained classification using images with and without compression, we find out that compressed images could make the fine-grained classification perform worse.

Keywords: compression, transfer learning, fine-grained classification

1 Introduction

Fine-grained classification [6] usually requires very detailed features because the model need to distinguish the differences among very similar objects. In this paper, we compared the results of fine-grained classification with and without image compression to discover whether compression would eliminate critical information for fine-grained classification.
There are two experiments for comparison, and the first experiments use a model with two components: one feed-forward network of three layers of processing neurons for feature extraction and one deep neuron network using one of best model developed in the recent years, called ResNet 152 [2] for classification. Like the progressive image compression model [1], the hidden layer in the feature extraction part has fewer neurons than the input layer, which means the hidden layer only extract the most important information from the input. The output layer has the same neurons as the input layer, so that it is easier to train the model by comparing the difference between the input and output. Then, I use the decoder structure of the first shallow network to reconstruct the original images and use them as the input of the second deep neuron network. For the first model, I am using the standard backpropagation method and gradient descent. The activation function of the hidden layers is sigmoid, which increase the non-linearity of these two models. In addition, I progressively remove the similar neurons of the first model during the training to reduce the feature dimension, so all the neurons are significant after the training process. For the second model, the general idea is transfer learning. I implement a pretrained model as feature extractor and freeze the parameters. Then I add a new canonical classifier to train for the fine-grained classification.
For the second experiment, we just use the ResNet 152 and the added classifier mentioned above as the model.

2 Method

2.1 Neuron pruning

For the first feature extraction model, the key part is to prune the hidden neurons based on their distinctiveness [3]. The distinctiveness of hidden neurons is determined from the neuron output activation vector over the pattern presentation set. Each pattern is fed into the input layer, and the hidden layer would decrease the dimension of that pattern to the number of neurons in the hidden layer. Once all the training data is fed into the network, the output of the activation function in the hidden layer is the activation vector which has the same dimension as the number of training data. If two neurons in the hidden layer output two similar vectors, it indicates that these two neurons have similar functionality, so one of the neurons can be removed such that the information can be further compressed.

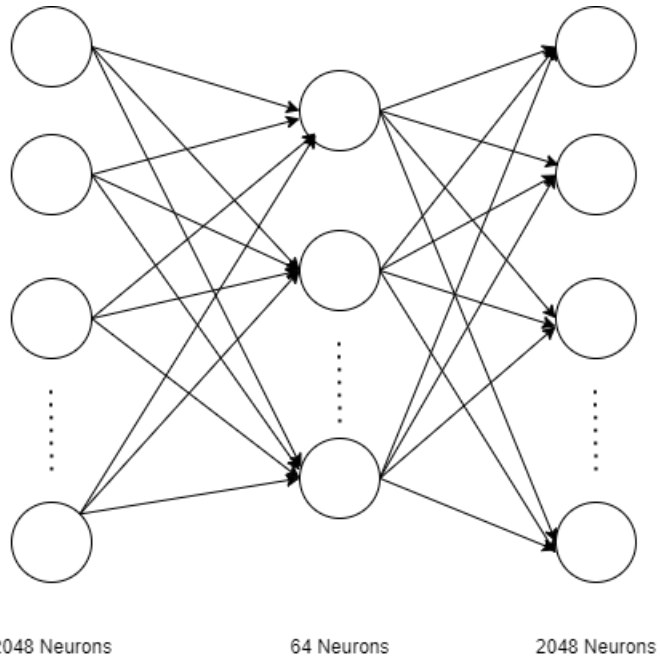

Fig. 1. The structure of compression network. The input layer and output layer have the same number of neurons, and the hidden layer has much less neurons.

In terms of distinctiveness, the angle between these two vectors represents the similarity between these two vectors. Therefore, if the angle between two activation vectors is less than 15°, these two neurons are too similar, so one of them should be removed and the weight of the removed neuron should be added to the maintained neuron. In addition, if the angle between two activation vectors is over 165°, these two neurons are complemented with each other, so we should perform the same operation as they are similar [4]. After pruning the similar and complemented neurons, the neuron network needs to be retained until it converges. Then the pruning process start again. Note that the pruning operation should be performed every time the network is converged, otherwise, we cannot make sure that the network has learnt all the functionality properly.

I start with 64 hidden neurons and prune them progressively to get the smallest dimension of the representation for the input patterns.

2.2 ResNet 152

The deep learning model ResNet 152 [2] is one of the best convolutional neuron networks developed in 2015. This model is based on the philosophy of VGG nets [7]. The residual network with 34 layers is shown in Fig 2.

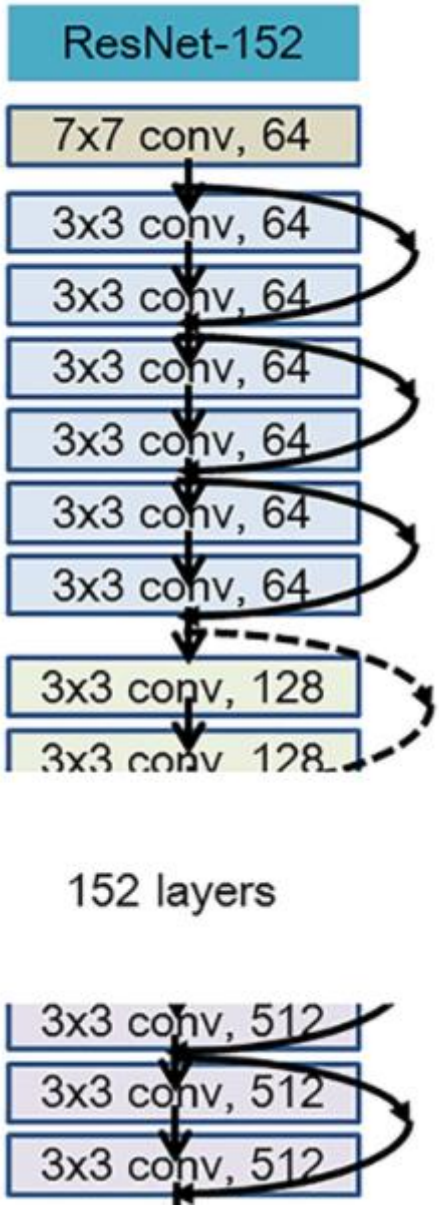

Fig. 2. The structure of ResNet 152. This network uses residuals to solve the problem of vanishing gradient. The network is very deep, so middle parts are omitted.

As we can see from Fig 2, the residual structure, which is the arrows in the figure, is shortcut connections between neighboring layers. The gradient will gradually disappear due to too many layers. However, with residual connections, we are able to generate more layers and get higher accuracy.

2.3 Canonical Neuron Network Classifier

The fine-grained classification task focuses on differentiating between hard-to-distinguish object classes, so it is a challenging task due to the large intra-class variance and small inter-class variance. In this case, instead of using the standard accuracy, I choose to use the top-10 accuracy metric as the measurement.

In terms of the model, I choose the standard shallow neuron network with one hidden layer. There are two reasons for choosing that. First, the input of this classification model is extracted feature representations rather than the original images, so it is not suitable to use deep neuron network like convolutional neuron network. Besides, the training data is not very large, which means we should either use pre-trained deep neuron network or choose a shallow network instead. Based on the above two reason, the most appropriate model should be a shallow canonical neuron network.

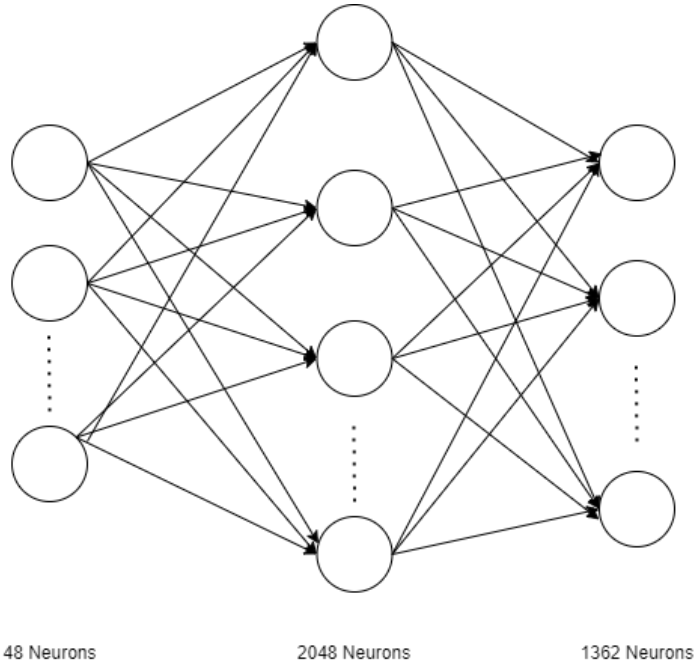

Fig. 3. The structure of classification model. The input layer has the same number of input neurons as the hidden layer of the compression model. Picking 48 input neurons and 2048 hidden neurons as an example. In general, sufficient dimension of features required for fine-grained classification is enough. The input layer must have the same number of neurons of the hidden layer in the first model. The output layer has 1362 neurons corresponding to 1362 classes we want to classify.

In this classification model shown in fig 3, the input layer can take the output from the hidden layer of the first model, so the classification model should have the same number of neurons as the hidden layer of the first model. For hidden layer, I choose to use 2048 neurons, which has the same dimension as the training set. The output layer has 1362 neurons, which matches the number of the classes for the dataset.

2.4 Transfer Learning

Transfer learning is where we train the weights of our model on a related (source) dataset and transfer those pre-trained weights over to the model for our target dataset rather than initializing the weights from scratch. In this paper, the dataset is vehicle-X, which is not a very large dataset. Training a very large model such as ResNet 152 from scratch requires way more data than those in this dataset. In this case, use transfer learning cannot only save a large amount of time, but also be more likely to get better results.

2.5 Implementation

Our implementation is the combination of the above components. There are two experiments for comparison: The first experiment uses the shallow neuron network to compress the images and use their reconstruction as the input of the second classifier model; the second experiment directly use the original images for classification. By comparing difference between those two experiments, we can find out whether image compression would eliminate some minor features which may not be useful for general cases but make critical differences in fine-grained classification.

3 Results and Discussion

In this section, we conduct experiments to evaluate the performance of fine-grained classification with and without image compression. We first introduce data preprocessing, hyper-parameter setting, and evaluation measures. Then we illustrate our results under the automatic evaluation. Finally, we do further comparisons and discussions over these two experiments.

3.1 Vehicle-X Dataset

The dataset we used is the vehicle-X dataset, and the training set, validation set, and test set have images of size 45438, 14936 and 15142, respectively. There are 1362 classes for this dataset, so the task is to use these features to perform a fine-grained classification over 1362 classes. The number of images in each class is not uniformly distributed in all three sets, which indicates some classes may not have sufficiently large amount of data.

3.2 Data Preprocessing

The preprocessing of the images is done using the built-in function "transforms" in PyTorch. To compress the images, we need to transform all the images into tensors and flatten them. Before feeding the inputs into the pretrained ResNet 152 model, we use the standard preprocessing steps as the training of ResNet 152 on ImageNet: reshape the all the images, resize them into 256 * 256 and crop the 224 * 224 from the center of the images. After that, normalizing all the tensors using mean of [0.485, 0.456, 0.406] as well as the standard deviation of [0.229, 0.224, 0.225].

3.3 Hyper-parameter Setting

For the implementation of compression model and classification model, we choose batch size of 32 and 600 epochs. The different part is, the compression model uses the Adam optimizer [8] with learning rate of 0.001 is used to train the model, whereas for the classification model, the SGD optimizer with learning rate of 0.01 is used as well as momentum of 0.5.

3.4 Metrics

We use two metrics for the multi-class classification task: Top-1 Categorical Accuracy and Top-5 Categorical Accuracy. Top-k Categorical Accuracy measures the proportion of times when the ground truth label can be found in the top-k predicted classes for that input. In most cases, Top-1 accuracy is sufficient, because we generally want to know whether the classifier can predict the correct class of that sample or not. However, for fine-grained classification, since the number of classes is enormous, it is reasonable to release some constraints and consider the most 5 likely classes during the prediction, because the differences of predicted likelihood among those classes may be very small.

3.5 Experiment for Image Compression

To get the minimum dimensional representation of the input patterns, we should start with larger number of hidden neurons. However, the initial choice should not be too large, since neuron pruning a generally a brute force process. In general, the model would evaluate the angle between each activation vector, which indicate the computational complexity is exponentially increased. Therefore, picking a good initial hidden neuron number is crucial. I tried out many different initial values starting from 8 to 64 every multiplying of 2. It turns out that 64 is the least number among the power of 2. Any number below it would fail to prune the hidden neurons, which means there are other important features need to be learnt. Picking 64 is also a relatively reasonable choice, since it is not an enormous value. The computation time is acceptable, so I do not have to give up some important information to speed up the pruning process.
As I tried many rounds, there are roughly 16 neurons (Table 1) being pruned during the pruning process, but the training loss and validation loss does not increase after pruning, which provide strong evidence that the pruning process does not take away essential information.

Table 1. Neuron Pruning Process in 5 Rounds.

Pruning Round	1	2	3	4	5
Pruned Neurons	20	14	18	21	16

One crucial thing is that the pruning process must be performed after the model converged, otherwise, it is possible that the removed neuron is not similar but in the process of learning. In this case, we cannot distinguish what condition they are in if the model is not converged. Therefore, I perform the pruning after 200 epochs, because the training loss and validation loss is relatively stable at that time.

Table 2. A record of one pruning process with pruned neurons, training loss and validation loss.

Epoch	200	400	600
Pruned Neurons	10	4	0
Training Loss	0.037365	0.037365	0.037365
Validation Loss	0.001171	0.001171	0.001171

As we can see from Table 2, the training loss and validation loss does not change after pruning, which provides strong evidence that the purning process does not harm the overall performance of the compression model. In addition, the validation loss is relatively small, the compression model has achieve a reasonbly good performance.

3.6 Experiment for Classification

The classification model is also a pretrained ResNet 152 with customized fully connected layer replacing the original classifier. The inputs are the images of size 224 * 224 after preprocessing.
To choose an ideal number of hidden neurons in the fully connected layer of the classification model, we choose to use the original images for comparison.

Table 3. Result of classification using different number of hidden neurons without image compression

Hidden Neurons	512	1024	2048
Rank-1	76.94	78.75	79.48
Rank-5	90.50	92.68	93.88

In the above Table 3, we can see that with more hidden neurons, both top-1 and top-5 accuracies increase. However, it does not bring significant improvement. In this case, for more than 2048 hidden neurons, it is not necessary to add more neurons for higher accuracy since it could also dramatically increase the training time.

3.7 Comparison with Original Paper

The original paper [5] has classification model achieving the rank-1 accuracy and rank-5 accuracy of 81.50% and 94.85% respectively.
My model without image compression gives much similar results: with 2048 hidden neurons, the rank-1 accuracy is only 79.48%, and the Rank-5 accuracy is 93.88%. The difference is probably caused by different model chosen, different hyperparameter setting and different weight initialization. However, these two similar results shows that my model has quite supportive result in terms of fine-grained classification.

Table 4. Method comparison of classification in original paper and my model. Rank-1 is the top-1 accuracy for classification. Rank-5 is the top-5 accuracy for classification.

Method	Rank-1	Rank-5
Original Paper Model	81.50	94.85
My Model without Image Compression	79.48	93.88
My Model with Image Compression	65.33	81.56

Table 4 also shows the classification results between with and without image compression. After compressing the image using the above compression model we described, the rank-1 accuracy decreases from 79.48% to 65.33%, along with the rank-5 accuracy from 93.88% to 81.56%.

3.8 Discussion

After comparing results from the above two models, it is obvious that compressing image using 64 hidden neurons would cause information loss so that the fine-grained classification would perform worse. One of the possible reasons is that image compression causes blurry to these images and make the minor difference between very similar cars hard to track. For example, if two cars have the same color and similar structure with only minor difference on automobile styling, it is likely that the difference is not obvious after compression, which would significantly reduce the accuracy of the classification model. However, if we keep the quality of the compressed images to some extent, it could reduce the effect that some features are abstracted so that the classification accuracy could be preserved after compression.

4 Conclusion

I have demonstrated that using the progressive compression model with 64 hidden neurons is not suitable for fine-grained classification tasks. However, it does not indicate progressive compression is useless for classification. As long as we can keep the quality of compressed model by increase the number of hidden neurons in the compression model, we can ultimately find a balance point between compression and preservation.
For future research, we can try different hidden neuron setting, such as 128 or 256, on the compression model to find the ideal hyperparameters. In this case, we cannot only save storing space for training data, but also speed up the training process by using the compressed images, which have lower dimensionality, directly. Also, a shallow network may not be

sufficiently large enough to express features. We can also try to add more layers in the compression model to construct a multilayer autoencoders to achieve better compression performance.

References

1. T.D. Gedeon & D. Harris, Progressive Image Compression. Proc. IEEE Int. Conf. on Neural Networks, 1992.
2. K. He, X. Zhang, S. Ren, and J. Sun, Deep residual learning for image recognition, arXiv:1512.03385, 2015.
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. Harris, D, Gedeon, TD, A meta-learning strategy to improve the performance of training algorithms, EXPERSYS-91, Paris, 1991.
5. Yao, Y., Zheng, L., Yang, X., Naphade, M., Gedeon, T.: Simulating content consistent vehicle datasets with attribute descent. arXiv preprint arXiv:1912.08855, 2019.
6. Ze Yang, Tiange Luo, Dong Wang, Zhiqiang Hu, Jun Gao, and Liwei Wang. Learning to navigate for fine-grained classification. In The European Conference on Computer Vision (ECCV), 2018.
7. K. Simonyan and A. Zisserman. Very deep convolutional networks for large-scale image recognition. In ICLR, 2015.
8. Kingma, D. P., & Ba, J. (2015, January). Adam: A Method for Stochastic Optimization. In ICLR (Poster).