Classification of Image using Convolutional Neural Network (CNN)

Md. Anwar Hossain[α] & Md. Shahriar Alam Sajib[σ]

Abstract- Computer vision is concerned with the automatic extraction, analysis, and understanding of useful information from a single image or a sequence of images. We have used Convolutional Neural Networks (CNN) in automatic image classification systems. In most cases, we utilize the features from the top layer of the CNN for classification; however, those features may not contain enough useful information to predict an image correctly. In some cases, features from the lower layer carry more discriminative power than those from the top. Therefore, applying features from a specific layer only to classification seems to be a process that does not utilize learned CNN's potential discriminant power to its full extent. Because of this property we are in need of fusion of features from multiple layers. We want to create a model with multiple layers that will be able to recognize and classify the images. We want to complete our model by using the concepts of Convolutional Neural Network and CIFAR-10 dataset. Moreover, we will show how MatConvNet can be used to implement our model with CPU training as well as less training time. The objective of our work is to learn and practically apply the concepts of Convolutional Neural Network.

Keywords: convolutional neural network, CIFAR-10 dataset, MatConvNet, relu, softmax.

I. Introduction

Convolutional Neural Networks (CNN) becomes one of the most appealing approaches recently and has been an ultimate factor in a variety of recent success and challenging applications related to machine learning applications such as challenge ImageNet object detection, image classification, and face recognition. Therefore, we consider CNN as our model for our challenging tasks of image classification. We use CNN for segmentation and classification of the images in academic and business transactions. We use image recognition in different areas for example automated image organization, stock photography, face recognition, and many other related works.

a) *CIFAR-10 Database*

The CIFAR-10 database (Canadian Institute for Advanced Research database) is a collection of images. We use this dataset to train machine learning and computer vision algorithms. CIFAR-10 database is the contribution of Alex Krizhevsky and Geoffrey Hinton. This dataset has 60,000 colored images. It has ten classes, and they are an airplane, automobile, bird, cat, deer, dog, frog, horse, ship, truck. The images are of size 32x32 pixels. The dataset consists of 50,000 training and 10,000 testing examples. It is a database for people who want to try learning techniques and pattern recognition methods on real-world data while spending minimal efforts on preprocessing and formatting. We will use this database in our experiment.

b) *Convolutional Neural Networks*

Convolutional neural networks are deep artificial neural networks. We use CNN to classify images, cluster them by similarity (photo search), and perform object recognition within scenes. It can be used to identify faces, individual, street signs, tumors, platypuses and many other aspects of visual data. The convolutional layer is the core building block of a CNN. The layer's parameters consist of a set of learnable filters (or kernels) which have a small receptive field but extend through the full depth of the input volume. During the forward pass, each filter is convolved across the width and height of the input volume, computing the dot product, and producing a 2-dimensional activation map of that filter. As a result, the network learns about the filters. The filter activates when they see some specific type of feature at some spatial position in the input. Then the activation maps are fed into a downsampling layer, and like convolutions, this method is applied one patch at a time. CNN has also fully connected layer that classifies output with one label per node.

II. Related Works

Image recognition has an active community of academics studying it. A lot of important work on convolutional neural networks happened for image recognition [1,2,3,4]. The most dominant recent works achieved using CNN is a challenging work introduced by Alex Krizhevsky [5], who used CNN for challenge classification ImageNet. Active areas of research are: object detection [14,15,16], scene labeling [17], segmentation [18,19], face recognition, and variety of other tasks [20,21,22].

Author α: Assistant Professor, Department of Information & Communication Engineering, Faculty of Engineering & Technology, Pabna University of Science & Technology, Pabna, Bangladesh.
e-mail: manwar.ice@gmail.com
Author σ: Student, Department of Information & Communication Engineering, Faculty of Engineering & Technology, Pabna University of Science & Technology, Pabna, Bangladesh.
e-mail: sajibpust130639@gmail.com

III. Methodology

Deep Learning has emerged as a main tool for self-perception problems like understanding images, the voice from humans, robots exploring the world. We aim to implement the concept of the Convolutional Neural Network for the recognition of images. Understanding CNN and applying it to the image recognition system is the target of the proposed model. Convolutional Neural Network extracts the feature maps from the 2D images by using filters. The Convolutional neural network considers the mapping of image pixels with the neighborhood space rather than having a fully connected layer of neurons. The Convolutional neural network has been proved to be a very dominant and potential tool in image processing. Even in the fields of computer vision such as handwriting recognition, natural object classification, and segmentation, CNN has become a much better tool compared to all other previously implemented tools.

a) The architecture of the Proposed Model

When one starts learning deep learning with the neural network, he realizes that one of the most supervised deep learning techniques is the Convolutional Neural Network. We design Convolutional Neural Network to recognize visual patterns directly from pixel images with minimal preprocessing. Almost all CNN architectures follow the same general design principles of successively applying convolutional layers to the input, periodically downsampling (Max pooling) the spatial dimensions while increasing the number of feature maps. Moreover, there are also fully connected layers, activation functions and loss function (e.g., cross entropy or softmax). However, among all the operations of CNN, convolutional layers, pooling layers, and fully connected layers are the most important ones. Therefore, we will quickly introduce these layers before presenting our proposed model.

The Convolutional layer is the very first layer where it can extract features from the images. Because pixels are only related to the adjacent and close pixels, convolution allows us to preserve the relationship between different parts of an image. Convolution is filtering the image with a smaller pixel filter to decrease the size of the image without losing the relationship between pixels. When we apply convolution to a 7x7 image by using a filter of size 3x3 with 1x1 stride (1-pixel shift at each step), we will end up having a 5x5 output.

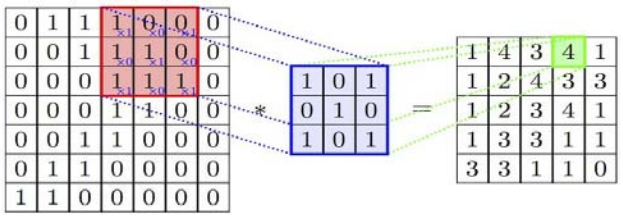


Fig. 1: Convolution Operation

When constructing CNN, it is common to insert pooling layers after each convolution layer, so that we can reduce the spatial size of the representation. This layer reduces the parameter counts, and thus reduces the computational complexity. Also, pooling layers help with the overfitting problem. We select a pooling size to reduce the amount of the parameters by selecting the maximum, average, or sum values inside these pixels. Fig.2 shows the max pooling and average pooling operation.

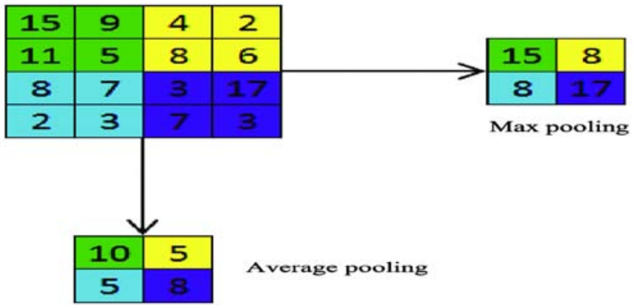


Fig. 2: Max pooling and Average pooling operation

A fully connected network is in any architecture where each parameter is linked to one another to determine the relation and effect of each parameter on the labels. We can vastly reduce the time-space complexity by using the convolution and pooling layers. We can construct a fully connected network in the end to classify our images.

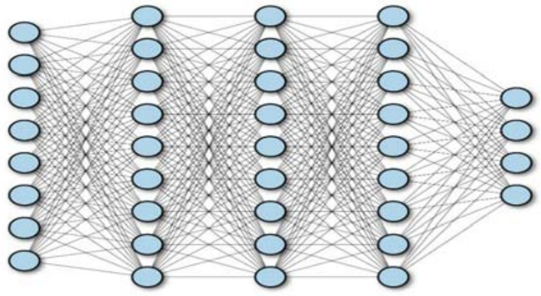


Fig. 3: Fully connected layer

Fig.4 shows the overview look of our proposed convolutional neural network. It is very much similar to the other image recognition architectures [1,2,3,4] but has changed in the number of filters, neurons and activation functions for better performance. We can divide our model into six sequences of layers.

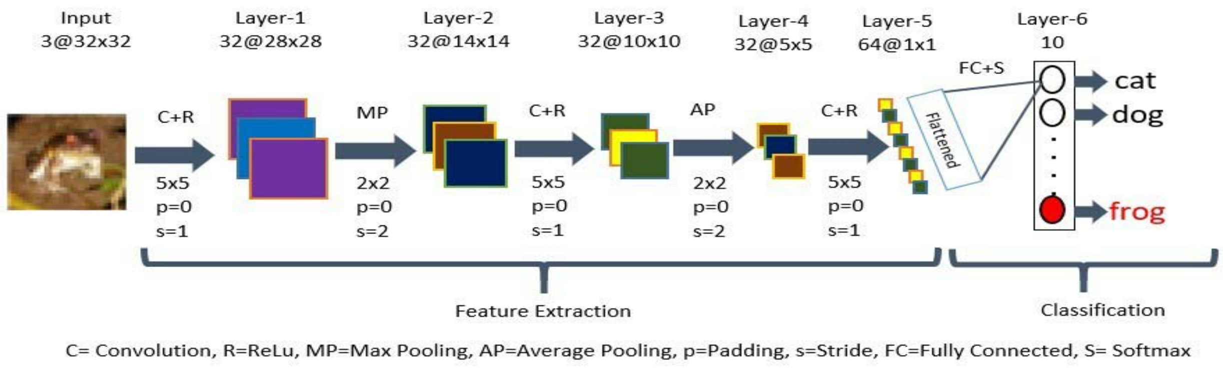


Fig. 4: The architecture of our proposed CNN

b) Explanation of the Model

A simple convolutional network is a sequence of layers. The layer transforms one volume of activations to another through a differentiable function. We use three main types of layers to build network architecture. They are a convolutional layer, pooling layer, and fully connected layer. We will stack these layers to form six layers of network architecture. We will go into more details below.

Fig.4 shows the architecture of our proposed CNN model. At first, we need some pre-processing on the images like resizing images, normalizing the pixel values, etc. After the necessary pre-processing, data is ready to be fed into the model.

Layer-1 consists of the convolutional layer with ReLu (Rectified Linear Unit) activation function which is the first convolutional layer of our CNN architecture. This layer gets the pre-processed image as the input of size n*n=32*32. The convolutional filter size (f*f) is 5*5, padding (p) is 0(around all the sides of the image), stride (s) is 1, and the number of filters is 32. After this convolution operation, we get feature maps of size 32@28*28 where 32 is the number of feature maps which is equal to the number of filters used, and 28 comes from the formula ((n+2p-f)/s) +1= ((32+2*0-5)/1) +1=28. Then the ReLu activation is done in each feature map.

Layer-2 is the max pooling layer. This layer gets the input of size 32@28*28 from the previous layer. The pooling size is 2*2; padding is 0 and stride is 2. After this max pooling operation, we get feature maps of size 32@14*14. Max pooling is done in each feature map independently, so we get same number feature maps as the previous layer, and 14 comes from the same formula ((n+2p-f)/s) +1. This layer has no activation function.

Layer-3 is the second convolutional layer with ReLu activation function. This layer gets the input of size 32@14*14 from the previous layer. The filter size is 5*5; padding is 0, the stride is 1, and the number of filters is 32. After this convolution operation, we get feature maps of size 32@10*10. Then ReLu activation is done in each feature map.

Layer-4 is the average pooling layer. This layer gets the input of size 32@10*10 from the previous layer. The pooling size is 2*2; padding is 0 and stride is 2. After this max pooling operation, we get a feature map of size 32@5*5.

Layer-5 is the third convolutional layer with ReLu activation function. This layer gets the input of size 32@5*5 from the previous layer. The filter size is 4*4; padding is 0, the stride is 1, and the number of filters is 64. After this convolution operation, we get feature maps of size 64@1*1. This layer acts as a fully connected layer and produces a one-dimensional vector of size 64 by being flattened.

Layer-6 is the last layer of the network. It is a fully connected layer. This layer will compute the class scores, resulting in a vector of size 10, where each of the ten numbers corresponds to a class score, such as among the ten categories of CIFAR-10 dataset. For final outputs, we use the softmax activation function.

In this way, CNN transforms the original image layer by layer from the main pixel values to the final class scores. Note that some layers contain parameters, and others don't. In particular, the convolution/fully connected layers perform transformations that are a function of not only the activations in the input volume but also of the parameters (the weights and biases of the neurons). On the other hand, the Relu/pooling layers will implement a fixed function. We train the parameters in the convolutional/fully connected layers with stochastic gradient descent. By this process, we will prepare the trained model which will be used to recognize the image present in the test data. Thus, we can classify the images as Class- airplanes, cars, birds, cats, deer, dogs, frogs, horses, ships, trucks.

IV. Implementation

To implement our CNN architecture, we will use MatConvNet. MatConvNet is an implementation of Convolutional Neural Networks (CNN) for MATLAB [23]. We built our model by using MatConvNet so that our model has greater simplicity and flexibility. It exposes the building blocks of CNN as easy-to-use MATLAB functions, providing routines for computing linear convolutions with filter banks, feature pooling and many more. In this manner, MatConvNet allows fast prototyping of new CNN architectures; at the same time, it supports efficient computation on CPU and GPU allowing to train complex models on large datasets such as ImageNet ILSVRC.

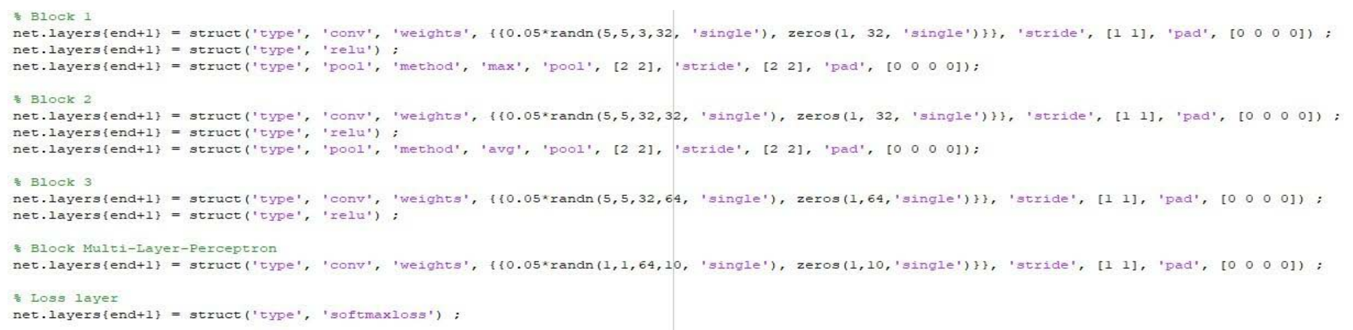
Convolutional Neural Networks (CNN) is the current state-of-art architecture for the image classification task. As shown in Fig.4 our proposed 2-D Convolutional Neural Network (CNN) model is designed using MatConvNet backend for the well known CIFAR-10 image recognition task. The whole workflow can be to preparing the data, building and compiling the model, training and evaluating the model and saving the model to disk for reuse.

Preparing the data is the first step of our approach. Before we build the network, we need to set up our training and testing data, combine data, combine labels and reshape into the appropriate size. We save the dataset of normalized data (single precision and zero mean), labels, and miscellaneous (meta) information.

Building and compiling the model is the second step. To create the CNN, we must initialize MatConvNets SimpleNN network and then define important initialization parameters for example batch size, number of epochs, learning rate, etc.

The batch size determines the number of samples for the training phase of the CNN. The CNN will process all the training data, but only in increments of the specified batch size. We can use batch size for computational efficiency, and its value will be dependent on the user's available hardware. An epoch is a successful forward pass and a backward pass through the network. It's usually beneficial to set its value high and then to reduce it once if one is satisfied with the convergence at a particular state (chosen epoch) in the network. Learning rate is a very sensitive parameter that pushes the model towards convergence. Finding its best value will be an experimental process unless one invokes more powerful techniques such as batch normalization. In our experiment, we use batch size 60, several epochs 300 and learning rate 0.0001 for maximum accuracy.

Now we can build our CNN by creating each layer individually as shown in fig 5. Afterward, we will invoke objective and error layers that will provide a graphical visualization of the training and validation convergence after completing each epoch. MatconvNet initializes the weights by using Gaussian distribution.



```
% Block 1
net.layers{end+1} = struct('type', 'conv', 'weights', {{0.05*randn(5,5,3,32, 'single'), zeros(1, 32, 'single')}}, 'stride', [1 1], 'pad', [0 0 0 0]) ;
net.layers{end+1} = struct('type', 'relu') ;
net.layers{end+1} = struct('type', 'pool', 'method', 'max', 'pool', [2 2], 'stride', [2 2], 'pad', [0 0 0 0]);

% Block 2
net.layers{end+1} = struct('type', 'conv', 'weights', {{0.05*randn(5,5,32,32, 'single'), zeros(1, 32, 'single')}}, 'stride', [1 1], 'pad', [0 0 0 0]) ;
net.layers{end+1} = struct('type', 'relu') ;
net.layers{end+1} = struct('type', 'pool', 'method', 'avg', 'pool', [2 2], 'stride', [2 2], 'pad', [0 0 0 0]);

% Block 3
net.layers{end+1} = struct('type', 'conv', 'weights', {{0.05*randn(5,5,32,64, 'single'), zeros(1,64,'single')}}, 'stride', [1 1], 'pad', [0 0 0 0]) ;
net.layers{end+1} = struct('type', 'relu') ;

% Block Multi-Layer-Perceptron
net.layers{end+1} = struct('type', 'conv', 'weights', {{0.05*randn(1,1,64,10, 'single'), zeros(1,10,'single')}}, 'stride', [1 1], 'pad', [0 0 0 0]) ;

% Loss layer
net.layers{end+1} = struct('type', 'softmaxloss') ;
```

Fig. 5: CNN layers in MatConvNet

The third step is the training and evaluating the model. Training a CNN requires computing the derivative of the loss concerning the network parameters. We calculate the derivatives using an algorithm called back propagation which is a memory-efficient implementation of the chain rule for derivatives. We built the model and performed a random gradient descent training according to the Stochastic Gradient Descent (SGD) training algorithm. We have used SGD training algorithm to adjust the weight of the connection between neurons so that the loss reaches a minimum value or stops after several epochs. We have used CPU training. It is important to note that GPU training will dramatically help training time for CNN.

Lastly, we can begin the training of CNN by supplying the training data, the constructed model and the current batch of data. When training the CNN, only the data specified for training play a role in minimizing error in the CNN. We feed the training data through the network for the forward pass and backward pass. The validation data is just used to see how the CNN responds to new similar data. We do not use the validation data to train the network. Afterward, we save the trained CNN and prepare for the testing phase.

During the training phase of the CNN, the simple network will produce three plots (Objective, Top1error, and Top5error) for each epoch. The top1 error is the chance that class with the highest probability is the correct target. In other words, CNN guesses the target correctly. The top5error is the chance that the true target is one of the top five probabilities. The objective for the simple network should mirror the form of the top1 and top5 error. In all the plots, we represent the training error and validation error by blue and orange respectively.

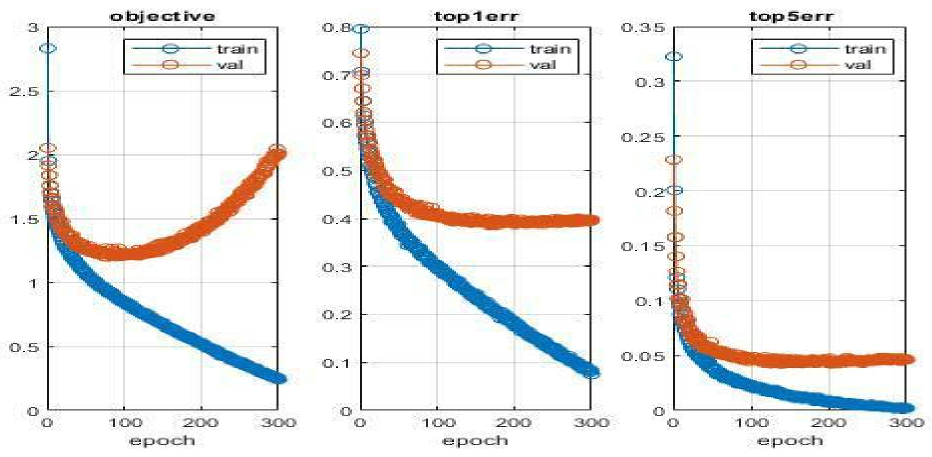


Fig. 6: Objective, Top1error, and Top5error during training

Finally, by using the testing data, we can evaluate our model. The following are an example of classification outputs from the simple network on the CIFAR-10 data.

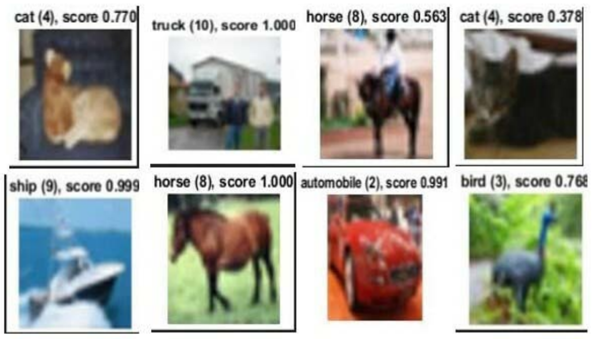


Fig. 7: Some correct recognized outputs

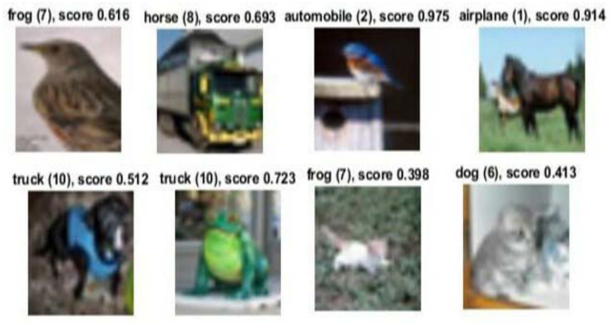


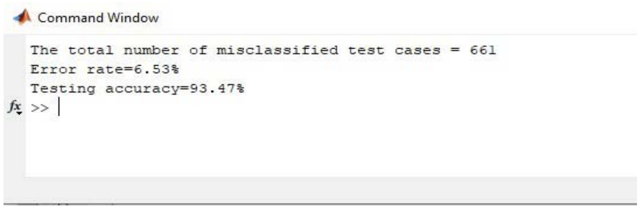
Fig. 8: Some wrong recognized outputs

We can determine the test cases that show failed classifications. The model can't identify some images because of limitations in the input of standard data images. Moreover, missing pixels caused by image compression and image sharpness problems are also reasons for misclassification.

The fourth and final step is to save the model in the disk for reuse. We store the trained model in a MATLAB file format. Hence the saved model can be reused later or easily ported to other environments too.

V. Results and Discussion

Among 10,118 test cases, our model misclassifies total of 661 images after three hundred epochs which correspond to 93.47% recognition rate shown in Table1. The results are pretty good for three hundred epochs and for such a simple model with CPU training and less training time (about 3 hours).



```
Command Window
  The total number of misclassified test cases = 661
  Error rate=6.53%
  Testing accuracy=93.47%
fx >> |
```

Fig. 9: Error rate and accuracy of our model.

Although there are some images which are difficult to identify, our model will be able to classify them correctly. For example, our model can recognize the following image and classify it as 'deer':

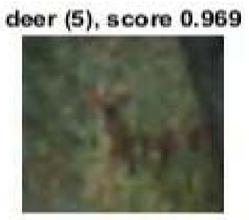


Fig. 10: Correct recognition of the bad image

Table 1: Summary of the experiment

Batch size	No. of epochs	Testing accuracy
100	250	76.82%
70	300	82.28%
60	300	93.47%

Test accuracy 93.47% implies that the model is trained well for prediction. Training set size affects the accuracy increases as the number of data increases. The more data in the training set, the smaller the impact of training error and test error and ultimately the accuracy can be improved.

VI. Conclusion and Future Work

Here we demonstrate a model which can recognize and classify the image. Later it can be extended for object recognition, character recognition, and real-time object recognition. Image recognition is an important step to the vast field of artificial intelligence and computer vision. As seen from the results of the experiment, CNN proves to be far better than other classifiers. The results can be made more accurate by increasing the number of convolution layers and hidden neurons. People can recognize the object from blurry images by using our model. Image recognition is an excellent prototype problem for learning about neural networks, and it gives a great way to develop more advanced techniques of deep learning. In the future, we are planning to develop a real-time image recognition system.

References Références Referencias

1. Kuntal Kumar Pal, Sudeep K. S.(2016)." Preprocessing for Image Classification by Convolutional Neural Networks", IEEE International Conference on Recent Trends in Electronics Information Communication Technology, May 20-21, 2016, India.
2. Hayder M. Albeahdili, Haider A. Alwzwazy, Naz E. Islam (2015)."Robust Convolutional Neural Networks for Image Recognition", (IJACSA) International Journal of Advanced Computer Science and Applications, Vol. 6, No. 11, 2015.
3. Jingkun Qin, Haihong E, Meina Song and Zhijun Ren(2018)."Image Retrieval Based on a Hybrid Model of Deep Convolutional Encoder", 2018 the International Conference of Intelligent Robotic and Control Engineering.
4. K Sumanth Reddy, Upasna Singh, Prakash K Uttam(2017)." Effect of Image Colourspace on Performance of Convolution Neural Networks", 2017 2nd IEEE International Conference on Recent Trends in Electronics Information & Communication Technology (RTEICT), May 19-20, 2017, India.
5. Krizhevsky, Alex, Sutskever, Ilya, and Hinton, Geoffrey. ImageNet classification with deep convolutional neural networks. In Advances in Neural Information Processing Systems 25 (NIPS'2012). 2012.
6. Marc'Aurelio Ranzato, Fu-Jie Huang, Y-Lan Boureau, Yann LeCun, —Unsupervised Learning of Invariant Feature Hierarchies with Applications to Object Recognition‖ CVPR, 2007.
7. Matthew D Zeiler and Rob Fergus. Stochastic pooling for regularization of deep convolutional neural networks. arXiv preprint arXiv:1301.3557, 2013.
8. Ian J Goodfellow, David Warde-Farley, Mehdi Mirza, Aaron Courville, and Yoshua Bengio. Maxout networks. arXiv preprint arXiv:1302.4389, 2013.
9. Nitish Srivastava, Geoffrey Hinton, Alex Krizhevsky, Ilya Sutskever, and Ruslan Salakhutdinov‖ Dropout: A Simple Way to Prevent Neural Networks from Overfitting‖ Journal of Machine Learning Research 15 (2014) 1929-1958.
10. Fabien Lauer, Ching Y. Suen, and G´erard Bloch —A trainable feature extractor for handwritten digit recognition‖ Journal Pattern Recognition 2007.
11. Chen-Yu Lee, Saining Xie, Patrick Gallagher, Zhengyou Zhang, Zhuowen Tu, — Deeply-Supervised Nets —NIPS 2014.
12. M. Fischler and R. Elschlager, —The representation and matching of pictorial structures,‖ IEEE Transactions on Computer, vol. 22, no. 1, 1973.
13. Kevin Jarrett, Koray Kavukcuoglu, Marc'Aurelio Ranzato and Yann LeCun ‖ What is the Best Multi-Stage Architecture for Object Recognition?‖ ICCV'09, IEEE, 2009.
14. Kaiming, He and Xiangyu, Zhang and Shaoqing, Ren and Jian Sun —Spatial pyramid pooling in deep convolutional networks for visual recognition‖ European Conference on Computer Vision, 2014.
15. Ross Girshick, —Fast R-CNN —arXiv preprint arXiv:1504.08083, 2015
16. X. Wang, M. Yang, S. Zhu, and Y. Lin. Regionlets for generic object detection. In ICCV, 2013. 8.
17. Karen Simonyan and Andrew Zisserman —VERY DEEP CONVOLUTIONAL NETWORKS FOR LARGE-SCALE IMAGE RECOGNITION‖ arXiv:1409.1556v5 [cs.CV] 23 Dec 2014.
18. C. Couprie, C. Farabet, L. Najman, and Y. LeCun. Indoor semantic segmentation using depth information. Internatinal Conference on Learning Representation, 2013. 2.
19. R. Girshick, J. Donahue, T. Darrell, and J. Malik. Rich feature hierarchies for accurate object detection and semantic segmentation. CoRR, abs/1311.2524, 2013. 4.
20. J. Deng, W. Dong, R. Socher, L.-J. Li, K. Li, and L. FeiFei. Imagenet: A large-scale hierarchical image database. In CVPR, 2009. 2.
21. L. N. Clement Farabet, Camille Couprie and Y. LeCun. Learning hierarchical features for scene labeling. PAMI, 35(8), 2013. 1, 2.
22. Honglak Lee, Roger Grosse, Rajesh Ranganath, and Andrew Y. Ng —Convolutional Deep Belief Networks for Scalable Unsupervised Learning of Hierarchical Representations‖.
23. "MatConvNet Convolutional Neural Networks for MATLAB" Andrea Vedaldi, Karel Lenc, Ankush Gupta.