Parametrization of Convolutional Neural Network for Image Classification

Srinivasan Dasarathi

Submitted as part of the requirements for the degree of MSc in Data Analytics at the School of Computing, National College of Ireland, Dublin, Ireland.

August 2015

Supervisor: Michael Bradford

Abstract

In Artificial Intelligence, convolutional neural network has been the most widely used machine learning methodology of recent times for object recognition. The focus of this research is to identify a combination of key parameters that help improve the accuracy of image classification on this neural network. The network model used in this study comprises of an input layer for normalization and extraction of image data, three hidden layers for convolution, activation and pooling of the feature maps, one fully connected layer for extraction of consolidated image features, followed by an output layer where the image is classified. Sample images of size 32x32 pixels from the Kaggle's CIFAR-10 image dataset belonging to 10 different classes has been used in this experiment.

The neural net is studied across Prototyping, Training, Validation and Testing phases, and the concept of Feed Forward and Backward Propagation has been applied in two stages – first in the hidden and fully connected layers, and later in the output layer – for different objectives as related to error convergence. The effectiveness of various parameterizations has been analysed in both these stages, including weights, bias, momentum, learning rate, regularization strength and iterative epochs. The application of different convolution, activation and pooling functions, key classifiers and novel concepts such as weight decay, weight dropouts and cross-entropy loss has been studied as part of this research project.

Keywords: image classification, convolutional neural network, convolution, activation, pooling, classifier, Softmax, CIFAR-10, error gradient and accuracy.

Acknowledgement

I would like to take this opportunity to thank the following faculties of the School of Computing for their timely support in aid of this research study.

- My project Supervisor **Mr. Michael Bradford**, Lecturer and Director of MSc Data Analytics Programme, for his continuous guidance and valuable inputs on Convolutional Neural Network.
- Lecturer **Dr. Simon Caton**, for his subject matter expert advice and guidance in the application of appropriate analytical and image data mining tools, techniques and methodologies.
- Lecturer **Dr. Adriana Hava Olariu**, for her guidance and valuable inputs on Literature Review and Research in Computing related work including induction to Harvard Referencing.
- Lecturer **Mr. Vikas Sahni**, for his valuable inputs on the overall dissertation as part of the MSc Surgeries workshop.
- **Mr. Keith Brittle**, Information Project Officer for the guidance in the use of online referencing resources at NCI portal, and the excellent knowledge repositories available at the Norma Smurfit Library.

In addition, thanks to all the domain experts who have been referenced in this literature and a special thanks to neural net scientists *Alex Krizhevsky, Geoffrey Hinton, Yann LeCun and Andrej Karpathy* who have made immense contribution to the world of Convolutional Neural Network through their research, publication and tutorials.

Declaration

I wish to hereby declare that the research work documented in this literature and the supportive application development done as part of this project is entirely my own work and contribution based on knowledge gained during the three months of independent study as part of the academic requirements. Authenticated publications of earlier work done on neural networks as well as the use of CIFAR-10 sample dataset for the training and validation of my neural network model as reported in here, has been duly referenced to avoid any scope for plagiarism. All other illustrations and diagrams portrayed in this report are my own creations and has no copyright violation.

Signed .... Date ....

Srinivasan Dasarathi

Table of Contents

1. Introduction

In Machine Learning, a subset of Artificial Intelligence, object recognition and classification of digital images has been an important topic of continuous research in recent times. As pictures speak a thousand words, images are widely used at largescale for scientific and educational purposes, including discovery of hidden knowledge and prediction of future outcome, across various social media portals including Flickr and Google. Image classification is more related to *Image Data Mining* which can be considered as an integration of computer vision, image processing, machine learning and data mining techniques [1], and has been widely in use in the field of biomedical, space research, meteorology and crime prevention including face recognition and handwriting detection. Convolutional Neural Network (CNN) is the state-of-the-art methodology applied in machine learning for image classification.

1.1. Research Question

This research is focused on finding the answer to the question;

"In Convolutional Neural Network, how can image classification accuracy be improved through parameter configuration across functions?"

The hypothesis is that, by tweaking and refining the values set for parameters that are used in different functions of a specified CNN model, the overall accuracy level in the image classification process can be enhanced.

1.2. Problem Definition

When image datasets are voluminous, manual classification of the images would have high time and cost implications. This necessitates the need for automated object recognition with the use of machine learning methodologies. The focus of this research is to analyse the influence of parameter values in improving the accuracy of image classification on CNN.

1.3. Challenges

The challenges encountered in this project are many-fold and of various dimensions;

1. *Field of Research:* Implementation of neural network is by itself a complex field of study in Machine Learning and CNN in specific is much more challenging to understand and implement, and can be considered to be a technical blackbox in many respects. Though there are many literatures and tutorials available for reference related to this, many knowledge gaps are created at each stage of the neural network. Impacting factors include the variation in terminologies used by the authors and the need to understand advanced neural network concepts and mathematical conventions such as calculus derivatives. For instance, though different models using classifiers such as Support Vector Machines (SVM) and Radial Basis Functions (RBF) as well as application of weight dropout mechanism were designed and developed, it could not be successfully implemented as the desired results were not obtained, and there was difficulty in causal analysis of the inaccuracy.
2. *Performance vs Accuracy:* In CNN, the number of hidden layers to be used depends on the choice of convolution function as well as the combination of different functions to

be applied. For instance, if convolution is done including the border pixels (‘same’ mode), the size of the output feature map will be the same as the input, and is likely to have an overall addition of one or two more layers. Likewise if after every convolution and activation a pooling function is applied for downsizing the feature map, then as well the number of layers would increase. This leads to decision-making on prioritization of overall accuracy or performance of the network, as more number of layers with additional functions would enhance accuracy but at the same time impact performance as more number of parameters have to be stored and processed. This trade-off has to be analysed in detail when large datasets are used, where accuracy increase could be marginal at the cost of performance.

3. *Convolution Filters:* Decision on the convolution filters to be used across hidden layers have to be made from different perspectives. The choice of the filter dimension, the number of filters to be used, the rationale for initialization of the filters and the combination of manual and random filters, are arbitrary as various combinations has been experimented earlier and are recommended by neural net experts. To overcome the curse of dimensionality [24] and to minimize performance impact only 16 filters have been used across layers. This size has been found to produce better results than 8 or 32 filters as experimented during the design stage of this project.
4. *Random Parameters:* In CNN, by process, parameters including weights (filters) and bias are initialized at the start of the training process. Though the rationale for initialization is under human control, the task of value assignment based on those rationale, such as random number generation, are machine controlled. As a result during development, detailed one-to-one feature map comparison is not feasible, as the output would vary across process runs; only the overall error gradient and classification accuracy could be measured.
5. *Result Verification:* Another factor is the non-availability of reference data for comparative study at each stage of the CNN process to confirm the mode correctness. Graphical presentation and understanding of the visual representations of the different convolutional layers is a highly complex task. During training, it is difficult to ensure that the graphically visualized feature maps and weights are correct, as there is no similar data for comparison. Likewise, at each stage of development, such as gradient error and classification loss, though it can be inferred from the displayed progress metrics that there is a gradual error and loss convergence, it is difficult to determine if the convergence is good enough, as there is no similar specific output or result from any previous study that can be used for comparison. Hence on most occasions, development progress has to be made on trial-and-error basis with assumptions on the correctness of the algorithm and by referencing several literatures.
6. *Defect Analysis:* At each stage of development, when desired accuracy or output is not achieved, it is difficult to determine if the issue is due to any incorrect coding of the algorithm, or parameter configuration or bug in the development tool used (python). This scenario is mostly faced during training when the model works fine for few images, but fails for an increased volume of few hundreds, and has huge impact on the time factor.
7. *Sample Dataset:* The CIFAR-10 dataset with image size of 32x32 pixels has been mainly used as the sample data in this research. There are several images with poor clarity and inaccurate object focus, which could confuse the network and impact the classification process, especially if these happen to be chosen as the class

representative at the Prototyping stage of CNN. In addition, during the process of the PNG dataset generation, due to huge memory consumption required for processing the 36MB batch files of 10000 images in each, the python program would not exit normally, in spite of generating the output files correctly.

8. *Hardware:* This is another major limitation factor. For this research a mid-range laptop with i7 CPU and 8GB memory has been used. Hence only limited data of few thousand images has been experimented with for the CNN training and testing. A high-end GPU based cloud computing environment would be required to process high volume data without performance degradation.
9. *Software:* As Python 2.7 on Enthought-Canopy GUI has been used for development, there are several open-source libraries such as 'Numpy' and 'Scipy' that are being used in it for various purposes including manipulation of image files, arrays and graphical display of the convolutional layers. Backend version updates to these libraries and tools, at times causes variation in their functional behaviour necessitating relevant code change.
10. *Process Log:* When large datasets such as 10000 images were trained, capturing of the process output on the window or on a file through command shell execution resulted in incomplete data capture. When displayed on window, the initial messages are lost due to flushing of the buffers. When redirected to a file from the command prompt, on completion of the process, the shell command does not exit even long after the process has completed which can be inferred from the CPU and Memory usage. When the process is killed, the log file does not capture the last messages as on the buffer. This necessitates code change to capture and write the process log onto a file.

1.4. Motivation

This project has been undertaken despite the above challenges, in consideration of the fact that CNN is a novel area in the field of neural network and has a larger scope for practical application including image detection in video stream use as highlighted later in this document as part of future work. The classification of images in CNN is more accurate when the training datasets are larger [2].

In addition, the most awaited research breakthrough in computer vision accuracy is to resolve the problem of detecting multiple objects in an image and automatic image annotation with correct and meaningful lexical semantics. There is a good potential to participate in global competitions such as, the 'ImageNet Large Scale Visual Recognition Challenge' (ILSVRC) and 'PASCAL Visual Object Classes' (PASCAL VOC) which promote research on image classification. These have been the key motivation factors for considering this subject as the research topic.

This paper is organized as follows in the forthcoming sections. The next section on 'Background' details the theoretical foundations that are required to be understood as a pre-requisite before reading further. The section after this is on 'Related Work' which evaluates related literatures as published by industry experts. This is followed by 'Methodology and Design' which how the research question was approached. Next is the 'Implementation' part which details the process flow as to how the practical aspects of this study was carried out. This is followed by 'Evaluation'. The final section is on 'Conclusion and Future Work'. This literature ends with a list of 'References' used for study and as referenced in the earlier sections.

2. Background

The application of CNN also known as *Convnets,* a branch of Artificial Neural Network is the emerging trend in the development of computing models for automated image classification. CNN is very much similar to a linear neural network. The main difference is that it performs an activity called *convolution,* which has been described in detail in the later part of this report*.* The following sub-sections give an overview of the various key terminologies and concepts as highlighted in italics, and methodologies that are used in CNN for supervised machine learning. More details on Machine Learning and Neural Network concepts and methodologies have been referenced in [28],[29],[30] and [32].

2.1 CNN Development Phase

The task of image classification is executed in four different phases on Convolutional Neural Network namely – Prototyping, Training, Validation and Testing. In this study the task of determining the class representative data has been considered as a separate Prototyping stage, and not included as part of training contrary to convention.

2.1.1 Prototyping

Prototyping is normally done as part of the training phase. But for activity differentiation, it has been considered as a distinct phase in this project. It involves defining the expected (desired) results across different classes of data. For each class of data, a sample image from the training dataset is selected and used as the base for comparison with other images of the same class in the subsequent phases. The class (*label)* to which the image belongs can either be provided in a compressed array file format or in a viewable format such as JPG or PNG with the label prefixed to the file name. For instance '0-ford.jpg' denotes the image has label '0' which is the 'automobile' class.

Rather than selecting an image at random for each class, to obtain faster classification during training phase, best practice would be to choose an image from the training dataset that could be considered as the class representative. One of the ways to achieve this, as applied in this research is the use of Mean Square Deviation (MSD) method which computes the Euclidean Distance between images to determine the one that has the least feature difference across images which would act as the class centroid representing the desired result of the class. The final output would be a vector of expected (desired) features of the class to which the image belongs.

2.1.2 Training

For Training the neural net, a new dataset across classes is used. For each input image the class it belongs to is notified to the neural net. Training involves two main process stages – Feed Forward and Back Propagation, which are described later in this document. The output would denote the predicted features of the image that is being trained. The predicted features for a class would be compared with expected features determined earlier in the Prototyping phase. The parameters (weights and bias) get trained till the difference between the predicted and desired gets least significant. At the end of training all the feature maps as generated across layers and the final predicted features can be disregarded, and only the trained weights and bias be retained for use in the subsequent phases of validation and testing.

2.1.3 Validation

Validation is the stage where the accuracy of the trained network parameters are verified and pre-tested using a subset of the test data. In this stage as well, the class value of each image can be made use of to compare and validate the desired results. This phase was introduced in the recent years as best practice of development to strengthen the training reliability. This is to ensure there is no 'overfitting', which refers to the scenario of the network model working well on the training data, but failing on the test data. All the trained parameters and hyper-parameters as applied during the end of training would be reused in this phase.

Programmatically, the two key differences in the processing logic for validation compared to the training phase is that, there is no back-propagation performed and the network is not pre-notified of the class to which the input image belongs. If the desired classification accuracy is not achieved, then the Training phase is re-executed.

2.1.4 Testing

This is the final stage of development where the trained weights and bias are validated across classes using untrained image repository. The same code as used for validation will be applied on a different test dataset.

2.2 CNN Processes

In artificial neural network including CNN, training phase comprises of two key stages, which are Feed Forward and Back Propagation [16].

2.2.1 Feed Forward

Feed Forward includes all tasks done from the input layer to the output classifier layer, starting from image normalization to loss computation. All phases of CNN excluding Training phase, terminate with the completion of this process.

2.2.2 Back Propagation

In the Training phase, at the end of Feed Forward, once the loss is computed and determined to be higher than the predefined loss threshold, the weights and bias across all layers have to be refined to minimize the loss. As a result, the process of *back propagation* is initiated. This combination of feed forward and back propagation is executed several times, to adjust the parameters and to re-compute the feature maps across layers till the loss converges to within the threshold limit and the desired classification is obtained.

In order to revise the parameters of weights and bias, CNN makes use of several additional parameters which are termed as the *hyper-parameters*. This mainly includes *learning rate* and *momentum* that are used to expedite the error convergence and would have a value in the range of (0,1). The dataset would be trained for multiple epochs, until the gradient error is within the predefined threshold. Each *epoch* refers to one round of training of the entire dataset. Whereas for verification of data loss and regularization loss, each image undergoes multiple loops of *iterations*. The maximum number of epochs and iterations are configured through parameters. A large learning rate can result in over-fitting or over-training [10]. The first stage of back propagation is to determine the error gradient across layers in the reverse order, starting from the fully connected layer to the first hidden layer. For the fully connected layer *error gradient* is computed as the product of the difference between the expected (desired) output and the predicted (actual) output, and the derivative of the activation function used in this layer. Then the weighted sum of the gradient is computed using the product of the weights and the previously computed gradient error of the fully connected layer. The weights in the fully connected layer are later adjusted accordingly using the product of the derivative and the error.

This is followed by back propagation for all the hidden layers starting from the last to the first. As done earlier, based on the activation function used, the corresponding derivative is applied to the weighted sum of the gradient error in the previous layer. These values are used along with the earlier computed gradients to determine the delta values for the weight and bias update in each hidden layer. Care has been taken to ensure that the manually generated kernels as used for edge detection are not updated while processing the weights of the first hidden layer.

2.3 CNN Layers

Convolutional neural network comprises of three layers, namely the Input, Hidden and the Output. Unlike Single Layer Perceptron (SLP), CNNs are similar to Multilayer Perceptron (MLP) and have more than one set of network connectivity with the layer.

2.3.1 Input Layer

In the *input layer* raw images are first pre-processed through *normalization* which involves data reduction through mean subtraction to reduce the scale of pixel values into a vector. This is done for eliminating computational errors due to large values especially during exponential and logarithmic calculations. In addition, conversion of colour images to grayscale images are done. Colour images have three channels of 'RGB' and comprise of three dimensional (3-D) array data which are complex to process. Hence they are simplified to grayscale images that have only one channel of two dimensional (2-D) array of pixels. Grayscale images have the advantage of not having data discrepancies due to variation in colour contrast. The value of each pixel in a grayscale image ranges from 0 (white) to 255 (black) with a gradation for the in-between shades of grey.

2.3.2 Hidden Layer

The *hidden layer* is the heart of CNN, and comprise of neurons which perform the key convolution, activation and pooling functions. For the first hidden layer the normalized image data would be the input. For the subsequent layers, feature maps generated in the previous hidden layer would become the input. The number of hidden layers required in a network model is determined based on the size of the image and filter.

Weights and bias factors, which are described more in detail in later sections, are the main configurable parameters that are to be tuned in the hidden layer to obtain the desired image classification output. In each layer, the number of output feature maps would be equal to the number of filters used in the convolution process. The feature of weights being shared across feature maps is what makes CNN perform better on vision problems [35].

2.3.3 Fully Connected Layer:

Following the last hidden layer is the *fully connected layer* where the single-node output feature maps of the previous layer are consolidated into a single vector. The data is then processed with the weights and bias of this layer, and the resulting output is sent to the final output layer. There is no need to perform pooling operation as the features are already in a single vector and there is no scope for further dimension reduction.

During training, the accuracy of the neural net is measured at two stages – in this layer and in the subsequent output layer. Here the difference between the class features and predicted feature are verified to be within a predefined error threshold limit which would normally be close to 0. The error is referred to as the *loss or cost* of the training. For instance an error threshold of 0.1% signifies an expected training accuracy of 99.99%. The overall loss comprises of *Data Loss,* that quantifies the difference in the predicted score and actual score of all examples, and the *Regularization Penalty* which refers to the correction of large weights. If the total loss is above the defined loss threshold, back-propagation process is initiated to train the weights and bias across layers.

2.3.4 Output Layer

The *output layer* performs the classification of the input image based on pre-defined classes. Some of the popular and widely used classifiers include Softmax, Multiclass Support Vector Machines (SVM), and Radial Basis Function Network (RBFN) which is claimed to produce high level of classification accuracy and ease of computation as compared to its former counterparts [4].

Score, Loss and Thresholds:

Second level of accuracy verification is done at the output layer to confirm the correct classification by computing the *Score* to determine the class of the image data, which is done using the product of the predicted features and the weights and bias of each class. The image is labelled after the class with the maximum score.

The classifiers make use of different hyper-parameters which are used to control the speed of convergence. For instance the Softmax Classifier applies the *Learning Rate* (*Step size) and L2 Regularization Strength* parameters for the said purpose as detailed in [26]. In Softmax, *Data Loss* which quantifies the dissatisfaction with the correctness of score and class prediction in a dataset, is to be determined for the entire training dataset [26]. The loss function initiates backpropagation in the output layer to update the weights and bias of the classifier.

2.4 CNN Functions

2.4.1 Convolution and Filters

Convolution is the first process of a hidden layer. In simple terms *convolution* involves scanning a data filter over the image data. For ease of understanding, it can be visualized as a magnifying lens that is used to view the pixels of the image, a small portion at a time. The filter also referenced as the *kernel* or *weights* are a 2-D array of data, usually of the size 3x3, 5x5 or 7x7 pixels. Filters are scanned over the image from the top left corner to the bottom right at a specified interval. The centre pixel of the filter will be made to overlap with the centre of the *Receptive Field* on the image. The filter will shift horizontally and vertically at a defined span size, and in each position the product of the image pixels and filter pixels will be computed. Swapping and transformation of image pixels is also done are part of convolution. The size of the Receptive field would be similar to the filter size. The concept of *stride* also needs to be understood which refers to the number of pixels the filter has to shift horizontally and vertically over the receptive field during convolution.

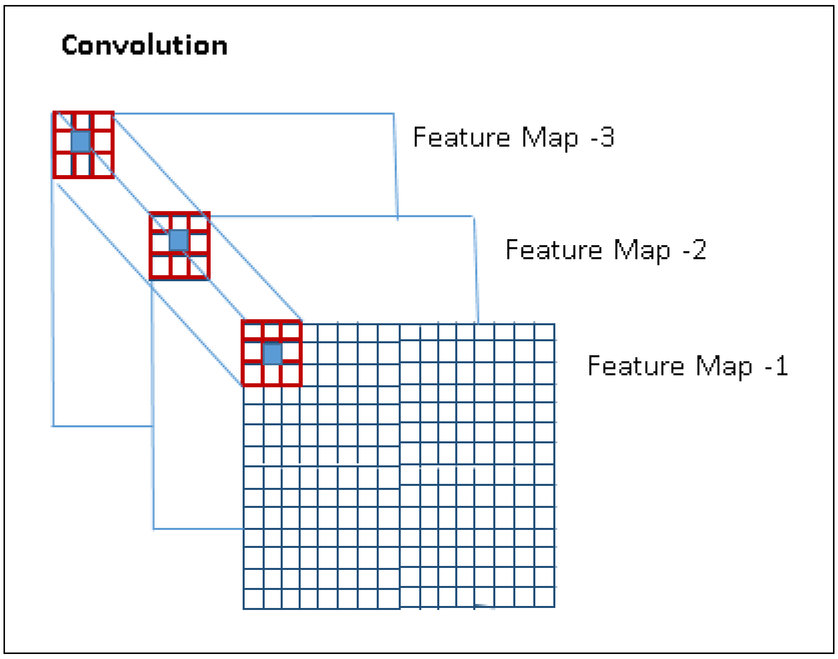


Figure-1: Convolution on three feature maps. The centre of the receptive field is highlighted in blue. In every convolution the filter is applied across all feature maps at the same depth location, as highlighted in red. The sum of convolution at each receptive field across all feature maps becomes the node value at (0.0) on the output feature map.

During convolution, as shown in figure-1, the filter would be applied to the depth at each receptive field location, across all input channels of that layer. Each convolution produces a single pixel value or an output node to which a *bias* parameter of value 0 or 1 is added. The weights and bias would be used in linear regression equations for computing the output based on the input across each layer. For ease of computation instead of having a separate bias, an additional weight (W-0) is added to the weight parameter and a dummy input with value 1 is included as well to facilitate matrix product computation.

A set of nodes produced as part of a convolution is termed as the *feature map*. It has to be ensured that the size of the input feature map of the last hidden layer does not fall below the size of the filter. The filters are usually odd-sized for the benefit of having a defined centre which would be used as the focal point during convolution. The size of the output feature map depends on the type of convolution applied. If it is a 'valid' mode, then additional extended pixels will not be used while convolving the boundary pixels with the filter centre, and hence the output feature map size will not be the same unlike as obtained in the 'same' mode of convolution method.

Likewise, the size and number of filters used in each hidden layer can be varied depending on the feature extraction needs at each layer. It should be noted that as the number and size of filters, and number of layers increase, there would be a proportionate increase in the total number of parameters to be computed and stored. This could result in the need for a high end computing environment with increased memory and processing capabilities to avoid performance degradation. In the first layer of convolution manual filters can be applied in combination with randomly generated filters. Manual filters would involve programmatic hardcoding of array values based on filter dimension to be used for purposes such as horizontal and vertical edge detection. Random filters would be a set of random numbers usually in the range of (-1,+1).

2.4.2 Activation Function

The output obtained from convolution is passed through activation function for a linear and non-linear transformation, such as;

- Logistic Sigmoid function (f(x) = 1/(1 + exp(- x))
- Softmax function (f(x) =max(0,x))
- Hyperbolic Tangent Sigmoid function (f(x) =tanh(x)) and
- Rectified Linear Unit (ReLU) function which is similar to Softmax.

Different activation functions can be applied across layers, if need be. These linear functions produce an output in the range of either (0,1) or (-1,+1). Each of these functions have a derivative as listed below, which is applied during back propagation to correct the gradient error.

- Logistic Sigmoid function (f(y) = f(x) * (1.0 – f(x), where f(x) is the related logistic sigmoid activation function)
- Softmax function (f(y) = (1 - x) * x)
- Hyperbolic Tangent Sigmoid function (f(y) =(1 - x) * (1+ x)) and
- Rectified Linear Unit (ReLU) function (f(y) =1, if x>0, and f(y)=0, if otherwise).

2.4.3 Pooling Function

The feature map thus derived after convolution, bias addition and application of activation function, is then processed through a *pooling* function. The most frequently applied pooling operations are the max-pooling and mean-pooling. In this research, max-pooling approach has been applied. The pooling function helps to down-sample and downsize the input feature map to half its size.

3. Related Work

1. Parameters across layers have to be properly trained. Significant performance enhancement can be achieved with proper weight initialization and use of simple heuristics [2]. To achieve higher level of classification accuracy it is proposed to linearly combine multiple neural network classifiers [18].

2. In some of the literatures it is recommended that a very small learning rate such as 0.0005 be used [11]. But in this model, the gradient error fails to converge if such a low learning rate is given. It has been found that 0.45 is the best Hence such discrepancies could arise due to variation in the combination of activation functions, kernel size and setting of hyper-parameters, which needs to be further explored.

3. Padding the filter during convolution has been proved to be ineffective [11]. Hence the alternate option would be to use the 'same' mode, but this would increase the computational time as more receptive fields need convolution, and the output feature map would be much higher, leading to the need to have additional hidden layers for the feature map to get smaller with nodes of size 1x1 or 2x2.

4. Another factor that needs to be experimented is the better stride to be used for convolution. During a convolution with the use of a 5x5 filter, if the stride is too small say 2, then for each convolution shift there would be an overlap of 2 pixels, resulting in higher number of feature map nodes, but at the same time has the advantage of more feature recognition. Hence a trade-off is required to determine the size of the stride. A novel concept of tiled CNN has be introduced [13] which claims to reduce the number of parameters. It has been recommended [14] that the use of smaller window size and stride in the first convolution layer improves performance.

5. Many of the researches have used the MINST handwritten digit dataset which are in grayscale. Also the general understanding is that processing grayscale images reduces the first layer complexity. Whereas in [18] it is mentioned that use of grayscale images increases error rate, and hence it is recommended to use the colour images.

6. It has been experimented that the trained layers and parameters can be reused for alternate datasets instead of creating the new network all from scratch [21]. An alternate and a more effective approach to initialization of weights, is to select random images from the dataset, and then to extract random patches at random positions on the images [20].

7. In many literatures *subsampling* is termed synonymous to max-pooling. But in [22] it has been considered as a distinct process as it has been stated that max pooling produces a much lesser error rate compared to subsampling. Also in [22] even dimension filters of size 6x6 has been used, as opposed to the convention of odd-sized filters.

8. SVM is stated to have limitations in comparison with other classifiers [7]. Whereas [27] has researched and highlighted that Deep Learning SVM offer better results in comparison to Softmax.

4. Methodology

For image classification and feature detection, CNN model has been applied with the Feed Forward and Back Propagation mechanism during the training phase. Two different activation functions have been applied in this model – (a) Hyperbolic Tangent Sigmoid function in all the hidden layers, as it has been verified to produce higher accuracy by many researchers; and (b) Softmax function in the fully connected layer as the Softmax Classifier is used in the output layer. Convolution is done in 'valid' mode which results in a slight dimension reduction after convolution, as the border pixels of the input image or feature map get ignored.

4.1 Design

The CNN designed for this project as shown in Figure-2 has a total of eight layers comprising one initial input layer (L-0), five hidden layers (L-1 to L-5) , followed by one fully connected layer (L-6) and then terminating with the output classifier layer (L-7). The number of hidden layers is not extended beyond five, as the output feature map of the last hidden layer converges to one node per map, and there is no further requirement to convolve or max-pool it. In this model, three hidden layers (L-2. L-3 and L-4). In the proposed model, pooling is done only after the first two convolution and activations, and not applicable to the last hidden layer as the output feature map is already of 1x1 dimension which is below the filter size of 5x5 pixels.

The network architecture with reduction in dimensionality of the feature maps across layers is designed very much similar to [10] and is as follows;

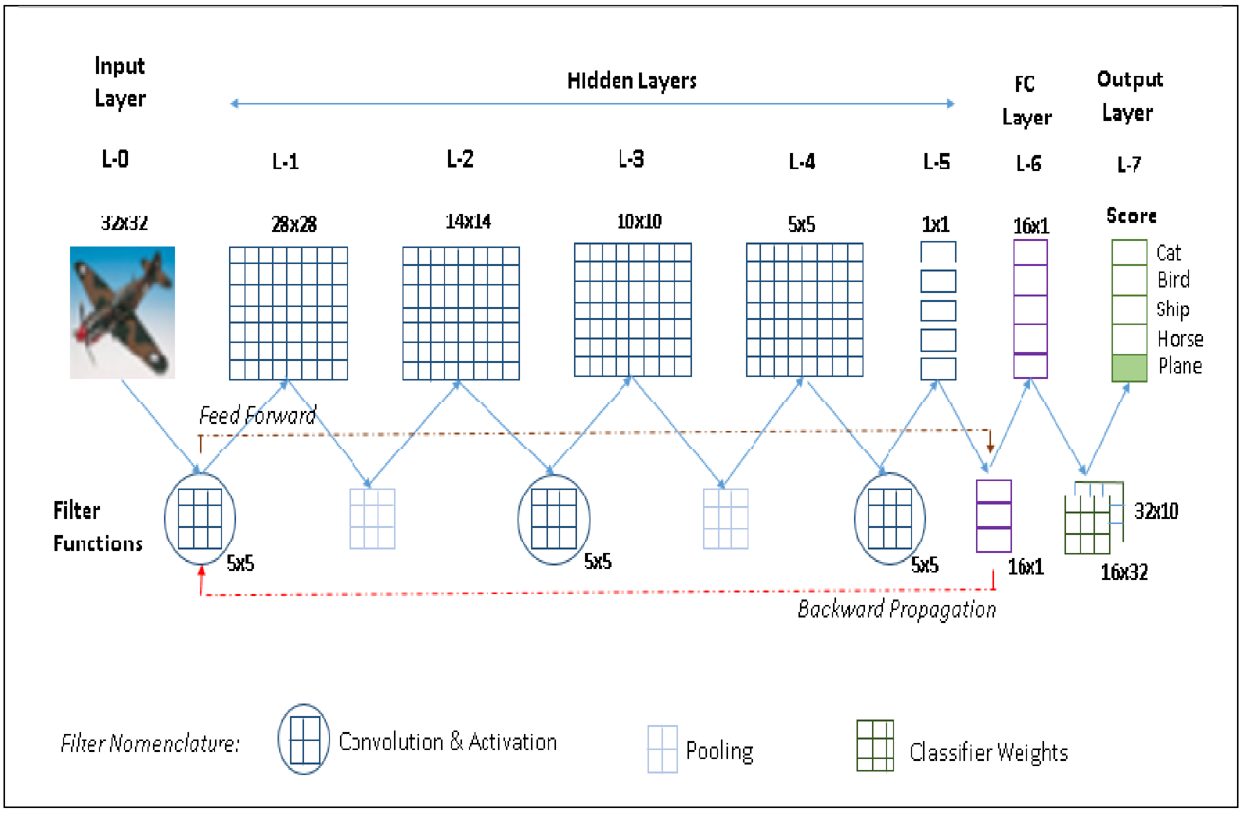


Figure-2: CNN architecture and process flow.

The details of various layers are as below;

- *L-0 Input Layer*: the raw input colour image is converted to grayscale, then normalization and produced as an output of same 32x32 single channel 2-D array of data.
- *L-1 Hidden Layer*: the 32x32 normalized input data is convolved on a 'valid' mode with 16 5x5 size filters, to obtain 16 feature maps of dimension 28x28 pixels.
- *L-2 Hidden Layer*: This undergoes activation and then max-pooled to generate 16 feature maps of 14x14 pixel dimension.
- *L-3 Hidden Layer*: the 16 input feature maps undergo similar process as in L-2. After convolution and activation the map size is reduced to 10x10.
- *L-4 Hidden Layer*: After max-pooling the inputs gets downsized to 5x5 pixels.
- *L-5 Hidden Layer*: this is the last hidden layer where the inputs after convolution and activation create 16 feature maps of size 1x1, with each feature map containing only one node.
- *L-6 Fully Connected Layer*: Here the 16 single-node input feature maps are consolidated into a single vector output. Instead of convolution, linear regression is performed with the use of weights and bias, followed by activation function execution.
- *L-7 Output Layer*: This is the final layer where classification is done using Softmax classifier which gives probability of scores and has been widely used in many research developments including [15]. For classification two sets of weight matrices are applied.

4.2 Processing

In the Prototyping phase, the weights and bias parameters across layers are initialized with normalization to minimize the weights. A set of seven manual filters for edge detection including horizontal lines, vertical lines (Prewitt edge detector) and diagonal lines (Sobel edge detector) are used in combination with weights that are randomly generated using the size of the filter as the base. The weights are initialized in line with normal Gaussian distribution to be in the range of -1.0 to +1.0, and the bias parameters of the output nodes are initialized to 0.001, rather than 0 as recommended in [26] to avoid zero saturation in the computed regression value.

For the initial Training, the weights and bias saved during Prototypying are uploaded into relevant arrays, and the Feed Forward and Backward Propagation process are cyclically executed for the hidden layers based on the error gradient of the predicted features. Subsequently the Softmax Classifier computes the score and cross-entropy loss before classifying the image. In case batch processing of the training dataset is done to overcome the hardware limitation, the trainable parameters that had been saved in the earlier training can be uploaded through a configuration change and used instead of applying the initial set of parameters as initialized during prototyping. On completion of training, the parameter data across layers are later saved for use in the validation and testing phases.

5. Implementation

The CNN model for image processing as proposed in this research, comprises of eight different neural net layers as in Figure-2 which are interconnected starting with an input layer, followed by hidden layers, a fully connected layer, and terminating with the final output layer.

5.1 Dataset

For this research a subset of about 1200 images from the Kaggle's CIFAR-10 dataset [3] as available on https://www.kaggle.com/c/cifar-10/data has been considered, as it comprises large volume (60,000 images) of smaller uniform dimensional images of 32x32 pixels, which are easy to compute on mid-range laptops. Moreover, this dataset has 10 different classes (airplane, automobile, bird, cat, deer, dog, frog, horse, ship and truck). The class labels are specified as part of the data. Also benchmark results of various testing done earlier by neural network experts using this dataset over a period is available for comparison.

The CIFAR-10 dataset as downloaded from the internet are in a compressed file format, and hence have to be converted to 'PNG' files using a separate extraction program developed by me. Though the compressed data can also be directly used in the neural net, the images are first extracted and then used for classification. This has the benefit of possibility to view the quality of the images before processing them, and the option to randomly choose and manipulate the batch contents of image files that are to be used across phases, by varying the number and type of images in a batch. The extraction code appends the class label to each image file, which is used as the indicator in the Prototyping and Training phases to determine the correct class of the image.

	CNN Stage	**CIFAR-10 Batch**	**Images Used**
1.	Prototyping and Training	Batch-2	100
2	-same-	Batch-2	500
3	-same-	Batch-1	1000
4	-same-	Batch-3	10000
5	Validation	Batch-4	100
6	Testing	Batch-4	200

Table-1: CIFAR-10 batch datasets used across CNN phases.

For the different CNN phases, sample datasets of different CIFAR-10 batches were made use of as in table-1. For Prototyping and Training, the same datasets are used, so as to extract the mean features across classes.

5.2 Technology

Python 2.7 on the User Interface tool Enthought's Canopy has been used for developing this CNN model. Additional python libraries such as numpy, scipy, matplotlib, cv2, math, and glob had been installed as part of the code. The data was tested on a standard laptop with Intel Core i7 processor, AMD Radeon graphics card, 8 Gb RAM and 500 Gb disk space and running on Windows 8.1.

5.3 Assumption

Some of the development assumptions and applicable pre-requisite conditions are;

- Each class should have minimum one and have the same number of records, to have equality in the determination of the mean image.
- Input Image dimension can be any size but should be symmetric. eg.32 x 32
- Number of filters used across layers is configurable but can be the same.
- Input images can be grayscale (1 channel) or colour (3 channels-RGB).
- The images can be of any file type (PNG, JPG, BMP etc.).

5.4 Key Features

Some of the common features applicable across processes are;

- The raw image files are initially converted to normalized before processing, wherein the colour intensities are removed and mean subtraction is done.
- In Prototyping and Training, on completion of the process, the parameters (weights and bias) of the hidden layer and fully connected layers along with the class features are saved in a numpy file format, for encrypted data security.
- The number of hidden layers required are dynamically determined based on the image and filter size.
- For convolution, 16 distinct 2-D filters of dimension 5x5 are used in each hidden layer.

5.5 Process Output

During execution of the application scripts across phases, informative details such as list of images being processed, error gradient and class predictions are displayed. Statistical summary of the classification to highlight the percentage of accuracy on correct and incorrect classes is reported at the end of training.

5.6 Visualization:

As part of the training, we can graphically visualize the feature maps and weights across hidden layers and across all kernels.

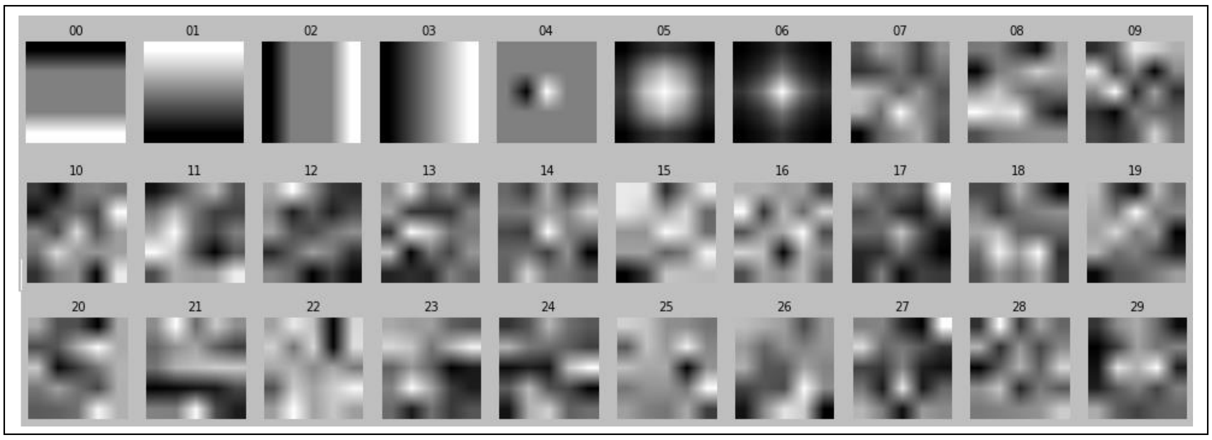
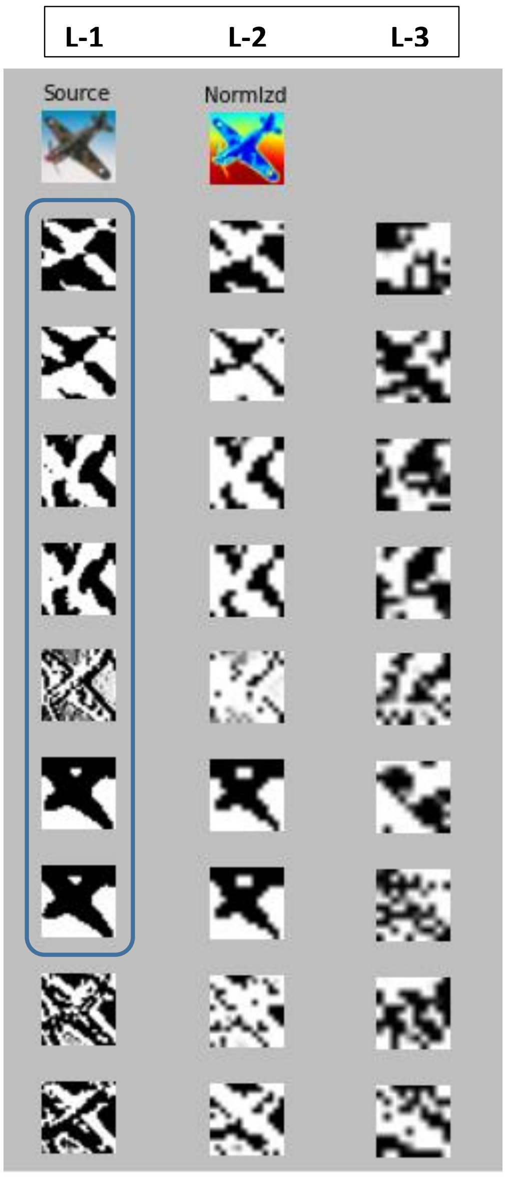


Figure-3: Sample 30 filters used as weights in the first convolutional layer

From Figure-3, we can clearly differentiate the initial manual filters used to identify horizontal lines, vertical lines and blur, and the random generated filters from filter 07 onwards. These filters are used only in the first convolutional layer. In subsequent layers all filters are random filters.

Figure-4 is the sample visualization of the feature maps of an image belonging to the class ‘aeroplane’ as in the file ‘0-fighter_aircraft_s_000655.png’.

0-fighter_aircraft_s_000655.png



Sample CIFAR-10 Image explanation:

The source and the normalized images are shown at the top for comparison. It can be seen that the under ‘Normalization’ mode the image is displayed without colour intensities.

Features of the first three hidden layers (L-1 to L-3) are shown as column data.

Each row represents the graphical representation of the first 9 kernels across the hidden layers L-1 to L-3.

In the first column the initial 7 feature maps as highlighted have been created after the input image was convolved with the 7 manual kernels made up of Prewitt, Sobel and Gaussian filters.

In the subsequent, layers the feature maps look distorted due to gradual the downsizing of the image.

The dimension of the image and feature map shown here are as follows;

1. Source Image : 32x32 pixels
2. Normalized Image : 32x32 pixels

(*After first convolution and activation*)

3. L-1 Feature Map : 28x28 pixels

(*After first max-pooling*)

4. L-2 Feature Map : 14x14 pixels

(*After second convolution and activation*)

5. L-3 Feature Map : 10x10 pixels

L-4 and L-5 are not show as the feature map dimensions are small – 5x5 and 1x1.

Figure-4: Sample Hidden Layers of a CIFAR-10 image.

6. Evaluation

CIFAR-10 dataset has been used to evaluate the CNN model developed for this project. This dataset has been used earlier in many other experiments, one of which is [31] where smaller size filters of 3x3, and large number of filters in the range of 100 – 400 have been used, and an error rate of 19.51% has been reported.

Following are the results of the experiment done on training and testing using the model developed for this project. The main objective was to observe the model behaviour with variation across a range of parameters including choice of functions across layers.

Activation function of Fully Connected Layer:
Initially Hyperbolic Tangent was used in all layers including fully connected layer. But the best accuracy obtained was 52% for 100 images. But subsequently it was learnt that while using Softmax as the Classifier the input should also be a Softmax output; as a result of which Softmax activation function was applied to the fully connected layer, and an accuracy of 98% was achieved proving the theory.

Number of Kernels:

Preliminary tests were done on a small dataset of 100 images to determine the number of kernels to be used across layers as in table-2-b.

	Kernels	Classifier Epochs	Min. Loss	Accuracy
1	8	6	2.17	18%
2	32	6	1.80	34%

Table-2: Kernel assessment

When 32 kernels were used, with a momentum of 0.01 the accuracy level was similar to what was achieved with 16 kernels. Hence 16 kernels has been standardized for this model.

Impact of change in Learning Rate (LR) of Error Gradient:

While testing for 100 images, with 16 filters and momentum of 0.1, the metrics as in table-3 were gathered for different values of the learning rate.

	LR	High Gradient Errors	Min. Loss	Accuracy
1	1e-10	Yes	2.05	27%
2	1e-5	Low	2.03	28%
3	1e-3	Low	2.24	14%

Table-3: Learning Rate assessment

It can be inferred that as the learning rate was increased from 1e-10 to 1e-5, high gradient errors were not encountered in the hidden layers, and the minimum cross-entropy loss marginally declined, with a proportionate marginal 1% increase in accuracy. When the learning rate was increased further, to 1e-3, the loss increased and the accuracy declined drastically. Thereby it was concluded that a learning rate of 1e-5 was the best for this model.

Impact of change in Momentum:

Subsequently, a test was done to check the impact of change in momentum by keeping the Learning Rate as 1e-5 and number of filters constant at 16.

	Momentum	High Gradient Errors	Min. Loss	Accuracy
1	0.01	Low as before	Marginally higher	27%
2	0.5	No change	2.03	28%

Table-4: Momentum assessment

It is worth noting that in one of the subsequent tests an accuracy of 34% was achieved with momentum at 1.0 and output weights initialized with a normalization factor of 0.0001 as against 0.01 as used in the above two tests. But the accuracy deteriorated when the dataset was increased from to more than 100 images. Hence it was determined to use a momentum of 0.5 as the standard for this model.

The following four constants as in table-5 have been used for further analysis as reported in the subsequent paragraphs.

Momentum	0.5
Error LR	1e-6
Weight Regularization	1e-6
Classifier LR	1.0

Table-5: Constants related to error gradient and loss minimization.

Prototyping using 10000 images took about an hour of processing time, whereas for 100 images the time taken was only 2 seconds.

	Classifier	**Classifier Epochs**				**Final**	
Images	**Loop Size**	**1**	**2**	**3**	**4**	**Min. Loss**	**Remarks**
100	25000	70	90	97	**98**	0.10	In the final epoch, threshold was reached below 14000 loops and the loss Gradient peaked to 2.72 before falling immediately below 0.1.
100	10000	26	30	**34**	33	1.94	Using dropout in Output layer
500	10000	31	39	39	**40**	1.65	Without dropout

Table-6: Epoch-Image Accuracy Matrix

The matrix presented in table-6 specifies the classification accuracy across 4 epochs during training phase with the use of 100 and 500 images. It can be observed that the accuracy has fallen drastically when the image count was increased from 100 to 500. Also it can be seen that the use of dropout algorithm in the output layer is not favourable to the model.

	Hidden Layer		**Classifier Loss**		**Accuracy / Classifier Epochs**				
Run	**LR**	**MTM**	**LRSS**	**WRS**	**E-0**	**E-1**	**E-2**	**E-3**	**E-4**
0	*1e-5*	*0.5*	*1.0*	*1e-10*	*83*	*83*	*87*	*92*	*92*
1	*1e-5*	*0.5*	*1.0*	*1e-10*	*84*	*91*	*92*	*92*	*93*
2				1e-5	54	72	87	65	85
3				1e-1	10	10	10	10	10
4			0.5		75	92	96	96	**97**
5			0.1		34	56	72	80	86
6			1.5		34	56	72	80	86
7	1e-10				76	84	77	94	96
8	1e-1				84	93	94	**97**	NA
9		1.0			75	88	87	94	95
10		0.1			78	88	80	87	92
11	**1e-1**	**0.5**	**0.5**	**1e-10**	**74**	**93**	**95**	**97**	**NA**

Table-7: Combination of parameters changes using 100 images

It can be inferred from the above table-7 that the parameters used in run-11 has the best results so far for training 100 images.

Chart-1 and Chart-2 below highlight the impact of variation in the hyper-parameters of the hidden layers on the error gradient.

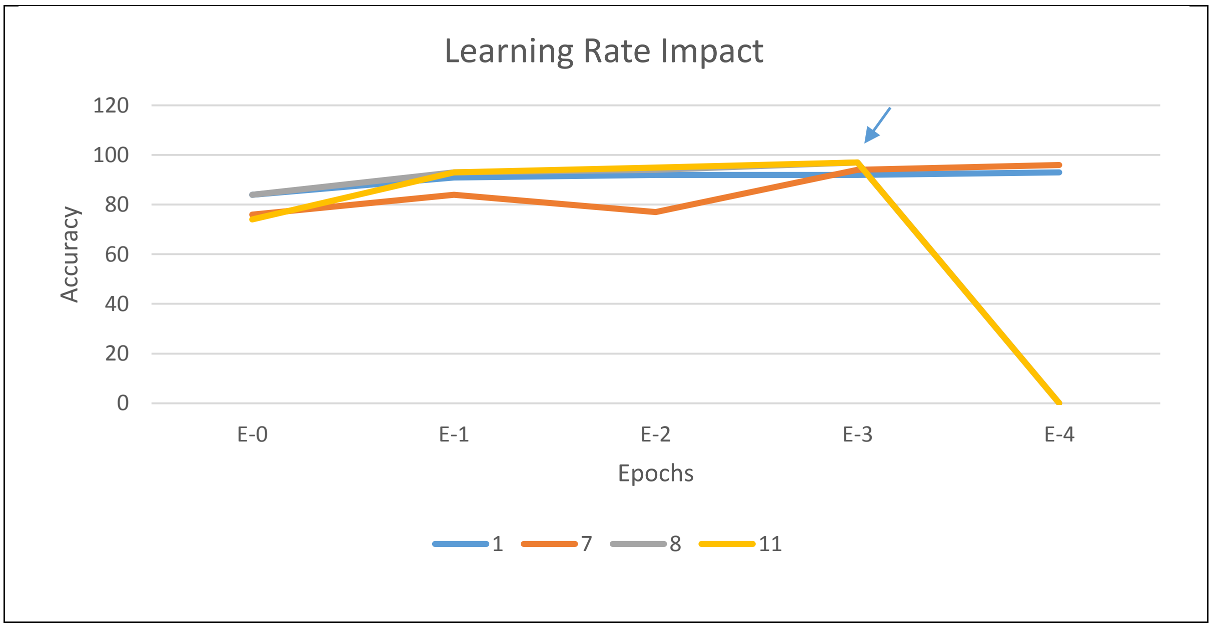


Chart-1: Accuracy impact due to variation in the Learning Rate. An overlapping of run-8 and run-11 values can be observed from epoch E-1 onwards.

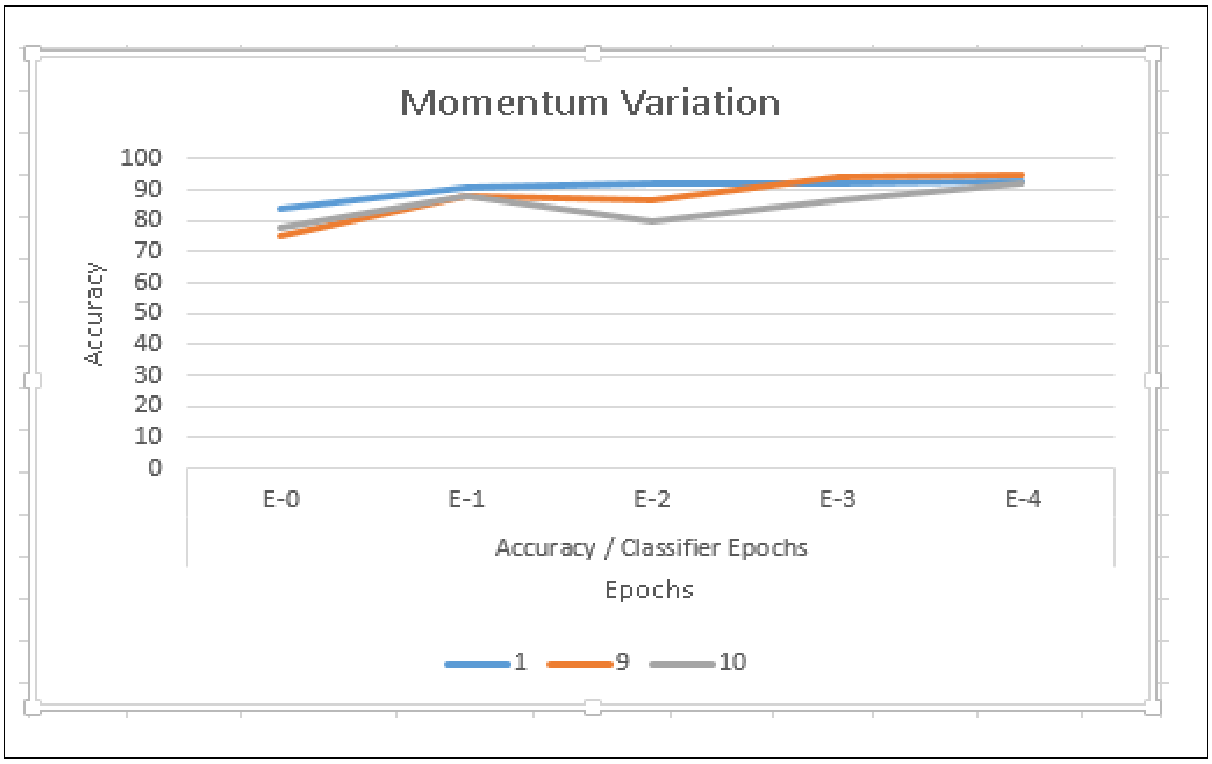


Chart-2: Accuracy impact due to variation in the Momentum

Chart-3 and Chart-4 below are related to variations that impact the classifier output.

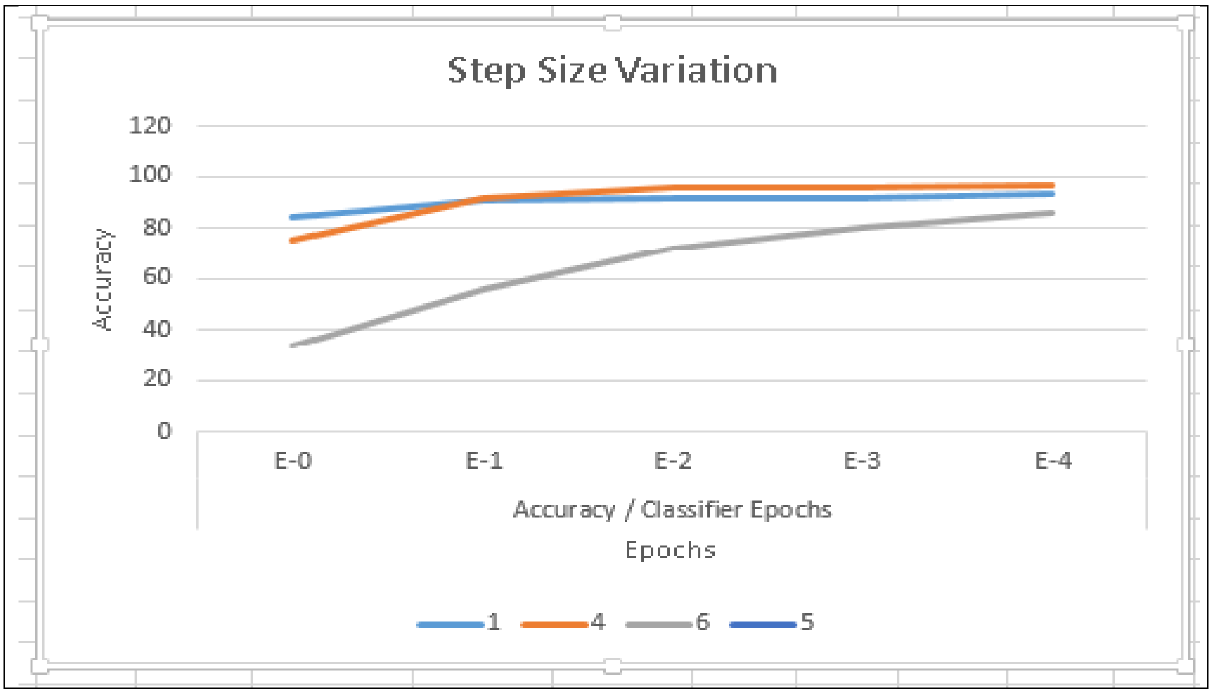


Chart-3: Accuracy impact due to variation in the Step Size of the Classifier

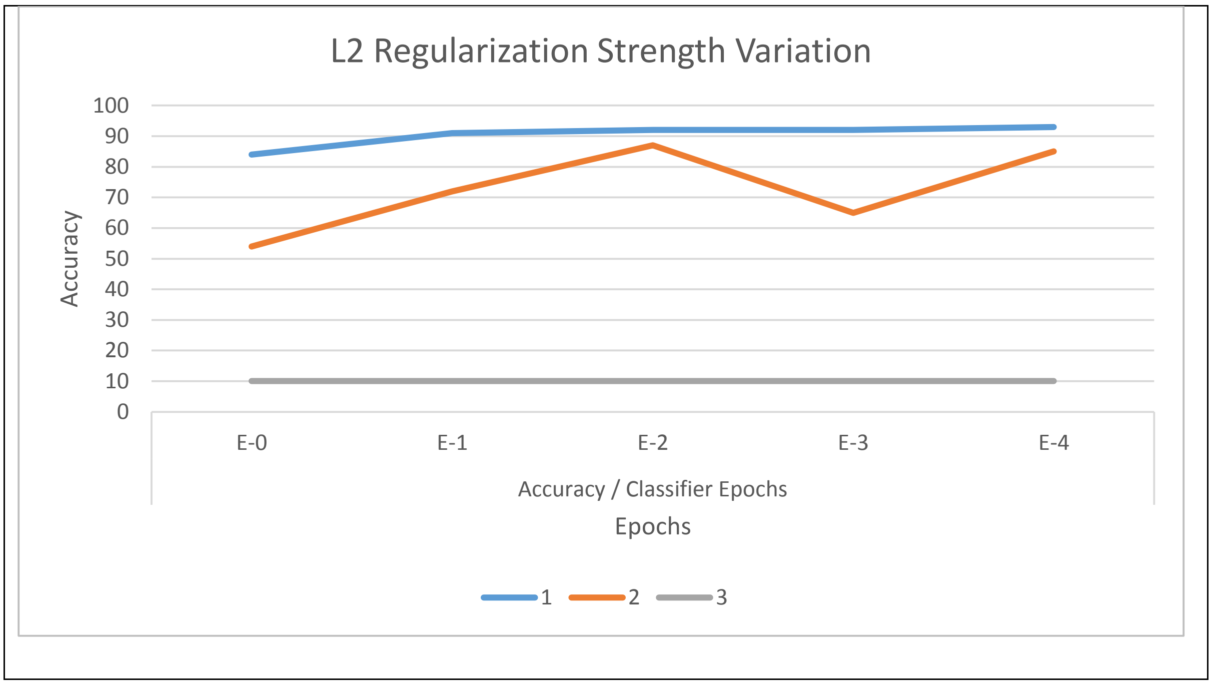


Chart-4: Accuracy impact due to variation in the L2 Regularization Strength of the Classifier

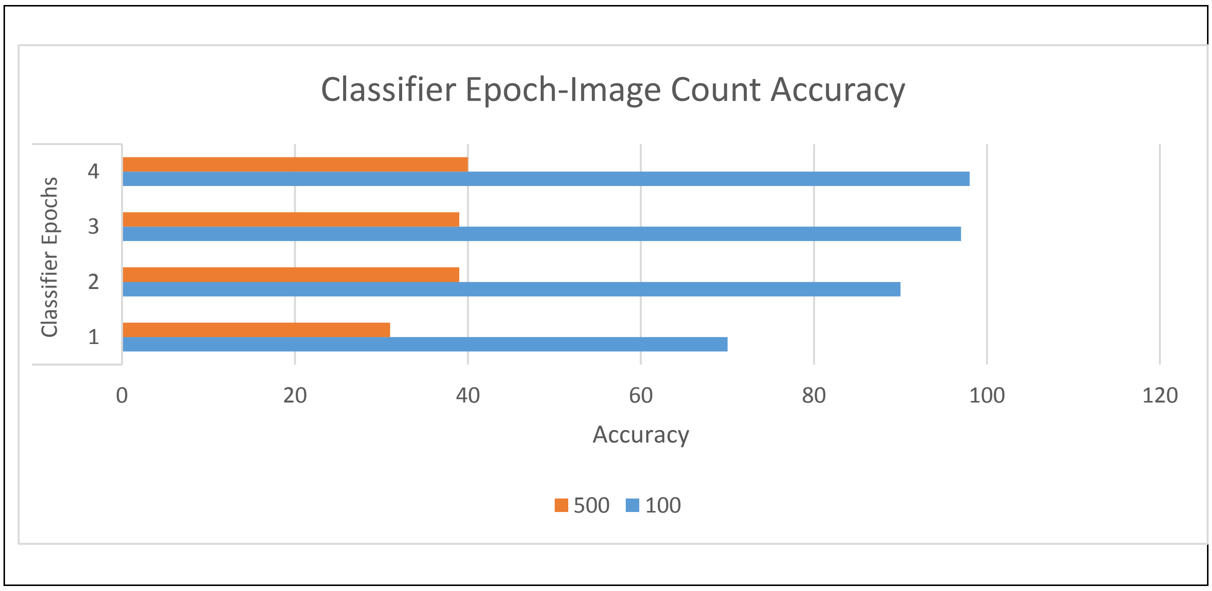


Chart-5: Accuracy comparison across image count and classifier epochs.

From chart-5, it can be inferred that the proposed model is more suitable for classifying small datasets, where the accuracy level is much higher. Also it can be seen that the accuracy stabilizes after 3 epochs for 100 image training.

Various sample output as generated during the training of 100 images are in the Appendix-A.

The summary of findings is that the model performs well and the predefined error threshold of 0.5 and loss threshold of 0.1 is achievable when the number of images is less – about 100. But as the input image count is increased, though a low error gradient is maintained, the desired accuracy in the loss minimization is not achieved in spite of multiple epochs.

7. Conclusion and Future Work

Based on this research it has been found that convolutional neural networks is a highly challenging subject to understand and implement. It is evident from this study that the various parameters of the CNN have a great influence on the image classification accuracy. Hence in addition to the use of right combination of activation functions, choice of number and size of kernels, the tuning of hyper-parameters such as learning rate and weight regularization strength are vital for a successful implementation of the model.

The proposed model is able to successfully classify within the expected threshold limits for small datasets. But with further analysis of the model and its algorithm, and fine-tuning of the parameters and hyper-parameters, the accuracy of this neural net model can be further enhanced.

Similar experiments using different datasets had been done for extended duration to see how the model performance after several 100 epochs. As referred in [34], MINST dataset had been used for training the CNN model using around 800 epochs with a time factor of 14 hours to achieve accuracy improvement. Likewise this neural net also can be studied for variation in its classification behaviour when trained for extended hours with large datasets.

Following are some of the key factors that can be considered with variations for further research as an enhancement to this project;

1. **Input**: images of higher dimension can be used to see the impact in the creation of additional hidden layers. Also direct processing of the colour images can be tested to understand the complexity involved in processing three channel inputs.
2. **Kernel:** The network can be experimented with the use of different number of filters along with variation in the size of filters across hidden layers. As the size and shape of the images progressively decline over the different layers, larger size kernels, for instance 7x7 pixels, can be used in the initial layers and a smaller size in the latter layers, such as 5x5 dimension. It has to be noted that more the number of kernels, better would be the feature detection. But implications would be increased computational time and memory requirements, as there will proportionate increase in the parameters being stored and trained. Decision to increase the number of kernels has to be rationally done based on the size of the image. In addition more manual filters can be introduced such as Gaussian blur in the first layer.
3. **Convolution:** Other modes of convolution, such as 'same', can be tried out, wherein the border pixels are also made use of especially in layers L-3 and L-5 where the number of pixels are less, and the border pixels could be of more value.
4. **Activation:** The model can be tested using other non-linear activation functions such as ReLU which are considered to have the feature of non-saturating non-linearity and have been tested to be several times faster [9] than Hyperbolic Tangent as used in this project.

5. **Classification:** In place of Softmax Classifier, the model can be modified to test using other proven classifiers mentioned earlier, especially RBF neural network, so as to improve the classification accuracy.
6. **Dataset:** To refine and test the model using large volume of CIFAR-10 dataset which would require a high-end computing GPU environment [9] to process such as the Amazon or Google cloud services. To make use of different image repositories such as ILSVRC and MNIST dataset.
7. **Application:** The neural net can be applied to classify snapshot of video images [12] and for identifying multiple objects within a single image. [5] have presented an approach for automatically annotating objects within images [17], using bag-of-visual-words concept. An advanced version of CNN could be applied to detect multiple objects in a single image.
8. **Overfitting:** The reason for the model proposed in this project not obtaining good level of accuracy, is very likely to be due to overfitting. Hence attempts could be made to refine it by adapting effective regularization method called 'dropout' [9], [23] has been suggested as a solution to the problem of overfitting. In dropout algorithm, during training neurons in each layer are randomly disabled using a drop-out map. These neurons are later activated during testing [36]. Another approach to minimize overfitting is the use of Rectified Linear Units (ReLU) as activation functions [24]. In addition, in every epoch of training, the dataset can be randomized before processing every image, to avoid overfitting of the training parameters.
9. **Parameter Tuning:** Performance of the model can be tested by further varying the values of the hyper-parameters such as momentum, learning rate etc. and also can see if any of those could be avoided as done by [11]. It has been mentioned in [33] that aspects such as decision on how weights and these parameters should be initialized has great impact on model performance and complexity. For instance, learning rate has been kept fixed in this experiment. But in [31] it has been tested by initializing to 0.001 and subsequently multiplied it by a factor of 0.993 after each epoch.
10. **Pyramid Reduction:** Experiments can also be done to see how the filters behave in tandem with *pyramid reduction* feature wherein large objects in an image are downsized so as to make it detectable as per the filter size [35].
11. **Process Output:** In addition to the on-window display, the process out can be written onto a file as part of the code. This will overcome the related challenges highlighted earlier.

References:

1. Sudhir, R. (2011) “A Survey on Image Mining Techniques: Theory and Applications”. Computer Engineering and Intelligent Systems, 2(6): pp. 44-52.
2. Wagner, R., Thom, M., Schweiger, R., Palm, G. and Rothermel, A. (2013) "Learning convolutional neural networks from few samples", In: IEEE, The 2013 International Joint Conference on Neural Networks (IJCNN), 2013, pp. 1-7.
3. Krizhevsky A., (2009), “Learning Multiple Layers of Features from Tiny Images”
4. Zhang, Y., Lu, Z. & Li, J. 2010, "Fabric defect classification using radial basis function network", Pattern Recognition Letters, vol. 31, no. 13, pp. 2033-2042.
5. Ries, C., Richter, F. and Lienhart, R. (2013) "Towards automatic object annotations from global image labels", In: ACM, Proceedings of the 3rd ACM conference on international conference on multimedia retrieval, April 2013, pp. 207-214.
6. Zhang, Y., Lu, Z. & Li, J. (2010), "Fabric defect classification using radial basis function network", Pattern Recognition Letters, vol. 31, no. 13, pp. 2033-2042.
7. Renjifo, C., Barsic, D., Carmen, C., Norman, K. & Peacock, G.S. (2008), "Improving radial basis function kernel classification through incremental learning and automatic parameter selection", Neurocomputing, vol. 72, no. 1, pp. 3-14.
8. Nando de Freitas (2015). “Deep Learning Lecture(10) – Convolutional Neural Networks”, Video file [Online]. Available from: https://www.youtube.com/watch?v=bEUX_56Lojc [Accessed 23rd August 2015].
9. Krizhevsky, A., Sutskever, I., and Hinton, G.E. (2012) “Imagenet classification with deep convolutional neural networks”. Unpublished paper, Canada: University of Toronto.
10. LeCun, Y., Bottou, L., Bengio, Y., Haffner, P., (1998), ‘Gradient Based Learning Applied to Document Recognition’, Proc. Of the IEEE, November 1998.
11. Simard, P.Y., Steinkraus, D. & Platt, J.C. (2003), "Best practices for convolutional neural networks applied to visual document analysis", pp. 958.
12. Ji, S., Xu, W., Yang, M. & Yu, K. (2013), "3D Convolutional Neural Networks for Human Action Recognition", IEEE Transactions on Pattern Analysis and Machine Intelligence, vol. 34, no. 1, pp. 221-231
13. Le, Q.,V., Ngiam,J., Chen,Z., Chia,D., Koh,P.w., Ng,A.,Y., ‘Tiled convolutional neural networks’, Stanford University, [Online]. Available from: http://ai.stanford.edu/~ang/papers/nips10-TiledConvolutionalNeuralNetworks.pdf [Accessed 23rd August 2015].
14. Simonyan, K., Zisserman, A., (2015), ‘Very Deep Comvolutional Networks for Large-scale Image Recognition’, Published as a conference paper at ICLR 2015. [Online]. Available from: http://arxiv.org/abs/1409.1556v6 [Accessed 23rd August 2015].
15. Zeiler, M.D. and Fergus, R. (2013) “Visualizing and Understanding Convolutional Networks”, Cornell University Library, arXiv:1311.2901v3. [Online]. Available from http://arxiv.org/abs/1311.2901 [Accessed 23rd August 2015].
16. Bouvrie, J., (2006), “Notes on Convolutional Neural Networks”, MIT, Cambridge, November 22, 2006.

17. Vinyals, O., Toshev, A., Bengio and S., Erhan, D. (2014) "Show and Tell: A Neural Image Caption Generator", Cornell University Library, arXiv:1411.4555v1. [Online]. Available from http://arxiv.org/abs/1411.4555 [Accessed 23rd August 2015].

18. Ciresan, D., Meier, U. & Schmidhuber, J. (2012), "Multi-column deep neural networks for image classification", IEEE, pp. 3642.

19. Ueda, N. (2000), "Optimal linear combination of neural networks for improving classification performance", IEEE Transactions on Pattern Analysis and Machine Intelligence, vol. 22, no. 2, pp. 207-215.

20. Wagner, R., Thom, M., Schweiger, R., Palm, G. & Rothermel, A. (2013), "Learning convolutional neural networks from few samples", IEEE, pp. 1.

21. Oquab, M., Bottou, L., Laptev, I. and Sivic, J. (2014) "Learning and Transferring Mid-level Image Representations Using Convolutional Neural Networks", IEEE, 2014 IEEE Conference on Computer Vision and Pattern Recognition, 2014, pp. 1717-1724.

22. Scherer, D., Müller, A. & Behnke, S. 2010, "Evaluation of Pooling Operations in Convolutional Architectures for Object Recognition" in Springer Berlin Heidelberg, Berlin, Heidelberg, pp. 92-101.

23. Srivastava,N., Hinton,G., Krizhevsky,A., Sutskever,I., Salakhutdinov,R., (2014), "Dropout: A Simple Way to Prevent Neural Networks from Overftting", The Journal of Machine Learning Research, vol. 15, no. 1, January 2014, pp. 1929-1958.

24. Egmont-Petersen, M., de Ridder, D. & Handels, H. 2002, "Image processing with neural networks—a review", Pattern Recognition, vol. 35, no. 10, pp. 2279-2301.

25. He,K., Zhang,X., Ren,S., Sun,J., (2015), "Delving Deep into Rectifiers: Surpassing Human-Level Performance on ImageNet Classification", Microsoft Research, [Online]. Available from http://arxiv.org/pdf/1502.01852.pdf [Accessed 24th August 2015].

26. Karpathy, A., "CS231n Convolutional Neural Networks for Visual Recognition", [Online]. Available from http://cs231n.github.io/ [Accessed 24th August 2015].

27. Tang, Y., "Deep Learning using Linear Support Vector Machines", [Online]. Available from http://arxiv.org/pdf/1306.0239.pdf [Accessed 24th August 2015].

28. Bishop, C.M., (2004), *Neural Networks for Pattern Recognition,* New York: Oxford University Press.

29. Mitchell, T.M., (1997), *Machine Learning,* Singapore: McGraw-Hill.

30. Barber, D., (2014), *Bayesian Reasoning and Machine Learning*, 5th edition, United Kingdom: Cambridge University Press.

31. Ciresan, D.C., Meier, U., Masci, J., Gambardella, L.M Schmidhuber,J., "Flexible, High Performance Convolutional Neural Networks for Image Classification", Proceedings of the Twenty-Second International Joint Conference on Artificial Intelligence [Online]. Available from http://people.idsia.ch/~juergen/ijcai2011.pdf [Accessed 24th August 2015].

32. Boden, M.A, (1996), *Artificial Intelligence - Connectionism and Neural Networks*, London: Academic Press.

33. Bruckner, D., Rosen, J., Sparks, E.R., "deepViz: Visualizing Convolutional Neural Networks for Image Classification", UC Berkeley, [Online]. Available from

http://vis.berkeley.edu/courses/cs294-10-fa13/wiki/images/f/fd/DeepVizPaper.pdfA [Accessed 24th August 2015].

34. Ciresan, D.C., Meier, U., Gambardella, L.M. & Schmidhuber, J. (2011), "Convolutional Neural Network Committees for Handwritten Character Classification", IEEE, pp. 1135.

35. Jaswal, D., Sowmya, V., Soman, K.P., (2014), "Image Classification using Convolutional Neural Networks", International Journal of Advancements in Research & Technology, Volume 3, Issue 6, June-2014, [Online]. Available from http://www.ijser.org/researchpaper/Image-Classification-Using-Convolutional-Neural-Networks.pdf.

36. Li, Q., Cai, W., Wang, X., Zhou, Y., Feng, D.D. & Chen, M. 2014, "Medical image classification with convolutional neural network", IEEE, =, pp. 844.

Appendix-A

Constants and Standard Settings:

Images = 100

Filters = 16

HL Error Gradient related :

Error Convergence Loop = 10

Error Epochs = 2

Error Threshold (ET) = 0.5

Initial Settings:

Learning Rate (LR) = 1e-5

Momentum (MTM) = 0.5

Classifier Loss related

Loss Convergence Loop = 50000

Epochs = 5

Loss Threshold = 0.1 # denotes 99 % accuracy

Initial Settings:

LR Step Size (LRSS) = 1.0 # Learning Rate

Weight Regularization Strength (WRS) = 1e-10 # L2 Regularization

Table-8: Initial Parameter settings

```
99 : 9-car_transporter_s_000146.png
1 High Gradient Error 0.103181320938
2 High Gradient Error 0.103179733317
3 High Gradient Error 0.103178146519
4 High Gradient Error 0.103176560542
5 High Gradient Error 0.103174975387
```

```
6 High Gradient Error 0.103173391052
7 High Gradient Error 0.103171807538
8 High Gradient Error 0.103170224844
9 High Gradient Error 0.103168642969
10 High Gradient Error 0.103167061912
```
Table-9: Sample output of convergence with high gradient error

```
95 : 9-camion_s_000388.png
1 Low Output Error 0.0715191300508

96 : 9-camion_s_000397.png
1 Low Output Error 0.133343497208

97 : 9-camion_s_001322.png
1 Low Output Error 0.0118169552257
```
Table-10: Sample output of convergence with low gradient error

```
TRAIN A LINEAR CLASSIFIER - Output Score and Loss Verification

iteration 0: loss 0.154869
iteration 2000: loss 1.511902
iteration 4000: loss 0.133649
iteration 6000: loss 0.126749
iteration 8000: loss 0.135375
iteration 10000: loss 0.129675
iteration 12000: loss 2.728299
Exiting Convergence Loop - Loss Threshold achieved
```
Table-11: Cross-Entropy loss convergence

```
 STATISTICS:
----------
(0, 'airplane') Images Correct: 9/10   Accuracy: 90.00 %
(1, 'automobile') Images Correct: 10/10   Accuracy: 100.00 %
(2, 'bird') Images Correct: 10/10   Accuracy: 100.00 %
(3, 'cat') Images Correct: 10/10   Accuracy: 100.00 %
(4, 'deer') Images Correct: 10/10   Accuracy: 100.00 %
(5, 'dog') Images Correct: 10/10   Accuracy: 100.00 %
(6, 'frog') Images Correct: 10/10   Accuracy: 100.00 %
(7, 'horse') Images Correct: 9/10   Accuracy: 90.00 %
(8, 'ship') Images Correct: 10/10   Accuracy: 100.00 %
(9, 'truck') Images Correct: 10/10   Accuracy: 100.00 %

Total Images: 100   Overall Accuracy: 98.00 %
Saving Trained Kernels and Predicted Features
Processing Time excluding Statistics and File Saving time
Started at : 2015-08-29 07:29:58
Finished at: 2015-08-29 07:30:05
```
Table-12: Final Classification Summary