Parametrization of Convolutional Neural Network for Image Classification

User Configuration Manual

Srinivasan Dasarathi
(Student Id: 14111314)

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

Table of Contents

1. Introduction 3
2. Application Environment 3
2.1 Hardware 3
2.2 Software 3
3. Application Artefacts 4
3.1 Image Extraction 4
3.2 CNN Prototyping 5
3.3 CNN Training 7
3.4 CNN Validation and Testing 10
4. Application Best Practices 11
5. Future Work 12
Appendix –A 13
A-1. CNN Image Extraction Code : 13
A-2. CNN Prototyping : 15
A-3. CNN Training : 24
A-4. CNN Validation and Testing : 39
Appendix –B 46
B-1. CNN Prototyping 46
B-2. CNN Training 46
B-3. CNN Validation 55
B-4. CNN Testing 61

1. Introduction

This configuration manual helps the users' understand the intricacies involved in the process flow of a Convolutional Neural Network(CNN) for image classification, as well as learn the internal details of the different artefacts developed in this project. Factors such as configurable parameters, process input and output, file storage and manipulations, as well as the application development environment have been elaborated.

This manual is supplementary to the dissertation thesis report submitted on the topic, and hence only technical aspects of the application are specified in the manual. The users are requested to refer the project report for details on functionalities and concepts related to CNN.

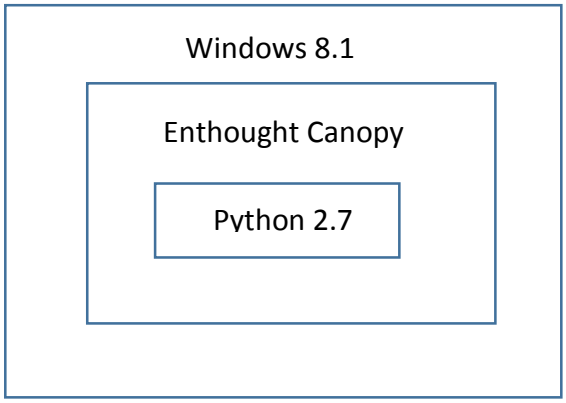
2. Application Environment

The artefacts of this project were developed and executed in the following environment.

2.1 Hardware

- Intel Core i7-4710U CPU @2.00 Ghz Processor
- AMD Radeon Graphics
- 8 GB RAM,
- 1 TB HDD

2.2 Software



- Enthought-Canopy GUI for Python 2.7 (there is also an option to choose v3.0 at the start of installation). This tool is used for coding in Python as well as for executing it. Canopy is a free download upon registration. It has a ton of python libraries including all those specified below. By default some of it are installed by default, few can be additionally installed freely, and the rest has to be subscribed. But if registered as a student with the college email-ID, then all libraries are available free of cost. Also there is a free and useful tutorial provided by Canopy online for students.
- Python Libraries used across artefacts:
  - numpy – numerical computations
  - scipy – scientific computations such as those used for convolution
  - cv2 - reading image file
  - matplotlib - graphical representation of the image data

- cPickle - to extract image data from compressed file,
- math - mathematical computations
- glob - reading of files
- os, sys,
- time
- copy – to deep copy contents of an array onto another array in a different address.

In case the normal '=' assignment operator is used, then the same address will be assigned. As a result, the old array variable will get updated as and when the new array is updated. Hence,'copy.deepcopy' function is used for copying arrays.

3. Application Artefacts

3.1 Image Extraction

Source Code: CIFAR-10-Extract.py (refer Appendix-A)

Purpose: To extract viewable images across classes from the compressed CIFAR-10 image dataset file as available on the net.

Data Source: https://www.kaggle.com/c/cifar-10/data contains 6 compressed batch data files, each containing 10,000 images.

Configuration Parameters:

1. folder - the directory of the source file
2. extract_img - number of images to extract in one run.
3. data_dict - location and source file name. This can be a pattern as well.

File Input: One compressed CIFAR-10 dataset file in a machine readable format, that contains a dictionary on the data, labels and file name. For each RGB image, data is stored as a vector, followed by the class label and file name.

File Output: A file is created for each image. File name is prefixed with the label.

Display : The name of each image file as it is processed is shown.

Process Steps:

1. Read each line, and convert the vector into 3 rows of array, one each for RGB, of size 32x32 pixel value, along with the label and file name. There are 5 images in every row. The concept of *unpickling* is applied here whereby the byte streams are converted back to object hierarchy.
2. Saves the array data as a PNG file. Before saving , the file name is prefixed with the label, so as to enable identification of the class to which the image belongs.
3. The process terminates once the 'extract_img' count is reached.

3.2 CNN Prototyping

Source Code: CNN_Prototyping.py (refer Appendix-A)

Purpose:

1. To initiate all weights and bias used across all hidden layers and fully connected layer.
2. To determine the mean image of each class and to identify the features across all 10 classes.
3. To save the arrays specified in #1 and #2 for use by the Training code.

Configuration Parameters:

Sample values as used in this project are specified within parenthesis.

1	data_folder	Folder where the input image data files are stored. ('C:\\SRINI\\CNN-DATASET\\CIFAR10\\ORIGINAL\data_b3\\b100-train') . This should be the same as the training data folder.
2	proc_folder	Folder where the trained parameter files of weights and bias, and class features are to be stored. ('C:\\SRINI\\CNN-DATASET\\CIFAR10\\CNN-PROC-FILES\\')
3	search_files	File pattern for selecting list of image files ("*.png")
4	max_imgs_to_process	Maximum images to process (10). Make sure this is minimum 10 as equal to total no. of classes - else Prototyping process will fail at runtime.
5	tot_kernels	Number of kernels (filters) to be used in each convolutional layer (16)
6	proto_wts_file	File to store the weights of the hidden layers ('CNN-Proto-Weights.npy')
7	proto_bias_file	File to store the bias of the hidden layers ('CNN-Proto-Biases.npy')
8	proto_fc_wts_file	File to store the weights of the fully connected layer 'CNN-Proto-FC-Weights.npy' [Note that bias has not been used in the fully connected layer on experimental basis. Normal practice is to attach a bias to the output node when weights are applied.]
9	proto_class_file	File to store the features of each class as determined as the output of the fully connected layer ('CNN-Proto-Class-Features.npy')
10	class_dict	A python dictionary with the label keys and values. {0:'airplane', 1:'automobile',2:'bird',3:'cat',4:'deer',5:'dog', 6:'frog', 7:'horse', 8:'ship', 9:'truck'}
11	kernel_dim	Filter dimension – (5 means 5x5 pixels)
12	bias_val	Bias of the hidden layers' output nodes (0.01)
13	last_manual_kernel_idx	The index of last manual filter hard-coded in the program as a 2-D array. Note that this is not the total no. of kernels but rather the index of the last kernel. A value of (6) indicates there are total 7 manual filters (0-6). This is specified for two purposes; (a) To generate random filters for the remaining filters. (b) To prevent update to these filters during backpropagation.

14	conv_mode	The mode of convolution, provided as a parameter to the scipy function ('valid') Other option is 'same'. These determine the use of boundary pixels of the input maps during convolution.
15	conv_function	Choice of convolution function to be used (1).
16	pool_function	Choice of pooling function to be used – max pooling or mean-pooling (1).
17	hid_activate_function	Choice of Activation function – Logistic Sigmoid, Hyperbolic Tangent, ReLu etc for use in the hidden layers (1)
18	fc_activate_function	Choice of Activation function – Logistic Sigmoid, Hyperbolic Tangent, ReLu etc for use in the fully connected layer (2)
19	tot_class	Total classes (length of the class_dict parameter value)
20	imgs_to_process	Total no. of images to process. Derived from the list of files.
21	cls_imgs_cnt	Average no. of images per class. Derived value.

Input: Same batch of image files (PNG) that would be used for training. There should be atleast 1 file for each class, and all classes should have same number of images to have uniformity during the selection of class representative images.

Output: The weights, bias and class features are stored in an encrypted numpy file format for data protection, just in case the files have to be sent across over the internet or external media. (refer Appendix-B for sample output)

Process Steps:

1. Images are read and normalized. As part of normalization the RGB files are converted to grayscale (by setting flatten = 0), converted into a vector, and subtracted from the mean value of the vector to minimize computational errors during execution of exponentiation and logarithmic functions.

2. All parameters and array variables are initialized. Weights (filters) and Bias are initialized across layers. For the first hidden layer, all the seven manual filters (5x5) that are hard-coded are applied. The manual filters are edge detectors for Horizontal and Vertical Gradient, Horizontal and Vertical Mask - Prewitt Edge, Diagonal Sobel Edge and Gaussian blur.

   For the remaining filters of the first layer and all filters in all other hidden layers are initialized to a Gaussian weight range of -1 to +1. The weights are also mean normalized before being applied, and are stored as a vector.

3. Based on the file name prefix, store the image labels.
   Process the following for each image, which involves the sequence of convolution, activation and pooling across hidden layers. The number of hidden layers required is computed by iteratively dividing the image size by the kernel size, until the image size becomes less than or equal to the kernel size. The number of iterations is considered as the number of layers. The logic behind this is, after every convolution, activation and pooling the image size gets downsized to almost half irrespective of whether the convolution mode is 'valid' or 'same'. For each hidden layer convolution and activation are first done, and the output of the activation function is stored in feature maps. Subsequently max pooling is done and the feature map is stored as well. Max pooling is done on all hidden layers except the last hidden layer as it already contains 1x1 pixel node, and there is no scope for further downsizing.

a) Convolution : There are two configurable options to perform convolution. If (conv_function == 1), then the scipy function 'convolve2d' is performed, which does normal convolution. Alternatively if (conv_function == 2), then 'fftconvolve' is executed which performs the Fast Fourier Transform function. In the process, for every rectifier field or window, the filter convolves across all the feature maps (input channels) and then sums up the value to which the bias factor is added and then the value is assigned to the node of the output feature map.

b) Activation: There are four options available for linear activation function that are used in the hidden layers and in the fully connected layer.
   - 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.

c) Pooling: Max-pooling is performed. Optionally mean-pooling could be experimented,

4. In the fully connected layer, the product of the feature map and the weights of that layer are computed, and then fed onto the activation function based on the choice of function as configured in 'fc_activate_function'. This output is considered as the feature of the class to which the mean image belongs. In the model Softmax activation function is applied, as Softmax Classifier is made used of in the Output layer.

   Euclidean Distance(ED) is a feature that is used often across programs in this project, to compute the distance between two data points. It is measured by the squared arithmetic difference between two data items (say x and y) , and then taking the square root of it. Here, *Mean Square Difference (MSD)* is computed by taking the Euclidean Distance between the data points in each class. For instance, the distance between the sum of all pixels in image-1 and image-2 in class 'plane' is determined. Then between image-1 and image-3, and so on. Finally the image in the class which has the least difference is chosen as the class representative.

5. Save the weights, bias and class features as numpy files
6. Record the process start and finish time.

3.3 CNN Training

Source Code: CNN_Training.py (refer Appendix-A)

Purpose:

1. To initiate weights and bias of the output classifier.
2. To train the filters (weights) and bias across hidden layers through back-propagation by cross validating the error gradient convergence against the predefined error threshold.
3. To train the filters and bias of the output classifier layer through back-propagation by cross validating the cost or loss convergence against the predefined loss threshold.
4. To save the array data of filters and bias across layers onto numpy files for later use in the Validation and Training phases.

Configuration Parameters:

Sample values as used in this project are specified within parenthesis.

1	data_folder	Folder where the input image data files are stored. ('C:\\SRINI\\CNN-DATASET\\CIFAR10\\ORIGINAL\data_b3\\b100-train')
2	proc_folder	Folder where the parameter files of weights and bias (-pre-training and post-training, and class features are to be stored. ('C:\\SRINI\\CNN-DATASET\\CIFAR10\\CNN-PROC-FILES\\')
3	search_files	File pattern for selecting list of image files ("*.png")
4	max_imgs_to_process	Maximum images to process (10). Make sure this is minimum 10 as equal to total no. of classes - else Prototyping process will fail at runtime.
5	tot_kernels	Number of kernels (filters) to be used in each convolutional layer (16)
6	proto_wts_file	File to store the weights of the hidden layers ('CNN-Proto-Weights.npy')
7	proto_bias_file	File to store the bias of the hidden layers ('CNN-Proto-Biases.npy')
8	proto_fc_wts_file	File to store the weights of the fully connected layer 'CNN-Proto-FC-Weights.npy' [Note that bias has not been used in the fully connected layer on experimental basis. Normal practice is to attach a bias to the output node when weights are applied.]
9	proto_class_file	File to store the features of each class as determined as the output of the fully connected layer ('CNN-Proto-Class-Features.npy')
10	train_wts_file	Trained weights of hidden layer ('CNN-Train-Weights.npy')
11	train_bias_file	Trained bias of hidden layer ('CNN-Train-Biases.npy')
12	train_fc_wts_file	Trained weights of fully connected layer ('CNN-Train-FC-Weights.npy')
13	train_out_wts_file	Trained weights of output layer ('CNN-Train-Output-Weights.npy')
14	train_out_bias_file	Trained bias of output layer ('CNN-Train-Output-Bias.npy')
15	train_out_wts2_file	Trained weights (additional) of output layer ('CNN-Train-Output-Weights2.npy')
16	train_out_bias2_file	Trained (additional) bias of output layer 'CNN-Train-Output-Bias2.npy'
17	class_dict	A python dictionary with the label keys and values. {0:'airplane', 1:'automobile',2:'bird',3:'cat',4:'deer',5:'dog', 6:'frog', 7:'horse', 8:'ship', 9:'truck'}
18	kernel_dim	Filter dimension – (5 means 5x5 pixels)
19	last_manual_kernel_idx	The index of last manual filter hard-coded in the program as a 2-D array. Note that this is not the total no. of kernels but rather the index of the last kernel. A value of (6) indicates there are total 7 manual filters (0-6). This is specified for two purposes;

		(c) To generate random filters for the remaining filters. (d) To prevent update to these filters during backpropagation.
20	conv_mode	The mode of convolution, provided as a parameter to the scipy function ('valid') Other option is 'same'. These determine the use of boundary pixels of the input maps during convolution.
21	conv_function	Choice of convolution function to be used (1).
22	pool_function	Choice of pooling function to be used – max pooling or mean-pooling (1).
23	hid_activate_function	Choice of Activation function – Logistic Sigmoid, Hyperbolic Tangent, ReLu etc for use in the hidden layers (1)
24	fc_activate_function	Choice of Activation function – Logistic Sigmoid, Hyperbolic Tangent, ReLu etc for use in the fully connected layer (2)
25	hid_deriv_function	Derivative of the activation function used in the hidden layer (1)
26	fc_deriv_function	Derivative of the activation function used in the fully connected layer (2)
27	tot_class	Total classes (length of the class_dict parameter value)
28	imgs_to_process	Total no. of images to process. Derived from the list of files.
29	cls_imgs_cnt	Average no. of images per class. Derived value.

Input: Batch of image files (PNG) that would be used for training. There should be atleast 1 file for each class, and all classes should have same number of images to have uniformity during the selection of class representative images.

Output: The trained weights and bias across layers are stored in an encrypted numpy file format for data protection. (refer Appendix-B for sample output)

Process Steps:

1. The feed forward process steps from image normalization in the input layer to feature generation in the fully connected layer are the same as done earlier for Prototyping.
2. Additional processes executed in Training are;
   a) back-propagation for error gradient convergence and loss minimization.
   b) Score computation and image classification.
3. At the end of the feed forward process, the predicted features are determined in the fully connected layer. These are compared with the class features generated earlier, and if the Mean Square Difference is higher than the expected threshold, back-propagation is executed to train the weights and bias parameters.
4. The loop of feed forward and back-propagation gets terminated as and when either one of the following happens;
   a) Error threshold is reached.
   b) Configured maxmium convergence loops and epochs are exhausted.
5. Similar loop of feed forward and back-propagation is done in the output classifier layer where the cross-entropy loss is minimized in comparison with the loss threshold.

6. Finally the scores are computed and the class of the image is determined.
7. The statistics of the classification is published and the trained parameters are saved for later use in the neural net Validation and Testing phases.

3.4 CNN Validation and Testing

Source Code: CNN_Testing.py (refer Appendix-A)

Purpose:

1. To use the trained parameters to classify a new set of data.

Configuration Parameters:

Sample values as used in this project are specified within parenthesis.

1	data_folder	Folder where the input image data files are stored. ('C:\\SRINI\\CNN-DATASET\\CIFAR10\\ORIGINAL\data_b3\\b100-test')
2	proc_folder	Folder where the parameter files of weights and bias (-pre-training and post-training, and class features are to be stored. ('C:\\SRINI\\CNN-DATASET\\CIFAR10\\CNN-PROC-FILES\\')
3	search_files	File pattern for selecting list of image files ("*.png")
4	max_imgs_to_process	Maximum images to process (100).
5	tot_kernels	Number of kernels (filters) to be used in each convolutional layer (16)
10	train_wts_file	Trained weights of hidden layer ('CNN-Train-Weights.npy')
11	train_bias_file	Trained bias of hidden layer ('CNN-Train-Biases.npy')
12	train_fc_wts_file	Trained weights of fully connected layer ('CNN-Train-FC-Weights.npy')
13	train_out_wts_file	Trained weights of output layer ('CNN-Train-Output-Weights.npy')
14	train_out_bias_file	Trained bias of output layer ('CNN-Train-Output-Bias.npy')
15	train_out_wts2_file	Trained weights (additional) of output layer ('CNN-Train-Output-Weights2.npy')
16	train_out_bias2_file	Trained (additional) bias of output layer 'CNN-Train-Output-Bias2.npy'
17	class_dict	A python dictionary with the label keys and values. {0:'airplane', 1:'automobile',2:'bird',3:'cat',4:'deer',5:'dog', 6:'frog', 7:'horse', 8:'ship', 9:'truck'}
18	kernel_dim	Filter dimension – (5 means 5x5 pixels)
19	last_manual_kernel_idx	The index of last manual filter hard-coded in the program as a 2-D array. Note that this is not the total no. of kernels but rather the index of the last kernel. A value of (6) indicates there are total 7 manual filters (0-6).

		This is specified for two purposes; (e) To generate random filters for the remaining filters. (f) To prevent update to these filters during backpropagation.
20	conv_mode	The mode of convolution, provided as a parameter to the scipy function ('valid') Other option is 'same'. These determine the use of boundary pixels of the input maps during convolution.
21	conv_function	Choice of convolution function to be used (1).
22	pool_function	Choice of pooling function to be used – max pooling or mean-pooling (1).
23	hid_activate_function	Choice of Activation function – Logistic Sigmoid, Hyperbolic Tangent, ReLu etc for use in the hidden layers (1)
24	fc_activate_function	Choice of Activation function – Logistic Sigmoid, Hyperbolic Tangent, ReLu etc for use in the fully connected layer (2)
27	tot_class	Total classes (length of the class_dict parameter value)
28	imgs_to_process	Total no. of images to process. Derived from the list of files.
29	cls_imgs_cnt	Average no. of images per class. Derived value.

Input: Batch of new image files (PNG) to be used for validation. As the same code will be used, different datasets have to be provided for validation and testing.

Output: The predicted classification of the images are displayed. (refer Appendix-B)

Process Steps:

1. In these phases only the feed forward process as specified for Prototyping and Training are performed. The images data is processed and the predicted classification based on the trained parameters is determined.

4. Application Best Practices

Following are the recommended do's and don'ts learnt from this project.

1. Use copy.deepcopy function for array assignments.
2. Canopy and Python libraries could get automatically upgraded if connected.
3. Backup – keep store of each working version of the artefacts, outputs and configuration
4. Restart kernel on Canopy before program run, to ensure the values of the previous run of the same or different file as on memory are not reused. File save is written completely to the files and not in-transit on buffers.

5. Future Work

In addition to the code updates that would be applicable for implementation of future work proposed in the thesis report, the following code related tasks are recommended and can be taken up as an enhancement activity;

- Use of OOAD ‘class’ concepts.
- Provision to use different number of kernels across hidden layers
- Provision to use different size filters across layers
- Store the runtime output that’s currently displayed in a log file.
- Develop a GUI apps to show the runtime progress such as error gradiance etc
- Improve visualization to display images being processed during runtime, and to make provision for display of kernels and feature maps as part of the code, which is currently commented and optional.
- Time the process s performance for large volume data or larger epochs using python’s timeit() function on a GPU platform.

Appendix –A

(Source code used in this project)

A-1. CNN Image Extraction Code :

```
# -*- coding: utf-8 -*-
#
# Python 2.7 on Enthought Canopy 1.5.2 (64-bit)
#
# Enthought Canopy Library's required
# 1. PIL - for Image Processing
#
# Code Author: Srinivasan Dasarathi(14111314), School of Computing, NCI
import cPickle
import numpy as np
from scipy.misc import imsave


def unpickle(file):
fo = open(file, 'rb')
dict = cPickle.load(fo)
fo.close()
return dict

folder= 'C:\\SRINI\\CNN-DATASET\\CIFAR10\\ORIGINAL\\data_b4\\'
extract_img = 10000

xs = []
ys = []
for j in range(4,5):
data_dict = unpickle(folder+'data_batch_'+`j`)
x = data_dict['data'] # image - stored as a vector
y = data_dict['labels'] # class of the image 0-9 range
f = data_dict['filenames'] # file name of the image

xs.append(x)
ys.append(y)

#print len(d['filenames']),len(x), len(y)

#print len(x[0]), len(ys)
#print x[0].shape
#print len(xs[0]), len(x[0])

#d = unpickle('test_batch')
#xs.append(d['data'])
#ys.append(d['labels'])

#x = np.concatenate(xs)
#y = np.concatenate(ys)

```

```
38.
39. #gs = gridspec.GridSpec(5, 5)
40.
41. i=0
42. j=0
43. # process first few images
44. for r in range(0,extract_img ):
45. img = np.dstack((x[r][:1024], x[r][1024:2048], x[r][2048:3072])) # restructures
    1D to 3 2Ds
46. org_img = img.reshape(32,32,3)
47. print f
48.
49. # ax = plt.subplot(gs[i,j])
50. # ax.set_title(y[r])
51. # ax.set_axis_off()
52. # ax.imshow(org_img)
53. # saves the images in the folder using the file name extracted from the data batch
54. imsave(folder+str(y[r])+'-'+f[r],org_img)
55. j+=1
56. if j==5: # autoincrement row for every 5 images
57. j=0
58. i+=1
59.
60. """
61. import matplotlib.pyplot as plt
62. import matplotlib.gridspec as gridspec
63. img_normalized = x[0:5] - np.mean(x[0:5]) # Normalization - 0-center the data
    (important)
64.
65. i=1
66. j=0
67. # process first 5 images
68. for r in range(0,5):
69.
70. img = np.dstack((img_normalized[r][:1024], img_normalized[r][1024:2048],
    img_normalized[r][2048:3072])) # restructures 1D to 3 2Ds
71. nor_img = img.reshape(32,32,3)
72.
73. ax = plt.subplot(gs[i,j])
74. ax.set_title(y[r])
75. ax.set_axis_off()
76. ax.imshow(nor_img)
77. j+=1
78. if j==5: # autoincrement row for every 5 images
79. j=0
80. i+=1
81.
82. covariance = np.dot(X.T, X) / X.shape[0] # get the data covariance matrix
83. U,S,V = np.linalg.svd(covariance) # compute SVD factorization, U is the eigenvector
84. # X shape (32L, 32L, 3L)
85. # U shape (3L, 32L, 32L, 32L)
86. # S (3L, 32L, 3L)
87. # V (3L, 32L, 3L, 3L)
```

```
88.
89. Xrot = np.dot(X, U) # decorrelate the data
90. img_reduced = np.dot(X, U[:,:2352]) # reduce by removing columns that have no
    variance
91. """
92. # EOF
```

A-2. CNN Prototyping :

```
1.  # -*- coding: utf-8 -*-
2.  #
3.  # Python 2.7 on Enthought Canopy 1.5.2 (64-bit)
4.  #
5.  # Enthought Canopy Library's required
6.  # 1. PIL - for Image Processing
7.  #
8.  # Code Author: Srinivasan Dasarathi(14111314), School of Computing, NCI
9.  #
10. # Process : PROTOTYPING for Convolutional Neural Networks (CNN) for 2D Image
11. #
12. # Python Libraries: Numpy 1.9.2-1, Scipy 0.15.1-2, Matplotlib 1.4.3-4,
13. # PIL, NeuroLab 0.2.0-1
14.
15. # Last Update: 30/08/2015
16. """
17. # Assumptions and Features:
18. #
19. # 1. Input Image can be Grayscale (1 channel) or Color (3 channels-RGB).
20. # But data will be processed in Grayscale
21. #
22. # 2. Input Image dimension can be any size but should be symmetric. eg.32 x 32 -
    CIFAR-10 Dataset
23. #
24. # 3. Filter/Kernel Size configured for dimension 5x5
25. #
26. # 4. No.of kernels across layers can be altered but has to be the same.
27. #
28. # 5. No. of kernels used in the prototype can be MAXIMUM,
29. # and then can be reduced during TRAINING.
30. """
31. #---------------------------------------------------------------
32. # ----- FUNCTION DEFINITIONS -----
33.
34. def save_data():
35. os.chdir(proc_folder)
36.
37. np.save(proto_wts_file, wts_conv_class)
38. np.save(proto_bias_file, bias_conv_class)
39. np.save(proto_fc_wts_file, fully_connected_wts)
40. np.save(proto_class_file,class_features)
```

```
41. return
42. #-----
43. def hidden_layers_reqd(img_list):
44. src_img = cv2.imread(img_list[0],0)
45. w_dim,h_dim = src_img.shape
46.
47. hid_layers_cnt = 0
48. image_size = w_dim
49. while (image_size > kernel_dim):
50. hid_layers_cnt += 1
51. image_size /= 2
52. return(hid_layers_cnt,w_dim)
53. #-----
54. def normalize_images(img_list):
55. proc_cnt = 0
56. for image_in in img_list:
57. in_image = cv2.imread(image_in,0)
58. # Mean Subtraction
59. normalize_img = in_image - np.mean(in_image)
60. # store the image contents
61. src_data.append(in_image.flatten())
62. nor_image_data.append(normalize_img.flatten())
63.
64. proc_cnt += 1
65. return
66.
67. #-----
68. def initialize_kernel_bias():
69. kernel_id = 0
70. for l in range(tot_conv_layers):
71.
72. for k in range(tot_kernels):
73. wts_conv_class[l][k] = filters_5(kernel_id,l)
74. bias_conv_class[l][k] = bias_val
75. kernel_id += 1
76. return
77.
78. #------
79. def filters_5(kernel_id, ker_lyr):
80. wts_size = kernel_dim**2
81. filter_mode="random"
82.
83. if ker_lyr == 0: # Apply manual filters for first C-0 layer only
84. # Manual Filters
85. if kernel_id == 0: # Horizontal Gradient
86. filter_mode="manual"
87. man_filter = np.array([ [-1,-1,-1,-1,-1],
88. [ 0, 0, 0, 0, 0],
89. [ 0, 0, 0, 0, 0],
90. [ 0, 0, 0, 0, 0],
91. [ 1, 1, 1, 1, 1] ])
92. elif kernel_id == 1: # Horizontal Mask - Prewitt Edge Detector
93. filter_mode="manual"
```

```
94. man_filter = np.array([ [ 2, 2, 2, 2, 2],
95. [ 1, 1, 1, 1, 1],
96. [ 0, 0, 0, 0, 0],
97. [-1,-1,-1,-1,-1],
98. [-2,-2,-2,-2,-2] ])
99. elif kernel_id == 2: # Vertical Gradient
100.    filter_mode="manual"
101.    man_filter = np.array([ [-1, 0, 0, 0, 1],
102.    [-1, 0, 0, 0, 1],
103.    [-1, 0, 0, 0, 1],
104.    [-1, 0, 0, 0, 1],
105.    [-1, 0, 0, 0, 1] ])
106.    elif kernel_id == 3: # Vertical Mask - Prewitt Edge Detector
107.    filter_mode="manual"
108.    man_filter = np.array([ [-2,-1, 0, 1, 2],
109.    [-2,-1, 0, 1, 2],
110.    [-2,-1, 0, 1, 2],
111.    [-2,-1, 0, 1, 2],
112.    [-2,-1, 0, 1, 2] ])
113.    elif kernel_id == 4: # Egde Detection-1
114.    filter_mode="manual"
115.    man_filter = np.array([ [0, 0, 0, 0, 0],
116.    [0, 0, 0, 0, 0],
117.    [0,-1, 1, 0, 0],
118.    [0, 0, 0, 0, 0],
119.    [0, 0, 0, 0, 0] ])
120.    elif kernel_id == 5: # Diagonal Sobel Edge detector
121.    filter_mode="manual"
122.    man_filter = np.array([ [2, 4, 5, 4, 2],
123.    [4, 9,12, 9, 4],
124.    [5,12,15,12, 5],
125.    [4, 9,12, 9, 4],
126.    [2, 4, 5, 4, 2] ])
127.    elif kernel_id == 6: # Gaussian Blur detector
128.    filter_mode="manual"
129.    man_filter = np.array([ [1, 2, 4, 2, 1],
130.    [2, 4, 8, 4, 2],
131.    [4, 8,16, 8, 4],
132.    [2, 4, 8, 4, 2],
133.    [1, 2, 4, 2, 1] ])
134.    if (filter_mode == "random"): # Guassian - weights range -1.0 to +1.0
135.    filter_val = 0.1 * np.random.randn(wts_size) #* math.sqrt(2.0/wts_size)
136.    #filter_val = np.random.randn(wts_size)
137.    # Mean Normalization
138.    filter_val = filter_val - np.mean(filter_val)
139.    else:
140.    filter_val = man_filter.reshape(wts_size)
141.    return (filter_val)
142.    #-----
143.    def auto_convolve(ac_conv_idx, ac_func_idx, ac_k, ac_kernel_2d,
    ac_in_channel_cnt):
144.    # Process for every Window
145.
```

```
146.  # Loop for every Input Feature Map or Image
147.  for loop in range(ac_in_channel_cnt):
148.  if (ac_conv_idx == 0):
149.  rf_2d = main_image.reshape(image_dim,image_dim)
150.  else:
151.  fm_in_arr = fea_map_data[ac_func_idx -1][ac_k]
152.
153.  arr_dim = fm_in_arr.shape[0]
154.  rf_2d = fm_in_arr.reshape(arr_dim,arr_dim)
155.  if (conv_function == 1):
156.  rf_conv_out = sg.convolve2d(rf_2d, ac_kernel_2d,
157.  mode=conv_mode, boundary='fill', fillvalue=0) \
158.  + bias_conv_class[ac_conv_idx][ac_k] #bias_val
159.  elif (conv_function == 2): # uses Fast Fourier Transform function
160.  rf_conv_out = sg.fftconvolve(rf_2d, ac_kernel_2d,
161.  mode=conv_mode) \
162.  + bias_conv_class[ac_conv_idx][ac_k] # bias_val
163.  if loop == 0:
164.  arr_dim = rf_conv_out.shape[0]
165.  rf_depth_arr = np.zeros((arr_dim,arr_dim))
166.  rf_depth_arr = np.add(rf_depth_arr,rf_conv_out)
167.
168.  return (rf_depth_arr)
169.  #-----
170.  def pooling(pool_in, pool_function):
171.  if (pool_function == 1):
172.
173.  ds_factor = 2
174.  ys,xs = pool_in.shape
175.  crarr = pool_in[:ys-(ys % int(ds_factor)),:xs-(xs % int(ds_factor))]
176.  pool_out = np.nanmax( np.concatenate([[crarr[i::ds_factor,j::ds_factor]
177.  for i in range(ds_factor)]
178.  for j in range(ds_factor)]), axis=0)
179.  return pool_out
180.  #-----
181.  def hid_activation(fm_in_act, activate_function, in_dim):
182.  hid_activate_out = np.zeros((in_dim, in_dim))
183.
184.  if (activate_function == 0):
185.  for j in range(0,in_dim): # Logistic Sigmoid Activation
186.  for i in range(0,in_dim):
187.  z = fm_in_act[i,j]
188.  if (z < -20.0): hid_activate_out[i,j] = 0.0
189.  elif (z > 20.0): hid_activate_out[i,j] = 1.0
190.  else: hid_activate_out[i,j] = 1.0/(1.0 + np.exp(-z))
191.
192.  elif (activate_function == 1): # Hyperbolic Tangent Activation
193.  for j in range(0,in_dim):
194.  for i in range(0,in_dim):
195.  hid_activate_out[i,j] = math.tanh(fm_in_act[i,j])
196.  elif (activate_function == 2): # Softmax Activation
197.  for j in range(0,in_dim):
198.  for i in range(0,in_dim):
```

```
199.    hid_activate_out[i,j] = np.maximum(0,fm_in_act[i,j])
200.    elif (activate_function == 3): # ReLU Activation - Analytic SoftPlus Function
201.    for j in range(0,in_dim):
202.    for i in range(0,in_dim):
203.    x = fm_in_act[i,j]
204.    hid_activate_out[i,j] = np.maximum(0,x)
205.
206.    return (hid_activate_out)
207.
208.    #-----
209.    def fc_out_activation(in_act,activate_function):
210.    if (activate_function == 0): # Logistic Sigmoid Activation
211.    if (in_act < -20.0): out_act = 0.0
212.    elif (in_act > 20.0): out_act = 1.0
213.    else: out_act = 1.0/(1.0 + np.exp(- in_act))
214.    elif (activate_function == 1): #Hyperbolic Tangent Activation
215.    out_act = math.tanh(in_act)
216.    elif (activate_function == 2): # Softmax Activation
217.    out_act = np.maximum(0,in_act)
218.
219.    elif (activate_function == 3): # ReLU Activation - Analytic SoftPlus Function
220.    out_act = np.maximum(0,in_act)
221.    return (out_act)
222.
223.    #----
224.    def hidden_layer_updt():
225.    # 3. Process for every HIDDEN LAYER
226.    conv_idx = 0
227.    func_idx = 0
228.    for ff_hid_layer in range(tot_conv_layers):
229.    # 4. Process for every CONVOLUTION-ACTIVATION LOOP
230.    if (func_idx != 0):
231.    in_channel_cnt = len(fea_map_data[func_idx -1])
232.    else:
233.    in_channel_cnt = 1 # 1 image for first function of first layer
234.
235.    for hlu_k in range(tot_kernels):
236.    kernel_vector = wts_conv_class[conv_idx][hlu_k]
237.    kernel_2d = kernel_vector.reshape(kernel_dim,kernel_dim)
238.    conv_out = auto_convolve(conv_idx,func_idx,hlu_k,kernel_2d,in_channel_cnt)
239.    fm_in_dim, ht = conv_out.shape # assign the dimentrsion of the new feature
    map
240.    if (ff_hid_layer != tot_conv_layers -1):
241.    activate_out = hid_activation(conv_out, hid_activate_function,fm_in_dim)
242.    fm_in_dim, ht = activate_out.shape # assign the dimension of the new feature
    map
243.    else: # last hidden layer
244.    activate_out = fc_out_activation(conv_out, fc_activate_function)
245.    fea_map_data[func_idx][hlu_k] = activate_out
246.
247.    # End of Kernel Loop - each kernel process all input feature maps
248.    conv_idx += 1
249.    func_idx += 1 # Conv and Activ performed
```

```
250.  #----- End of Conv-Activation Loops per Hidden Layer
251.  if (ff_hid_layer != tot_conv_layers -1):
252.  # 5. POOLING for every hidden layer except the last layer
253.  for k in range(tot_kernels):
254.  activate_out = fea_map_data[func_idx -1][k]
255.  pool_out = pooling(activate_out,pool_function)
256.  fm_in_dim,ht = pool_out.shape
257.  fea_map_data[func_idx][k] = pool_out
258.  func_idx += 1
259.  #---End of Hidden Layer Loop
260.  return
261.
262.  #-----
263.  def fully_connected_layer_updt():
264.  fc_layer_data = [0 for j in range(tot_kernels)]
265.
266.  for k in range(tot_kernels):
267.
268.  fc_out = float(fea_map_data[tot_net_layers -1][k] \
269.  * fully_connected_wts[0][k] )
270.  fc_layer_data[k] = fc_out_activation(fc_out, fc_activate_function)
271.  #fc_layer_data[k] = fc_out
272.
273.  class_features[cls_idx] = fc_layer_data
274.  return
275.
276.  #----
277.  def class_mean_image(): # using Mean Square Deviation Algorithm
278.  prototype_img_list = []
279.  norm_factor = 1e-05 # to avoid computational overflow in runtime
280.  tot_imgs = 0
281.  for c in range(tot_class):
282.  cls_srch_pat = str(c) + search_files
283.  cls_file_list = glob.glob(cls_srch_pat)
284.
285.  cls_imgs_cnt = len(cls_file_list)
286.
287.  mean_sqr_dist = [0.0 for j in range(cls_imgs_cnt)]
288.  for i in range(cls_imgs_cnt):
289.  in_image = cv2.imread(cls_file_list[i],0)
290.  input_x = np.sum(np.sum(in_image)) * norm_factor
291.  x_diff_sum = 0.0
292.  for j in range(cls_imgs_cnt):
293.
294.  nxt_image = cv2.imread(cls_file_list[j],0)
295.
296.  input_y = np.sum(np.sum(nxt_image)) * norm_factor
297.  x_diff_sum += abs(input_x - input_y)**2 # Euclidean Distance
298.  mean_sqr_dist[i] = x_diff_sum
299.  mean_idx = np.argmin(mean_sqr_dist) # mean_sqr_dist.index(mean_val)
300.  mean_file = cls_file_list[mean_idx]
301.  prototype_img_list.append(mean_file)
302.  print 'Class',c, 'Images Processed:',cls_imgs_cnt
```

```
303.    tot_imgs += cls_imgs_cnt
304.    print ''
305.    print 'Total Images Processed:', tot_imgs
306.    print ''
307.    return(prototype_img_list)
308.
309.    #-----
310.    #--------------------------------------------------
311.    # ----- MAIN PROGRAM ------
312.    #--------------------------------------------------
313.    from scipy import signal as sg # Convolution function
314.    from time import gmtime, strftime
315.    import cv2
316.    import numpy as np
317.    import math # Activation function
318.    import glob # read file patterns from folder
319.    import os,sys
320.
321.    print 'PROTOTYPING PROCESS COMMENCED'
322.    proc_start = strftime("%Y-%m-%d %H:%M:%S", gmtime())
323.    #----- USER CONFIGURABLE PARAMETERS
324.    data_folder = 'C:\\SRINI\\CNN-DATASET\\CIFAR10\\ORIGINAL\data_b3\\'
    #Training-Images\\b2_500\\'
325.
326.    proc_folder = 'C:\\SRINI\\CNN-DATASET\\CIFAR10\\CNN-PROC-FILES\\'
327.
328.    os.chdir(data_folder)
329.
330.    search_files = "*.png"
331.    #max_imgs_to_process = 10000 # make sure this is minimum equal to total no.
    of classes - else Prototype process wiil fail
332.    tot_kernels = 16 # in every conv layer
333.    proto_wts_file = 'CNN-Proto-Weights.npy' # in weights_conv_class
334.    proto_bias_file = 'CNN-Proto-Biases.npy' # in bias_conv_class
335.    proto_fc_wts_file = 'CNN-Proto-FC-Weights.npy' # in fully_connected_wts
336.    proto_class_file = 'CNN-Proto-Class-Features.npy' # in class_features
337.
338.    # PARAMETERS BELOW have to be the same as used for Training
339.
340.    class_dict = {0:'airplane', 1:'automobile',2:'bird',3:'cat',4:'deer',5:'dog',
341.    6:'frog', 7:'horse', 8:'ship', 9:'truck'}
342.
343.    # class_keys = class_dict.keys() # keys
344.    # class_vals = class_dict.values() # only value
345.    class_keyvals = class_dict.items() # both key and value pair
346.
347.    # print 'Define Kernel and Bias settings'
348.    kernel_dim = 5 # 5 means 5x5
349.    rf_stride = kernel_dim
350.    bias_val = 0.001 # refer notes of Andrej Karpathy
351.
352.    """ this has to be CORRECTLY SYNCHRONIZED with filters_5 function"""
353.    last_manual_kernel_idx = 6
```

```
354.
355.    conv_mode = 'valid'
356.    conv_function = 1
357.    pool_function = 1
358.
359.    hid_activate_function = 1
360.    fc_activate_function = 2
361.
362.    # print 'Define Arrays and Load Files based on Task'
363.    tot_class = len(class_dict)
364.
365.    print 'Computing the Mean Image for each Class - using Mean Squared Error'
366.    image_file_list = class_mean_image()
367.
368.    #image_file_list = class_cluster_median()
369.
370.    tot_conv_layers, image_dim = hidden_layers_reqd(image_file_list) # based on
    image dimension
371.
372.    tot_net_layers = (tot_conv_layers * 2) -1 # last layers has no Pooling
373.
374.    imgs_per_class = [0 for j in range(tot_class)]
375.    fea_map_data = [[0.0 for j in range(tot_kernels)]
376.    for i in range(tot_net_layers)]
377.
378.    wts_conv_class = [[0.0 for j in range(tot_kernels)]
379.    for i in range(tot_conv_layers)]
380.    bias_conv_class = [[0.0 for j in range(tot_kernels)]
381.    for i in range(tot_conv_layers)]
382.
383.    fully_connected_wts = 0.01 * np.random.randn(1,tot_kernels)
384.    #fully_connected_wts = np.ones((1,tot_kernels))
385.
386.    class_features = [0.0 for j in range(tot_class)] # Desired Features
387.
388.    #-----------------------------------------------
389.    print 'PROCESS SAMPLE IMAGES ACROSS ALL CLASSES'
390.    src_data =[] # original input
391.    nor_image_data =[] # normalized input
392.    print 'Normalizing Image Files'
393.
394.    normalize_images(image_file_list)
395.    imgs_to_process = len(image_file_list)
396.    if imgs_to_process == 0:
397.    sys.exit("Images not available for processing")
398.    #-----
399.    cls_imgs_cnt = int(round(imgs_to_process / tot_class))
400.
401.    class_mean_idx = [0 for i in range(tot_class)]
402.
403.    img_labels = np.array([0 for i in range(imgs_to_process)])
404.
405.    initialize_kernel_bias()
```

```
406.
407.    for i in range(imgs_to_process):
408.    img_labels[i] = int(image_file_list[i][:1])
409.    #-------
410.    print ''
411.    for img_idx in range(imgs_to_process):
412.    main_image = nor_image_data[img_idx]
413.    cls_idx = int(image_file_list[img_idx][:1])
414.    print ''
415.    print 'Prototype File',img_idx, ':', image_file_list[img_idx], \
416.    'Class:',str(cls_idx)
417.    print ''
418.    hidden_layer_updt()
419.    fully_connected_layer_updt()
420.    # print "class_features", class_features[cls_idx]
421.    # END OF IMAGE LOOP
422.    proc_end = strftime("%Y-%m-%d %H:%M:%S", gmtime())
423.
424.    print ''
425.    print 'Saving Prototype Kernels and Class Features'
426.    save_data()
427.
428.    print "Processing Time excluding File Saving time"
429.    print "Started at :", proc_start
430.    print "Finished at:", proc_end
         # --- END OF PROTOTYPE PROCESS
```

A-3. CNN Training :

```
1. # -*- coding: utf-8 -*-
2. #
3. # Python 2.7 on Enthought Canopy 1.5.2 (64-bit)
4. #
5. # Enthought Canopy Library's required C:\SRINI\CNN-DATASET\CIFAR10\Training-
   Images\Initial
6. # 1. PIL - for Image Processing
7. #
8. # Code Author: Srinivasan Dasarathi(14111314), School of Computing,
   NCIC:\SRINI\CNN-DATASET\CIFAR10\Training-Images\Initial
9. #
10. # Process : TRAINING for Convolutional Neural Networks (CNN) for 2D Image
11. #
12. # Python Libraries: Numpy 1.9.2-1, Scipy 0.15.1-2, Matplotlib 1.4.3-4,
13. # PIL, NeuroLab 0.2.0-1
14. #
15. # Last Update: 31/08/2015
16. #
17. """
18. # Assumptions and Features:
19. #
20. # 1. Input Image can be Grayscale (1 channel) or Color (3 channels-RGB).
21. # But data will be processed in Grayscale
22. #
23. # 2. Input Image dimension can be any size but should be symmetric. eg.32 x 32 -
   CIFAR-10 Dataset
24. #
25. # 3. Filter/Kernel Size configured for dimension 5x5
26. #
27. # 4. No.of kernels across layers can be altered but will be the same across layers
28. #
29. # 5. No. of kernels used in the prototype can be MAXIMUM,
30. # and then can be reduced during TRAINING.
31. """
32. #----------------------------------------------------------------------
33. # ----- FUNCTION DEFINITIONS -----


34. def statistics():
35. print ' '
36. print 'STATISTICS:'
37. print '----------'


38. for i in range(tot_class):
39. cls_tot_imgs = class_correct[i] + class_incorrect[i]
40. if cls_tot_imgs != 0:
41. accuracy = (float(class_correct[i]) / cls_tot_imgs) * 100.00
42. print class_keyvals[i], "Images Correct:",str(class_correct[i]) +'/' +
   str(cls_tot_imgs), " Accuracy:", "%.2f" % accuracy, '%'
```

```
overall_accuracy = (float(sum(class_correct)) / imgs_to_process) * 100.00
print ''
print "Total Images:", imgs_to_process, " Overall Accuracy:","%.2f" %
    overall_accuracy, '%'
return()


#-------
def save_data():
os.chdir(proc_folder)
np.save(train_wts_file, wts_conv_class)
np.save(train_bias_file, bias_conv_class)
np.save(train_fc_wts_file, fully_connected_wts)
np.save(train_out_wts_file, output_wts)
np.save(train_out_bias_file, output_bias)
np.save(train_out_wts2_file, output_wts2)
np.save(train_out_bias2_file, output_bias2)
return


#-----
def hidden_layers_reqd(img_list):
src_img = cv2.imread(img_list[0], 0) # RGB to grayscale
w_dim,h_dim = src_img.shape


hid_layers_cnt = 0
image_size = w_dim
while (image_size > kernel_dim):
hid_layers_cnt += 1
image_size /= 2
return(hid_layers_cnt,w_dim)
#-----
def normalize_images(image_file):
# cv2 imread gives integer values whereas scipy returns float accuracy
in_image = cv2.imread(image_file, 0) # RGB to grayscale
#scipy_image = misc.imread(image_file, flatten=1) # RGB to grayscale


# Normalization
normalize_img = in_image - np.mean(in_image)
ni_data = normalize_img.flatten()
return(ni_data)
#-----
def auto_convolve(ac_conv_idx, ac_func_idx, ac_k, ac_kernel_2d,
    ac_in_channel_cnt):
# Process for every Window


# Loop for every Input Feature Map or Image
for loop in range(ac_in_channel_cnt):
if (ac_conv_idx == 0):
rf_2d = nor_image_data.reshape(image_dim,image_dim)
else:
fm_in_arr = fea_map_data[ac_func_idx -1][ac_k]
```

```
arr_dim = fm_in_arr.shape[0]
rf_2d = fm_in_arr.reshape(arr_dim,arr_dim)
if (conv_function == 1):
rf_conv_out = sg.convolve2d(rf_2d, ac_kernel_2d,
mode=conv_mode, boundary='fill', fillvalue=0) \
+ bias_conv_class[ac_conv_idx][ac_k]
elif (conv_function == 2): # uses Fast Fourier Transform function
rf_conv_out = sg.fftconvolve(rf_2d, ac_kernel_2d,
mode=conv_mode) \
+ bias_conv_class[ac_conv_idx][ac_k]
if loop == 0:
arr_dim = rf_conv_out.shape[0]
rf_depth_arr = np.zeros((arr_dim,arr_dim))
rf_depth_arr = np.add(rf_depth_arr,rf_conv_out)


    return (rf_depth_arr)
    #-----
    def pooling(pool_in, pool_function):
    if (pool_function == 1):


    ds_factor = 2
    ys,xs = pool_in.shape
    crarr = pool_in[:ys-(ys % int(ds_factor)),:xs-(xs % int(ds_factor))]
    pool_out = np.nanmax( np.concatenate([[crarr[i::ds_factor,j::ds_factor]
    for i in range(ds_factor)]
    for j in range(ds_factor)]), axis=0)
    return pool_out
    #-----
    def hid_activation(fm_in_act, activate_function, in_dim):
    hid_activate_out = np.zeros((in_dim, in_dim))


    if (activate_function == 0):
    for j in range(0,in_dim): # Logistic Sigmoid Activation
    for i in range(0,in_dim):
    z = fm_in_act[i,j]
    if (z < -20.0): hid_activate_out[i,j] = 0.0 # with threshold limit
    elif (z > 20.0): hid_activate_out[i,j] = 1.0
    else: hid_activate_out[i,j] = 1.0/(1.0 + np.exp(-z))


    elif (activate_function == 1): # Hyperbolic Tangent Activation
    for j in range(0,in_dim):
    for i in range(0,in_dim):
    hid_activate_out[i,j] = math.tanh(fm_in_act[i,j])
    elif (activate_function == 2): # SOFTMAX Activation and Rectified Linear Unit -
    ReLU Activation - Analytic SoftPlus Function'
    for j in range(0,in_dim):
    for i in range(0,in_dim):
    hid_activate_out[i,j] = np.maximum(0,fm_in_act[i,j])


    elif (activate_function == 3): # ReLU Activation - Analytic SoftPlus Function
```

```
130.  for j in range(0,in_dim):
131.  for i in range(0,in_dim):
132.  x = fm_in_act[i,j]


133.  hid_activate_out[i,j] = np.maximum(0,x)
134.  return (hid_activate_out)


135.  #-----
136.  def fc_out_activation(in_act, activate_function):
137.  if (activate_function == 0): # Logistic Sigmoid Activation
138.  if (in_act < -20.0): out_act = 0.0
139.  elif (in_act > 20.0): out_act = 1.0
140.  else: out_act = 1.0/(1.0 + np.exp(- in_act))
141.  elif (activate_function == 1): # Hyperbolic Tangent Activation
142.  out_act = math.tanh(in_act)
143.  elif (activate_function == 2): # SOFTMAX Activation and ReLU Activation -
    Analytic SoftPlus Function
144.  out_act = np.maximum(0,in_act)


145.  elif (activate_function == 3): # ReLU Activation - Analytic SoftPlus Function
146.  out_act = np.maximum(0,in_act)
147.  return (out_act)


148.  #------
149.  def derivative_func(x_val, deriv_function):


150.  if deriv_function == 0: # Deriv of Logistic Sigmoid Function
151.  logsig = 1.0/(1.0 + np.exp(- x_val))
152.  z_val = logsig * (1.0 - logsig)
153.  elif (deriv_function == 1): # Deriv of Hyperbolic Tangent
154.  z_val = (1.0 - x_val) * (1.0 + x_val)
155.  elif (deriv_function == 2): # Deriv of Softmax
156.  z_val = (1.0 - x_val) * x_val
157.  elif (deriv_function == 3): # Derivative of ReLU
158.  z_val = 1.0 #(known that x_val > 0.0) # 1.0/(1.0 + np.exp(- x_val))
159.  return(z_val)


160.  #----
161.  def hidden_layer_updt():
162.  # 3. Process for every HIDDEN LAYER
163.  conv_idx = 0
164.  func_idx = 0
165.  for ff_hid_layer in range(tot_conv_layers):
166.  # 4. Process for every CONVOLUTION-ACTIVATION LOOP
167.  if (func_idx != 0):
168.  in_channel_cnt = len(fea_map_data[func_idx -1])
169.  else:
170.  in_channel_cnt = 1 # 1 image for first function of first layer
```

```
171.  for hlu_k in range(tot_kernels):
172.  kernel_vector = wts_conv_class[conv_idx][hlu_k]
173.  kernel_2d = kernel_vector.reshape(kernel_dim,kernel_dim)


174.  conv_out = auto_convolve(conv_idx,func_idx,hlu_k,kernel_2d,in_channel_cnt)
175.  fm_in_dim, ht = conv_out.shape # assign the dimentrsion of the new feature
    map
176.  if (ff_hid_layer != tot_conv_layers -1):
177.  activate_out = hid_activation(conv_out, hid_activate_function,fm_in_dim)
178.  fm_in_dim, ht = activate_out.shape # assign the dimension of the new feature
    map
179.  else: # last hidden layer
180.  activate_out = fc_out_activation(conv_out, fc_activate_function)
181.  #print 'activate_out',activate_out
182.  fea_map_data[func_idx][hlu_k] = activate_out
183.  # End of Kernel Loop - each kernel process all input feature maps
184.  conv_idx += 1
185.  func_idx += 1 # Conv and Activ performed
186.  #----- End of Conv-Activation Loops per Hidden Layer
187.  if (ff_hid_layer != tot_conv_layers -1):
188.  # 5. POOLING for every hidden layer except the last layer
189.  for k in range(tot_kernels):
190.  activate_out = fea_map_data[func_idx -1][k]
191.  pool_out = pooling(activate_out,pool_function)
192.  fm_in_dim,ht = pool_out.shape
193.  fea_map_data[func_idx][k] = pool_out


194.  func_idx += 1


195.  #---End of Hidden Layer Loop
196.  return


197.  #-----
198.  def fully_connected_layer_updt():
199.  for k in range(tot_kernels):


200.  fc_out = float(fea_map_data[tot_net_layers -1][k] \
201.  fully_connected_wts[0][k] )


202.  pred_features[0][k] = fc_out_activation(fc_out, fc_activate_function)
203.  return


204.  #-----


205.  def chk_err_threshold(chk_cnt):
206.  # COMPUTE ERROR GRADIENTSC:\SRINI\CNN-DATASET\CIFAR10\Prototype-
    Images
207.  err_sum = 0.0
```

```
208.  for k in range(0,tot_kernels):
209.  expected_out = class_features[cls_idx][k]
210.  pred_out = pred_features[0][k]
211.  output_err = abs(expected_out - pred_out) # BETTER CONVERGENCE THAN
      WITH ABS
212.  err_sum += output_err


213.  if err_sum > error_threshold :
214.  high_error = 1
215.  print chk_cnt,'High Gradient Error', err_sum
216.  else:
217.  high_error = 0
218.  print chk_cnt,'Low Output Error', err_sum
219.  return(high_error)


220.  #------
221.  def backpropagate():


222.  #-----
223.  # BACK PROPAGATE
224.  for k in range(0,tot_kernels):
225.  expected_out = class_features[cls_idx][k]
226.  pred_out = pred_features[0][k]
227.  derivative = derivative_func(pred_out, fc_deriv_function)
228.  fc_grad[k] = derivative * abs(expected_out - pred_out)
229.  conv_grad = [[0.0 for i in range(tot_kernels)]
230.  for j in range(tot_conv_layers)]


231.  # 1.b. FOR THE LAST CONVOLUTIONAL LAYERS (eg.C-2 and FM-4)
232.  fc_grad_wt_sum = 0.0
233.  for k in range(0,tot_kernels):
234.  x = fc_grad[k] * fully_connected_wts[0][k]
235.  fc_grad_wt_sum += x
236.  layer = tot_conv_layers -1
237.  hid_idx = layer * 2
238.  for k in range(0,tot_kernels):
239.  hid_node = fea_map_data[hid_idx][k] # since last conv layer is 1x1 SUM is not
      applicable
240.  derivative = derivative_func(hid_node, hid_deriv_function)
241.  conv_grad[layer][k] = derivative * fc_grad_wt_sum # deriv * next layer error
242.  #----
243.  # 2.COMPUTE ERROR GRADIENT FOR THE CONVOLUTIONAL LAYERS
244.  # FROM LAST BUT ONE (eg.C-1 and FM-2) TO FIRST (eg. C-0 and FM-0)


245.  for layer in reversed(range(tot_conv_layers -1)): # its -1 since already processed
      for last Conv Layer
246.  conv_grad_wt_sum = 0.0
247.  for k in range(0,tot_kernels):
248.  x = conv_grad[layer+1][k] * sum(wts_conv_class[layer+1][k]) # of the next
      layer
```

```
249.    conv_grad_wt_sum += x
250.    hid_idx = layer * 2
251.    for k in range(0,tot_kernels):


252.    hid_node = sum(sum(fea_map_data[hid_idx][k]))


253.    derivative = derivative_func(hid_node, hid_deriv_function)
254.    conv_grad[layer][k] = derivative * conv_grad_wt_sum
255.    # 2. UPDATE WEIGHTS FOR FULLY CONVERTED LAYER. BIAS ADDITION NOT
    USED
256.    fc_wts_delta = 0.0 # previous weights
257.    inp_idx = tot_net_layers -1 # feature map of previous conv layer (input for this
    hidden layer)
258.    for k in range(0,tot_kernels):
259.    # Update Weights


260.    inp_node = fea_map_data[inp_idx][k] # last hid_node is the input
261.    wt_delta = float(learn_rate * fc_grad[k] * inp_node)
262.    fully_connected_wts[0][k] += wt_delta
263.    fully_connected_wts[0][k] += float(momentum * fc_wts_delta)
264.    fc_wts_delta = wt_delta
265.    # 3. UPDATE WEIGHTS AND BIAS FOR ALL CONV-LAYERS IN REVERSE ORDER
266.    # FROM LAST (eg.C-2) TO FIRST(eg. C-0)
267.    input_img_sum = sum(nor_image_data)
268.    for layer in reversed(range(tot_conv_layers)): # range will further deduct 1 from
    the value within backets
269.    hid_wts_delta = 0.0
270.    hid_bias_delta = 0.0
271.    inp_idx = (layer * 2) - 2 # feature map of previous conv layer (input for this
    hidden layer)
272.    if layer != 0:
273.    for k in range(0,tot_kernels): # UPDATE ALL FILTERS
274.    # Update Weights
275.    inp_node = sum(sum(fea_map_data[inp_idx][k])) # hid_node is the input


276.    wt_delta = float(learn_rate * conv_grad[layer][k] * inp_node)
277.    wts_conv_class[layer][k] += wt_delta
278.    wts_conv_class[layer][k] += float(momentum * hid_wts_delta)
279.    hid_wts_delta = wt_delta
280.    # Update Bias
281.    bias_delta = float(learn_rate * conv_grad[layer][k])
282.    bias_conv_class[layer][k] += bias_delta
283.    bias_conv_class[layer][k] += float(momentum * hid_bias_delta)
284.    hid_bias_delta = bias_delta
285.    else: # NOT TO UPDATE MANUAL FILTERS in the FIRST CONV LAYER
286.    for k in range(st_rdm_kernel_idx,tot_kernels):
287.    inp_node = input_img_sum
288.    wt_delta = float(learn_rate * conv_grad[layer][k] * inp_node)
289.    wts_conv_class[layer][k] += wt_delta
290.    wts_conv_class[layer][k] += float(momentum * hid_wts_delta)
291.    hid_wts_delta = wt_delta
```

```
292.    # Update Bias
293.    bias_delta = float(learn_rate * conv_grad[layer][k])
294.    bias_conv_class[layer][k] += bias_delta
295.    bias_conv_class[layer][k] += float(momentum * hid_bias_delta)
296.    hid_bias_delta = bias_delta
297.    return


298.    #----
299.    #-----------------------------------------------------
300.    # ----- MAIN PROGRAM ------
301.    #-----------------------------------------------------
302.    from scipy import signal as sg # Convolution function
303.    from time import gmtime, strftime
304.    #from scipy import misc
305.    import cv2
306.    import numpy as np
307.    import math # Activation function
308.    import glob # read file patterns from folder
309.    import os,sys
310.    import copy


311.    print 'TRAINING PROCESS COMMENCED'
312.    proc_start = strftime("%Y-%m-%d %H:%M:%S", gmtime())


313.    #----- USER CONFIGURABLE PARAMETERS
314.    data_folder = 'C:\\SRINI\\CNN-DATASET\\CIFAR10\\ORIGINAL\data_b3\\b100-
    train' #Training-Images' #\\b2_500\\' #Trg-Batch01\\'


315.    proc_folder = 'C:\\SRINI\\CNN-DATASET\\CIFAR10\\CNN-PROC-FILES\\'


316.    os.chdir(data_folder)


317.    kernel_to_use_for_training = 'P' # or (T)rained to use existing kernels in training
    folder
318.    search_files = "*.png" # search file pattern - str(cls_idx) + "*.png"


319.    max_imgs_to_process = 100 # make sure this is minimum equal to total no. of
    classes - else Prototype process wiil fail
320.    tot_kernels = 16
321.    output_dim = tot_kernels # no. of nodes or pixels in the final hidden layer.


322.    max_converge_loop = 10
323.    max_epochs = 2
324.    min_epochs = 1


325.    learn_rate = 1e-1 # Best value 1e-5
326.    momentum = 0.1 # Best value 0.5
```

```
error_threshold = 0.5 # denotes 99.5 % accuracy # 0.1 not reached, but 0.3 reachable in most cases

clsfr_epochs = 5
max_loss_loop = 50000
display_loop = 2000
step_size = 0.5 # Best value 1.0 # for softmax classifer training
regu_strength = 1e-10 # Best value 1e-5 # L2 Regularization strength - lambda
loss_threshold = 0.1 # denotes 99 % accuracy

#------
proto_wts_file = 'CNN-Proto-Weights.npy' # in weights_conv_class
proto_bias_file = 'CNN-Proto-Biases.npy' # in bias_conv_class
proto_fc_wts_file = 'CNN-Proto-FC-Weights.npy' # in fully_connected_wts
proto_class_file = 'CNN-Proto-Class-Features.npy' # in class_features

train_wts_file = 'CNN-Train-Weights.npy' # in weights_conv_class
train_bias_file = 'CNN-Train-Biases.npy' # in bias_conv_class
train_fc_wts_file = 'CNN-Train-FC-Weights.npy'

train_out_wts_file = 'CNN-Train-Output-Weights.npy'
train_out_bias_file = 'CNN-Train-Output-Bias.npy'

train_out_wts2_file = 'CNN-Train-Output-Weights2.npy'
train_out_bias2_file = 'CNN-Train-Output-Bias2.npy'
# PARAMETERS BELOW have to be the same as used for Training

class_dict = {0:'airplane', 1:'automobile',2:'bird',3:'cat',4:'deer',5:'dog',
6:'frog', 7:'horse', 8:'ship', 9:'truck'}

class_keys = class_dict.keys() # keys
# class_vals = class_dict.values() # only value
class_keyvals = class_dict.items() # both key and value pair
#--------------------------------------`--------------------
#----- Hidden Layer
# print 'Define Kernel and Bias settings'
kernel_dim = 5 # 5 means 5x5
rf_stride = kernel_dim

# this has to be CORRECTLY SYNCHRONIZED with filters_5 function
last_manual_kernel_idx = 6

conv_mode = 'valid'
conv_function = 1
pool_function = 1

hid_activate_function = 1
```

```
363.  hid_deriv_function = 1

364.  fc_activate_function = 2
365.  fc_deriv_function = 2

366.  #-----
367.  # print 'Define Arrays and Load Files based on Task'
368.  os.chdir(data_folder)

369.  tot_class = len(class_dict)

370.  image_file_list = glob.glob(search_files)
371.  imgs_to_process = len(image_file_list)

372.  tot_conv_layers,image_dim = hidden_layers_reqd(image_file_list) # based on
      image dimension



373.  os.chdir(proc_folder)

374.  tot_net_layers = (tot_conv_layers * 2) -1 # last layers has no Pooling
375.  fea_map_data = [[0.0 for j in range(tot_kernels)]
376.  for i in range(tot_net_layers)]
377.  pred_features = np.zeros((1,tot_kernels))
378.  class_features = np.load(proto_class_file,mmap_mode=None)

379.  output_features = np.zeros((imgs_to_process,output_dim))

380.  img_labels = np.array([0 for i in range(imgs_to_process)])

381.  output_score = [[0.0 for j in range(tot_class)]
382.  for i in range(imgs_to_process)]

383.  if kernel_to_use_for_training == 'P':

384.  wts_conv_class = np.load(proto_wts_file,mmap_mode=None)
385.  bias_conv_class = np.load(proto_bias_file,mmap_mode=None)
386.  fully_connected_wts = np.load(proto_fc_wts_file ,mmap_mode=None)

387.  hidden_nodes = tot_kernels * 2 # will be 32 neurons if kernel is 16
388.  output_wts = 0.01 * np.random.randn(output_dim,hidden_nodes)
389.  output_bias = np.zeros((1,hidden_nodes))

390.  output_wts2 = 0.01 * np.random.randn(hidden_nodes,tot_class)
391.  output_bias2 = np.zeros((1,tot_class))
```

```
392.  else: # to use the existing trained weights for further training
393.  wts_conv_class = np.load(train_wts_file,mmap_mode=None)
394.  bias_conv_class = np.load(train_bias_file,mmap_mode=None)
395.  fully_connected_wts = np.load(train_fc_wts_file,mmap_mode=None)
396.  output_wts = np.load(train_out_wts_file,mmap_mode=None)
397.  output_bias = np.load(train_out_bias_file,mmap_mode=None)


398.  output_wts2 = np.load(train_out_wts2_file,mmap_mode=None)
399.  output_bias2 = np.load(train_out_bias2_file,mmap_mode=None)


400.  #----------------------------------------------
401.  # CNN - FEED FORWARD PROCESS
402.  # 1. Process for every CLASS


403.  # print 'PROCESS IMAGES ACROSS CLASSES TILL FINAL HIDDEN LAYER'


404.  print 'Stage-1 Processing - Normalizing Images'
405.  os.chdir(data_folder)
406.  if imgs_to_process > max_imgs_to_process: # Restricting no. of images to
    process
407.  imgs_to_process = max_imgs_to_process
408.  if imgs_to_process == 0:
409.  sys.exit("Images not available for processing")


410.  st_rdm_kernel_idx = last_manual_kernel_idx +1 # TO ALTER ONLY RANDOM
    KERNELS AND NOT MANUAL


411.  for i in range(imgs_to_process):
412.  img_labels[i] = int(image_file_list[i][:1])
413.  # ****** TRAINING AND BACK-PROPAGATION


414.  print 'Stage-2 Processing - Gradient Error Convergence'


415.  for epoch in range(max_epochs):
416.  class_correct = [0 for j in range(tot_class)]
417.  class_incorrect = [0 for j in range(tot_class)]
418.  print ''
419.  print 'Epoch Count:', epoch +1
420.  #img_sequence = [i for i in range(imgs_to_process)]
421.  #np.random.shuffle(img_sequence) # Shuffle to prevent Overfitting
422.  # print "Random Sequence:",img_sequence
423.  print ' '


424.  file_cnt = 0
425.  low_grad_err = 0
426.  for rnd_idx in range(imgs_to_process):
427.  #for rnd_idx in img_sequence: # use this list of arrays as it contains normalized
    image data
```

```
428.  file_cnt += 1
429.  nor_image_data = normalize_images(image_file_list[rnd_idx])
430.  cls_idx = img_labels[rnd_idx]
431.  print ''
432.  print file_cnt,':',image_file_list[rnd_idx]
433.  verify_cnt = 0
434.  bp_done = 0
435.  for loop in range(max_converge_loop):
436.  # Forward Pass
437.  hidden_layer_updt()
438.  fully_connected_layer_updt()
439.  verify_cnt += 1
440.  #ed_err = chk_err_threshold(verify_cnt)
441.  err_sum = abs(np.sum(class_features[cls_idx]) - np.sum(pred_features[0]))
442.  #print ''
443.  #print 'expected_out', class_features[cls_idx]
444.  #print ''
445.  #print 'pred_out',pred_features[0]
446.  if err_sum > error_threshold :
447.  high_error = 1
448.  print verify_cnt,'High Gradient Error', err_sum
449.  else:
450.  high_error = 0
451.  print verify_cnt,'Low Output Error', err_sum
452.  ed_err = high_error
453.  if ed_err == 1:
454.  fc_grad = [0.0 for i in range(tot_kernels)]
455.  backpropagate()
456.  #print 'bp', pred_features
457.  bp_done = 1
458.  else:
459.  break
460.  if bp_done == 0 :
461.  low_grad_err += 1
462.  # end of gradient error loop
463.  #print ''
464.  #print 'pred_features',cls_idx,pred_features[0]
465.  #print ''
466.  output_features[rnd_idx] = copy.deepcopy(pred_features[0])
467.  # END OF IMAGE LOOP
468.  if low_grad_err == imgs_to_process:
469.  print 'Gradient Error within Threshold - Terminating Epochs'
470.  break
471.  # END OF EPOCH LOOP
472.  # END OF GRADIENT ERROR CORRECTION


473.  print 'Stage-3 Processing - Score - Cost-Loss'
474.  print ''
475.  print 'TRAIN A LINEAR CLASSIFIER - Output Score and Loss Verification'
476.  print ''


477.  for e in range(clsfr_epochs):
```

```
478.  print ''
479.  print 'CLASSIFIER EPOCH IDX:',e


480.  for loop in range(max_loss_loop +1):
481.  # evaluate class scores # ReLU activation
482.  hidden_layer = np.maximum(0,np.dot(output_features, output_wts) \
483.  + output_bias)
484.  output_score = np.dot(hidden_layer, output_wts2) + output_bias2
485.  # compute class probabilities
486.  score_exp = np.exp(output_score)
487.  score_loss_prob = score_exp / np.sum(score_exp, axis=1,keepdims=True)
488.  # compute the loss: average cross-entropy loss and regularization
489.  correct_logprobs = - np.log(score_loss_prob[range(imgs_to_process),
    img_labels])
490.  data_loss = np.sum(correct_logprobs)/imgs_to_process
491.  reg_loss1 = 0.5 * regu_strength * np.sum(np.dot(output_wts,output_wts.T))
492.  reg_loss2 = 0.5 * regu_strength * np.sum(np.dot(output_wts2,output_wts2.T))
493.  loss = data_loss + reg_loss1 + reg_loss2
494.  if loop % display_loop == 0:
495.  print "iteration %d: loss %f" % (loop, loss)
496.  if loss <= loss_threshold:
497.  print 'Exiting Cross-Entrophy Loss Computation Loop - Loss Threshold achieved'
498.  break
499.  # compute the gradient on scores
500.  dscores = copy.deepcopy(score_loss_prob)
501.  dscores[range(imgs_to_process), img_labels] -= 1
502.  dscores /= imgs_to_process
503.  dW2 = np.dot(hidden_layer.T, dscores)
504.  db2 = np.sum(dscores, axis=0, keepdims = True)
505.  dhidden = np.dot(dscores, output_wts2.T)
506.  dhidden[hidden_layer <=0] = 0
507.  dW = np.dot(output_features.T, dhidden)
508.  db = np.sum(dhidden, axis=0, keepdims = True)
509.  dW2 += regu_strength * output_wts2
510.  dW += regu_strength * output_wts
511.  # perform a parameter update
512.  output_wts2 += -step_size * dW2
513.  output_bias2 += -step_size * db2
514.  output_wts += -step_size * dW
515.  output_bias += -step_size * db
516.  # end of Score and Loss Computation loop
517.  # End of CLASSIFIER TRAINING
518.  # evaluate training set accuracy
519.  #scores = np.dot(output_features, output_wts ) + output_bias
520.  #predicted_class = np.argmax(scores, axis=1)
521.  #print predicted_class
522.  #print 'training accuracy: %.2f' % (np.mean(predicted_class == img_labels))
523.  # END OF FORWARD AND BACK-PROPAGATION
524.  print 'CHECK ACCURACY'
525.  class_correct = [0 for j in range(tot_class)]
526.  class_incorrect = [0 for j in range(tot_class)]
527.  for i in range(imgs_to_process):
```

```
528.  c_idx = int(image_file_list[i][:1])
529.  pred_cls_idx = np.argmax(output_score[i])
530.  #print ''
531.  #print i,c_idx, pred_cls_idx, output_score[i]
532.  if pred_cls_idx == c_idx :
533.  class_correct[c_idx] += 1
534.  else:
535.  class_incorrect[c_idx] += 1
536.  proc_end = strftime("%Y-%m-%d %H:%M:%S", gmtime())
537.  #statistics()
538.  print ' '
539.  print 'STATISTICS:'
540.  print '----------'
541.  for i in range(tot_class):
542.  cls_tot_imgs = class_correct[i] + class_incorrect[i]
543.  if cls_tot_imgs != 0:
544.  accuracy = (float(class_correct[i]) / cls_tot_imgs) * 100.00
545.  print class_keyvals[i], "Images Correct:",str(class_correct[i]) +'/' +
    str(cls_tot_imgs), " Accuracy:", "%.2f" % accuracy, '%'
546.  overall_accuracy = (float(sum(class_correct)) / imgs_to_process) * 100.00
547.  print ''
548.  print "Total Images:", imgs_to_process, " Overall Accuracy:","%.2f" %
    overall_accuracy, '%'
549.  print 'Saving Trained Kernels and Predicted Features'
550.  save_data()
551.  if loss <= loss_threshold:
552.  print 'Exiting Cross-Entrophy EPOCH - Loss Threshold achieved'
553.  break
554.  # End of Loss Epoch


555.  print "Processing Time excluding Statistics and File Saving time"
556.  print "Started at :", proc_start
557.  print "Finished at:", proc_end


558.  # --- END OF TRAINING PROCESS


559.  """
560.  # 11: GRAPHICAL REPRESENTATION OF CNN LAYERS
561.  import matplotlib.gridspec as gridspec
562.  import matplotlib.pyplot as plt
563.  import scipy.misc as sm


564.  os.chdir(data_folder)


565.  image_dim = 32
566.  img_idx = 0 #rnd_idx
567.  rows = 10
568.  cols = tot_net_layers


569.  org_image = sm.imread(image_file_list[img_idx])
```

```
570.  normalized_image = nor_image_data.reshape(image_dim,image_dim)

571.  gs = gridspec.GridSpec(rows,cols, hspace=0.5)

572.  # show original source image for first row only
573.  fm = plt.subplot(gs[0,0])
574.  plt.draw()
575.  fm.set_axis_off()
576.  img_title="Source"
577.  fm.set_title(img_title,fontsize=9)
578.  fm.imshow(org_image)
579.  fm = plt.subplot(gs[0,1])
580.  plt.draw()
581.  fm.set_axis_off()
582.  img_title="Normlzd"
583.  fm.set_title(img_title,fontsize=9)
584.  fm.imshow(normalized_image)


585.  # show original source image for first row only
586.  fm = plt.subplot(gs[0,0])
587.  plt.draw()
588.  fm.set_axis_off()
589.  fm.imshow(org_image)

590.  krnl_st = 0
591.  for layer_idx in range(0,cols):
592.  for krnl_idx in range(krnl_st,krnl_st+9):
593.  print layer_idx, krnl_idx
594.  fm = plt.subplot(gs[krnl_idx - krnl_st +1,layer_idx])
595.  plt.draw()
596.  fm.set_axis_off()
597.  layer_image = fea_map_data[layer_idx][krnl_idx]

598.  fm.imshow(layer_image, cmap='gray')

599.  # ----------------------------------------

600.  # 11: GRAPHICAL WEIGHTS

601.  import matplotlib.gridspec as gridspec
602.  import matplotlib.pyplot as plt

603.  image_dim = 32

604.  layer_idx = 0
605.  krnl_idx = 0
```

```
606.    #rows = len(np.array( fea_map_data[layer_idx]) )
607.    rows = 10
608.    cols = tot_conv_layers


609.    gs = gridspec.GridSpec(cols,rows+1,hspace=0.5)


610.    for layer_idx in range(0,cols):
611.    for krnl_idx in range(0,rows):
612.    fm = plt.subplot(gs[layer_idx,krnl_idx+1])
613.    plt.draw()
614.    fm.set_axis_off()
615.    img_title = str(layer_idx) + str(krnl_idx)
616.    if img_title != '0':
617.    fm.set_title(img_title,fontsize=9)
618.    layer_image = wts_conv_class[layer_idx][krnl_idx].reshape(5,5)
619.    fm.imshow(layer_image, cmap='gray')
620.    #-----EOC
"""
```

A-4. CNN Validation and Testing :

```
1.  # -*- coding: utf-8 -*-
2.  #
3.  # Python 2.7 on Enthought Canopy 1.5.2 (64-bit)
4.  #
5.  # Enthought Canopy Library's required
6.  # 1. PIL - for Image Processing
7.  #
8.  # Code Author: Srinivasan Dasarathi(14111314), School of Computing, NCI
9.  #
10. # Process : TRAINING for Convolutional Neural Networks (CNN) for 2D Image
11. #
12. # Python Libraries: Numpy 1.9.2-1, Scipy 0.15.1-2, Matplotlib 1.4.3-4,
13. # PIL, NeuroLab 0.2.0-1
14. #
15. # Last Update: 30/08/2015
16. """
17. # Assumptions and Features:
18. #
19. # 1. Input Image can be Grayscale (1 channel) or Color (3 channels-RGB).
20. # But data will be processed in Grayscale
21. #
22. # 2. Input Image dimension can be any size but should be symmetric. eg.32 x 32 -
    CIFAR-10 Datase
23. """
24. #-----
25. def hidden_layers_reqd(img_list):
26. src_img = cv2.imread(img_list[0], 0)
27. w_dim,h_dim = src_img.shape
```

```
hid_layers_cnt = 0
image_size = w_dim
while (image_size > kernel_dim):
hid_layers_cnt += 1
image_size /= 2
return(hid_layers_cnt,w_dim)


#-----
def normalize_images(image_file):


in_image = cv2.imread(image_file, 0)


# Mean Subtraction
normalize_img = in_image - np.mean(in_image)
ni_data = normalize_img.flatten()
return(ni_data)
#-----
def auto_convolve(ac_conv_idx, ac_func_idx, ac_k, ac_kernel_2d,
    ac_in_channel_cnt):
# Process for every Window


# Loop for every Input Feature Map or Image
for loop in range(ac_in_channel_cnt):
if (ac_conv_idx == 0):
rf_2d = nor_image_data.reshape(image_dim,image_dim)
else:
fm_in_arr = fea_map_data[ac_func_idx -1][ac_k]
arr_dim = fm_in_arr.shape[0]
rf_2d = fm_in_arr.reshape(arr_dim,arr_dim)


if (conv_function == 1):
rf_conv_out = sg.convolve2d(rf_2d, ac_kernel_2d,
mode=conv_mode, boundary='fill', fillvalue=0) \
+ bias_conv_class[ac_conv_idx][ac_k] #bias_val
elif (conv_function == 2): # uses Fast Fourier Transform function
rf_conv_out = sg.fftconvolve(rf_2d, ac_kernel_2d,
mode=conv_mode) \
+ bias_conv_class[ac_conv_idx][ac_k] # bias_val
if loop == 0:
arr_dim = rf_conv_out.shape[0]
rf_depth_arr = np.zeros((arr_dim,arr_dim))
rf_depth_arr = np.add(rf_depth_arr,rf_conv_out)


return (rf_depth_arr)
#-----
def pooling(pool_in, pool_function):
if (pool_function == 1):
```

```
ds_factor = 2
ys,xs = pool_in.shape
crarr = pool_in[:ys-(ys % int(ds_factor)),:xs-(xs % int(ds_factor))]
pool_out = np.nanmax( np.concatenate([[crarr[i::ds_factor,j::ds_factor]
for i in range(ds_factor)]
for j in range(ds_factor)]), axis=0)
return pool_out
#-----
def hid_activation(fm_in_act, activate_function, in_dim):
hid_activate_out = np.zeros((in_dim, in_dim))


if (activate_function == 0):
for j in range(0,in_dim): # Logistic Sigmoid Activation
for i in range(0,in_dim):
z = fm_in_act[i,j]
if (z < -20.0): hid_activate_out[i,j] = 0.0
elif (z > 20.0): hid_activate_out[i,j] = 1.0
else: hid_activate_out[i,j] = 1.0/(1.0 + np.exp(-z))


elif (activate_function == 1): # Hyperbolic Tangent Activation
for j in range(0,in_dim):
for i in range(0,in_dim):
hid_activate_out[i,j] = math.tanh(fm_in_act[i,j])
elif (activate_function == 2): # SOFTMAX Activation and Rectified Linear Unit - ReLU
    Activation - Analytic SoftPlus Function'
for j in range(0,in_dim):
for i in range(0,in_dim):
hid_activate_out[i,j] = np.maximum(0,fm_in_act[i,j])


elif (activate_function == 3): # ReLU Activation - Analytic SoftPlus Function
for j in range(0,in_dim):
for i in range(0,in_dim):
x = fm_in_act[i,j]


hid_activate_out[i,j] = np.maximum(0,x)
return (hid_activate_out)


#-----
    def fc_out_activation(in_act,activate_function):
    if (activate_function == 0): # Logistic Sigmoid Activation
    if (in_act < -20.0): out_act = 0.0
    elif (in_act > 20.0): out_act = 1.0
    else: out_act = 1.0/(1.0 + np.exp(- in_act))
    elif (activate_function == 1): #Hyperbolic Tangent Activation
    out_act = math.tanh(in_act)
    elif (activate_function == 2): # SOFTMAX Activation
    out_act = np.maximum(0,in_act)
    elif (activate_function == 3): # ReLU Activation - Analytic SoftPlus Function
    out_act = np.maximum(0,in_act)
    return (out_act)
```

```
112.  #----
113.  def hidden_layer_updt():
114.  # 3. Process for every HIDDEN LAYER
115.  conv_idx = 0
116.  func_idx = 0
117.  for ff_hid_layer in range(tot_conv_layers):
118.  # 4. Process for every CONVOLUTION-ACTIVATION LOOP
119.  if (func_idx != 0):
120.  in_channel_cnt = len(fea_map_data[func_idx -1])
121.  else:
122.  in_channel_cnt = 1 # 1 image for first function of first layer


123.  for hlu_k in range(tot_kernels):
124.  kernel_vector = wts_conv_class[conv_idx][hlu_k]
125.  kernel_2d = kernel_vector.reshape(kernel_dim,kernel_dim)


126.  conv_out = auto_convolve(conv_idx,func_idx,hlu_k,kernel_2d,in_channel_cnt)
127.  fm_in_dim, ht = conv_out.shape # assign the dimentrsion of the new feature
   map
128.  if (ff_hid_layer != tot_conv_layers -1):
129.  activate_out = hid_activation(conv_out, hid_activate_function,fm_in_dim)
130.  fm_in_dim, ht = activate_out.shape # assign the dimension of the new feature
   map
131.  else: # last hidden layer
132.  activate_out = fc_out_activation(conv_out, fc_activate_function)
133.  fea_map_data[func_idx][hlu_k] = activate_out
134.  # End of Kernel Loop - each kernel process all input feature maps
135.  conv_idx += 1
136.  func_idx += 1 # Conv and Activ performed
137.  #----- End of Conv-Activation Loops per Hidden Layer
138.  if (ff_hid_layer != tot_conv_layers -1):
139.  # 5. POOLING for every hidden layer except the last layer
140.  for pool_k in range(tot_kernels):
141.  activate_out = fea_map_data[func_idx -1][pool_k]
142.  pool_out = pooling(activate_out,pool_function)
143.  fm_in_dim,ht = pool_out.shape
144.  fea_map_data[func_idx][pool_k] = pool_out


145.  func_idx += 1
146.  #---End of Hidden Layer Loop
147.  return


148.  #-----
149.  def fully_connected_layer_updt():


150.  for k in range(tot_kernels):


151.  fc_out = float(fea_map_data[tot_net_layers -1][k] \
152.  fully_connected_wts[0][k] )
```

```
153.    test_features[0][k] = fc_out_activation(fc_out, fc_activate_function)

154.    return

155.    #----
156.    #---------------------------------------------------
157.    # ----- MAIN PROGRAM ------
158.    #---------------------------------------------------
159.    #from scipy import misc # image reading
160.    from scipy import signal as sg # Convolution function
161.    from time import gmtime, strftime
162.    import cv2
163.    import numpy as np
164.    import math # Activation function
165.    import glob # read file patterns from folder
166.    import os,sys

167.    print 'TESTING PROCESS COMMENCED'
168.    proc_start = strftime("%Y-%m-%d %H:%M:%S", gmtime())

169.    #----- USER CONFIGURABLE PARAMETERS
170.    data_folder = 'C:\\SRINI\\CNN-DATASET\\CIFAR10\\ORIGINAL\\data_b3\\b100-
    test' #Batch01\\Test\\' #Batch01-Images' #Training-Images\\Batch01\\'

171.    proc_folder = 'C:\\SRINI\\CNN-DATASET\\CIFAR10\\CNN-PROC-FILES\\'

172.    search_files = "*.png" # search file pattern - str(cls_idx) + "*.png"
173.    max_imgs_to_process = 100 # make sure this is minimum equal to total no. of
    classes - else Prototype process wiil fail
174.    tot_kernels = 16
175.    output_dim = tot_kernels

176.    train_wts_file = 'CNN-Train-Weights.npy' # in weights_conv_class
177.    train_bias_file = 'CNN-Train-Biases.npy' # in bias_conv_class
178.    train_fc_wts_file = 'CNN-Train-FC-Weights.npy'
179.    train_out_wts_file = 'CNN-Train-Output-Weights.npy'
180.    train_out_bias_file = 'CNN-Train-Output-Bias.npy'
181.    train_out_wts2_file = 'CNN-Train-Output-Weights2.npy'
182.    train_out_bias2_file = 'CNN-Train-Output-Bias2.npy'

183.    # PARAMETERS BELOW have to be the same as used for Training

184.    class_dict = {0:'airplane', 1:'automobile',2:'bird',3:'cat',4:'deer',5:'dog',
185.    6:'frog', 7:'horse', 8:'ship', 9:'truck'}

186.    class_keyvals = class_dict.items() # both key and value pair
```

```
187.  #-------------------------------------------------------
188.  #----- Hidden Layer
189.  # print 'Define Kernel and Bias settings'
190.  kernel_dim = 5 # 5 means 5x5
191.  rf_stride = kernel_dim


192.  """ this has to be CORRECTLY SYNCHRONIZED with filters_5 function"""
193.  last_manual_kernel_idx = 6


194.  conv_mode = 'valid'
195.  conv_function = 1
196.  pool_function = 1


197.  hid_activate_function = 1
198.  fc_activate_function = 2


199.  # print 'Define Arrays and Load Files based on Task'
200.  os.chdir(proc_folder)


201.  tot_class = len(class_dict)


202.  os.chdir(data_folder)
203.  image_file_list = glob.glob(search_files)
204.  imgs_to_process = len(image_file_list)
205.  tot_conv_layers,image_dim = hidden_layers_reqd(image_file_list) # based on
    image dimension
206.  tot_net_layers = (tot_conv_layers * 2) -1 # last layers has no Pooling
207.  fea_map_data = [[0.0 for j in range(tot_kernels)]
208.  for i in range(tot_net_layers)]
209.  os.chdir(proc_folder)
210.  wts_conv_class = np.load(train_wts_file,mmap_mode=None)
211.  bias_conv_class = np.load(train_bias_file,mmap_mode=None)
212.  fully_connected_wts = np.load(train_fc_wts_file,mmap_mode=None)
213.  output_wts = np.load(train_out_wts_file,mmap_mode=None)
214.  output_bias = np.load(train_out_bias_file,mmap_mode=None)
215.  output_wts2 = np.load(train_out_wts2_file,mmap_mode=None)
216.  output_bias2 = np.load(train_out_bias2_file,mmap_mode=None)
217.  test_features = np.zeros((1,tot_kernels))
218.  #-----------------------------------------------
219.  # CNN - FEED FORWARD PROCESS
220.  # 1. Process for every CLASS
221.  # print 'PROCESS IMAGES ACROSS CLASSES TILL FINAL HIDDEN LAYER'
222.  print 'Stage-1 Processing:'
223.  os.chdir(data_folder)
224.  if imgs_to_process > max_imgs_to_process: # Restricting no. of images to
    process
225.  imgs_to_process = max_imgs_to_process
226.  if imgs_to_process == 0:
227.  sys.exit("Images not available for processing")
228.  #----
```

```
229.  classified_class = [0 for j in range(tot_class)]
230.  tot_correct = 0
231.  tot_incorrect = 0
232.  print 'Stage-2 Processing'
233.  #img_sequence = [i for i in range(imgs_to_process)]
234.  #np.random.shuffle(img_sequence)
235.  #for img_idx in img_sequence:
236.  for img_idx in range(imgs_to_process):
237.  print ''
238.  print 'Image:',image_file_list[img_idx]
239.  nor_image_data = normalize_images(image_file_list[img_idx])
240.  cls_idx = int(image_file_list[img_idx][:1])
241.  # Forward Pass
242.  hidden_layer_updt()
243.  fully_connected_layer_updt()
244.  hidden_layer = np.maximum(0,np.dot(test_features, output_wts) \
245.  + output_bias)
246.  output_score = np.dot(hidden_layer, output_wts2) + output_bias2
247.  test_cls_idx = np.argmax(output_score)
248.  if test_cls_idx == cls_idx: # correctly classified
249.  classified_class[test_cls_idx] += 1
250.  tot_correct += 1
251.  else:
252.  tot_incorrect += 1
253.  print img_idx,'Image:',image_file_list[img_idx],
    'Classification:',class_keyvals[test_cls_idx]
254.  #print output_score
255.  # END OF IMAGE LOOP
256.  print 'No. of Classifications across Classes'
257.  # Print Classification Summary
258.  for c in range(tot_class):
259.  print class_keyvals[c],':', str(classified_class[c])
260.  print ''
261.  print 'Correct Classification:', tot_correct
262.  print 'Incorrect Classification:', tot_incorrect
263.  print ''
264.  proc_end = strftime("%Y-%m-%d %H:%M:%S", gmtime())
265.  print "Processing Time excluding Statistics and File Saving time"
266.  print "Started at :", proc_start
267.  print "Finished at:", proc_end
```

--- END OF TESTING PROCESS

Appendix –B

(Sample Display Outputs)

B-1. CNN Prototyping

B-2. CNN Training

```
Welcome to Canopy's interactive data-analysis environment!
 with pylab-backend set to: qt
Type '?' for more information.

In [1]: %run "C:\SRINI\RESEARCH\CLASSIFIER - SOFTMAX\CNN-Training-
3008.py"
TRAINING PROCESS COMMENCED
Stage-1 Processing - Normalizing Images
Stage-2 Processing - Gradient Error Convergence

Epoch Count: 1

1 : 0-aeroplane_s_000004.png
1 Low Output Error 0.0177706039911

2 : 0-aeroplane_s_000025.png
1 Low Output Error 0.0788742856779

3 : 0-aeroplane_s_000035.png
1 Low Output Error 0.0676869015821

4 : 0-aeroplane_s_000037.png
1 Low Output Error 0.0731254882032

5 : 0-aeroplane_s_000121.png
1 Low Output Error 0.184560112896

6 : 0-airbus_s_000055.png
1 Low Output Error 0.107304419887

7 : 0-airbus_s_000059.png
1 Low Output Error 0.13694498457

8 : 0-airbus_s_000063.png
1 Low Output Error 0.00663804674167

9 : 0-airbus_s_000075.png
1 Low Output Error 0.0626016199565

10 : 0-airbus_s_000109.png
1 Low Output Error 0.202733289912

11 : 1-automobile_s_000019.png
1 Low Output Error 0.234473670797
```

```
12 : 1-automobile_s_000025.png
1 Low Output Error 0.240699992047

13 : 1-automobile_s_000028.png
1 Low Output Error 0.219137078247

14 : 1-automobile_s_000029.png
1 Low Output Error 0.1671316032

15 : 1-automobile_s_000049.png
1 Low Output Error 0.199469622195

16 : 1-auto_s_002335.png
1 Low Output Error 0.237481169666

17 : 1-auto_s_002336.png
1 Low Output Error 0.0772537134522

18 : 1-auto_s_002356.png
1 Low Output Error 0.0145522061671

19 : 1-auto_s_002360.png
1 Low Output Error 0.268043309851

20 : 1-auto_s_002367.png
1 Low Output Error 0.241342224822

21 : 2-cassowary_s_000712.png
1 Low Output Error 0.0467944945916

22 : 2-cassowary_s_000791.png
1 Low Output Error 0.075651552045

23 : 2-cassowary_s_000796.png
1 Low Output Error 0.180520021781

24 : 2-cassowary_s_000801.png
1 Low Output Error 0.219020500766

25 : 2-cassowary_s_000820.png
1 Low Output Error 0.239022996878

26 : 2-cassowary_s_000904.png
1 Low Output Error 0.136228730735

27 : 2-cassowary_s_000952.png
1 Low Output Error 0.131355390914

28 : 2-cassowary_s_000992.png
1 Low Output Error 0.226402166201

29 : 2-cassowary_s_001017.png
1 Low Output Error 0.188661860267

30 : 2-cassowary_s_001027.png
```

```
1 Low Output Error 0.0404364460153

31 : 3-domestic_cat_s_001130.png
1 Low Output Error 0.0792008447503

32 : 3-domestic_cat_s_001146.png
1 Low Output Error 0.176822852036

33 : 3-domestic_cat_s_001156.png
1 Low Output Error 0.0355484304588

34 : 3-domestic_cat_s_001165.png
1 Low Output Error 0.0667550318985

35 : 3-domestic_cat_s_001181.png
1 Low Output Error 0.112172749605

36 : 3-domestic_cat_s_001208.png
1 Low Output Error 0.0486807952609

37 : 3-domestic_cat_s_001216.png
1 Low Output Error 0.0478471381905

38 : 3-domestic_cat_s_001224.png
1 Low Output Error 0.115278296317

39 : 3-domestic_cat_s_001257.png
1 Low Output Error 0.181185132502

40 : 3-domestic_cat_s_001265.png
1 Low Output Error 0.0815699692172

41 : 4-fawn_s_002049.png
1 Low Output Error 0.105324104526

42 : 4-fawn_s_002112.png
1 Low Output Error 0.0800090384642

43 : 4-fawn_s_002165.png
1 Low Output Error 0.0431786803424

44 : 4-fawn_s_002168.png
1 Low Output Error 0.0702613791994

45 : 4-fawn_s_002245.png
1 Low Output Error 0.0844089053194

46 : 4-greenland_caribou_s_000054.png
1 Low Output Error 0.155236754149

47 : 4-japanese_deer_s_000075.png
1 Low Output Error 0.117647859488

48 : 4-japanese_deer_s_000089.png
1 Low Output Error 0.0331963095629
```

```
49 : 4-japanese_deer_s_000226.png
1 Low Output Error 0.0826195270135

50 : 4-japanese_deer_s_000594.png
1 Low Output Error 0.159675363179

51 : 5-pekingese_s_001049.png
1 Low Output Error 0.116761574497

52 : 5-pekingese_s_001052.png
1 Low Output Error 0.0148216155199

53 : 5-pekingese_s_001058.png
1 Low Output Error 0.106688841673

54 : 5-pekingese_s_001122.png
1 Low Output Error 0.035773782056

55 : 5-pekingese_s_001141.png
1 Low Output Error 0.0417255205836

56 : 5-pekingese_s_001143.png
1 Low Output Error 0.210266990021

57 : 5-pekingese_s_001152.png
1 Low Output Error 0.00656621289806

58 : 5-pekingese_s_001153.png
1 Low Output Error 0.0386303442344

59 : 5-pekingese_s_001170.png
1 Low Output Error 0.083704216641

60 : 5-pekingese_s_001171.png
1 Low Output Error 0.0845537848237

61 : 6-alytes_obstetricans_s_000086.png
1 Low Output Error 0.0209628941986

62 : 6-alytes_obstetricans_s_000130.png
1 Low Output Error 0.058347565045

63 : 6-alytes_obstetricans_s_000167.png
1 Low Output Error 0.0499109908517

64 : 6-alytes_obstetricans_s_000184.png
1 Low Output Error 0.0348742535143

65 : 6-alytes_obstetricans_s_000253.png
1 Low Output Error 0.11509185062

66 : 6-alytes_obstetricans_s_000293.png
1 Low Output Error 0.0472338041207

67 : 6-alytes_obstetricans_s_000364.png
1 Low Output Error 0.0390034602887
```

```
68 : 6-alytes_obstetricans_s_000397.png
1 Low Output Error 0.00124094103962

69 : 6-alytes_obstetricans_s_000411.png
1 Low Output Error 0.0109405818716

70 : 6-alytes_obstetricans_s_000414.png
1 Low Output Error 0.0384631754277

71 : 7-arabian_s_001843.png
1 Low Output Error 0.0887072945129

72 : 7-arabian_s_001846.png
1 Low Output Error 0.0363495147026

73 : 7-arabian_s_001874.png
1 Low Output Error 0.0712235258595

74 : 7-arabian_s_001882.png
1 Low Output Error 0.0784936173408

75 : 7-arabian_s_001894.png
1 Low Output Error 0.0586717016842

76 : 7-arabian_s_001899.png
1 Low Output Error 0.0996529117305

77 : 7-arabian_s_001901.png
1 Low Output Error 0.0767769904314

78 : 7-arabian_s_001904.png
1 Low Output Error 0.118349870545

79 : 7-arabian_s_001910.png
1 Low Output Error 0.0846496440813

80 : 7-arabian_s_001911.png
1 Low Output Error 0.0745730715341

81 : 8-cargo_vessel_s_002295.png
1 Low Output Error 0.0130990574208

82 : 8-cattle_boat_s_000250.png
1 Low Output Error 0.0200245589276

83 : 8-cattle_boat_s_000294.png
1 Low Output Error 0.0193867126653

84 : 8-cattle_boat_s_000324.png
1 Low Output Error 0.0200044042833

85 : 8-container_ship_s_000024.png
1 Low Output Error 0.0595511661234

86 : 8-container_ship_s_000025.png
```

```
1 Low Output Error 0.0149591736871

87 : 8-container_ship_s_000039.png
1 Low Output Error 0.00722400292804

88 : 8-container_ship_s_000061.png
1 Low Output Error 0.0822682845598

89 : 8-container_ship_s_000081.png
1 Low Output Error 0.0770050448375

90 : 8-container_ship_s_000196.png
1 Low Output Error 0.10397621903

91 : 9-trucking_rig_s_000028.png
1 Low Output Error 0.0731866166241

92 : 9-trucking_rig_s_000037.png
1 Low Output Error 0.106515612012

93 : 9-trucking_rig_s_000047.png
1 Low Output Error 0.0284625466639

94 : 9-trucking_rig_s_000048.png
1 Low Output Error 0.0072782841033

95 : 9-truck_s_002018.png
1 Low Output Error 0.0710533049785

96 : 9-truck_s_002131.png
1 Low Output Error 0.0262181550784

97 : 9-truck_s_002181.png
1 Low Output Error 0.0242417979584

98 : 9-truck_s_002273.png
1 Low Output Error 0.0280755903407

99 : 9-truck_s_002298.png
1 Low Output Error 0.0700405969009

100 : 9-truck_s_002315.png
1 Low Output Error 0.117815925317
Gradient Error within Threshold - Terminating Epochs
Stage-3 Processing - Score - Cost-Loss

TRAIN A LINEAR CLASSIFIER - Output Score and Loss Verification


CLASSIFIER EPOCH IDX: 0
iteration 0: loss 2.302585
iteration 2000: loss 2.252446
iteration 4000: loss 2.166321
iteration 6000: loss 2.047556
iteration 8000: loss 1.926859
iteration 10000: loss 1.805121
```

```
iteration 12000: loss 1.726731
iteration 14000: loss 1.657883
iteration 16000: loss 1.581127
iteration 18000: loss 1.557170
iteration 20000: loss 1.469265
iteration 22000: loss 1.426903
iteration 24000: loss 1.336866
iteration 26000: loss 1.357177
iteration 28000: loss 1.200977
iteration 30000: loss 1.191415
iteration 32000: loss 1.186031
iteration 34000: loss 1.091175
iteration 36000: loss 0.955906
iteration 38000: loss 0.945873
iteration 40000: loss 0.931019
iteration 42000: loss 0.976741
iteration 44000: loss 0.929791
iteration 46000: loss 0.845219
iteration 48000: loss 0.802442
iteration 50000: loss 0.707240
CHECK ACCURACY

STATISTICS:
----------
(0, 'airplane') Images Correct: 9/10   Accuracy: 90.00 %
(1, 'automobile') Images Correct: 9/10   Accuracy: 90.00 %
(2, 'bird') Images Correct: 9/10   Accuracy: 90.00 %
(3, 'cat') Images Correct: 5/10   Accuracy: 50.00 %
(4, 'deer') Images Correct: 7/10   Accuracy: 70.00 %
(5, 'dog') Images Correct: 7/10   Accuracy: 70.00 %
(6, 'frog') Images Correct: 8/10   Accuracy: 80.00 %
(7, 'horse') Images Correct: 4/10   Accuracy: 40.00 %
(8, 'ship') Images Correct: 9/10   Accuracy: 90.00 %
(9, 'truck') Images Correct: 7/10   Accuracy: 70.00 %

Total Images: 100   Overall Accuracy: 74.00 %
Saving Trained Kernels and Predicted Features

CLASSIFIER EPOCH IDX: 1
iteration 0: loss 0.694564
iteration 2000: loss 0.628253
iteration 4000: loss 0.616346
iteration 6000: loss 0.648255
iteration 8000: loss 0.758267
iteration 10000: loss 0.533598
iteration 12000: loss 0.526769
iteration 14000: loss 0.467375
iteration 16000: loss 0.410173
iteration 18000: loss 1.100598
iteration 20000: loss 0.380011
iteration 22000: loss 0.396512
iteration 24000: loss 0.365300
iteration 26000: loss 0.329673
iteration 28000: loss 0.331363
iteration 30000: loss 0.339049
iteration 32000: loss 0.326666
```

```
iteration 34000: loss 2.219172
iteration 36000: loss 0.328618
iteration 38000: loss 0.331297
iteration 40000: loss 0.298835
iteration 42000: loss 0.272406
iteration 44000: loss 0.273549
iteration 46000: loss 0.269701
iteration 48000: loss 0.265549
iteration 50000: loss 0.273158
CHECK ACCURACY

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: 8/10   Accuracy: 80.00 %
(4, 'deer') Images Correct: 9/10   Accuracy: 90.00 %
(5, 'dog') Images Correct: 10/10   Accuracy: 100.00 %
(6, 'frog') Images Correct: 10/10   Accuracy: 100.00 %
(7, 'horse') Images Correct: 8/10   Accuracy: 80.00 %
(8, 'ship') Images Correct: 10/10   Accuracy: 100.00 %
(9, 'truck') Images Correct: 9/10   Accuracy: 90.00 %

Total Images: 100   Overall Accuracy: 93.00 %
Saving Trained Kernels and Predicted Features

CLASSIFIER EPOCH IDX: 2
iteration 0: loss 0.274401
iteration 2000: loss 0.648201
iteration 4000: loss 0.274650
iteration 6000: loss 0.255820
iteration 8000: loss 0.251451
iteration 10000: loss 3.169521
iteration 12000: loss 0.237498
iteration 14000: loss 1.636900
iteration 16000: loss 0.373675
iteration 18000: loss 0.223964
iteration 20000: loss 0.212428
iteration 22000: loss 0.203946
iteration 24000: loss 0.201282
iteration 26000: loss 0.192702
iteration 28000: loss 0.184038
iteration 30000: loss 0.184546
iteration 32000: loss 0.178884
iteration 34000: loss 0.172271
iteration 36000: loss 0.167719
iteration 38000: loss 0.167285
iteration 40000: loss 0.170452
iteration 42000: loss 0.156703
iteration 44000: loss 0.159787
iteration 46000: loss 0.153035
iteration 48000: loss 0.148384
iteration 50000: loss 0.153465
CHECK ACCURACY
```

```
STATISTICS:
----------
(0, 'airplane') Images Correct: 10/10   Accuracy: 100.00 %
(1, 'automobile') Images Correct: 10/10   Accuracy: 100.00 %
(2, 'bird') Images Correct: 10/10   Accuracy: 100.00 %
(3, 'cat') Images Correct: 9/10   Accuracy: 90.00 %
(4, 'deer') Images Correct: 9/10   Accuracy: 90.00 %
(5, 'dog') Images Correct: 10/10   Accuracy: 100.00 %
(6, 'frog') Images Correct: 8/10   Accuracy: 80.00 %
(7, 'horse') Images Correct: 10/10   Accuracy: 100.00 %
(8, 'ship') Images Correct: 10/10   Accuracy: 100.00 %
(9, 'truck') Images Correct: 9/10   Accuracy: 90.00 %

Total Images: 100   Overall Accuracy: 95.00 %
Saving Trained Kernels and Predicted Features

CLASSIFIER EPOCH IDX: 3
iteration 0: loss 0.153246
iteration 2000: loss 0.148504
iteration 4000: loss 0.140457
iteration 6000: loss 0.139625
iteration 8000: loss 0.141711
iteration 10000: loss 0.132980
iteration 12000: loss 0.128370
iteration 14000: loss 0.121708
iteration 16000: loss 0.122219
iteration 18000: loss 0.119391
iteration 20000: loss 0.120798
iteration 22000: loss 0.124661
iteration 24000: loss 0.108956
iteration 26000: loss 0.112310
iteration 28000: loss 0.105440
iteration 30000: loss 0.102582
iteration 32000: loss 0.105085
Exiting Cross-Entrophy Loss Computation Loop - Loss Threshold
achieved
CHECK ACCURACY

STATISTICS:
----------
(0, 'airplane') Images Correct: 10/10   Accuracy: 100.00 %
(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: 9/10   Accuracy: 90.00 %
(5, 'dog') Images Correct: 10/10   Accuracy: 100.00 %
(6, 'frog') Images Correct: 9/10   Accuracy: 90.00 %
(7, 'horse') Images Correct: 10/10   Accuracy: 100.00 %
(8, 'ship') Images Correct: 10/10   Accuracy: 100.00 %
(9, 'truck') Images Correct: 9/10   Accuracy: 90.00 %

Total Images: 100   Overall Accuracy: 97.00 %
Saving Trained Kernels and Predicted Features
Exiting Cross-Entrophy EPOCH - Loss Threshold achieved
Processing Time excluding Statistics and File Saving time
Started at : 2015-08-31 07:11:19
```

```
Finished at: 2015-08-31 07:12:17

In [2]:
```

B-3. CNN Validation

(Output generated when validating the same trained dataset with the testing code)

```
Welcome to Canopy's interactive data-analysis environment!
 with pylab-backend set to: qt
Type '?' for more information.

In [2]: %run "C:\SRINI\RESEARCH\CLASSIFIER - SOFTMAX\CNN-Test-
3008.py"
TESTING PROCESS COMMENCED
Stage-1 Processing:
Stage-2 Processing

Image: 0-aeroplane_s_000004.png
0 Image: 0-aeroplane_s_000004.png Classification: (0, 'airplane')

Image: 0-aeroplane_s_000025.png
1 Image: 0-aeroplane_s_000025.png Classification: (0, 'airplane')

Image: 0-aeroplane_s_000035.png
2 Image: 0-aeroplane_s_000035.png Classification: (0, 'airplane')

Image: 0-aeroplane_s_000037.png
3 Image: 0-aeroplane_s_000037.png Classification: (0, 'airplane')

Image: 0-aeroplane_s_000121.png
4 Image: 0-aeroplane_s_000121.png Classification: (0, 'airplane')

Image: 0-airbus_s_000055.png
5 Image: 0-airbus_s_000055.png Classification: (0, 'airplane')

Image: 0-airbus_s_000059.png
6 Image: 0-airbus_s_000059.png Classification: (0, 'airplane')

Image: 0-airbus_s_000063.png
7 Image: 0-airbus_s_000063.png Classification: (0, 'airplane')

Image: 0-airbus_s_000075.png
8 Image: 0-airbus_s_000075.png Classification: (0, 'airplane')

Image: 0-airbus_s_000109.png
9 Image: 0-airbus_s_000109.png Classification: (0, 'airplane')

Image: 1-automobile_s_000019.png
10 Image: 1-automobile_s_000019.png Classification: (1,
'automobile')
```

```
Image: 1-automobile_s_000025.png
11 Image: 1-automobile_s_000025.png Classification: (1,
'automobile')

Image: 1-automobile_s_000028.png
12 Image: 1-automobile_s_000028.png Classification: (1,
'automobile')

Image: 1-automobile_s_000029.png
13 Image: 1-automobile_s_000029.png Classification: (1,
'automobile')

Image: 1-automobile_s_000049.png
14 Image: 1-automobile_s_000049.png Classification: (1,
'automobile')

Image: 1-auto_s_002335.png
15 Image: 1-auto_s_002335.png Classification: (1, 'automobile')

Image: 1-auto_s_002336.png
16 Image: 1-auto_s_002336.png Classification: (1, 'automobile')

Image: 1-auto_s_002356.png
17 Image: 1-auto_s_002356.png Classification: (1, 'automobile')

Image: 1-auto_s_002360.png
18 Image: 1-auto_s_002360.png Classification: (1, 'automobile')

Image: 1-auto_s_002367.png
19 Image: 1-auto_s_002367.png Classification: (1, 'automobile')

Image: 2-cassowary_s_000712.png
20 Image: 2-cassowary_s_000712.png Classification: (2, 'bird')

Image: 2-cassowary_s_000791.png
21 Image: 2-cassowary_s_000791.png Classification: (2, 'bird')

Image: 2-cassowary_s_000796.png
22 Image: 2-cassowary_s_000796.png Classification: (2, 'bird')

Image: 2-cassowary_s_000801.png
23 Image: 2-cassowary_s_000801.png Classification: (7, 'horse')

Image: 2-cassowary_s_000820.png
24 Image: 2-cassowary_s_000820.png Classification: (2, 'bird')

Image: 2-cassowary_s_000904.png
25 Image: 2-cassowary_s_000904.png Classification: (2, 'bird')

Image: 2-cassowary_s_000952.png
26 Image: 2-cassowary_s_000952.png Classification: (2, 'bird')

Image: 2-cassowary_s_000992.png
27 Image: 2-cassowary_s_000992.png Classification: (2, 'bird')

Image: 2-cassowary_s_001017.png
```

```
28 Image: 2-cassowary_s_001017.png Classification: (2, 'bird')

Image: 2-cassowary_s_001027.png
29 Image: 2-cassowary_s_001027.png Classification: (2, 'bird')

Image: 3-domestic_cat_s_001130.png
30 Image: 3-domestic_cat_s_001130.png Classification: (3, 'cat')

Image: 3-domestic_cat_s_001146.png
31 Image: 3-domestic_cat_s_001146.png Classification: (3, 'cat')

Image: 3-domestic_cat_s_001156.png
32 Image: 3-domestic_cat_s_001156.png Classification: (3, 'cat')

Image: 3-domestic_cat_s_001165.png
33 Image: 3-domestic_cat_s_001165.png Classification: (3, 'cat')

Image: 3-domestic_cat_s_001181.png
34 Image: 3-domestic_cat_s_001181.png Classification: (3, 'cat')

Image: 3-domestic_cat_s_001208.png
35 Image: 3-domestic_cat_s_001208.png Classification: (3, 'cat')

Image: 3-domestic_cat_s_001216.png
36 Image: 3-domestic_cat_s_001216.png Classification: (3, 'cat')

Image: 3-domestic_cat_s_001224.png
37 Image: 3-domestic_cat_s_001224.png Classification: (3, 'cat')

Image: 3-domestic_cat_s_001257.png
38 Image: 3-domestic_cat_s_001257.png Classification: (6, 'frog')

Image: 3-domestic_cat_s_001265.png
39 Image: 3-domestic_cat_s_001265.png Classification: (3, 'cat')

Image: 4-fawn_s_002049.png
40 Image: 4-fawn_s_002049.png Classification: (4, 'deer')

Image: 4-fawn_s_002112.png
41 Image: 4-fawn_s_002112.png Classification: (4, 'deer')

Image: 4-fawn_s_002165.png
42 Image: 4-fawn_s_002165.png Classification: (4, 'deer')

Image: 4-fawn_s_002168.png
43 Image: 4-fawn_s_002168.png Classification: (4, 'deer')

Image: 4-fawn_s_002245.png
44 Image: 4-fawn_s_002245.png Classification: (4, 'deer')

Image: 4-greenland_caribou_s_000054.png
45 Image: 4-greenland_caribou_s_000054.png Classification: (4,
'deer')

Image: 4-japanese_deer_s_000075.png
46 Image: 4-japanese_deer_s_000075.png Classification: (4, 'deer')
```

```
Image: 4-japanese_deer_s_000089.png
47 Image: 4-japanese_deer_s_000089.png Classification: (3, 'cat')

Image: 4-japanese_deer_s_000226.png
48 Image: 4-japanese_deer_s_000226.png Classification: (4, 'deer')

Image: 4-japanese_deer_s_000594.png
49 Image: 4-japanese_deer_s_000594.png Classification: (4, 'deer')

Image: 5-pekingese_s_001049.png
50 Image: 5-pekingese_s_001049.png Classification: (5, 'dog')

Image: 5-pekingese_s_001052.png
51 Image: 5-pekingese_s_001052.png Classification: (5, 'dog')

Image: 5-pekingese_s_001058.png
52 Image: 5-pekingese_s_001058.png Classification: (5, 'dog')

Image: 5-pekingese_s_001122.png
53 Image: 5-pekingese_s_001122.png Classification: (5, 'dog')

Image: 5-pekingese_s_001141.png
54 Image: 5-pekingese_s_001141.png Classification: (5, 'dog')

Image: 5-pekingese_s_001143.png
55 Image: 5-pekingese_s_001143.png Classification: (5, 'dog')

Image: 5-pekingese_s_001152.png
56 Image: 5-pekingese_s_001152.png Classification: (5, 'dog')

Image: 5-pekingese_s_001153.png
57 Image: 5-pekingese_s_001153.png Classification: (5, 'dog')

Image: 5-pekingese_s_001170.png
58 Image: 5-pekingese_s_001170.png Classification: (5, 'dog')

Image: 5-pekingese_s_001171.png
59 Image: 5-pekingese_s_001171.png Classification: (5, 'dog')

Image: 6-alytes_obstetricans_s_000086.png
60 Image: 6-alytes_obstetricans_s_000086.png Classification: (6,
'frog')

Image: 6-alytes_obstetricans_s_000130.png
61 Image: 6-alytes_obstetricans_s_000130.png Classification: (3,
'cat')

Image: 6-alytes_obstetricans_s_000167.png
62 Image: 6-alytes_obstetricans_s_000167.png Classification: (6,
'frog')

Image: 6-alytes_obstetricans_s_000184.png
63 Image: 6-alytes_obstetricans_s_000184.png Classification: (6,
'frog')
```

```
Image: 6-alytes_obstetricans_s_000253.png
64 Image: 6-alytes_obstetricans_s_000253.png Classification: (6,
'frog')

Image: 6-alytes_obstetricans_s_000293.png
65 Image: 6-alytes_obstetricans_s_000293.png Classification: (6,
'frog')

Image: 6-alytes_obstetricans_s_000364.png
66 Image: 6-alytes_obstetricans_s_000364.png Classification: (6,
'frog')

Image: 6-alytes_obstetricans_s_000397.png
67 Image: 6-alytes_obstetricans_s_000397.png Classification: (6,
'frog')

Image: 6-alytes_obstetricans_s_000411.png
68 Image: 6-alytes_obstetricans_s_000411.png Classification: (7,
'horse')

Image: 6-alytes_obstetricans_s_000414.png
69 Image: 6-alytes_obstetricans_s_000414.png Classification: (6,
'frog')

Image: 7-arabian_s_001843.png
70 Image: 7-arabian_s_001843.png Classification: (7, 'horse')

Image: 7-arabian_s_001846.png
71 Image: 7-arabian_s_001846.png Classification: (7, 'horse')

Image: 7-arabian_s_001874.png
72 Image: 7-arabian_s_001874.png Classification: (7, 'horse')

Image: 7-arabian_s_001882.png
73 Image: 7-arabian_s_001882.png Classification: (7, 'horse')

Image: 7-arabian_s_001894.png
74 Image: 7-arabian_s_001894.png Classification: (7, 'horse')

Image: 7-arabian_s_001899.png
75 Image: 7-arabian_s_001899.png Classification: (7, 'horse')

Image: 7-arabian_s_001901.png
76 Image: 7-arabian_s_001901.png Classification: (7, 'horse')

Image: 7-arabian_s_001904.png
77 Image: 7-arabian_s_001904.png Classification: (6, 'frog')

Image: 7-arabian_s_001910.png
78 Image: 7-arabian_s_001910.png Classification: (6, 'frog')

Image: 7-arabian_s_001911.png
79 Image: 7-arabian_s_001911.png Classification: (7, 'horse')

Image: 8-cargo_vessel_s_002295.png
80 Image: 8-cargo_vessel_s_002295.png Classification: (8, 'ship')
```

```
Image: 8-cattle_boat_s_000250.png
81 Image: 8-cattle_boat_s_000250.png Classification: (8, 'ship')

Image: 8-cattle_boat_s_000294.png
82 Image: 8-cattle_boat_s_000294.png Classification: (8, 'ship')

Image: 8-cattle_boat_s_000324.png
83 Image: 8-cattle_boat_s_000324.png Classification: (8, 'ship')

Image: 8-container_ship_s_000024.png
84 Image: 8-container_ship_s_000024.png Classification: (8, 'ship')

Image: 8-container_ship_s_000025.png
85 Image: 8-container_ship_s_000025.png Classification: (8, 'ship')

Image: 8-container_ship_s_000039.png
86 Image: 8-container_ship_s_000039.png Classification: (8, 'ship')

Image: 8-container_ship_s_000061.png
87 Image: 8-container_ship_s_000061.png Classification: (8, 'ship')

Image: 8-container_ship_s_000081.png
88 Image: 8-container_ship_s_000081.png Classification: (8, 'ship')

Image: 8-container_ship_s_000196.png
89 Image: 8-container_ship_s_000196.png Classification: (8, 'ship')

Image: 9-trucking_rig_s_000028.png
90 Image: 9-trucking_rig_s_000028.png Classification: (9, 'truck')

Image: 9-trucking_rig_s_000037.png
91 Image: 9-trucking_rig_s_000037.png Classification: (3, 'cat')

Image: 9-trucking_rig_s_000047.png
92 Image: 9-trucking_rig_s_000047.png Classification: (9, 'truck')

Image: 9-trucking_rig_s_000048.png
93 Image: 9-trucking_rig_s_000048.png Classification: (9, 'truck')

Image: 9-truck_s_002018.png
94 Image: 9-truck_s_002018.png Classification: (9, 'truck')

Image: 9-truck_s_002131.png
95 Image: 9-truck_s_002131.png Classification: (9, 'truck')

Image: 9-truck_s_002181.png
96 Image: 9-truck_s_002181.png Classification: (9, 'truck')

Image: 9-truck_s_002273.png
97 Image: 9-truck_s_002273.png Classification: (9, 'truck')

Image: 9-truck_s_002298.png
98 Image: 9-truck_s_002298.png Classification: (9, 'truck')

Image: 9-truck_s_002315.png
```

```
99 Image: 9-truck_s_002315.png Classification: (9, 'truck')
No. of Classifications across Classes
(0, 'airplane') : 10
(1, 'automobile') : 10
(2, 'bird') : 9
(3, 'cat') : 9
(4, 'deer') : 9
(5, 'dog') : 10
(6, 'frog') : 8
(7, 'horse') : 8
(8, 'ship') : 10
(9, 'truck') : 9

Correct Classification: 92
Incorrect Classification: 8

Processing Time excluding Statistics and File Saving time
Started at : 2015-08-31 06:08:53
Finished at: 2015-08-31 06:08:55

In [3]:
```

B-4. CNN Testing

(Output generated when testing a new dataset with the testing code)

```
Welcome to Canopy's interactive data-analysis environment!
 with pylab-backend set to: qt
Type '?' for more information.

In [2]: %run "C:\SRINI\RESEARCH\CLASSIFIER - SOFTMAX\CNN-Test-
3008.py"
TESTING PROCESS COMMENCED
Stage-1 Processing:
Stage-2 Processing

Image: 0-aeroplane_s_000004.png
0 Image: 0-aeroplane_s_000004.png Classification: (0, 'airplane')

Image: 0-aeroplane_s_000025.png
1 Image: 0-aeroplane_s_000025.png Classification: (0, 'airplane')

Image: 0-aeroplane_s_000035.png
2 Image: 0-aeroplane_s_000035.png Classification: (0, 'airplane')

Image: 0-aeroplane_s_000037.png
3 Image: 0-aeroplane_s_000037.png Classification: (0, 'airplane')

Image: 0-aeroplane_s_000121.png
4 Image: 0-aeroplane_s_000121.png Classification: (0, 'airplane')

Image: 0-airbus_s_000055.png
5 Image: 0-airbus_s_000055.png Classification: (0, 'airplane')
```

```
Image: 0-airbus_s_000059.png
6 Image: 0-airbus_s_000059.png Classification: (0, 'airplane')

Image: 0-airbus_s_000063.png
7 Image: 0-airbus_s_000063.png Classification: (0, 'airplane')

Image: 0-airbus_s_000075.png
8 Image: 0-airbus_s_000075.png Classification: (0, 'airplane')

Image: 0-airbus_s_000109.png
9 Image: 0-airbus_s_000109.png Classification: (0, 'airplane')

Image: 1-automobile_s_000019.png
10 Image: 1-automobile_s_000019.png Classification: (1,
'automobile')

Image: 1-automobile_s_000025.png
11 Image: 1-automobile_s_000025.png Classification: (1,
'automobile')

Image: 1-automobile_s_000028.png
12 Image: 1-automobile_s_000028.png Classification: (1,
'automobile')

Image: 1-automobile_s_000029.png
13 Image: 1-automobile_s_000029.png Classification: (1,
'automobile')

Image: 1-automobile_s_000049.png
14 Image: 1-automobile_s_000049.png Classification: (1,
'automobile')

Image: 1-auto_s_002335.png
15 Image: 1-auto_s_002335.png Classification: (1, 'automobile')

Image: 1-auto_s_002336.png
16 Image: 1-auto_s_002336.png Classification: (1, 'automobile')

Image: 1-auto_s_002356.png
17 Image: 1-auto_s_002356.png Classification: (1, 'automobile')

Image: 1-auto_s_002360.png
18 Image: 1-auto_s_002360.png Classification: (1, 'automobile')

Image: 1-auto_s_002367.png
19 Image: 1-auto_s_002367.png Classification: (1, 'automobile')

Image: 2-cassowary_s_000712.png
20 Image: 2-cassowary_s_000712.png Classification: (2, 'bird')

Image: 2-cassowary_s_000791.png
21 Image: 2-cassowary_s_000791.png Classification: (2, 'bird')

Image: 2-cassowary_s_000796.png
22 Image: 2-cassowary_s_000796.png Classification: (2, 'bird')
```

```
Image: 2-cassowary_s_000801.png
23 Image: 2-cassowary_s_000801.png Classification: (7, 'horse')

Image: 2-cassowary_s_000820.png
24 Image: 2-cassowary_s_000820.png Classification: (2, 'bird')

Image: 2-cassowary_s_000904.png
25 Image: 2-cassowary_s_000904.png Classification: (2, 'bird')

Image: 2-cassowary_s_000952.png
26 Image: 2-cassowary_s_000952.png Classification: (2, 'bird')

Image: 2-cassowary_s_000992.png
27 Image: 2-cassowary_s_000992.png Classification: (2, 'bird')

Image: 2-cassowary_s_001017.png
28 Image: 2-cassowary_s_001017.png Classification: (2, 'bird')

Image: 2-cassowary_s_001027.png
29 Image: 2-cassowary_s_001027.png Classification: (2, 'bird')

Image: 3-domestic_cat_s_001130.png
30 Image: 3-domestic_cat_s_001130.png Classification: (3, 'cat')

Image: 3-domestic_cat_s_001146.png
31 Image: 3-domestic_cat_s_001146.png Classification: (3, 'cat')

Image: 3-domestic_cat_s_001156.png
32 Image: 3-domestic_cat_s_001156.png Classification: (3, 'cat')

Image: 3-domestic_cat_s_001165.png
33 Image: 3-domestic_cat_s_001165.png Classification: (3, 'cat')

Image: 3-domestic_cat_s_001181.png
34 Image: 3-domestic_cat_s_001181.png Classification: (3, 'cat')

Image: 3-domestic_cat_s_001208.png
35 Image: 3-domestic_cat_s_001208.png Classification: (3, 'cat')

Image: 3-domestic_cat_s_001216.png
36 Image: 3-domestic_cat_s_001216.png Classification: (3, 'cat')

Image: 3-domestic_cat_s_001224.png
37 Image: 3-domestic_cat_s_001224.png Classification: (3, 'cat')

Image: 3-domestic_cat_s_001257.png
38 Image: 3-domestic_cat_s_001257.png Classification: (6, 'frog')

Image: 3-domestic_cat_s_001265.png
39 Image: 3-domestic_cat_s_001265.png Classification: (3, 'cat')

Image: 4-fawn_s_002049.png
40 Image: 4-fawn_s_002049.png Classification: (4, 'deer')

Image: 4-fawn_s_002112.png
41 Image: 4-fawn_s_002112.png Classification: (4, 'deer')
```

```
Image: 4-fawn_s_002165.png
42 Image: 4-fawn_s_002165.png Classification: (4, 'deer')

Image: 4-fawn_s_002168.png
43 Image: 4-fawn_s_002168.png Classification: (4, 'deer')

Image: 4-fawn_s_002245.png
44 Image: 4-fawn_s_002245.png Classification: (4, 'deer')

Image: 4-greenland_caribou_s_000054.png
45 Image: 4-greenland_caribou_s_000054.png Classification: (4,
'deer')

Image: 4-japanese_deer_s_000075.png
46 Image: 4-japanese_deer_s_000075.png Classification: (4, 'deer')

Image: 4-japanese_deer_s_000089.png
47 Image: 4-japanese_deer_s_000089.png Classification: (3, 'cat')

Image: 4-japanese_deer_s_000226.png
48 Image: 4-japanese_deer_s_000226.png Classification: (4, 'deer')

Image: 4-japanese_deer_s_000594.png
49 Image: 4-japanese_deer_s_000594.png Classification: (4, 'deer')

Image: 5-pekingese_s_001049.png
50 Image: 5-pekingese_s_001049.png Classification: (5, 'dog')

Image: 5-pekingese_s_001052.png
51 Image: 5-pekingese_s_001052.png Classification: (5, 'dog')

Image: 5-pekingese_s_001058.png
52 Image: 5-pekingese_s_001058.png Classification: (5, 'dog')

Image: 5-pekingese_s_001122.png
53 Image: 5-pekingese_s_001122.png Classification: (5, 'dog')

Image: 5-pekingese_s_001141.png
54 Image: 5-pekingese_s_001141.png Classification: (5, 'dog')

Image: 5-pekingese_s_001143.png
55 Image: 5-pekingese_s_001143.png Classification: (5, 'dog')

Image: 5-pekingese_s_001152.png
56 Image: 5-pekingese_s_001152.png Classification: (5, 'dog')

Image: 5-pekingese_s_001153.png
57 Image: 5-pekingese_s_001153.png Classification: (5, 'dog')

Image: 5-pekingese_s_001170.png
58 Image: 5-pekingese_s_001170.png Classification: (5, 'dog')

Image: 5-pekingese_s_001171.png
59 Image: 5-pekingese_s_001171.png Classification: (5, 'dog')
```

```
Image: 6-alytes_obstetricans_s_000086.png
60 Image: 6-alytes_obstetricans_s_000086.png Classification: (6,
'frog')

Image: 6-alytes_obstetricans_s_000130.png
61 Image: 6-alytes_obstetricans_s_000130.png Classification: (3,
'cat')

Image: 6-alytes_obstetricans_s_000167.png
62 Image: 6-alytes_obstetricans_s_000167.png Classification: (6,
'frog')

Image: 6-alytes_obstetricans_s_000184.png
63 Image: 6-alytes_obstetricans_s_000184.png Classification: (6,
'frog')

Image: 6-alytes_obstetricans_s_000253.png
64 Image: 6-alytes_obstetricans_s_000253.png Classification: (6,
'frog')

Image: 6-alytes_obstetricans_s_000293.png
65 Image: 6-alytes_obstetricans_s_000293.png Classification: (6,
'frog')

Image: 6-alytes_obstetricans_s_000364.png
66 Image: 6-alytes_obstetricans_s_000364.png Classification: (6,
'frog')

Image: 6-alytes_obstetricans_s_000397.png
67 Image: 6-alytes_obstetricans_s_000397.png Classification: (6,
'frog')

Image: 6-alytes_obstetricans_s_000411.png
68 Image: 6-alytes_obstetricans_s_000411.png Classification: (7,
'horse')

Image: 6-alytes_obstetricans_s_000414.png
69 Image: 6-alytes_obstetricans_s_000414.png Classification: (6,
'frog')

Image: 7-arabian_s_001843.png
70 Image: 7-arabian_s_001843.png Classification: (7, 'horse')

Image: 7-arabian_s_001846.png
71 Image: 7-arabian_s_001846.png Classification: (7, 'horse')

Image: 7-arabian_s_001874.png
72 Image: 7-arabian_s_001874.png Classification: (7, 'horse')

Image: 7-arabian_s_001882.png
73 Image: 7-arabian_s_001882.png Classification: (7, 'horse')

Image: 7-arabian_s_001894.png
74 Image: 7-arabian_s_001894.png Classification: (7, 'horse')

Image: 7-arabian_s_001899.png
```

```
75 Image: 7-arabian_s_001899.png Classification: (7, 'horse')

Image: 7-arabian_s_001901.png
76 Image: 7-arabian_s_001901.png Classification: (7, 'horse')

Image: 7-arabian_s_001904.png
77 Image: 7-arabian_s_001904.png Classification: (6, 'frog')

Image: 7-arabian_s_001910.png
78 Image: 7-arabian_s_001910.png Classification: (6, 'frog')

Image: 7-arabian_s_001911.png
79 Image: 7-arabian_s_001911.png Classification: (7, 'horse')

Image: 8-cargo_vessel_s_002295.png
80 Image: 8-cargo_vessel_s_002295.png Classification: (8, 'ship')

Image: 8-cattle_boat_s_000250.png
81 Image: 8-cattle_boat_s_000250.png Classification: (8, 'ship')

Image: 8-cattle_boat_s_000294.png
82 Image: 8-cattle_boat_s_000294.png Classification: (8, 'ship')

Image: 8-cattle_boat_s_000324.png
83 Image: 8-cattle_boat_s_000324.png Classification: (8, 'ship')

Image: 8-container_ship_s_000024.png
84 Image: 8-container_ship_s_000024.png Classification: (8, 'ship')

Image: 8-container_ship_s_000025.png
85 Image: 8-container_ship_s_000025.png Classification: (8, 'ship')

Image: 8-container_ship_s_000039.png
86 Image: 8-container_ship_s_000039.png Classification: (8, 'ship')

Image: 8-container_ship_s_000061.png
87 Image: 8-container_ship_s_000061.png Classification: (8, 'ship')

Image: 8-container_ship_s_000081.png
88 Image: 8-container_ship_s_000081.png Classification: (8, 'ship')

Image: 8-container_ship_s_000196.png
89 Image: 8-container_ship_s_000196.png Classification: (8, 'ship')

Image: 9-trucking_rig_s_000028.png
90 Image: 9-trucking_rig_s_000028.png Classification: (9, 'truck')

Image: 9-trucking_rig_s_000037.png
91 Image: 9-trucking_rig_s_000037.png Classification: (3, 'cat')

Image: 9-trucking_rig_s_000047.png
92 Image: 9-trucking_rig_s_000047.png Classification: (9, 'truck')

Image: 9-trucking_rig_s_000048.png
93 Image: 9-trucking_rig_s_000048.png Classification: (9, 'truck')
```

```
Image: 9-truck_s_002018.png
94 Image: 9-truck_s_002018.png Classification: (9, 'truck')

Image: 9-truck_s_002131.png
95 Image: 9-truck_s_002131.png Classification: (9, 'truck')

Image: 9-truck_s_002181.png
96 Image: 9-truck_s_002181.png Classification: (9, 'truck')

Image: 9-truck_s_002273.png
97 Image: 9-truck_s_002273.png Classification: (9, 'truck')

Image: 9-truck_s_002298.png
98 Image: 9-truck_s_002298.png Classification: (9, 'truck')

Image: 9-truck_s_002315.png
99 Image: 9-truck_s_002315.png Classification: (9, 'truck')
No. of Classifications across Classes
(0, 'airplane') : 10
(1, 'automobile') : 10
(2, 'bird') : 9
(3, 'cat') : 9
(4, 'deer') : 9
(5, 'dog') : 10
(6, 'frog') : 8
(7, 'horse') : 8
(8, 'ship') : 10
(9, 'truck') : 9

Correct Classification: 92
Incorrect Classification: 8

Processing Time excluding Statistics and File Saving time
Started at : 2015-08-31 06:08:53
Finished at: 2015-08-31 06:08:55

In [3]:


 Welcome to Canopy's interactive data-analysis environment!
 with pylab-backend set to: qt
Type '?' for more information.

In [1]: %run "C:\SRINI\RESEARCH\CLASSIFIER - SOFTMAX\CNN-Test-
3008.py"
TESTING PROCESS COMMENCED
Stage-1 Processing:
Stage-2 Processing

Image: 0-airbus_s_000120.png
0 Image: 0-airbus_s_000120.png Classification: (5, 'dog')

Image: 0-airbus_s_000149.png
1 Image: 0-airbus_s_000149.png Classification: (2, 'bird')

Image: 0-airbus_s_000151.png
```

```
2 Image: 0-airbus_s_000151.png Classification: (1, 'automobile')

Image: 0-airbus_s_000158.png
3 Image: 0-airbus_s_000158.png Classification: (6, 'frog')

Image: 0-airbus_s_000226.png
4 Image: 0-airbus_s_000226.png Classification: (3, 'cat')

Image: 0-airbus_s_000228.png
5 Image: 0-airbus_s_000228.png Classification: (1, 'automobile')

Image: 0-airbus_s_000246.png
6 Image: 0-airbus_s_000246.png Classification: (0, 'airplane')

Image: 0-airbus_s_000248.png
7 Image: 0-airbus_s_000248.png Classification: (3, 'cat')

Image: 0-airbus_s_000359.png
8 Image: 0-airbus_s_000359.png Classification: (2, 'bird')

Image: 0-airbus_s_000364.png
9 Image: 0-airbus_s_000364.png Classification: (6, 'frog')

Image: 1-cab_s_000619.png
10 Image: 1-cab_s_000619.png Classification: (3, 'cat')

Image: 1-cab_s_000626.png
11 Image: 1-cab_s_000626.png Classification: (4, 'deer')

Image: 1-cab_s_000642.png
12 Image: 1-cab_s_000642.png Classification: (5, 'dog')

Image: 1-cab_s_000882.png
13 Image: 1-cab_s_000882.png Classification: (0, 'airplane')

Image: 1-cab_s_001013.png
14 Image: 1-cab_s_001013.png Classification: (0, 'airplane')

Image: 1-cab_s_001229.png
15 Image: 1-cab_s_001229.png Classification: (8, 'ship')

Image: 1-car_s_000025.png
16 Image: 1-car_s_000025.png Classification: (2, 'bird')

Image: 1-car_s_000027.png
17 Image: 1-car_s_000027.png Classification: (3, 'cat')

Image: 1-car_s_000029.png
18 Image: 1-car_s_000029.png Classification: (9, 'truck')

Image: 1-car_s_000032.png
19 Image: 1-car_s_000032.png Classification: (5, 'dog')

Image: 2-pipit_s_001669.png
20 Image: 2-pipit_s_001669.png Classification: (1, 'automobile')
```

```
Image: 2-pipit_s_001727.png
21 Image: 2-pipit_s_001727.png Classification: (3, 'cat')

Image: 2-pipit_s_001747.png
22 Image: 2-pipit_s_001747.png Classification: (3, 'cat')

Image: 2-pipit_s_001844.png
23 Image: 2-pipit_s_001844.png Classification: (5, 'dog')

Image: 2-pipit_s_001885.png
24 Image: 2-pipit_s_001885.png Classification: (9, 'truck')

Image: 2-prunella_modularis_s_000003.png
25 Image: 2-prunella_modularis_s_000003.png Classification: (1,
'automobile')

Image: 2-prunella_modularis_s_000065.png
26 Image: 2-prunella_modularis_s_000065.png Classification: (4,
'deer')

Image: 2-prunella_modularis_s_000075.png
27 Image: 2-prunella_modularis_s_000075.png Classification: (9,
'truck')

Image: 2-prunella_modularis_s_000157.png
28 Image: 2-prunella_modularis_s_000157.png Classification: (5,
'dog')

Image: 2-prunella_modularis_s_000161.png
29 Image: 2-prunella_modularis_s_000161.png Classification: (9,
'truck')

Image: 3-cat_s_002262.png
30 Image: 3-cat_s_002262.png Classification: (7, 'horse')

Image: 3-cat_s_002308.png
31 Image: 3-cat_s_002308.png Classification: (8, 'ship')

Image: 3-cat_s_002311.png
32 Image: 3-cat_s_002311.png Classification: (5, 'dog')

Image: 3-cat_s_002313.png
33 Image: 3-cat_s_002313.png Classification: (3, 'cat')

Image: 3-cat_s_002320.png
34 Image: 3-cat_s_002320.png Classification: (3, 'cat')

Image: 3-cat_s_002360.png
35 Image: 3-cat_s_002360.png Classification: (7, 'horse')

Image: 3-cat_s_002379.png
36 Image: 3-cat_s_002379.png Classification: (7, 'horse')

Image: 3-cat_s_002389.png
37 Image: 3-cat_s_002389.png Classification: (8, 'ship')
```

```
Image: 3-cat_s_002391.png
38 Image: 3-cat_s_002391.png Classification: (8, 'ship')

Image: 3-cat_s_002406.png
39 Image: 3-cat_s_002406.png Classification: (1, 'automobile')

Image: 4-cervus_elaphus_s_000666.png
40 Image: 4-cervus_elaphus_s_000666.png Classification: (8, 'ship')

Image: 4-cervus_elaphus_s_000682.png
41 Image: 4-cervus_elaphus_s_000682.png Classification: (7, 'horse')

Image: 4-cervus_elaphus_s_000881.png
42 Image: 4-cervus_elaphus_s_000881.png Classification: (0,
'airplane')

Image: 4-cervus_elaphus_s_000887.png
43 Image: 4-cervus_elaphus_s_000887.png Classification: (1,
'automobile')

Image: 4-cervus_elaphus_s_000895.png
44 Image: 4-cervus_elaphus_s_000895.png Classification: (6, 'frog')

Image: 4-cervus_elaphus_s_000904.png
45 Image: 4-cervus_elaphus_s_000904.png Classification: (3, 'cat')

Image: 4-cervus_elaphus_s_000929.png
46 Image: 4-cervus_elaphus_s_000929.png Classification: (4, 'deer')

Image: 4-cervus_elaphus_s_000950.png
47 Image: 4-cervus_elaphus_s_000950.png Classification: (9, 'truck')

Image: 4-cervus_elaphus_s_000974.png
48 Image: 4-cervus_elaphus_s_000974.png Classification: (3, 'cat')

Image: 4-cervus_elaphus_s_001037.png
49 Image: 4-cervus_elaphus_s_001037.png Classification: (0,
'airplane')

Image: 5-feist_s_000469.png
50 Image: 5-feist_s_000469.png Classification: (1, 'automobile')

Image: 5-feist_s_000475.png
51 Image: 5-feist_s_000475.png Classification: (8, 'ship')

Image: 5-feist_s_000593.png
52 Image: 5-feist_s_000593.png Classification: (5, 'dog')

Image: 5-feist_s_000684.png
53 Image: 5-feist_s_000684.png Classification: (7, 'horse')

Image: 5-japanese_spaniel_s_000001.png
54 Image: 5-japanese_spaniel_s_000001.png Classification: (0,
'airplane')

Image: 5-japanese_spaniel_s_000009.png
```

```
55 Image: 5-japanese_spaniel_s_000009.png Classification: (4,
'deer')

Image: 5-japanese_spaniel_s_000018.png
56 Image: 5-japanese_spaniel_s_000018.png Classification: (0,
'airplane')

Image: 5-japanese_spaniel_s_000063.png
57 Image: 5-japanese_spaniel_s_000063.png Classification: (8,
'ship')

Image: 5-japanese_spaniel_s_000106.png
58 Image: 5-japanese_spaniel_s_000106.png Classification: (7,
'horse')

Image: 5-japanese_spaniel_s_000165.png
59 Image: 5-japanese_spaniel_s_000165.png Classification: (9,
'truck')

Image: 6-bufo_speciosus_s_000027.png
60 Image: 6-bufo_speciosus_s_000027.png Classification: (6, 'frog')

Image: 6-bufo_speciosus_s_000034.png
61 Image: 6-bufo_speciosus_s_000034.png Classification: (8, 'ship')

Image: 6-bufo_speciosus_s_000037.png
62 Image: 6-bufo_speciosus_s_000037.png Classification: (0,
'airplane')

Image: 6-bufo_speciosus_s_000081.png
63 Image: 6-bufo_speciosus_s_000081.png Classification: (4, 'deer')

Image: 6-bufo_speciosus_s_000097.png
64 Image: 6-bufo_speciosus_s_000097.png Classification: (8, 'ship')

Image: 6-bufo_speciosus_s_000121.png
65 Image: 6-bufo_speciosus_s_000121.png Classification: (7, 'horse')

Image: 6-bufo_s_001873.png
66 Image: 6-bufo_s_001873.png Classification: (7, 'horse')

Image: 6-bufo_s_001876.png
67 Image: 6-bufo_s_001876.png Classification: (4, 'deer')

Image: 6-bufo_s_001962.png
68 Image: 6-bufo_s_001962.png Classification: (8, 'ship')

Image: 6-bufo_s_001992.png
69 Image: 6-bufo_s_001992.png Classification: (7, 'horse')

Image: 7-walking_horse_s_001565.png
70 Image: 7-walking_horse_s_001565.png Classification: (8, 'ship')

Image: 7-walking_horse_s_001581.png
71 Image: 7-walking_horse_s_001581.png Classification: (7, 'horse')
```

```
Image: 7-walking_horse_s_001593.png
72 Image: 7-walking_horse_s_001593.png Classification: (0,
'airplane')

Image: 7-walking_horse_s_001806.png
73 Image: 7-walking_horse_s_001806.png Classification: (3, 'cat')

Image: 7-walking_horse_s_001963.png
74 Image: 7-walking_horse_s_001963.png Classification: (2, 'bird')

Image: 7-walking_horse_s_001986.png
75 Image: 7-walking_horse_s_001986.png Classification: (2, 'bird')

Image: 7-walking_horse_s_002028.png
76 Image: 7-walking_horse_s_002028.png Classification: (3, 'cat')

Image: 7-walking_horse_s_002125.png
77 Image: 7-walking_horse_s_002125.png Classification: (8, 'ship')

Image: 7-walking_horse_s_002129.png
78 Image: 7-walking_horse_s_002129.png Classification: (3, 'cat')

Image: 7-warhorse_s_000031.png
79 Image: 7-warhorse_s_000031.png Classification: (5, 'dog')

Image: 8-abandoned_ship_s_000034.png
80 Image: 8-abandoned_ship_s_000034.png Classification: (3, 'cat')

Image: 8-abandoned_ship_s_001058.png
81 Image: 8-abandoned_ship_s_001058.png Classification: (0,
'airplane')

Image: 8-abandoned_ship_s_001170.png
82 Image: 8-abandoned_ship_s_001170.png Classification: (3, 'cat')

Image: 8-banana_boat_s_000016.png
83 Image: 8-banana_boat_s_000016.png Classification: (3, 'cat')

Image: 8-banana_boat_s_000145.png
84 Image: 8-banana_boat_s_000145.png Classification: (7, 'horse')

Image: 8-banana_boat_s_000265.png
85 Image: 8-banana_boat_s_000265.png Classification: (2, 'bird')

Image: 8-banana_boat_s_000269.png
86 Image: 8-banana_boat_s_000269.png Classification: (0, 'airplane')

Image: 8-banana_boat_s_000271.png
87 Image: 8-banana_boat_s_000271.png Classification: (9, 'truck')

Image: 8-banana_boat_s_000609.png
88 Image: 8-banana_boat_s_000609.png Classification: (5, 'dog')

Image: 8-banana_boat_s_000731.png
89 Image: 8-banana_boat_s_000731.png Classification: (7, 'horse')
```

```
Image: 9-tipper_lorry_s_000634.png
90 Image: 9-tipper_lorry_s_000634.png Classification: (9, 'truck')

Image: 9-tipper_lorry_s_000659.png
91 Image: 9-tipper_lorry_s_000659.png Classification: (0,
'airplane')

Image: 9-tipper_lorry_s_000687.png
92 Image: 9-tipper_lorry_s_000687.png Classification: (7, 'horse')

Image: 9-tipper_lorry_s_000703.png
93 Image: 9-tipper_lorry_s_000703.png Classification: (8, 'ship')

Image: 9-tipper_s_000081.png
94 Image: 9-tipper_s_000081.png Classification: (2, 'bird')

Image: 9-tipper_s_000106.png
95 Image: 9-tipper_s_000106.png Classification: (2, 'bird')

Image: 9-tipper_s_000116.png
96 Image: 9-tipper_s_000116.png Classification: (8, 'ship')

Image: 9-tipper_s_000237.png
97 Image: 9-tipper_s_000237.png Classification: (8, 'ship')

Image: 9-tipper_s_000288.png
98 Image: 9-tipper_s_000288.png Classification: (5, 'dog')

Image: 9-tipper_s_000291.png
99 Image: 9-tipper_s_000291.png Classification: (2, 'bird')
No. of Classifications across Classes
(0, 'airplane') : 1
(1, 'automobile') : 0
(2, 'bird') : 0
(3, 'cat') : 2
(4, 'deer') : 1
(5, 'dog') : 1
(6, 'frog') : 1
(7, 'horse') : 1
(8, 'ship') : 0
(9, 'truck') : 1

Correct Classification: 8
Incorrect Classification: 92

Processing Time excluding Statistics and File Saving time
Started at : 2015-08-31 06:10:12
Finished at: 2015-08-31 06:10:14

In [2]:

======================================================================
```