ImageNet Classifier using TFLearn with TensorFlow

Narendra Kumar Manoharan, Pradeepa Shri Jothinathan
The University of Texas at Arlington

Abstract

Image recognition and classification have become household names similar to that of artificial intelligence and machine learning. The advent of artificial intelligence in our day-to-day activities have been inevitable ever since the major breakthrough of high-powered low-energy consuming devices and technologies. Here we can see the effect of one such technology, Deep Convolutional nets. This technique has brought about major breakthroughs in the processing of images, video and audio. Especially in image recognition and classification. We train a deep convolutional network of 5 layers to classify the Cifar-10 dataset, 102 category Flower datset,test data and we were able to achieve reduced error rates of 8-9%. To reduce overfitting and optimize performance we use fractional max-pooling to augment the results. We use both traditional convolutional network and a 5-layer implementation of Alexnet to build our model.

Introduction

Convolutional neural networks are very similar to normal neural networks but with nonlinear activation functions such as Rectified Linear Unit(ReLU) and TanH functions that result in sparsely connected networks which learns the values of its filters by itself. There are four key concepts behind ConvNets that take advantage of the properties of natural signals: local connections, shared weights, pooling and the use of many layers.

A typical convolutional network architecture consists of Convolutional layers, Polling Layer and Fully-connected Layer. In high level, the working of an Image Classifier built with a CNN can be summarized as the following. The first layer can be used to detect edges from raw pixels, then use these edges to detect simple shapes in the second layer, and then use these shapes to detect higher-level features, such as facial shapes in higher layers. The last layer is then a classifier that uses these high-level features. The activation functions play a vital role in the process of backward propagation of the output values. And the use of non-linear functions also increases the sparse ativation of the network.

Convolutional Neural Network

The architecture of a typical ConvNet is structured as a series of stages. The first few stages are composed of two types of layers: convolutional layers and pooling layers. Units in a convolutional layer are organized in feature maps, within which each unit is connected to local patches in the feature maps of the previous layer through a set of weights called a filter bank. The result of this local weighted sum is then passed through a non-linearity such as a ReLU.

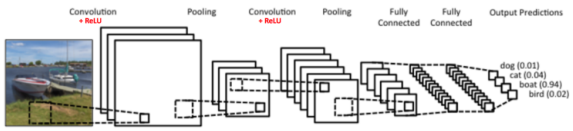

Fig 1 | Typical Convolutional Neural Network with Convolution, Pooling and fully connected layers.

Convolution Layer

The primary purpose of Convolution in case of a ConvNet is to extract features from the input image. Convolution preserves the spatial relationship between pixels by learning image features using small squares of input data. The CONV layer's parameters consist of a set of learnable filters. Every filter is small spatially (along width and height), but extends through the full depth of the input volume. For example, a typical filter on a first layer of a ConvNet might have size 5x5x3 (i.e. 5 pixels width and height, and 3 because images have depth 3, the color channels).

During the forward pass, we slide (more precisely, convolve) each filter across the width and height of the input volume and compute dot products between the entries of the filter and the input at any position. As we slide the filter over the width and height of the input volume we will produce a 2-dimensional activation map that gives the re-

sponses of that filter at every spatial position. Intuitively, the network will learn filters that activate when they see some type of visual feature such as an edge of some orientation or a blotch of some color on the first layer, or eventually entire honeycomb or wheel-like patterns on higher layers of the network. Now, we will have an entire set of filters in each CONV layer (e.g. 12 filters), and each of them will produce a separate 2-dimensional activation map. We will stack these activation maps along the depth dimension and produce the output volume.

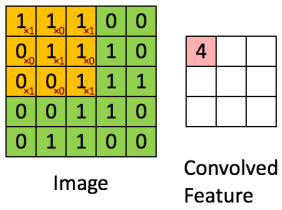

Fig 2 | The covolution operation with the sliding window with the resulting feature map.

Pooling Layer

It is common to insert a Pooling layer in-between successive Conv layers in a ConvNet architecture. Its function is to progressively reduce the spatial size of the representation to reduce the amount of parameters and computation in the network, and hence to also control overfitting. The Pooling Layer operates independently on every depth slice of the input and resizes it spatially, using the MAX operation. The most common form is a pooling layer with filters of size 2x2 applied with a stride of 2 downsamples every depth slice in the input by 2 along both width and height, discarding 75% of the activations. Every MAX operation would in this case be taking a max over 4 numbers (little 2x2 region in some depth slice). The depth dimension remains unchanged.

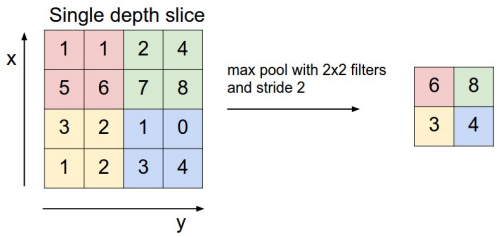

Fig 3 | Max-pooling layer with stride 2

Fully Connected Layer

The Fully Connected layer is a traditional Multi Layer Perceptron that uses a softmax activation function in the output layer (other classifiers like SVM can also be used, but will stick to softmax in this post). The term "Fully Connected" implies that every neuron in the previous layer is connected to every neuron on the next layer. The sum of output probabilities from the Fully Connected Layer is 1. This is ensured by using the Softmax as the activation function in the output layer of the Fully Connected Layer. The Softmax function takes a vector of arbitrary real-valued scores and squashes it to a vector of values between zero and one.

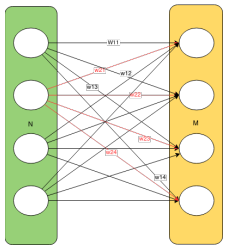

Fig 4 | Fully connected neuron network

Dataset

In order to test the model that we implemented, we use both high resolution images from the 102 Category flower dataset and Low resolution images from the CIFAR 10 dataset. And for the purpose of testing the performance of the models we use both large and small training and testing validation sets. This will give us a better idea of how the models might perform in real life situations. We use categorical cross-entropy as our loss function to calculate the performance of the models.

Attributes	CIFAR - 10	102 Flower
Classes	10	102
Number of Images	60,000	8,189
Resolution	32 X 32	227 X 227
Images in Each Class	6,000	40 - 258
Baseline Accuracies	89.36	91.52

Implementation

The models used for Classification are constructed using TensorFlow with the TFlearn wrapper which helps access the underlying TensorFlow API in a simpler manner. Also, the scipy stack is a pre-requisite for TFlearn. While implementing the models, there have been several changes to the TensorFlow API without backwards compatibility. This has lead us to move from using Python 3.5 to Python 2.7 for the sake of backward compatibility.

Technical Specifications

- Python 2.7
- TensorFlow 0.10.0
- TFLearn 0.2.1
- CUDA 8
- CuDNN v5

Amazon Web Services (EC2 Instance)

Since CNN deals with images and graphic processing, the help of GPU's might be helpful in reducing the computation time a great deal. Using Amazon Web Services helped us a lot in that regards. We were able to rent Amazon EC2 with 32 vCPUs and 80 gigabytes of storage to help us out with the execution of our models. We were able to cut down our computation time by almost 85%. The specification of our EC2 instances are listed below.

API Name	g2.8x
Storage	260GB SSD
vCPU	32
ECU	104
Memory	60GB
GPU	4 NVIDIA GRID GPU
GPU Memory	4 GB
CUDA	1536 cores

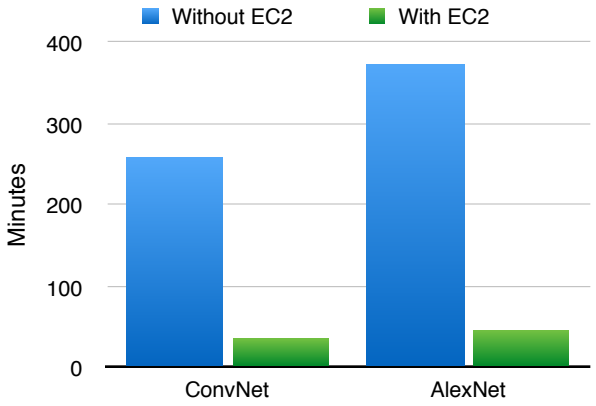

Fig 5 | Comparison of execution time with and without EC2 instances for the CIFAR 10 dataset

Model

We initially planned to construct complex CNN models for classification of the images with the highest accuracy possible. With that in mind, we implemented three models, the basic convolutional neural network for image classification, Alex net which has been tuned for image classification and GoogLeNet which is the industry standard of image classifiers in the industry right now.

LeNet

The typical LeNet consists of three layers of convolution with an input image size of 32x32. In order to input other images into the network they have to intially be downsampled to the appropriate dimensions. The LeNet consists of alternating layers of convolution and max-pooling followed by a fully connected layer for classifying the samples using a Softmax function. Also the convolution is done via a ReLU activation function.

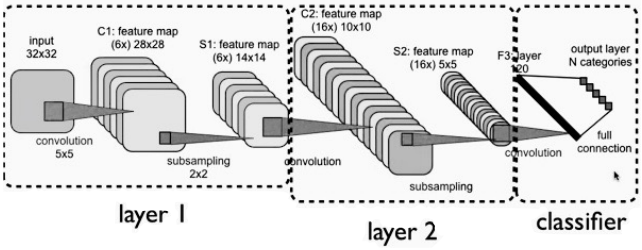

Fig 6 | LeNet Architecture

AlexNet

The AlexNet consists of five layer of alternating convolution and max pooling layers followed by two fully connected layers which converges with a SVM classifier for classification of the images. The input image size is 227 x 227 and so images with low resolution have to be augmented in order to be fed into this net. Because of the higher number of convolutions and higher resolution, we can expect an improved accuracy rating from this network.

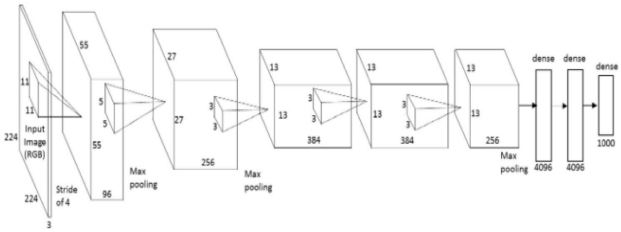

Fig 7 | Alex Net Architecture

GoogLeNet

This net is one of the industry standard for image classification and has 22 layers for the purpose of classification. The basic ideology behind GoogLeNet is to reduce the computation cost and overfitting that occurs throughout other convolutional neural networks. This can be constructed by the use of a deep sparse network rather than a tightly coupled shallow network. And 1×1 convolutions are used to compute reductions before the expensive 3×3 and 5×5 convolutions. Also this net employs random interpolation methods (bilinear, area, nearest neighbor and cubic, with equal probability) for resizing the images and uses a technique called Photometric distortions to combat overfitting.

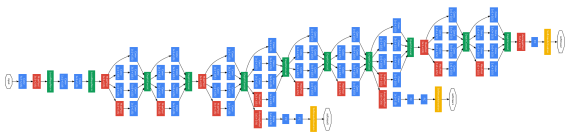

Fig 8 | GoogLeNet Architecture

Results

LeNet CNN - CIFAR-10

With a training dataset of size 50,000 and validation set of size 10,000 these are the accuracies, with data augmentation.

Iteration	Accuracy	Loss
1	0.8204	0.4830
2	0.8283	0.4628
3	0.8402	0.4493
4	0.8567	0.4179
5	0.8626	0.4044

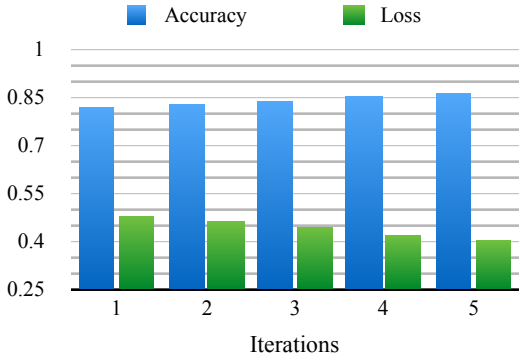

Fig 9 | Table showing the accuracy and loss for each iteration for the CIFAR 10 dataset in the LeNet model

LeNet CNN - 102 Category Flower Dataset

With a training dataset of size 7200 and validation set of size 900 these are the accuracies, with data augmentation.

Iteration	Accuracy	Loss
1	0.7693	0.5454
2	0.7876	0.5239
3	0.7996	0.4932
4	0.8018	0.4621
5	0.8191	0.4430

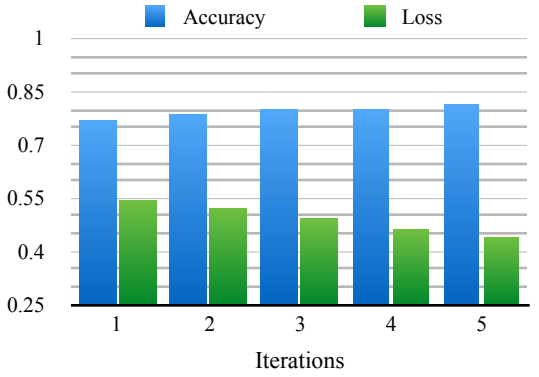

Fig 10 | Table showing the accuracy and loss for each iteration for the 102 Category flower dataset in the LeNet model

Performance of LeNet model comparision

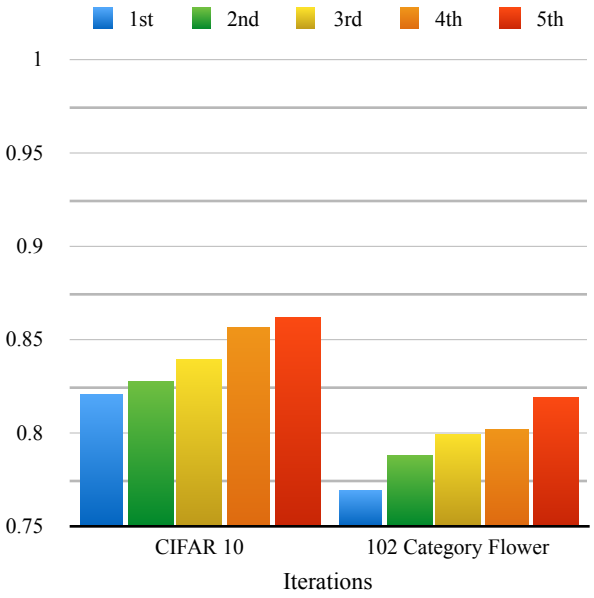

Fig 11 | Comparison between the accuracies for both datasets in the LeNet model

We can see that the model performs better with the CIFAR 10 dataset because of the higher number of samples and better classification in each classes.

AlexNet CNN - CIFAR-10

With a training dataset of size 50,000 and validation set of size 10,000 these are the accuracies, with data augmentation.

Iteration	Accuracy	Loss
1	0.8394	0.5102
2	0.8473	0.4932
3	0.8507	0.4706
4	0.8598	0.4322
5	0.8722	0.4131

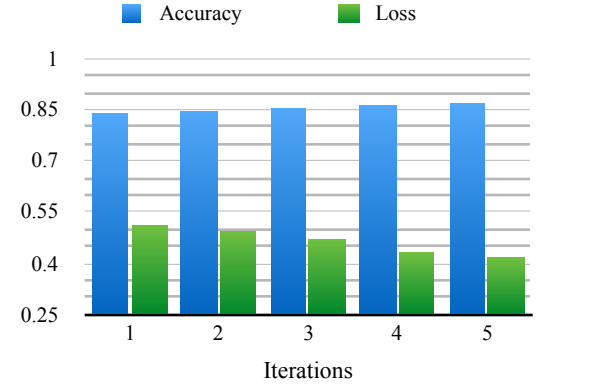

Fig 12 | Table showing the accuracy and loss for each iteration for the CIFAR 10 dataset in the AlexNet model

AlexNet CNN - 102 Category Flower Dataset

With a training dataset of size 7200 and validation set of size 900 these are the accuracies, with data augmentation.

Iteration	Accuracy	Loss
1	0.7560	0.5853
2	0.7756	0.5660
3	0.8058	0.5318
4	0.8360	0.5292
5	0.8467	0.5182

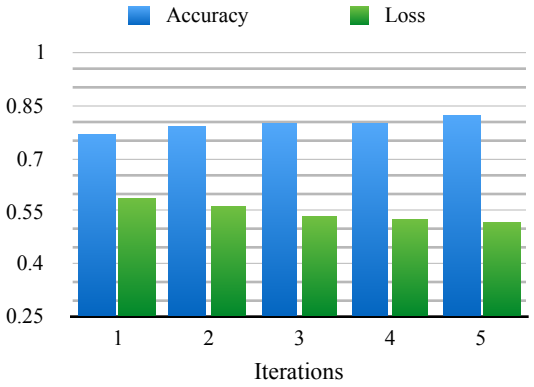

Fig 13 | Table showing the accuracy and loss for each iteration for the 102 Category flower dataset in the AlexNet model

Performance of AlexNet model comparision

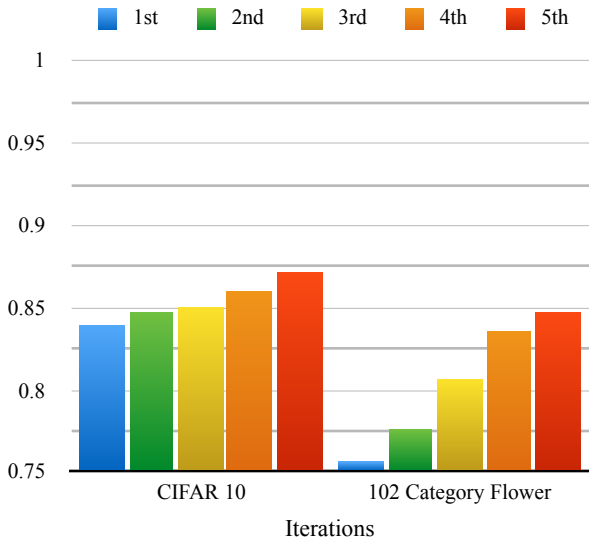

Fig 14 | Comparison between the accuracies for both datasets in the AlexNet model

We can see that the model have an irregular behaviour performance for both datasets. This is because of the fully connected layers which increase the cohesiveness and randomness of the prediction in the model.

Fractional Max-Pooling

As an extension of our previous models, we decided to implement fractional max-pooling instead of the conventional MP2 which usually result in the reduction of the 4x4 matrices to 2x2 with a 2x2 sliding window. But fractional max-pooling actually increases the window size by a factor of 2. Resulting in non-overlapping structures and thereby increasing the randomness of the sliding window. Hence, resulting in a spatial reduction and Non - overlapping pooling regions. Reduces the spatial space by a factor alpha belongs to (1,2). The pooling regions are chosen in a pseudo-random manner. Now let us discuss the accuracies for the models that are constructed with the fractional max-pooling.

LeNet CNN - CIFAR-10 (FMP)

With a training dataset of size 50,000 and validation set of size 10,000 these are the accuracies, with data augmentation.

Iteration	Accuracy	Loss
1	0.8472	0.4639
2	0.8477	0.4483
3	0.8618	0.4214
4	0.8833	0.4093
5	0.8921	0.3988

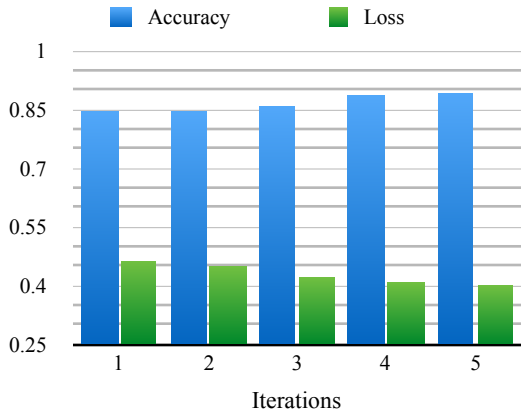

Fig 15 | Table showing the accuracy and loss for each iteration for the CIFAR 10 dataset in the LeNet model with FMP

LeNet CNN - 102 Category Flower (FMP)

With a training dataset of size 7200 and validation set of size 900 these are the accuracies, with data augmentation.

Iteration	Accuracy	Loss
1	0.7992	0.3432
2	0.8193	0.3629
3	0.8429	0.3321
4	0.8796	0.2994
5	0.8863	0.2623

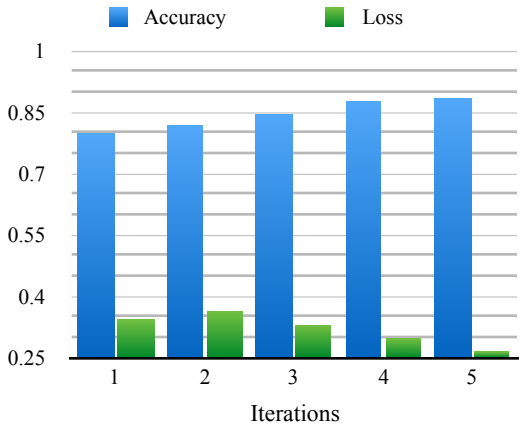

Fig 16 | Table showing the accuracy and loss for each iteration for the 102 Category flower dataset in the LeNet model with FMP

Performance of LeNet model comparision

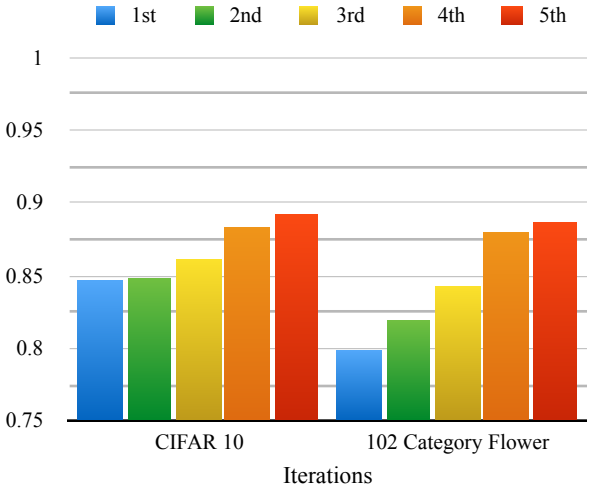

Fig 17 | Comparison between the accuracies for both datasets in the LeNet model with FMP

We can see that the models perform in a very similar manner with both datasets when implementing fractional max pooling. This is due to the randomness and spatial reduction that FMP provides.

AlexNet CNN - CIFAR-10 (FMP)

With a training dataset of size 50,000 and validation set of size 10,000 these are the accuracies, with data augmentation.

Iteration	Accuracy	Loss
1	0.8820	0.3760
2	0.8731	0.3894
3	0.8947	0.3647
4	0.9102	0.3538
5	0.9240	0.3266

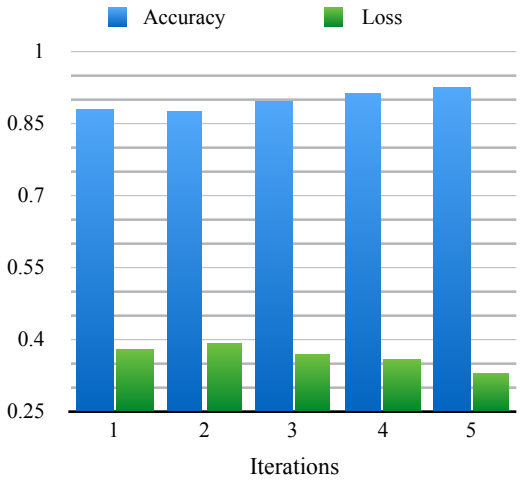

Fig 18 | Table showing the accuracy and loss for each iteration for the CIFAR 10 dataset in the AlexNet model with FMP

AlexNet CNN - 102 Category Flower (FMP)

With a training dataset of size 7200 and validation set of size 900 these are the accuracies, with data augmentation. Here we also perform Fractional max-pooling instead of normal MP2. In this dataset, the images are not uniformly spread out into all the classes. It ranges from 48 images to228 images in each class. This gives us a sense of how real life data actually occurs.

Iteration	Accuracy	Loss
1	0.8204	0.4175
2	0.8532	0.3830
3	0.8705	0.3432
4	0.8832	0.2901
5	0.9074	0.2621

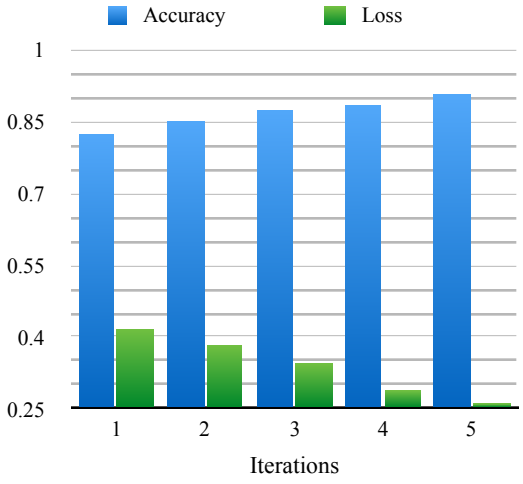

Fig 19 | Table showing the accuracy and loss for each iteration for the 102 Category flower dataset in the AlexNet model with FMP

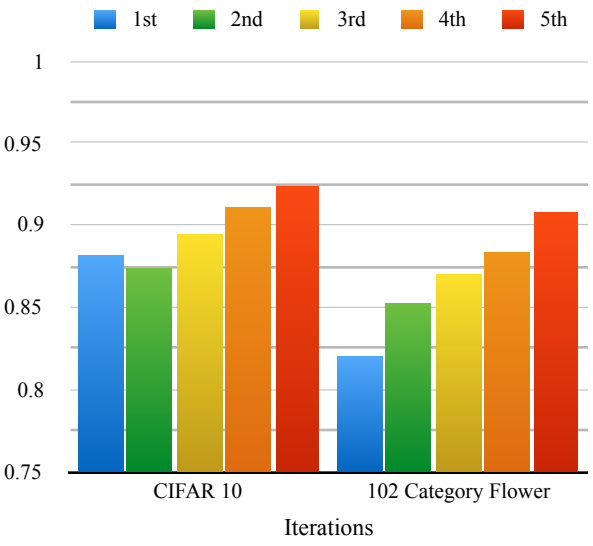

Fig 20 | Comparison between the accuracies for both datasets in the AlexNet model with FMP

As we expected, the model gives better performance with the uniformly distributed CIFAR 10 dataset with Fractional Max-Polling. We are able to achieve an accuracy of about 90% which aligns with the industry standard for image classifiers.

Conclusion

From the experiments conducted we can conclude that deep layered convolutional networks perform better than shallow networks because the additional layers provide the opportunity to look at the image from different perspectives that helps in classifying the images. Although, deep convolutional networks with fully connected layers come with the drawback of increased computational time and overfitting of data. Results from our experiments conclude that sparse networks with deep layers perform better than fully connected networks with shallow layers by reducing the computational complexity which is an important factor when used in real world applications. Max pooling rapidly reduces the spatial space which increases the probability of leaving out important information. Also, there is a restriction being imposed on the number of hidden layers and max pooling layers because of the rapid reduction in spatial space. Replacing max pooling with fractional max pooling overcomes the problem of spatial space reduction and the number of hidden layers since spatial space is reduced by a factor of 1.414. Also, there is a randomness being introduced in selecting the pooling regions that minimizes the probability of leaving out important information. Generating pseudo random numbers for identifying the max pooling regions performs better than random numbers.

One of the possible extensions to our project is developing a google net with more number of hidden layers which we tried to implement in our application. We could not exactly build the model since there were few parameters that needed to be tuned to get the expected level of accuracy. Therefore, we have developed a classifier with an accuracy of 90% that could be incorporated into any real world application. And we were also able to achieve greater accuracies than that were posted by the original authors of the CIFAR 10 dataset. They posted an error rate of about 11-13% whereas we were able to get that down to about 8-9%. We have achieved all the goals that we had previously set to achieve during the start of this project.

References

[1] Deep learning by Yann LeCun, Yoshua Bengio & Geoffrey Hinton in doi:10.1038/nature14539

[2] Krizhevsky, A., Sutskever, I. & Hinton, G. ImageNet classification with deep convolutional neural networks. In Proc. Advances in Neural Information Processing Systems 25 1090–1098 (2012)

[3] Going Deeper with Convolutions by Christian Szegedy, Wei Liu, Yangqing Jia, Pierre Sermanet, Scott Reed, Dragomir Anguelov, Dumitru Erhan, Vincent Vanhoucke, Andrew Rabinovich (2014)

[4] Fast Algorithms for Convolutional Neural Networks by Andrew Lavin, Scott Gray (2015)

[5] Fractional Max-Pooling by Benjamin Graham (2014)

[6] Deep Inside Convolutional Networks: Visualising Image Classification Models and Saliency Maps by Karen Simonyan, Andrea Vedaldi, Andrew Zisserman (2014)

[7] Deep convolutional filter banks for texture recognition and segmentation by Mircea Cimpoi, Subhransu Maji, Andrea Vedaldi (2014)

[8] Deep Convolutional Inverse Graphics Network by Tejas D. Kulkarni, Will Whitney, Pushmeet Kohli, Joshua B. Tenenbaum (2015)

[9] Flower Classification with Few Training Examples via Recalling Visual Patterns from Deep CNN by Kevin Lin, Huei-Fang Yang, Chu-Song Chen (2014)

[10] Spatial Transformer Networks by Max Jaderberg Karen Simonyan Andrew Zisserman Koray Kavukcuoglu (2016)

[11] Some improvements on deep convolutional neural network based image classification A. G. Howard (2013)

[12] Visualizing and Understanding Convolutional Networks by Matthew D. Zeiler, Rob Fergus (2013)

[13] Multi- column deep neural networks for image classification by Ciresan, D. C., Meier, J., and Schmidhuber, J. (2012)

[14] High-performance neural networks for visual object classification by D.C. Cires¸an, U. Meier, J. Masci, L.M. Gambardella, and J. Schmidhuber (2011)