Deep Learning for Real-Time Robust Facial Expression Recognition on a Smartphone

Inchul SONG, Hyun-Jun KIM, and Paul Barom JEON
Samsung Advanced Institute of Technology, Youngin-si, Korea
{inchul2.song,dannis.kim,paul.barom.jeon}@samsung.com

Abstract—**We developed a real-time robust facial expression recognition function on a smartphone. To this end, we trained a deep convolutional neural network on a GPU to classify facial expressions. The network has 65k neurons and consists of 5 layers. The network of this size exhibits substantial overfitting when the size of training examples is not large. To combat overfitting, we applied data augmentation and a recently introduced technique called "dropout". Through experimental evaluation over various face datasets, we show that the trained network outperformed a classifier based on hand-engineered features by a large margin. With the trained network, we developed a smartphone app that recognized the user's facial expression. In this paper, we share our experiences on training such a deep network and developing a smartphone app based on the trained network.**

I. INTRODUCTION

With technological breakthroughs in recent years, it has now become possible to train a deep neural network with many layers and parameters [1][2]. The largest neural network that has ever been successfully trained contained 60 million parameters and was of 8 layers, which was unprecedentedly large [2]. The network was trained with the ImageNet dataset, which contains over 15 million labeled high-resolution images belonging to around 22k categories, and showed a record-breaking performance. Such a large network has long been thought hard or impossible to train, but its structure and efficient implementation over highly-optimized GPUs made it possible to successfully train such a huge network [3]. In some pattern recognition applications such as hand-written digit recognition and traffic sign recognition, a deep neural network either came close to human performance or even outperformed [4].

Motivated by the success story of training such a deep neural network, we implemented a deep convolutional neural network based on the publicly available code [5] and trained a large, deep convolutional neural network to classify facial expressions for various face datasets. To overcome overfitting commonly occurred in the large network of this size, we applied data augmentation and a recently introduced technique called "dropout", which resembles model averaging. The network trained on a highly parallelized GPU for a couple of hours showed superior classification performances compared to a traditional approach, a classifier based on human-engineered features.

With the trained model, we developed a real-time facial expression recognition function on a smartphone based on a client-server architecture, the demo of which is shown in Fig. 1. The developed smartphone app captures the user's face image and reports the predicted facial expression in real-time. The real-time robust facial expression recognition function we developed can be used in many application scenarios such as a music recommendation system that suggests a song that may comfort the user based on the emotion extracted from the user's facial expression. Implementing the best shot feature in a camera app on a smartphone is another possible application.

Fig. 1. Demo on a smartphone. It shows that the user's current facial expression is predicted as sadness with high probability.

In this paper, we share our experiences on training such a large neural network and implementing a real-time robust facial expression recognition function. In Section II, we outline the structure of the deep architecture and describe how to train it. With a network of this size, overfitting becomes a serious problem. We also explain how to combat overfitting. In Section III, we present the performance evaluation results over various face datasets. In Section IV, we describe how to implement a real-time facial expression recognition function on a smartphone. In Section V, we discuss the lessons we learned and some research directions tackling the difficulties of training deep architecture. Finally, we conclude the paper in Section VI.

II. THE NETWORK

The structure of our network is shown in Fig. 2. It contains five layers; the first four layers are convolutional with optional max pooling and the last one is fully-connected. The output of the last layer is fed into a 5-way softmax that produces a distribution over 5 facial expression types. The first and second convolutional layers (each of which is denoted by a pair of $\text{conv}_{\{1,2\}}$ and $\text{pool}_{\{1,2\}}$) filter input images with 64

978-1-4799-1291-9/14/$31.00 ©2014 IEEE

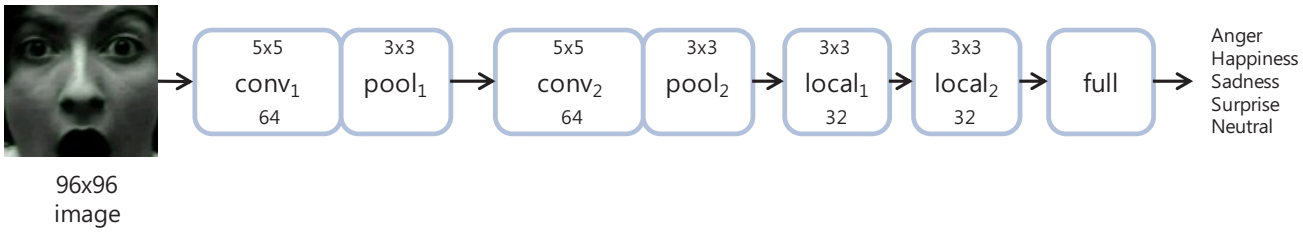

Fig. 2. Network structure. Each box indicates the type of operation applied to its input. The size NxN indicated in each box represents the size of area to which a given operation is applied. The numbers below in some boxes are the number of different applications of a given operation to the input.

kernels of size 5x5 with a stride of 1 pixel and pool with a grid of pooling units spaced 2 pixels apart, each summarizing a neighborhood of size 3x3 centered at the location of the pooling unit. Fig. 3 shows the network connections of the first convolutional layer in more detail. In the figure, a sold arrow connecting a squared area to a point indicates that all units in the squared area are connected to the unit pointed by the arrow. The same filter is applied to every squared area in the same input image. As a result of applying 64 different filters to the input, 64 output images are produced. Pooling is applied to each image separately. After pooling with stride 2, the size of the input gets reduced by half.

The connection patterns of the second convolutional layer are more complicated and shown in Fig. 4. Each unit on the right-hand side is connected to the units in every squared area from all input images on the left-hand side. And different filters are applied to different input images although the same filter is applied within the same input image. Pooling is the same as in the first convolutional layer. In the third and fourth layers (denoted by $local_1$ and $local_2$ in Fig. 2), a different set of filters is applied to every location in the input with 32 sets of filters for each layer; they are just like the first two convolutional layers, but without any weight sharing. The neurons of the last fully connected layer (denoted by full) are connected to all neurons in the fourth layer.

With a network of this size, overfitting becomes a serious problem. We applied a recently-introduced technique, called "dropout" [9], to the third and fourth layers of the network to combat overfitting. Dropout is an efficient version of model averaging applied to big neural networks. During training, the output of each hidden unit is set to zero with probability of 0.5, which is similar to choosing a different neural network every time. The difference from the traditional model averaging is that those chosen sub-networks share their weights. At test time, the outputs of neurons are multiplied by 0.5, which is an approximation to taking the average of the predictive distributions produced by the different sub-networks. Without dropout, the network exhibited substantial overfitting, i.e., the gap between the training error and test error was large. Dropout increased the generalization performance of the network by 1.9% to 3.9% for various face datasets.

We also used data augmentation, a well-known technique to overcome overfitting especially for big neural networks. We performed data augmentation that enlarged the datasets using image transformations. For each 96x96 training image, we extracted five 88x88 patches from the four corners and the center of the image and also generated their horizontal reflections, which increased the size of the training examples by a factor of 10. Similarly, at test time, we generated ten patches from each test image and averaged the predictions over the ten patches.

The network was trained on a GPU using stochastic gradient descent with a batch size of 128 examples, momentum of 0.9, and weight decay of 0.0005. The weights were updated as follows:

$$v_{i+1} := 0.9 \cdot v_i - 0.0005 \cdot \epsilon \cdot w_i - \epsilon \cdot \langle \frac{\partial L}{\partial w_i} \rangle D_i$$

$$w_{i+1} := w_i + v_{i+1}$$

where i is the iteration index, v is the momentum variable, ϵ is the learning rate, and $\langle \frac{\partial L}{\partial w_i} \rangle D_i$ is the average over the ith batch D_i of the derivative of the objective with respect to w, evaluated at w_i. Derivatives were calculated by using the backpropagation algorithm as described in [10]. Equal running rates were used for all layers. It took a couple of hours to train the network on a NVIDIA GeForce Titan 6GB GPU. For more details of learning, refer to [2].

III. Results

TABLE I
Performance Comparisons Over Various Face Datasets

Dataset	No. of images	LBP+SVM	Ours
CK+	1,400	93%	99.2%
SAIT	7,970	90%	97.1%
SAIT2	8,510	83.5%	95.5%
Internet	1,860	78.8%	84.5%

The evaluation results on various face datasets are summarized in Table I. We considered four different datasets: the Cohn-Kanade Plus (CK+) dataset, SAIT, SAIT2, and Internet. The CK+ dataset is a standard dataset for facial expression recognition. SAIT and SAIT2 are datasets we collected from people inside and outside of SAIT. The difference between the two datasets is that SAIT contains only frontal faces, whereas SAIT2 has slightly rotated faces as well. The Internet dataset was created by downloading face images from the Internet and manually labeling them with five facial expressions. For all datasets, we considered the following facial expressions: anger, happiness, sadness, surprise, and neutral.

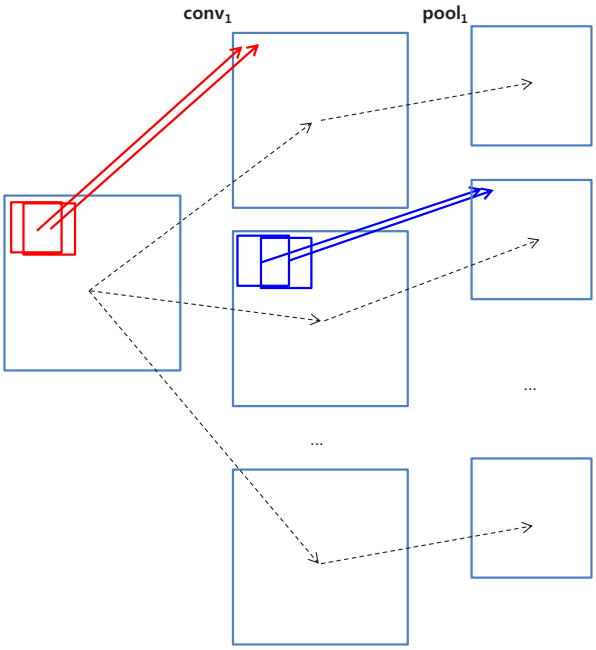

Fig. 3. Network connections of the first convolutional layer (best viewed in color). Each squared area in the input is connected to the unit in the output on the right-hand side. $conv_1$ produces 64 filtered output images. Pooling is applied to each of these images separately. After pooling, images get reduced by half. Dashed arrows show directions from input to output.

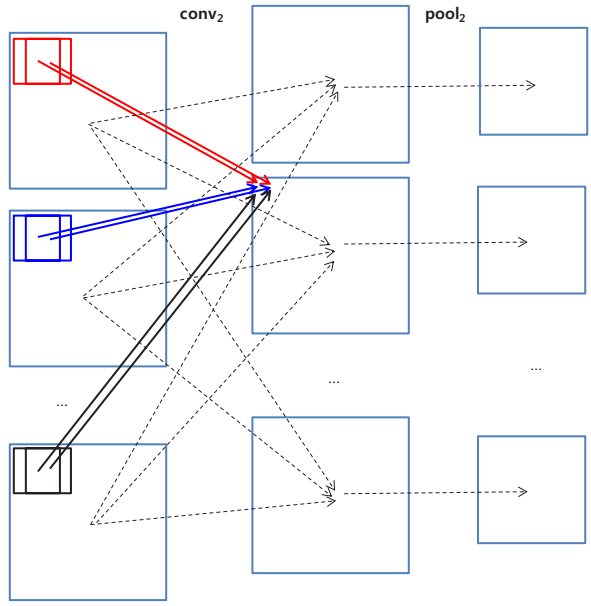

Fig. 4. Network connections of the second convolutional layer (best viewed in color). Each squared area from all input images is connected to the unit in the output on the right-hand side. Different filters are applied to different input images. As before, 64 different output images are produced after convolution. Pooling is also applied separately. Dashed arrows show the directions from input to output.

For each dataset, we trained the network as described in the previous section. In order to compare the deep architecture with a traditional approach based on hand-engineered features, we implemented an SVM classifier with popular hand-crafted features, called Local Binary Patterns (LBPs) [6], which encodes the differences between neighboring pixels. We performed 10-fold cross validation to obtain average test accuracies for our network and SVM classifier, which are shown in Table I. As shown in Table I, our network showed superior performances compared to LBP+SVM over various face datasets. For the face datasets CK+ and SAIT that contain only frontal faces and are well labeled, the network showed 97.1%-99.2% test accuracies, which outperformed LBP+SVM by 6.2-7%. For SAIT2 that contains non-frontal faces as well, the test accuracy of the network was 95.5%. A relatively lower test accuracy of 84.5% was obtained for the Internet face dataset. This is because the dataset contained insufficient training examples and it turned out that there were many confusing images downloaded from the Internet that were hard to decide which facial expression to label.

IV. Real-Time Facial Expression Recognition

We developed a real-time robust facial expression recognition function on a smartphone based on a client-server architecture, which is shown in Fig. 5. Given input from a remote smartphone client, the server responds with the most probable facial expression predicted by the trained model. We developed a smartphone app that captures the user's face with the frontal camera, sends a request containing the captured image to the server, and reports the predicted facial expression to the user. We added a pre-processing step so that the app sent a request only when it detected the user's full face. Fig. 1 shows a live demo of the smartphone app. It shows that the user's current facial expression is detected as sadness with

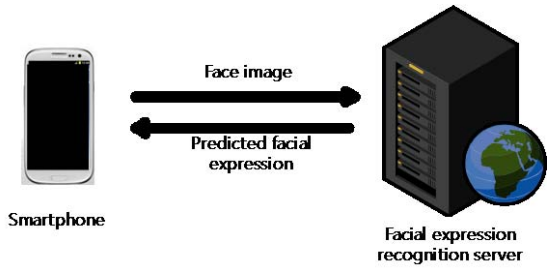

Fig. 5. Implementation of a facial expression recognition app on a smartphone. A smartphone app captures the user's face, sends a request, and reports the facial expression predicted by the server. The server predicts facial expressions by the trained model.

high probability. The prediction time per image on the server was around 50ms, and the round-trip time per image was less than 100ms. Up to hundreds of facial expressions can be recognized at once without further increase of prediction time since GPUs are highly parallel computing devices.

V. Discussions

In this paper, we trained a deep, large convolutional neural network for facial expression recognition and obtained excellent performances compared to a classifier based on hand-engineered features. However, this result came with cost. The deep architecture had numerous hyper-parameters to optimize such as learning rate, the number of layers, the type of each layer, the area size of convolution and pooling, and so on. Typically hyper-parameter optimization is done by grid search and manual search. This tuning is often a "black art" requiring expert experience, rule of thumbs, or sometimes brute-force search. In this paper, we started with the network structure that showed a good performance for the CIFAR-10 dataset, a standard dataset for object recognition, and manually tuned the hyper-parameters with a validation set. Recently,

computer clusters and GPU processors make it possible to use algorithmic approaches to hyper-parameter optimization. In [11], the authors showed that randomly chosen trials were more efficient than trials on a grid and suggested that random search should be used as a substitute and baseline instead of grid search. There is another body of work that uses sequential model-based global optimization (SMBO) algorithms where evaluation of the fitness function is expensive [12][13] (for a neural network, the fitness function can be defined as the test accuracy on the validation set). In an SMBO algorithm, the fitness function is modeled by the Gaussian Process based on the observation history. At every evaluation of the fitness function, the model for the function is adjusted and the next-to-evaluate hyper-parameters are chosen based on the estimation provided by the current model. In some cases, a deep neural network whose hyper-parameters were optimized by a SMBO algorithm achieved the state-of-the-art performance [13]. More recently, a generic method to incorporate knowledge from previous experiments was also proposed [14]. We believe that this line of work will improve the usability of the deep architecture significantly and accelerate the adoption of the deep architecture in many real-world applications.

VI. Conclusions

In this paper, we have described the experiences on developing a real-time robust facial expression recognition function on a smartphone. Our deep convolutional neural network trained on a GPU has shown superior performances for various face datasets compared to a classifier based on hand-crafted features. The model can be used to provide accurate real-time facial expression prediction services on consumer electronics devices such as smartphones and digital TVs.

As future work, we plan to use unsupervised pre-training that is known to be helpful to train deep architectures [7]. Another possible direction of our work is to implement the network on custom hardware architectures such as FPGAs for fast and energy-efficient operations in battery-powered mobile devices [8].

References

[1] Y. Bengio, "Learning deep architectures for AI," *Foundations and Trends in Machine Learning*, pp. 1-127, 2009.

[2] A. Krizhevsky, I. Sutskever, and G. E. Hinton, "ImageNet classification with deep convolutional neural networks," *NIPS*, 2012.

[3] X. Glorot and Y. Bengio, "Understanding the difficulty of training deep feedforward neural networks," *AISTATS*, 2010.

[4] D. Cireşan, U. Meier, and J. Schmidhuber, "Multi-column deep neural networks for image classification," *CVPR*, 2012.

[5] https://code.google.com/p/cuda-convnet

[6] T. Ojala and M. Pietikäinen, "Multiresolution gray-scale and rotation invariant texture classification with local binary patterns," *TPAMI*, vol. 24, no. 7, pp. 971-987, 2002.

[7] D. Erhan, P.-A. Manzagol, Y. Bengio, S. Bengio, and P. Vincent, "The difficulty of training deep architectures and the effect of unsupervised pre-training," *AISTATS*, 2009.

[8] C. Farabet, B. Martini, B. Corda, P. Akselrod, E. Culurciello, and Y. LeCun, "Neuflow: a runtime reconfigurable dataflow processor for vision," *CVPR Workshops*, 2011.

[9] G. E. Hinton, N. Srivastava, A. Krizhevsky, I. Sutskever, and R. R. Salakhutdinov, "Improving neural networks by preventing co-adaptation of feature detectors", CoRR, abs/1207.0580, 2012.

[10] C. M. Bishop, *Pattern Recognition and Machine Learning*, Springer, 2006.

[11] J. Bergstra and Y. Bengio, "Random search for hyper-parameter optimization," *JMLR*, vol. 13, no. 1, pp. 281-305, 2012.

[12] J. Bergstra, R. Bardenet, Y. Bengio, and B. Kégl, "Algorithms for hyper-parameter optimization," *NIPS*, 2011.

[13] J. Snoek, H. Larochelle, and R. P. Adams, "Practical Bayesian optimization of machine learning algorithms," *NIPS*, 2012.

[14] R. Bardenet, M. Brendel, B. Kégl, and M. Sebag, "Collaborative hyperparameter tuning," *ICML*, 2013.