Model Compression in Neural Network

Nigel Chen

Research School of Computer Science, Australian National University, Canberra Australia
u6862130@anu.edu.au

Abstract. In this paper, we trained a simple one-hidden-layer neural network and finetuned ResNet, MobileNetV2 and MobileNetV3, which are pre-trained on ImageNet, to do classification on SFEW. After fitting the model on the SFEW data set, we performed experiments on compressing the model. For a one-hidden-layer neural network, we pruned the neurons within a hidden layer by the distinctiveness of neurons and re-trained the neural network. The pruning process is one neuron at a time until there are no neurons that need to be pruned. For MobileNetV2, we quantized the model. After evaluation of the performance, we found that after compressing, the performance of the neural network will decrease, but still better than the performance stated in some former papers. Considering the computational resources, A certain degree of performance decline is acceptable.

Keywords: Deep Neural Network, Distinctiveness Pruning, Quantization, ResNet, MobileNet

1 Introduction

In recent years, with the rapid development of GPUs and the advent of the era of big data, deep neural networks (DNN) have achieved state-of-the-art performance in various fields of artificial intelligence [1], including voice recognition [2], natural language process [3], image classification [4] etc. Deep neural networks have greatly surpassed the accuracy of human recognition in many recognition tasks. It also had broken through the huge performance gain brought by a traditional technique such as LPQ [5] and PHOG [6]. These performance gains are due to the ability of deep neural networks to extract high-level features from large data set to obtain an effective representation of the input data. In addition, the powerful GPU computing power has greatly improved the computational efficiency of deep neural networks compared with traditional computing on CPU platforms, resulting in a significant increase in the training speed of models.

As the performance of deep neural network models increases, the depth of the neural network becomes deeper and deeper, which is followed by the disadvantage of high storage and high-power consumption of deep network models, which severely limits the application of deep neural networks in resource-limited application environments and real-time online processing. This raises the need for compression and acceleration of deep neural networks. There are two main approaches to compressing deep neural networks – pruning [8] and quantization [14].

In this paper, we set up several experiments on the data provided by [7]. The quality of the data often determines the final performance of the machine learning model. [7] present a static facial expression data set Static Facial Expressions in the Wild (SFEW) which was extracted from frames of movies. Such a way of generating data can make the content of the data set closer to the real production environment. SFEW provides data with seven emotion labels, so the task of our model is to predict the emotion label based on the data in SFEW.

Then we compare the performance results among different methods trained on the data described above. First, we trained a simple one hidden layer feed-forward neural network and 3 commonly used deep learning models – ResNet [11], MobileNetV2 [12] and MobileNetV3 [13]. Then we utilized a pruning method introduced in [8] and a quantization method [14] to compress the size of some of the models we trained and explore how the performance and resources required by the model could be affected. From a common-sense perspective, neural networks tend to perform better with larger architecture, or more neurons. But the computation cost could be relatively large in production. Even though the state-of-the-art software and hardware can handle huge neural networks, there are still a lot of circumstances, smartphone for example, that the hardware, which is the computation power, is restricted. This is where pruning and quantization can be utilized to reduce the demand for neural networks for computational resources while maintaining acceptable performance.

2 Method

In this section, we first describe the data set we used and how we pre-process the data, then we describe the details of our experiments. Finally, we describe the technique we used to compress the neural network.

2.1 Data Set

The *Static Facial Expression in the Wild (SFEW)* [7] has been generated upon *AFEW* [9]. In detail, SFEW has selected frames from AFEW, which contains an unconstrained facial expression, different resolution of faces. SFEW contains a

total of 700 images and is labelled with seven emotion labels - *angry, disgust, fear, happy, sad, surprise* and *neutral*. As described in [7], the process of generating data is shown in Fig.1 from the original paper. When the title with the corresponding frame contains the keyword 'laughing', the corresponding clip is played to human labelers. Then, human labelers annotate the subjects in the scene using the provided GUI tools. The result of annotation is shown at the bottom of Fig.1. In the data set folder, there are seven subfolders, each named with different emotion labels. Each of these seven folders contains the frame image of the corresponding label.

Fig. 1. Process of data set generation described in the original paper

2.2 Data Preprocess

The preprocess step of the data set contains two parts described below:

Extracting Features Manually. Performing Principal Component Analysis (PCA) on the features generate by Local Phase Quantization (LPQ) and Pyramid of Histogram of Gradients (PHOG). And then, extract the first 5 PCA of these two descriptors, which can keep 98% of the variance, as our processed input feature of the simple feed-forward neural network.

Dividing the Data for Training. Dividing the data set into training/validation sub-set with the ratio 80/20. The training set and the validation set will have the same structure as the original data set. That is, there are seven subfolders, each named with different emotion labels. Next, we resized all the images into 224 by 224 resolution and normalized them using the mean and standard deviation of ImageNet [15] to fit with the input requirements of the pre-trained models that we are going to use. The preprocessed images are as shown in Fig. 2.

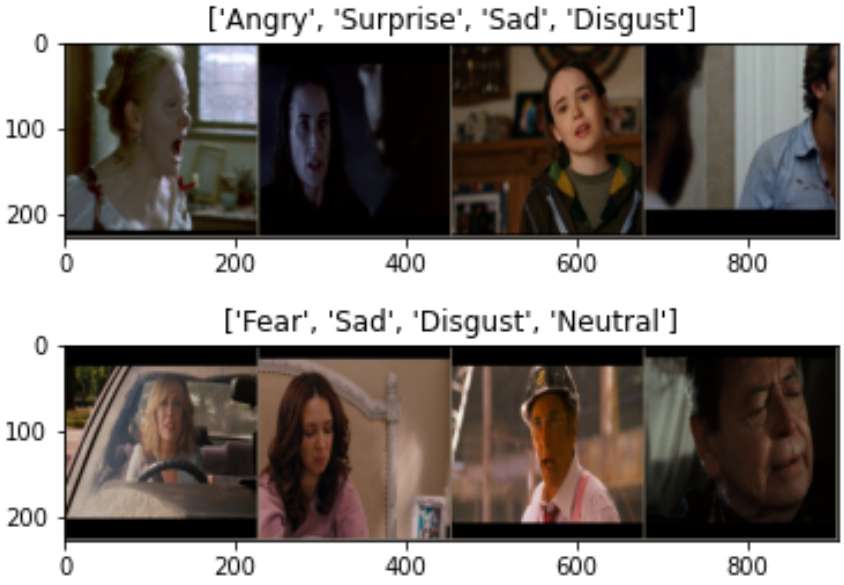

Fig. 2. Sample of resized and normalized images

2.3 Model Structure

To do model compression, we first need a baseline model to be trained. In this paper, we use a simple feed-forward neural network as shown in Fig.3 with three layers including one input layer, one hidden layer and one output layer. Specifically, the input layer has 10 neurons which corresponding to the two first 5 principal component features from LPG and PHOG. The hidden layer has 128 neurons which is set by hand so to guarantee that some neurons within this layer can be compressed or pruned. The output layer has 7 neurons corresponding to the 7 classes of emotion. We added one more parameter, mask, to the forward process of the neural network. This mask has the same shape as the weights of the hidden layer and is used to activate or deactivate a neuron in the hidden layer. The formula of applying the mask is

$$HiddenNeurons = \text{HiddenNeurons} \odot \text{Mask}.$$

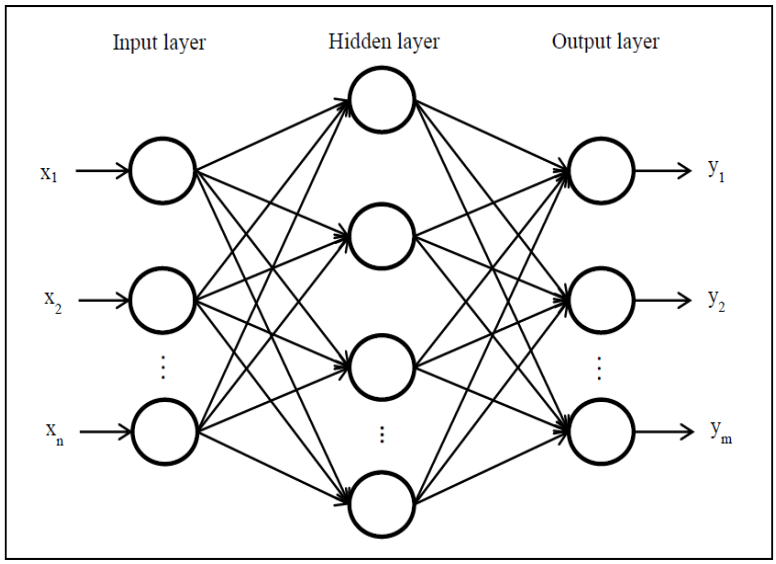

Fig. 3. Baseline Model Architecture

Furthermore, the optimization algorithm used in our model is Adaptive Moment Estimation (Adam) introduced by [10], which is proved to be a better default optimizer. Cross-entropy loss, which measures the performance of a classification model whose output is a probability value between 0 and 1, is suitable for this task since the output of our model is a probability that indicates which class the input is in.

2.4 Deep Learning Models

For comparing the performance of different size of neural networks, we further applied ResNet18 [11], MobileNetV2 [12] and MobileNetV3 [13] on our training/validation divided data set. Since the SFEW data contains only 100 images for each label, it is not feasible to use such a data set to train these deep models. Instead of random initialized the parameters of these models, we initialized them with pre-trained networks that are trained on ImageNet. Next, we change the last fully connected layer of these models so that it can fit with the number of our labels. Then we fine-tune the model with our divided data set. For each of these models, we finetuned it 100 epochs.

2.5 Network Pruning

As described in [8], the distinctiveness of the hidden units is determined by the vector of unit output activations on the pattern display set. So, in a hidden layer, if a pair of neurons perform a similar operation, then we consider one of these pair of neurons is lack distinctiveness and can be pruned. If a pair of neurons perform an opposite operation, then we consider both of these neurons are lack distinctiveness since their operations inhibit each other, resulting in no significant impact on the results.

By defining the similarity of a pair of neurons in a hidden layer, we calculate the angle between these two neurons in pattern space:

$$similarity(A, B) = \frac{A \cdot B}{\|A\| \times \|B\|} = \frac{\sum_{i=1}^{n} A_i \times B_i}{\sqrt{\sum_{i=1}^{n} A_i^2} \times \sqrt{\sum_{i=1}^{n} B_i^2}}$$

We consider two neurons to be similar if the angle between them is less than 15 degrees. For similar neurons, we add the value from one (a) of these two neurons to another one (b) and use 0 to mask (a) so to make this neuron non-functional in the network. We consider two neurons to be complementary to each other if the angle between them is larger than 165 degrees. For this kind of pairs of neurons, we use 0 to mask both of them so to make them non-functional in the network.

For each of the pruning step, we prune only one of the neurons, or two for the complementary situation, then perform the model training step before performing the next pruning. This pruning is taken until there are no similar or complementary neurons.

2.6 Quantization

Quantization involves improving the efficiency of deep learning computations by smaller representations of model weights, such as representing 32-bit floating-point weights as 8-bit integers. The specific quantization technique we utilize in the MobileNetV2 model is "post-training static quantization". This technique involves quantizing the weights after training rather than during training (this is referred to as quantization-aware training). Compared to quantization-aware training, post-training static quantization usually cost less time for quantization and higher accuracy lost.

3 Result and Discussion

3.1 Experiment Result

One-hidden-layer Network and Pruning. In [7], the baseline classification of over accuracy is 19.0%. So, for our 1-hidden-layer model, if the test accuracy can be higher than 19%, then it is acceptable. We train our baseline model without any pruning for 2000 epochs with a learning rate set to 1e-4, and the loss is 1.8041, the accuracy is 30.18% for training, 22.58% for testing. The changes in loss during training is shown in Fig.4

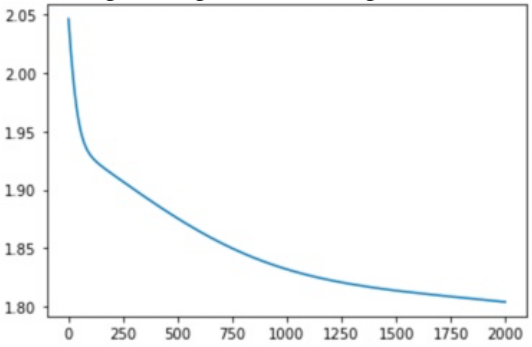

Fig. 4. Loss during training of baseline model

In our experiment, 2 pairs of neurons appear to be similar and none of the pairs of neurons is the complement. After performing the first pruning and re-train for 2000 epochs, the testing accuracy is 27.42%. After performing the second pruning and re-train, the test accuracy is 20.97%.

Finetuning Pretrained Models and Quantization. The changes in loss and accuracy during finetuning ResNet18, MobileNetV2 and MobileNetV3 are shown in Fig.5, Fig.6 and Fig.7 correspondingly. Finetuning these models using GPU is also somewhat time-consuming for 100 epochs. However, As shown in the figures, we can clearly see that all models converge around 25 epochs. Considering this, the time cost becomes acceptable.

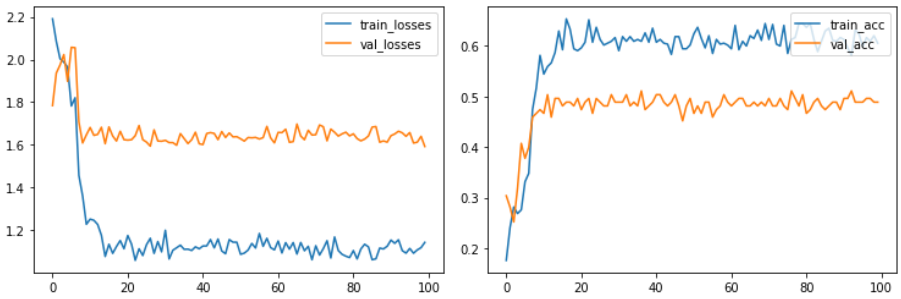

Fig. 5. Loss and accuracy change during finetuning ResNet18

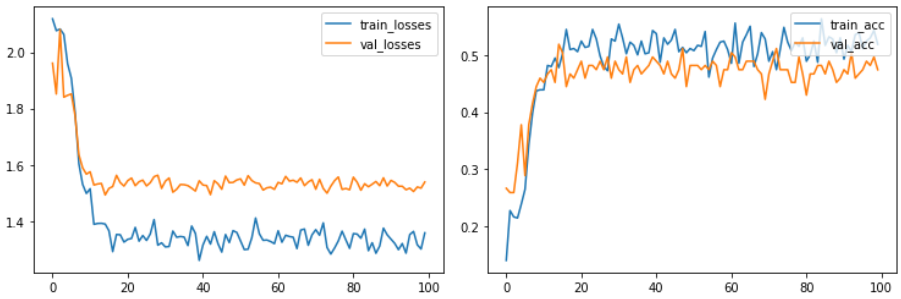

Fig. 6. Loss and accuracy change during finetuning MobileNetV2

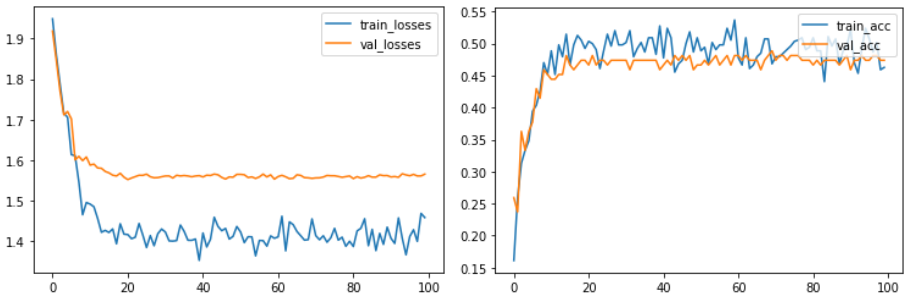

Fig. 7. Loss and accuracy change during finetuning MobileNetV3

The accuracy of all the experiments are shown in Table 1.

Table 1. The validation accuracy result of different models

Model	Accuracy (%)	Accuracy after compress (%)
One-hidden-layer Network	22.58	20.97
ResNet18	51.11	-
MobileNetV2	51.85	40.74
MobileNetV3	48.89	-

3.2 Discussion

The testing accuracy of our one-hidden-layer network is better than the baseline classification accuracy in [7]. After the first pruning and re-train, the test accuracy growth dramatically to 27.42%. This is somehow counter-intuitive since lesser neurons usually perform worse. Since our pruned model inherits the weights of the baseline model, the trained weight may overfit the test data even we have never used the test date for training or tuning hyper-parameters based on the result of the testing accuracy. After the second pruning and re-train, the test accuracy falls back to 20.97%, which is now reasonable. The testing accuracy dropped by 1.61%, but still better than the baseline classification accuracy in [7] which indicates that the performance of this pruned model is acceptable.

As shown in Table 1, the Accuracy of ResNet18, MobileNetV2 and MobileNetV3 are 51.11%, 51.85% and 48.89% correspondingly. Comparing to our one-hidden-layer network and the baseline accuracy in [7], the accuracy of these

models is significantly better. And of course, these models take significantly more time in training and inferencing, and more space to store the model. Using distinctiveness of the neurons to prune such deep neural networks with a large number of parameters is not feasible. Because to determine the distinctiveness, cosine similarity is needed to be calculated for each combination of any two neurons. This calculation is time-consuming and cannot be accelerated using GPUs like training a deep model. The purpose of using the pruning method is to compress the size of neural networks and improve the computation efficiency while doing inference. To fulfil this purpose with another approach, we used quantization. After quantized MobileNetV2 from float32 to int8, the size of the model drops from 9.18 MB to 2.36 MB. And the precision loss is (51.85% - 40.74% =) 11.11%. This is a big loss in accuracy. But it still outperforms the baseline model significantly.

4 Conclusion and Future Work

Pruning one neuron per step sometimes could result in dramatically growth of testing accuracy, which could because the weights of keep neurons inherit from the model before pruning is somehow overfitting to the test data. This strange behavior cannot be explained from our current experiments, which provides a direction for further analysis of the performance during pruning neural networks. After all the pruning operations are done, the performance of the pruned neural network is worse than the original neural network, however, it is still better than the baseline performance in [7]. In our approach to implementing pruning is using a 0 mask to force the corresponding neurons non-functional, but the linear computation process is still run through the pruned neurons, which is causing the time usage of running the neural network to stay almost the same. A further improvement of our implementation could be to really prune the neurons by deleting the neurons rather than using a mask to simulate the pruning.

From our experiment results, it is obvious that when the architecture of the neural network is larger and more complex, the performance of the model becomes better. To compress and accelerate the deep neural network, we could also use quantization, which is smaller representations of model weights, such as representing 32-bit floating-point weights as 8-bit integers. But a big loss in accuracy occurs after quantization. The reason is that the technique we used is post-training static quantization. In future work, we could use quantization-aware training instead of post-training static quantization, which may help to decrease the loss in accuracy while benefits from quantization.

References

1. LeCun, Y., Bengio, Y., & Hinton, G. (2015). Deep learning. *nature*, *521*(7553), 436-444.
2. Deng, L., Li, J., Huang, J. T., Yao, K., Yu, D., Seide, F., ... & Acero, A. (2013, May). Recent advances in deep learning for speech research at Microsoft. In *2013 IEEE International Conference on Acoustics, Speech and Signal Processing* (pp. 8604-8608). IEEE.
3. Collobert, R., Weston, J., Bottou, L., Karlen, M., Kavukcuoglu, K., & Kuksa, P. (2011). Natural language processing (almost) from scratch. *Journal of machine learning research*, *12*(ARTICLE), 2493-2537.
4. Krizhevsky, A., Sutskever, I., & Hinton, G. E. (2012). Imagenet classification with deep convolutional neural networks. *Advances in neural information processing systems*, *25*, 1097-1105.
5. Ojansivu, V., & Heikkilä, J. (2008, July). Blur insensitive texture classification using local phase quantization. In *International conference on image and signal processing* (pp. 236-243). Springer, Berlin, Heidelberg.
6. Bosch, A., Zisserman, A., & Munoz, X. (2007, July). Representing shape with a spatial pyramid kernel. In *Proceedings of the 6th ACM international conference on Image and video retrieval* (pp. 401-408).
7. Dhall, A., Goecke, R., Lucey, S., & Gedeon, T. (2011, November). Static facial expression analysis in tough conditions: Data, evaluation protocol and benchmark. In 2011 IEEE International Conference on Computer Vision Workshops (ICCV Workshops) (pp. 2106-2112). IEEE.
8. Gedeon, T. D., & Harris, D. (1992, June). Progressive image compression. In [Proceedings 1992] IJCNN International Joint Conference on Neural Networks (Vol. 4, pp. 403-407). IEEE.
9. Dhall, A., Goecke, R., Lucey, S., & Gedeon, T. (2011). Acted facial expressions in the wild database. Australian National University, Canberra, Australia, Technical Report TR-CS-11, 2, 1.
10. Kingma, D. P., & Ba, J. (2014). Adam: A method for stochastic optimization. arXiv preprint arXiv:1412.6980.
11. He, K., Zhang, X., Ren, S., & Sun, J. (2016). Deep residual learning for image recognition. In *Proceedings of the IEEE conference on computer vision and pattern recognition* (pp. 770-778).
12. Howard, A., Sandler, M., Chu, G., Chen, L. C., Chen, B., Tan, M., ... & Adam, H. (2019). Searching for mobilenetv3. In *Proceedings of the IEEE/CVF International Conference on Computer Vision* (pp. 1314-1324).
13. Sandler, M., Howard, A., Zhu, M., Zhmoginov, A., & Chen, L. C. (2018). Mobilenetv2: Inverted residuals and linear bottlenecks. In *Proceedings of the IEEE conference on computer vision and pattern recognition* (pp. 4510-4520).
14. Krishnamoorthi, R. (2018). Quantizing deep convolutional networks for efficient inference: A whitepaper. *arXiv preprint arXiv:1806.08342*.
15. Deng, J., Dong, W., Socher, R., Li, L. J., Li, K., & Fei-Fei, L. (2009, June). Imagenet: A large-scale hierarchical image database. In *2009 IEEE conference on computer vision and pattern recognition* (pp. 248-255). Ieee.