Optimizing Deep Convolutional Neural Network for Facial Expression Recognitions

Umesh Chavan and Dinesh Kulkarni

Abstract Facial expression recognition (FER) systems have attracted much research interest in the area of Machine Learning. We designed a large, deep convolutional neural network to classify 40,000 images in the dataset into one of seven categories (disgust, fear, happy, angry, sad, neutral, surprise). In this project, we have designed deep learning Convolution Neural Network (CNN) for facial expression recognition and developed model in Theano and Caffe for training process. The proposed architecture achieves 61% accuracy. This work presents results of accelerated implementation of the CNN with graphic processing units (GPUs). Optimizing Deep CNN is to reduce training time for system.

Keywords Convolutional neural network · Deep learning · Graphical processing unit (GPU)

1 Introduction

1.1 Background

Facial expression recognition have found applications in technical fields such as Human–computer Interaction (HCI) which detect people's emotions using their facial expressions and security monitoring [1]. Use of Machine learning is powerful approach to detect and classify images. To improve their performance, it is necessary to collect larger datasets, as well as need to build powerful models. The weakest point of machine learning is that it cannot do feature engineering. The limitations of machine learning in many cases learned model does not generalize well. An algorithm can only work well on data with assumption of the training data. The biggest drawback is it is time consuming for learning with large datasets with powerful model. Deep Learning (DL) is a new advancement in area of machine

U. Chavan (✉) · D. Kulkarni
Walchand College of Engineering, Sangli, Maharashtra, India
e-mail: umesh.chavan@walchandsangli.ac.in

© Springer Nature Singapore Pte Ltd. 2019
V. E. Balas et al. (eds.), *Data Management, Analytics and Innovation*,
Advances in Intelligent Systems and Computing 808,
https://doi.org/10.1007/978-981-13-1402-5_14

learning research whose motivation is moving closer to the objective of Artificial Intelligence (AI). Convolutional neural networks (CNNs) are a special kind of DL method. CNNs are useful in the area of computer vision. Facial expression recognition (FER) techniques detect people's emotions using their facial expressions. We build a model for FER using deep CNN. CNNs have much fewer parameters as compared to neural networks. So they are easier to train. More training time in CNN with large dataset is major bottleneck. We designed model in Theano framework [2] and exploited Graphics Processing unit (GPU) computation. The result shows that it is achieving 2–5 times speedup with training on GPU. Also shows that it achieves 61% accuracy. Our intention is to exhibit the performance and scalability improvement for FER using deep CNN.

2 Convolution Neural Network (CNN)

A CNN [3] is an advance in neural network evolution. It consists of sequences of one or more convolutional layers (CLs). CL's are mostly with pooling layers (PLs). PLs are succeeding by one or more fully connected layers (FCs), FCs are just as standard neural network. Accurate and correct feature extraction is necessary. This is the base for CNN. Input to a neural network is fed from output of feature extractor. It is challenging work to select a "suitable" feature extractor. It cannot adapt to network configuration. It is not part of learning procedure. These layers arranged in feed-forward structure as shown in Fig. 1. In a CNN only a small region/subset of input layer connects neurons in hidden layer. These regions are referred as Local receptive fields. The local receptive field is translated across an image to create a feature map. It can use convolution to implement this operation efficiently. So it's called as a CNN. The typical neural network has parameters—weights and biases [4]. The model learns these values during the training process and it's continuously updates with each new training examples. However, in the case of CNN, weights and bias values are the same for all hidden neurons in a given layer. This means that all hidden neurons detect the same feature such as an edge or

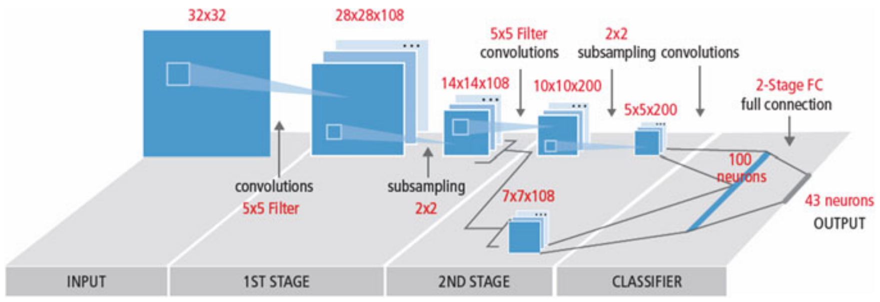


Fig. 1 A typical CNN model

blobs in different regions of the input image. This makes network tolerant to translation of object in an image. Activation step applies the transformation to the output of each neuron by using activation functions Rectified Linear Unit commonly known as a ReLUs [5]. Most DL network use ReLU for hidden layers. The power of ReLU is it trains much faster is more expressive than other alternatives—logistic function. Also ReLU prevents the gradient vanishing problem. It takes output of neurons and maps it the highest positive value or if the output is negative the function maps it to zero. We can further transform the activation step by applying a pooling step. Pooling reduced the dimensionality of the feature map by condensing the output of small regions of neurons into a single output. The CNN with its layer(s) is briefly explored in [6].

3 GPU (Graphics Processor Units) Programming

GPUs are massively parallel numeric processors. It is programmed in C with extensions for GPU programmers. It has application programming interfaces for programmers. It takes advantages of heterogeneous computing systems that contain both CPUs and massively parallel GPU's. For a GPU developer, the computing environment consists of a host that is traditional CPU and one or more devices that are processors with massive number of arithmetic units. GPU is a typically known as device in CUDA. Use of GPUs together with a CPU is GPU-accelerated computing. It accelerates deep learning applications. This work presents results of accelerated implementation of the deep CNN in graphic processing units (GPUs) for FER.

3.1 CUDA Programming Architecture

CUDA architecture allows the programmer to write one code that will run on both CPU and GPU. In CUDA GPU is referred as a device and CPU is referred as Host. CUDA assumes that the device and the host have their own separate memories where they store data. The function that executes on GPU is known as a Kernel. The kernel is invoked and executed by 100 s or even 1000 s of threads at a time. The CPU launches the kernels with a specific syntax to let GPU know how many threads should be used. The kernel function reserves memory in the device on board global memory. It takes one of the device's pointers as the first argument and the second argument is how many bytes to reserve. This function copies data from the host and device memories. CUDA provides a scalable approach to express parallelism. The CUDA is suitable for large amount of data and having lots of computations like image convolution. It achieves high throughput. CUDA uses thousands of threads executing in parallel, all these threads are executing the same function which is known as a kernel. Programmer can organize threads into blocks.

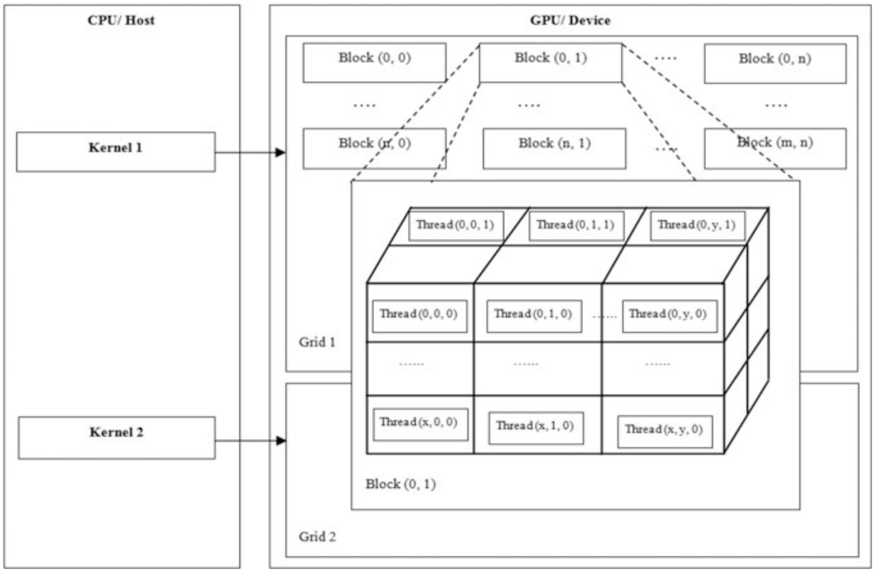


Fig. 2 CUDA programming model

These thread blocks are again arranged into grids of multiple thread blocks. All thread blocks at the same time work together through shared memory. A thread block has its own ID for its grid. Streaming microprocessors (SMs) are the part of GPU that actually runs this kernel. SMs are the heart of the architecture. They perform computations which have their own control units, execution pipelines, caches and registers. A typical CUDA architecture is shown in Fig. 2.

3.2 System Design

We built a CNN that has two convolution layers and two fully connected (FC) layers. In the first convolution layer, have 20, 5 $\times$ 5 filters with pooling. In the second convolution layer, we had 20, 5 $\times$ 5 filters and also pooling. In all convolution layers ReLU activation function is used. In the first FC layer it have 500 neurons and in second FC layer have 300 neurons. In both FC layers same as in the convolution layer we used ReLU activation function. Also we used softmax as loss function. We trained the network for varying number of epochs on each run (for 2, 10, 30, 50, 70 epochs) and with batch size of 30 samples. We cross-validated the hyper-parameters.

3.3 CNN Model

For an image with a size $M \times N$ of and kernels with a size of $m \times n$, the convolution is represented as

$$z_{ij}^{(k)} = \sum_{s=0}^{m-1} \sum_{t=0}^{n-1} w_{st}^{(k)} x(i+s)(j+t) \tag{1}$$

Here w is the weight of the kernel, which is the model parameter. Above Eq. (1) is for one kernel. For multi-convolution layers, the equation is

$$z_{ij}^{(k)} = \sum_{c} \sum_{s=0}^{m-1} \sum_{t=0}^{n-1} w_{st}^{(k,c)} x_{(i+s)(j+t)}^{(c)}(i+s)(j+t) \tag{2}$$

Here, c denotes the channel of the image. If the number of kernels is k and the number of channels is c, we have $w \in R^{k \times c \times m \times n}$. Then we see from the Eq. (2) that the size of convolved image is $(M - m + 1) \times (N - n + 1)$. After the convolution, all the convolved values will be activated by the activation function. We will implement CNN with the ReLU (rectified Linear Unit) function. With the activation we have

$$a_{ij}^{(k)} = h(z_{ij}^{(k)} + b^{(k)} = \max(o, z_{ij}^{(k)} + b^{(k)})) \tag{3}$$

where $b \in R^k$, a one-dimensional array. Next is the max-pooling layer. The propagation can simply be expressed as

$$y_{ij}^{(k)} = \max(z_{(l_1 i+s)(l_2 j+t)}^{(k)}) \tag{4}$$

Here l_1 and l_2 are the size of pooling layers and $s \in [0, l_1], t \in [0, l_2]$.

Usually l_1 and l_2 are set to same sizes (2 or 4). The simple Multilayer Perceptron Network (MLP) follows after sequences of convolutional layers and pooling layers to classify data. MLP can accept one-dimensional data. Output of CLs and PLs are two-dimensional, we need to flatten the down sampled/pooled data as preprocessing to adapt it to input to input layer of MLP. The error from input layer of MLP is back-propagated to the max-pooling layer, and this time it is un-flattened to two dimensions to be adapted properly to the model. Max-pooling layer simply back-propagates error to its previous layer as max-pooling layer does not have parameters. The equation can be expressed as

$$\frac{\partial E}{\partial_{(l_1 i+s)(l_2 j+t)}^{(k)}} = \begin{cases} \frac{\partial E}{\partial y_{ij}^{(k)}} \text{ if}, y_{ij}^{(k)} = a_{(l_1 i+s)(l_2 j+t)}^{(k)} \\ 0 \end{cases} \tag{5}$$

Here E denotes the evaluation function, the error is then back-propagated to the CL, and with it can calculate the gradient of the weight and bias. Gradient of bias is represented as

$$\frac{\partial E}{\partial b^{(k)}} = \sum_{i=0}^{M-m}\sum_{j=0}^{N-m} \frac{\partial E}{\partial a_{ij}^{(k)}} \frac{\partial a_{ij}^{(k)}}{\partial b^{(k)}} \tag{6}$$

$$\partial_{ij}^{(k)} = \frac{\partial E}{\partial a_{ij}^{(k)}} \tag{7}$$

$$c_{ij}^{(k)} = z_{ij}^{(k)} + b^{(k)} \tag{8}$$

Then we get,

$$\frac{\partial E}{\partial b^{(k)}} = \sum_{i=0}^{M-m}\sum_{j=0}^{N-m} d_{ij}^{(k)} \partial_{ij}^{(k)} \frac{\partial a_{ij}^{(k)}}{\partial c_{ij}^{(k)}} \frac{\partial c_{ij}^{(k)}}{\partial b_{ij}^{(k)}} = \sum_{i=0}^{M-m}\sum_{j=0}^{N-m} d_{ij}^{(k)} h'(c_{y}^{(k)}) \tag{9}$$

In the same way the gradient for weight (kernel) is

$$\frac{\partial E}{\partial w_{st}^{(k,c)}} = \sum_{i=0}^{M-m}\sum_{j=0}^{N-m} d_{ij}^{(k)} h'(c_{y}^{(k)}) x_{(i+s)(j+t)}^{(c)} \tag{10}$$

When we think for multi-convolutional layers it is necessary to calculate the error of convolutional layers.

$$\frac{\partial E}{\partial w_{st}^{(c)}} = \sum_{k}\sum_{s=0}^{M-m}\sum_{t=0}^{N-m} \frac{\partial E}{\partial z_{i(i-s)(j-t)}^{(k)}} \frac{\partial z_{i(i-s)(j-t)}^{(k)}}{\partial x_{ij}^{(c)}} = \frac{\partial E}{\partial z_{i(i-s)(j-t)}^{(k)}} w_{st}^{(k,c)} \tag{11}$$

so, the error can be expressed as

$$\frac{\partial E}{\partial x_{ijst}^{(c)}} = \sum_{k}\sum_{s=0}^{M-m}\sum_{t=0}^{N-m} \partial_{(i-s)(j-t)}^{(k)} h'(c_{(i-s,j-t)}^{(k,c)}) \tag{12}$$

4 Experiments and Results

All benchmark in this paper were performed in machine having computation platform with (1) CPU: AMD Phenom II X4 B97—processor; (2) GPU is GeForce GTX520, compute capability 2.1, 48 cores. The software platform is composed of: Ubuntu 14.04 Operating system, CUDA 7.5, Python with Theano. All training and testing are in single precisions.

4.1 Program Code

In the part of code snippet; the model is created in Python having two convolutional layers and one FC layer. The first CL has 32 filters of size 32 × 32. Second CL has 64 filters of 3 × 3 sizes. The object CNN is instantiated with parameters for CNN layers.

```
def main():
X, Y = getImageData()
model = CNN(convpool_layer_sizes=[(32, 3,3), (64, 3, 3),(96,3,3),
(128,3,3)], hidden_layer_sizes=[200],
)
model.fit(X, Y,epochs=3,batch_sz=30)
if __name__ == '__main__':
main()
```

4.2 Dataset

The experiment was conducted on the dataset provided by Kaggle [7] website for Facial Expression Recognition Challenge [8]. This dataset consists of 37,000—48 × 48 pixel gray-scale images of faces. Each image is labeled with one of seven expression categories: Fear, Happy, Sad, Angry, Disgusts, Surprise, and Neutral. We used a training set of 36,000 samples, a validation set of thousand examples.

The emotions are labeled in each image. Network is trained on the dataset, which comprises 48-by-48-pixel gray-scale images of human faces each labeled with one of seven expressions. Some samples images with labeled expression are shown in Fig. 3. There are variations in the dataset considerably in scale, rotation, and illumination.

4.3 Experiment

We built a CNN that had two convolution layers and two fully connected (FC) layers. In the first convolution layer, we had 20, 5 × 5 filters with pooling. In the second convolution layer, we had 20, 5 × 5 filters and also pooling. In all convolution layers ReLU activation function is used. In the first FC layer we had 500 neurons and in second FC layer we had 300 neurons. In both FC layers same as in the convolution layer we used ReLU activation function. Also we used softmax as our loss function. Figure 4 shows the architecture of this deep network. We trained the network for varying number of epochs on each run (for 2, 10, 30, 50, 70 epochs) and

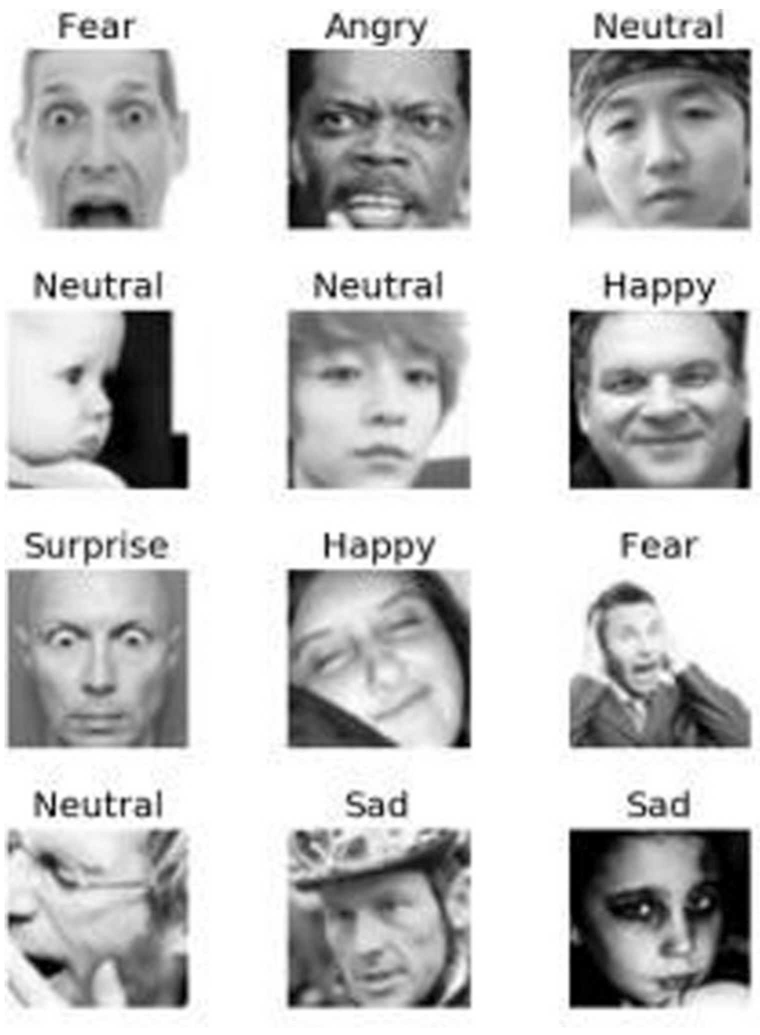


Fig. 3 Example images from Kaggle dataset [7]

with batch size of 30 samples. We cross-validated the hyper-parameters (Learning rate, regularization, decay, epsilon, Batch size) as shown in Table 1. To make the model training faster, we exploited GPU-accelerated deep learning facilities on Theano [2] library in using Python.

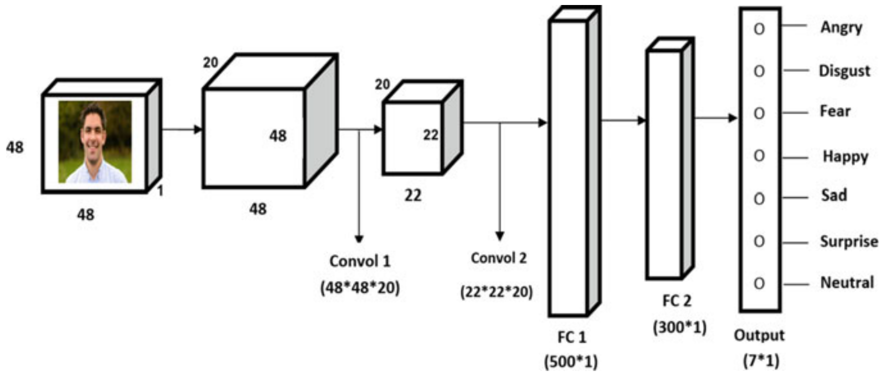


Fig. 4 FER CNN architecture

Table 1 The hyper-parameters for model

Parameters	Value
Learning rate	0.00001
Regularization	0.0000001
Decay	0.9999
Epsilon	0.001
Batch size	30

4.4 Results and Evaluation

The final validation accuracy we obtained is 61% for training with epochs =10. Training loss plot is shown in Fig. 5. The confusion matrix for classification is shown in Fig. 7. The performance in GPU speedup over Kaggle [7] FER 2013 dataset is shown in Table 2. We can see the performance improvement in speed of execution time of CPU and GPU training with different number of epochs which is shown in Table 2. The average speedup gain is approximately five times (Figs. 6 and 7).

5 Future Scope

The proposed work can be further extended by increasing the number of different expressions other than the six universal expressions (anger, fear, disgust, joy, surprise, sadness). The classification of other facial expressions may require the

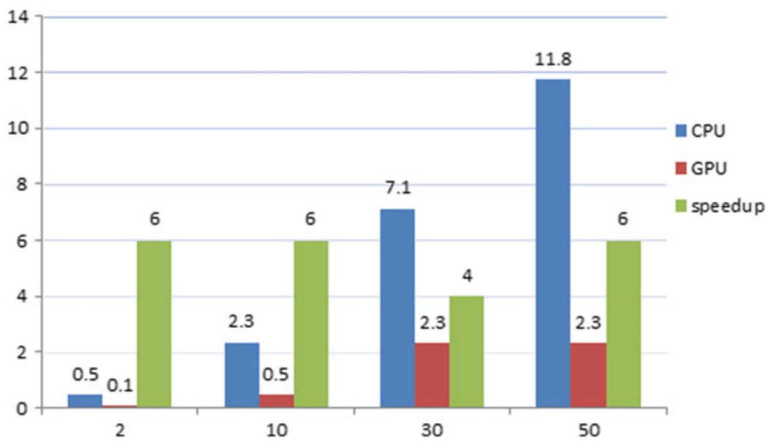


Fig. 5 Execution time (in h) and speedup with GPU

Table 2 Performance: execution time and accuracy

Epochs	Execution time (in minutes)		Speedup	Accuracy (%)
	CPU	GPU		
2	28	5	6	39
10	141	27	6	55
30	427	141	4	59
50	706	141	6	61

extraction and tracing of additional facial points and corresponding features. The system can be improved by using a wider training set so as to cover a wider range of poses and cases of low quality of images.

6 Conclusions

Although topology structure of convolution neural network is simple, it still needs a huge amount of work in calculation. NVIDIA GPU based on hardware architecture of stream processor has significant improvement in face expression recognition based on convolution neural network in support of programming model in CUDA. Compared with CPU, it has amazing advantages. Experiments show that stream

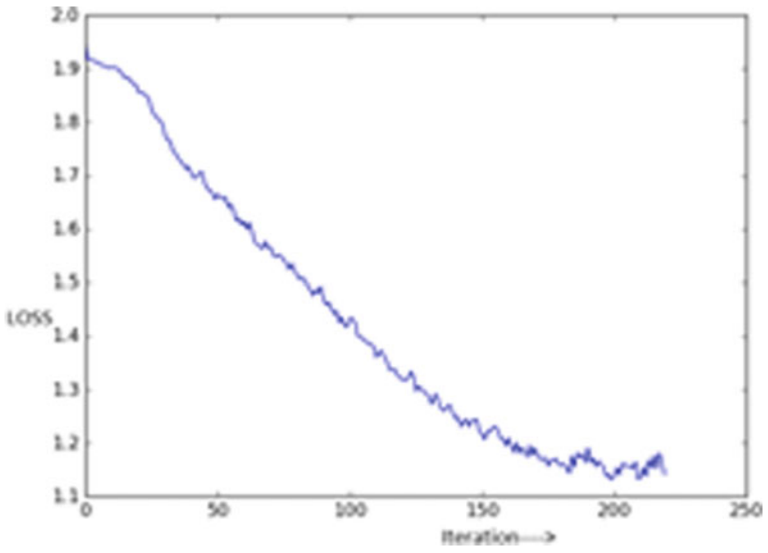
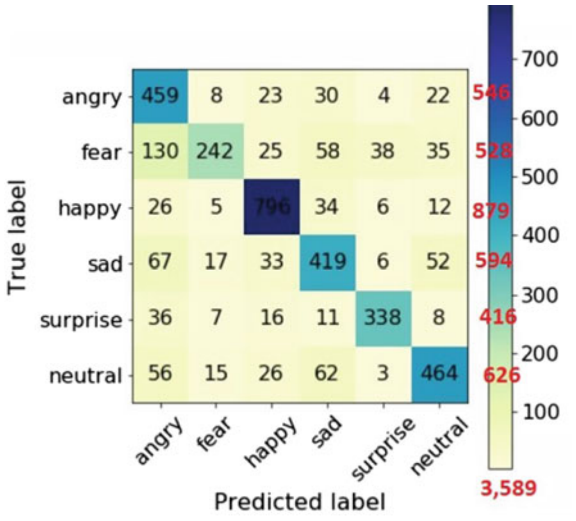


Fig. 6 Training loss

Fig. 7 Confusion matrix



	angry	fear	happy	sad	surprise	neutral
angry	459	8	23	30	4	22
fear	130	242	25	58	38	35
happy	26	5	796	34	6	12
sad	67	17	33	419	6	52
surprise	36	7	16	11	338	8
neutral	56	15	26	62	3	464

processor is suitable for convolution neural network. The results in this work show that GPUs are just as fast and efficient for deep learning. In this work, we evaluated their performance using different performance measurement and visualization techniques. Some of the difficulties with improving this is that images are very small and some cases it is difficult to distinguish which emotion is on each image.

Bibliography

1. Pantic, M., & Rothkrantz, L. J. M. (2004). Facial action recognition for facial expression analysis from static face images. *IEEE Transactions on Systems, 34*(3).
2. Bergstra, J., Breuleux, O., Bastien, F., Lamblin, P., Pascanu, R., Desjardins, G., et al. (2010). Theano: A CPU and GPU math compiler in Python. In *Proceedings 9th Python in Science Conference* (pp. 1–7).
3. Hinton, G. E., Osindero, S., & Teh, Y.-W. (2006). A fast learning algorithm for deep belief nets. *Neural Computation, 18*(7), 1527–1554.
4. Krizhevsky, A., Sutskever, I., & Hinton, G. E. (2012). Imagenet classification with deep convolutional. In *Advances in Neural Information Processing Systems*.
5. Nair, V., & Hinton, G. E. (2010). Rectified linear units improve restricted Boltzmann machines. In *ICML*, 2010.
6. Chavan, U., & Kulkarni, D. (2016). Accelerating learning performance of facial expression recognition using convolution neural network. *International Journal of Control Theory and Applications, 9*(43).
7. www.kaggle.com/c/challenges-in-representation-learning-facial-expression-recognition-challenge.
8. LeCun, Y., Cortes, C., & Burges, C. J. C. (1998). The MNIST database of handwritten digits.