Machine Learning

Assignment 3: Introduction To Deep Learning

Darshan Agarwal (201225189)

Introduction

Deep Learning is an area of Machine Learning research, which has been introduced with the objective of moving Machine Learning closer to one of its original goals: Artificial Intelligence. It is a branch of machine learning based on a set of algorithms that attempt to model high-level abstractions in data by using multiple processing layers with complex structures or otherwise, composed of multiple non-linear transformations. It is part of a broader family of machine learning methods based on learning representations of data. An observation (e.g., an image) can be represented in many ways such as a vector of intensity values per pixel, or in a more abstract way as a set of edges, regions of particular shape, etc.. Some representations make it easier to learn tasks (e.g., face recognition or facial expression recognition) from examples. One of the promises of deep learning is replacing handcrafted features with efficient algorithms for unsupervised or semi-supervised feature learning and hierarchical feature extraction. Deep learning has been constantly causing shock-waves in the field of machine learning, achieving and surpassing the performance of most traditional models. It finds its applications in a wide variety of applications from speech to natural language processing, from robotics to computer vision.

In machine learning, a convolutional neural network (CNN, or ConvNet) is a type of feed-forward artificial neural network where the individual neurons are tiled in such a way that they respond to overlapping regions in the visual field. Convolutional networks were inspired by biological processes and are variations of multilayer perceptrons which are designed to use minimal amounts of preprocessing. They are widely used models for image and video recognition.

STAGE 1 : Discussion on recent success of deep learning

Paper : Mikolov, Tomas, et al. "Recurrent neural network based language model."

I have selected the following research paper : "Recurrent neural network based language model" by Tomas Mikolov, et al. This paper introduces implementation of language model using a recurrent neural network. Results indicate that it is possible to obtain around 50% reduction of perplexity by using mixture of several RNN LMs, compared to a state of the art backoff language model. Speech recognition experiments show around 18% reduction of word error rate on the Wall Street Journal task when comparing models trained on the same amount of data, and around 5% on the much harder NIST RT05 task, even when the backoff model is trained on much more data than the RNN LM. Since Language modelling is a sequential data prediction problem, which requires previous words as a context to predict the next word. So this paper, investigates recurrent neural networks for modelling sequential data. The goal is to build a Language Model using a Recurrent Neural Network. Here's what that means. Let's say we have sentence of m words. A language model allows us to predict the probability of observing the sentence (in a given dataset) as:

$$P(w_1, ..., w_m) = \prod_{i=1}^{m} P(w_i \mid w_1, ..., w_{i-1})$$

It has used an architecture, that is usually called a simple recurrent neural network or Elman network. This network is very easy to implement and train. The network has an input layer x, hidden layer s (also called context layer or state) and output layer y. Input to the network in time t is x (t), output is denoted as y(t), and s(t) is state of the network (hidden layer). Input vector x(t) is formed by concatenating vector w representing current word, and output from neurons in context layer s at time t - 1. Input, hidden and output layers are then computed as follows :

- $x(t) = w(t) + s(t-1)$
- $s_j(t) = f(\Sigma_i x_i(t)\, u_{ji})$
- $y_k(t) = g(\Sigma_j s_j(t)\, v_{kj})$

where *f* is sigmoid activation function and *g* is softmax function.

s(0) is initialized with small value like 0.1. Networks are trained in several epochs, in which all data from training corpus are sequentially presented. Weights are initialized to small

values (random Gaussian noise with zero mean and 0.1 variance). To train the network, we use the standard backpropagation algorithm with stochastic gradient descent. After each epoch, the network is tested on validation data. If log-likelihood of validation data increases, training continues in new epoch. If no significant improvement is observed, learning rate α is halved at start of each new epoch.

The error rate is computed as : ***error(t) = desired(t) − y(t).*** where desired is a vector using 1-of-N coding representing the word that should have been predicted in a particular context and y(t) is the actual output from the network.

The parametric assumptions are : In each layer, weights are initialised from a zero-mean Gaussian distribution with standard deviation 0.01. The neuron biases are set in the second, fourth, and fifth convolutional layers, as well as in the fully-connected hidden layers, with the constant 1. This initialization accelerates the early stages of learning by providing the ReLUs with positive inputs. All other neuron biases are initialised with the constant 0. They used an equal learning rate for all layers, which they adjusted manually throughout training. The heuristic was to divide the learning rate by 10 when the validation error rate stopped improving with the current learning rate. The learning rate was initialized at 0.01 and reduced three times prior to termination. The network is trained for roughly 90 cycles through the training set of 1.2 million images, which took five to six days on two NVIDIA GTX 580 3GB GPUs. The non parametric assumptions are : The kernels on GPU 1 are largely color-agnostic, while the kernels on on GPU 2 are largely color-specific. They did not use any unsupervised pre-training. The two images produce feature activation vectors with a small Euclidean separation, due to which the higher levels of the neural network consider them to be similar.

Recurrent neural networks outperformed significantly state of the art backoff models in all our experiments, most notably even in case when backoff models were trained on much more data than RNN LMs. One of major differences between feedforward neural networks as used by baseline methods and recurrent neural networks is in amount of parameters that need to be tuned or selected ad hoc before training. For RNN LM, only size of hidden (context) layer needs to be selected. For feedforward networks, one needs to tune the size of layer that projects words to low dimensional space, the size of hidden layer and the context-length 2 . The drawback is it needs a large training data. It is possible that learning RNN through Backpropagation through time algorithm or implementing LSTM will provide additional improvements. Since simple recurrent neural networks does not seem to capture truly long context information, as cache models still provide complementary information even to dynamic models trained with BPTT. As the paper did not make any task or language specific assumption in our work, it is easy to use RNN based models almost effortlessly in any kind of application that uses backoff language models, like machine translation or OCR. The proposed approach has no tendency to retrieve images with similar patterns of edges, whether or not they are semantically similar. The second major limitation is the use of Euclidean distance between two 4096-dimensional, real-valued vectors, which is inefficient.

STAGE 2

Dataset

Cifar-10 and Cifar-100 datasets are the smaller versions of 1.2 million high resolution ImageNet dataset. As the names go, they contain lower resolution (32x32) images for 10 classes and 20(coarse)/100(fine) classes respectively. The experiments are performed on the Cifar-10 and the fine version of Cifar-100. The CIFAR-10 dataset consists of 60000 32x32 colour images in 10 classes, with 6000 images per class. There are 50000 training images and 10000 test images.

Experiment 1

In this experiment, a CNN is trained using raw pixels as features with a softmax classifier on top for 100 epochs.

Architecture

The network architecture has three layers of the convolution, pooling, ReLU, normalise nodes, then finally an inner-product layer to pass on the features to the softmax layer for classification.

Experiment 2

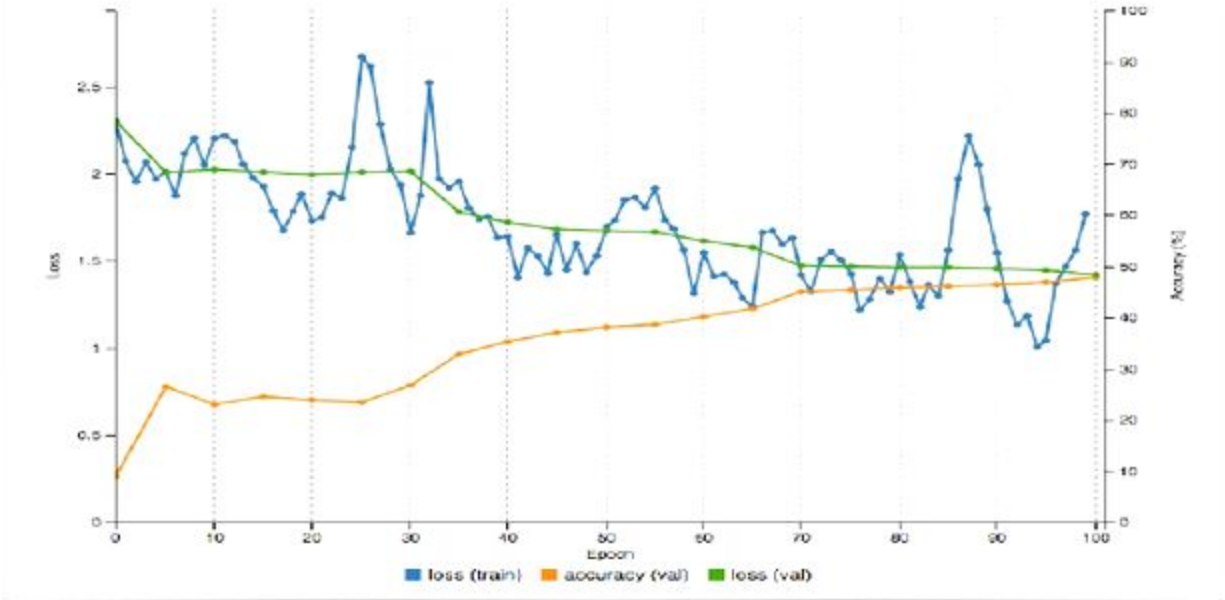
Using the trained model from experiment 1 , chop off the softmax classifier. Extract features from this model, and train a SVM using these features.

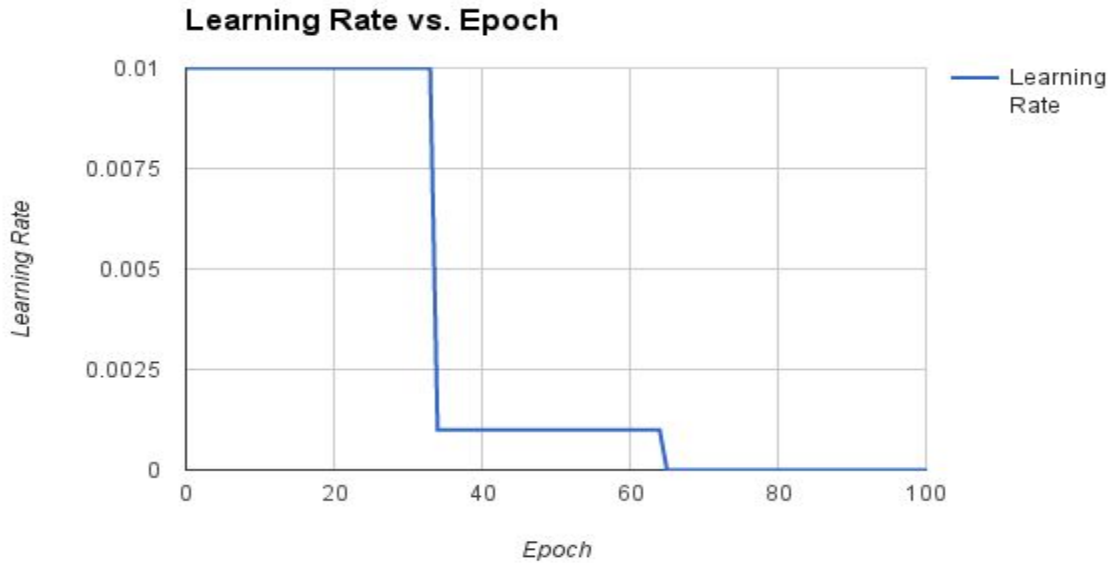
Observations

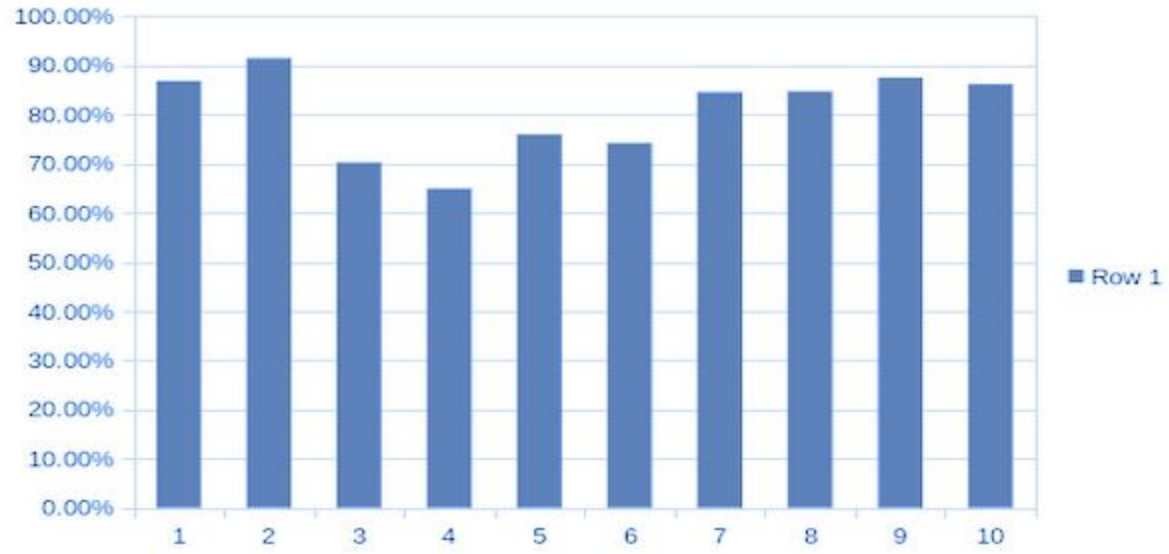
For experiment 1 , our network achieves 74.6% accuracy on the Cifar-10 dataset. The job took a total of 11 hours and 41 minutes to train for 100 epochs.

Plots for first model

overall accuracy : 74.6%





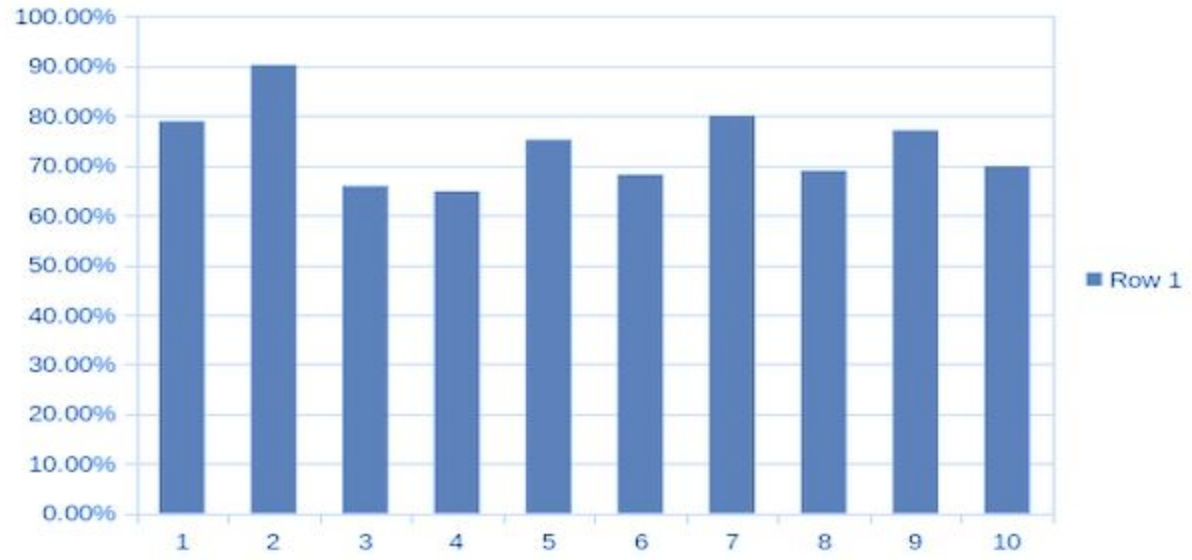


For training the SVM, using the trained model from experiment 1, the parameters used were cache_size=200, class_weight=None, coef0=0.0, degree=3, gamma=0.0, kernel='rbf', max_iter=-1, probability=False, random_state=None, shrinking=True, tol=0.001, verbose=False). The following are the results:

CifarNet-SVM	Precision	Recall	F1-Score	Support
0	0.1	0.2	0.15	1000
1	0.2	0.2	0.2	1000
2	0.05	0.1	0.07	1000
3	0.17	0.3	0.22	1000
4	0.03	0.1	0.07	1000
5	0.12	0.09	0.1	1000
6	0.14	0.1	0.12	1000
7	0.1	0.1	0.1	1000

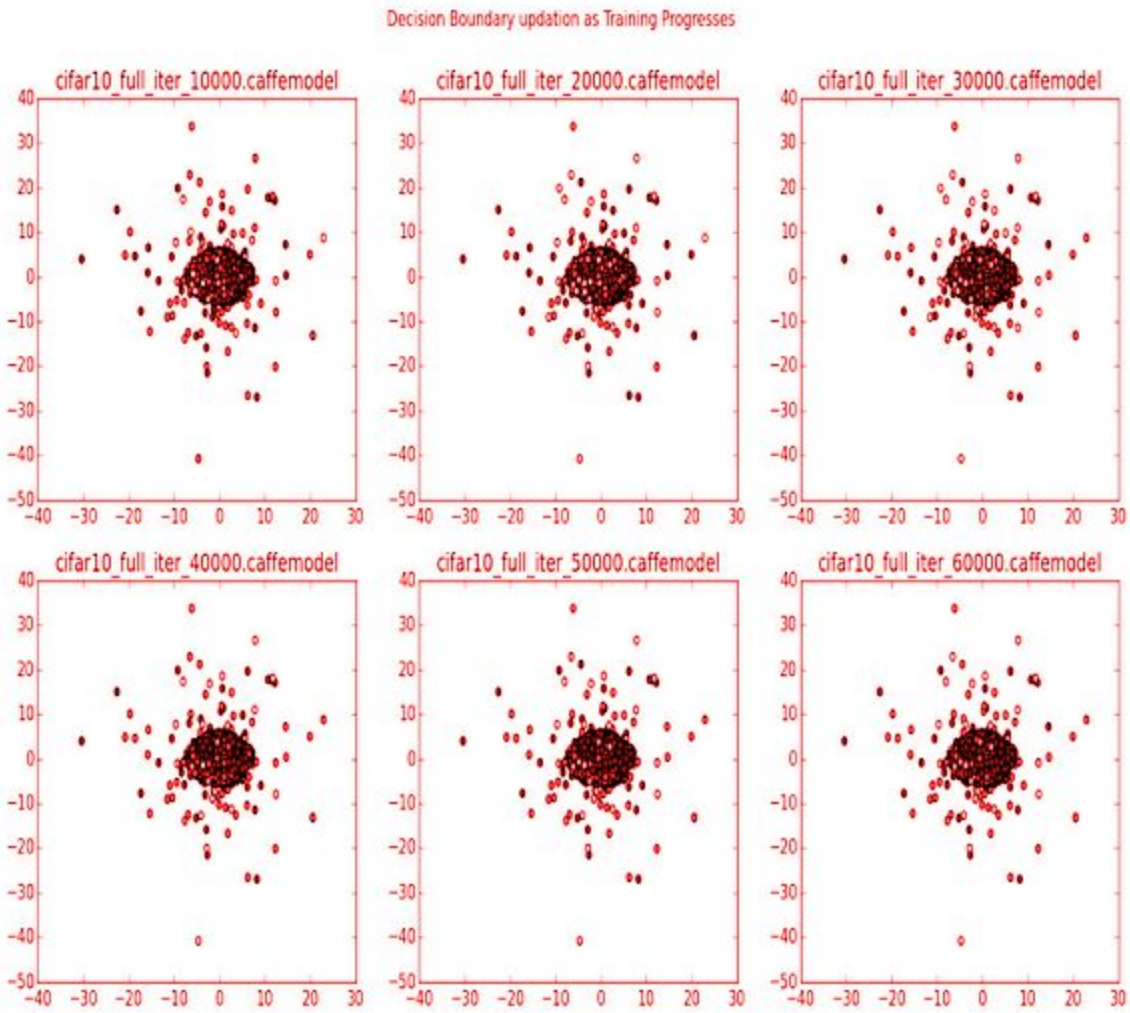
8	0.23	0.21	0.22	1000
9	0.9	1.1	1.0	1000
Average	**0.1**	**0.2**	**0.15**	**1000**

Label wise accuracy



From the results, we observe that SVM is failing to learn the underlying boundaries between these feature points. The performance of SVM is miserable. For not normalized datapoint, the classification accuracy is as low as 7%. However, even after normalising the input features before training and testing, the overall precision turns out to be as low as nearly 10%. The statistics clearly show that using SVM after training model from CNN fails to be of any significance. The reason for this must be the way of computing features is different for the shallow networks and deep networks or the reason may be that Cifar-10/100 are very small and low resolution datasets and due to which the features captured are not really well.

Model training snapshots were taken every 10 epochs (10000 iterations through the dataset images) and tSNE visualisations for the same look as follows.



The tSNE visualisations were computed using PCA to bring the 3-D images to lower dimensional manifolds and then computing the nearest neighbour techniques along with the tSNE algorithm to bring about the 2-D representation as shown above. The visualisation show some progress in the classification bounds of the model but don't really shown distinctive boundaries due to the presence of outliers and not so perfect mapping to a lower dimensional manifold.

STAGE 3 : Parameter tuning using CIFAR-100 dataset

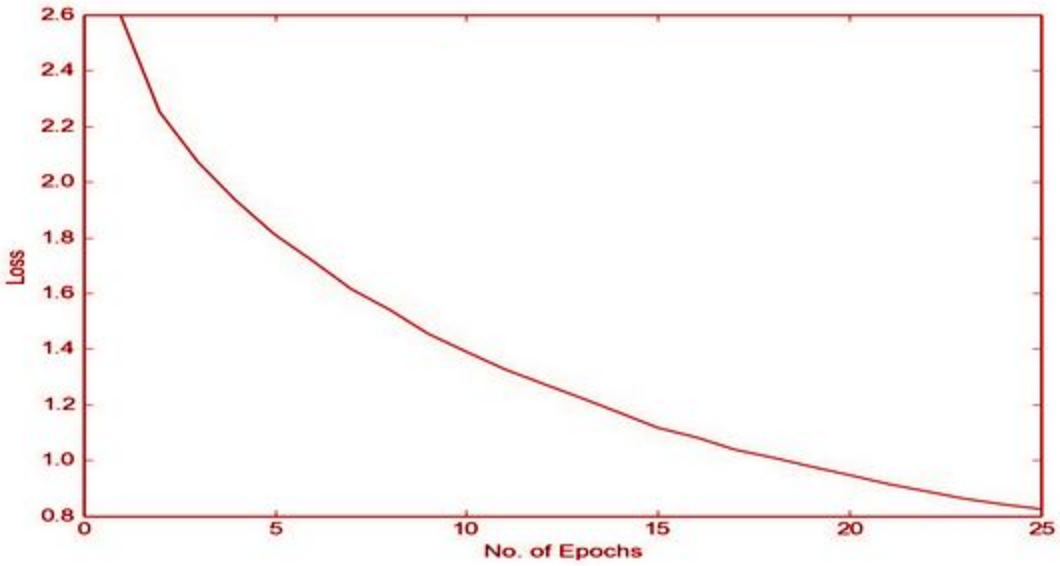
Experiment 3

In this experiment, using the trained model from experiment 1, we chopped off the softmax classifier and the last fully connected layer. Added a new fully connected layer, along with a new softmax classifier, and retrain the model on the new data for about 25 epochs. Repeated this experiment at least thrice with different learning rates for the new layers.

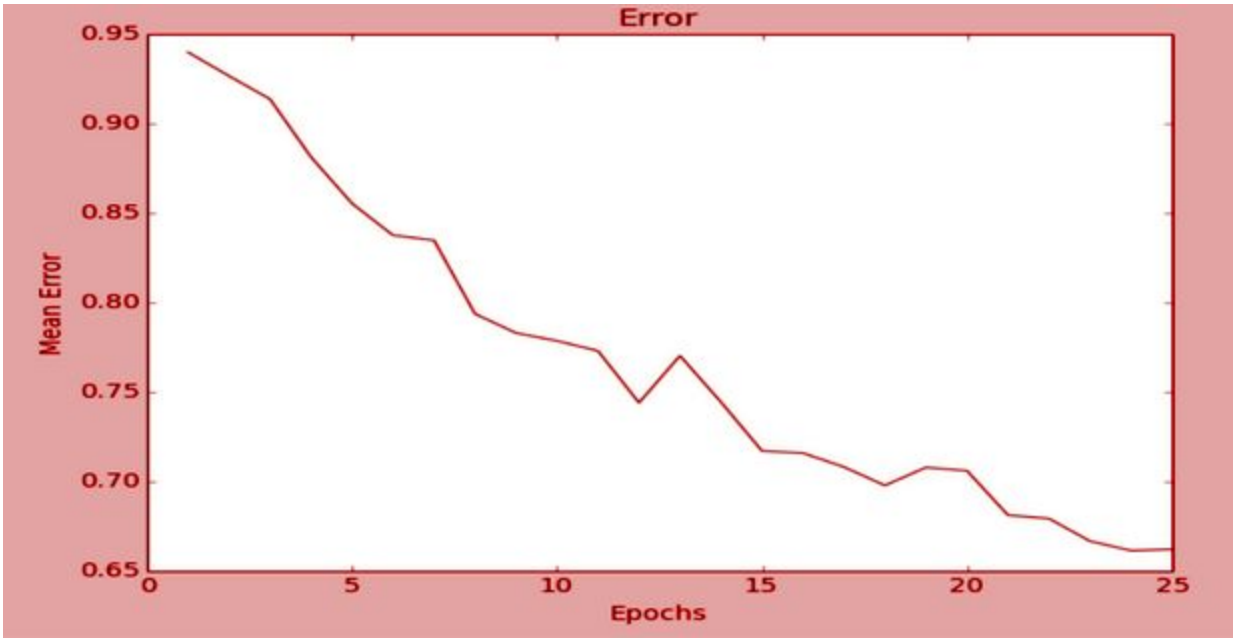
The model took much less time to train (thanks to the optimization techniques used) and has a similar efficiency for classification as that of its predecessor. Such techniques are widely used in machine learning experiments to speed up the process of training and evaluation and as seen from the experiment results, are highly useful.

Accuracy	Learning Rates
59.7%	0.005
59.34	0.001
58.57	0.01
58.30	0.98

Objective :



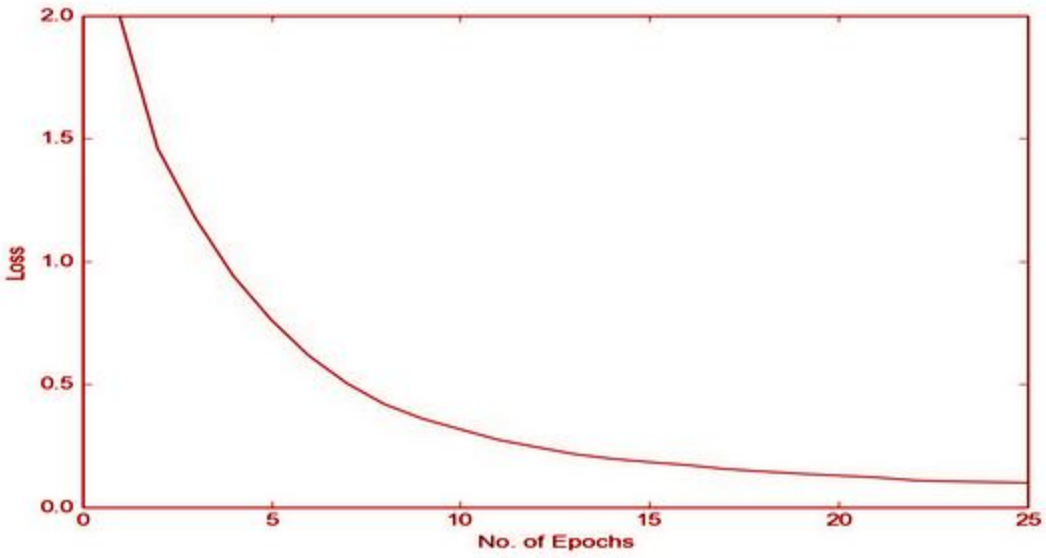
Error:



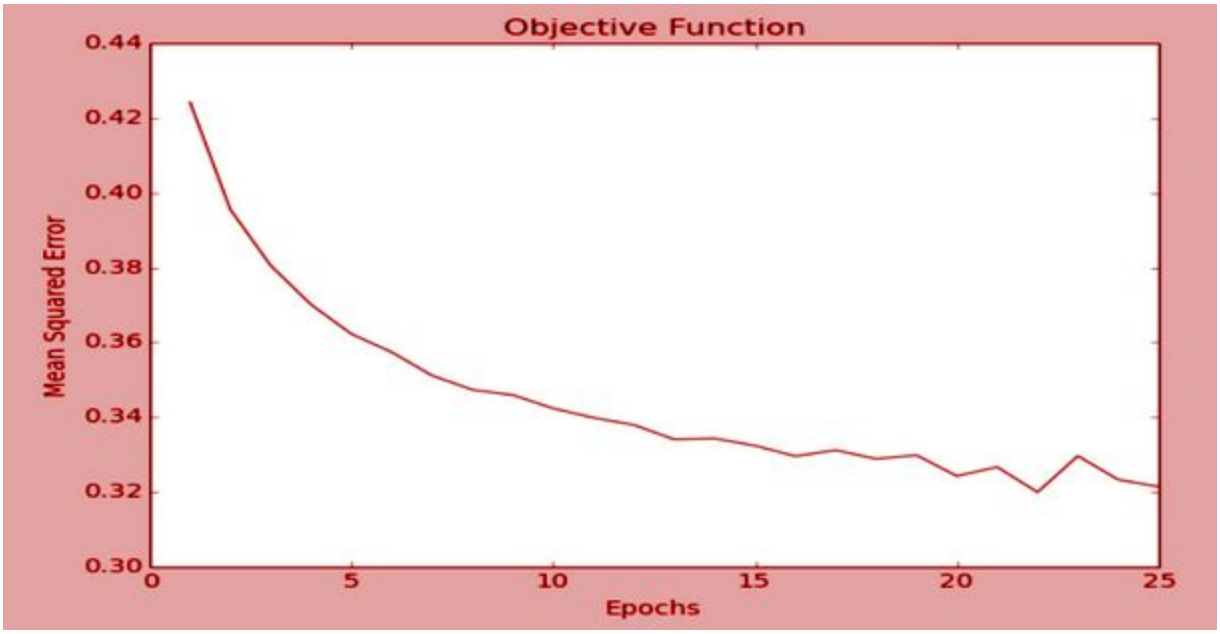
Experiment 4

In this experiment, We need to train a CNN using raw pixels as features with a softmax classifier on top for 25 epochs only. the pretrained convnet from CIFAR-10 experiment is used for feature extraction, and then then the weights in the fully-connected layer and the softmax layer are fine-tuned using the CIFAR-100 data. The whole architecture is trained for 25 epochs.

Overall accuracy: 54.46%



Error:



From the results the observation is that the improvement in performance on training from scratch is not as significant as the improvement in training time between the two methods. Fine-tuning takes just 450 seconds to train as opposed to around 29700 seconds for training from scratch. This method of using pretrained network from one domain to fine-tune the performance for another domain is called Transfer Learning. Transfer learning can be done in one of the two ways: Using the pretrained convnet as a fixed feature extractor, and training the fully connected and softmax layers on top, on the new data. Replacing the fully connected and softmax layers on top of the pretrained network, but fine-tuning the weights of the entire network by continuing backpropagation. For this stage, the first method was used. Since CIFAR-100 is a small dataset, and is similar to CIFAR-10, it is not a good idea to fine-tune the weights of the entire convnet architecture due to overfitting concerns. Transfer learning works between CIFAR-10 and CIFAR-100 because the datasets are the subsets of the same 80 million tiny

images collected by Krizhevsky et al. So the higher level features extracted by the convnet for CIFAR-10 is also relevant for CIFAR-100 dataset.

STAGE 4

Paper : ImageNet Classification with Deep Convolutional Neural Networks

In this paper, a large, deep convolutional neural network is trained to classify the 1.2 million high-resolution images in the ImageNet LSVRC-2010 contest into the 1000 different classes. On the test data, their technique achieved top-1 and top-5 error rates of 37.5% and 17.0% which is considerably better than the previous state-of-the-art. They trained one of the largest convolutional neural networks to date on the subsets of ImageNet used in the ILSVRC-2010 and ILSVRC-2012 competitions and achieved by far the best results ever reported on these datasets. A highly-optimized GPU implementation of 2D convolution is written. Their network contains a number of new and unusual features which improve its performance and reduce its training time. Several effective techniques like Data Augmentation and Dropout are implemented for preventing overfitting. They used RELU to model the neuron's output, since deep convolutional neural networks with ReLUs train several times faster than their equivalents with tanh units.

The network architecture contains eight layers with weights; the first five are convolutional and the remaining three are fully-connected. The output of the last fully-connected layer is fed to a 1000-way softmax which produces a distribution over the 1000 class labels. Our network maximizes the multinomial logistic regression objective, which is equivalent to maximizing the average across training cases of the log-probability of the correct label under the prediction distribution. The kernels of the second, fourth, and fifth convolutional layers are connected only to those kernel maps in the previous layer which reside on the same GPU. The kernels of the third convolutional layer are connected to all kernel maps in the second layer. The neurons in the fully-connected layers are connected to all neurons in the previous layer. Response-normalization layers follow the first and second convolutional layers. Max-pooling layers follow both response-normalization layers as well as the fifth convolutional layer. The ReLU non-linearity is applied to the output of every convolutional and fully-connected layer.

The main drawback of the system is it is highly dependent on computational power and data. It is also seen that their network's performance degrades if a single convolutional layer is removed. Since removing any of the middle layers results in a loss of about 2% for the top-1 performance of the network. So the depth is much important for achieving their results, thus more the depth more computational power. A modification to this approach can be using unsupervised pre-training if we enough computational power is obtained to significantly increase the size of the network without obtaining a corresponding increase in the amount of labeled data.