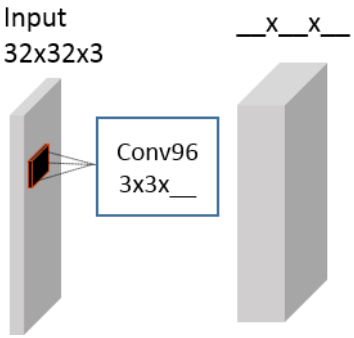
ROCHESTER INSTITUTE OF TECHNOLOGY
DEPARTMENT OF COMPUTER ENGINEERING
CMPE 677 Machine Intelligence
HW #8

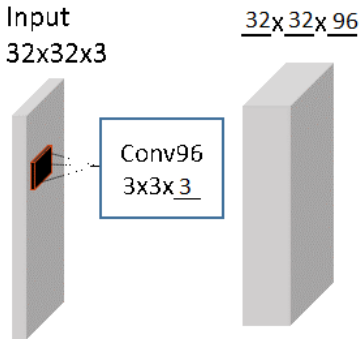
Name: ___Ram Longman_____

Due Fri, Dec 9th, 11:55pm. Submit via Dropbox. Work alone. All questions pertain to Matlab/Octave.

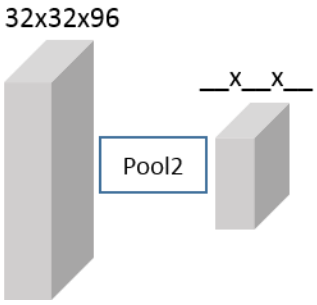
1. (4 pts) Given below is the input and output to a convolutional layer of a convolutional neural network. The input to this particular layer is a RGB image of dimensions 32×32×3. The layer has 96 filters each of spatial extent 3×3. The filters have a stride of 1 and padding of size 1 to the left, right, top, and bottom. Fill in the blanks in the below figure.



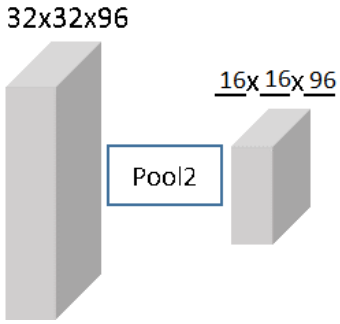
Ans:



2. (3 pts) Given below is the input and output to a pooling layer of a convolutional neural network. The input to this particular layer has dimensions 32×32×96. The pooling layer has filters of size 2×2 and stride of 2. Determine the dimensions of the output of this pooling layer.



Ans:



3. We will now explore and learn MatConvNet (A Matlab toolbox for implementing Convolutional Neural Networks for computer vision applications) for coding a CNN architecture. Installing a deep learning library is a very crucial step and can be very tricky in some cases. In this exercise, we will learn how to install MatConvNets and run some examples.

   Step-1 Follow the instructions below to download and install (set up) the MatConvNet environment.

   a. Download the latest version of MatConvNet from the following link- http://www.vlfeat.org/matconvnet/download/matconvnet-1.0-beta13.tar.gz
   b. Unpack the library in a directory of your choice.
   c. Start Matlab and cd to the unpacked MatConvNet directory.
   d. Run the following commands-
      >> addpath matlab
      >> vl_compilenn
      >> vl_setupnn
      NOTE: Depending upon your compiler you may need to comment out line 324 in the file 'vl_compilenn.m' as the –cxx flag may not get recognized by the visual studio compiler.
      Optional- For more details read the following link: http://www.vlfeat.org/matconvnet/

   Step-2a (6 points) In this part we will introduce MatConvnNets while using the same MNIST dataset from the previous homework. We have created a simple CNN architecture in the file 'mnist_cnn_initialize_sample.m' and the corresponding training file in 'mnist_cnn_cmpe677hw8.m'. Read both files and make sure you understand various terms used and answer the following questions.

   a. How many activation layers are present in this network?
   b. How many Pooling layers are present in this network?
   c. What are the filter dimensions in the first Convolutional layer?
   d. How many filters are present in the second Convolutional layer?
   e. What is the batch size we are using in this model?
   f. What is the learning rate value used in this model?

   Download the data 'ex4data1.mat' from the Dropbox. Make a directory named "MIL677" in the MatConvNet root folder. We will keep all files and data for this homework in this directory. Copy the downloaded data and files in this folder. This dataset contains 5000 samples from MNIST digit

database where each sample is a grayscale image of dimensions 20×20. Execute the following command to get the error values.
>> addpath <MIL677path>\MIL677

>> [net, info] = mnist_cnn_cmpe677hw8(mnist_cnn_initialize_sample);
Note: You will have to modify path in line 6. You should get values of top5 training error close to 0.424 and top5 validation error close to 0.344. (You would get these values after you run the given code successfully).

Ans:

a) There are 3 activation layers in the network.
b) There are 2 pooling layers in the network.
c) The dimensions of the filter in the first convolutional layer are 5x5x1.
d) There are 32 filters in the second convolutional layer.
e) The batch size in this model is 10.
f) The learning rate value used in this model is 0.001.

Step-2b (10 pts) Copy 'mnist_cnn_initialize_sample.m' to 'mnist_cnn_initialize.m'. Modify 'mnist_cnn_initialize.m' such that the second convolutional layer has 64 filters and the last convolutional layer has 256 filters. DO NOT make any other modifications to the architecture to obtain the correct error values. Note that you will have to modify the function name in line 1. Run your modified architecture by executing the following command-

>> [net, info] = mnist_cnn_cmpe677hw8(mnist_cnn_initialize);

What are the top1 training and validation errors? Show your code.

Ans: Top1 training error = 0.8422

Top1 validation error = 0.7590

I changed the weights of the second convolutional layer to this:

```
'weights', {{0.05*randn(5,5,32,64, 'single'), zeros(1,64,'single')}}, ...
```

I changed the weights of the third convolutional layer (fully connected layer) to this:

```
'weights', {{0.05*randn(5,5,32,256, 'single'), zeros(1,256,'single')}}, ...
```

Last, I changed the weights of layer 9 to be compatible with the 256 filters:

```
'weights', {{0.05*randn(1,1,256, 10, 'single'), zeros(1,10,'single')}}, ...
```

Step-3a (10 pts) In this step we will train a CNN for the CIFAR-10 dataset. After copying the Matlab files from the Dropbox into the MIL677 folder, run the script cifar_cnn_cmpe677hw8.m to train a CNN for CIFAR-10 image classification:

>> [net, info] = cifar_cnn_cmpe677hw8(cifar_cnn_initialize_sample);

This script implements an entire CNN and will also download the CIFAR-10 dataset. CIFAR-10 data consists of 60,000 RGB images of dimensions 32×32×3 for ten classes- airplane, automobile,

bird, cat, deer, dog, frog, horse, ship, truck. 50,000 of these images comprise the training set and 10,000 as testing. More information on this dataset, please refer to: http://www.cs.toronto.edu/~kriz/cifar.html.

What are the top-1 and top-5 training and validation error rates do you get after 1 epoch?

What do top-1 and top-5 errors mean?

Ans:

Validation: top-1 = 0.4282, top-5 = 0.0555

Training: top-1 = .5606, top-5 =0.1262

Top-1 error is the error for an incorrect first guess of the class (the class was incorrectly classified in the first guess). Top-5 error is the error for an incorrect first five guesses of the class (the class was incorrectly classified in the top five guesses).

Step-3b (13 pts) In this part, we will modify and train our own CNN model for classifying CIFAR-10 image data set. Copy 'cifar_cnn_initialize_sample.m' to 'cifar_cnn_initialize.m' and make the following changes and report the results.
The original network architecture in 'cifar_cnn_initialize_sample.m ' can be represented as:
conv3(32) – relu – conv3(32) – relu – pool2,2 – conv3(64) – relu – conv3(64) – relu – pool2,2 – conv8(64) – FC(10) – Softmax
where:

conv**A**(**B**) represents a convolutional layer with filter size **A×A** with **B** filters;
pool**C**,**D** represents max pooling, filter size **C×C**, stride of **D**;
FC(**E**) is a fully connected to **E** neuron layer;
relu is the rectified linear unit activation function; and
Softmax is the softmax probabilistic classification function.

Modify your 'cifar_cnn_initialize.m' such that it represents the following network architecture:
conv7(32) – relu – pool3,2 – conv5(32) – relu – pool3,2 – conv3(64) – relu – pool3,2 – conv4(128) – FC(10) – Softmax

Note: In the pooling step, you will need to use the pad command to add a zero line to the right and bottom of the image such that the output size of pool3,2 is the same as with the pool2,2.

Train this network for 1 epoch and report the top1 and top5 training and validation error rates.

Explain why you need to pad a zero line to the right and bottom of the image during the pooling step.

Ans:

Training: Top1 error = 0.5789, Top5 error = 0.1303
Validation: Top1 error = 0.4603, Top5 error = 0.0643

It is needed to pad a zero line to the right and bottom of the image during the pooling step so that during pooling there are integers and no data is lost.

4. (7 pts) The previous problem implements a mini-VGGNet (http://arxiv.org/pdf/1409.1556.pdf) architecture. In this problem, we will learn to visualize a CNN architecture as defined in MatConvNets. We will use the 'cnn_initialize_sample.m' file in the MIL677 folder. Report below the parameters either by reading the 'cnn_initialize_sample.m' file or by running the following commands from the MIL677 directory.
   >> opts.inputSize = [32,32,3,1]; % input image dimensions
   >> vl_simplenn_display(cnn_initialize_sample,opts)

   Fill in the missing values in the table (some values have been filled in for you). Note that the 'Data size' and 'Data depth' values are image size and no. of channels respectively, at a layer output.

Layer No.	0	1	2	5	6	7	10	11	12	13	14
Layer Name	input	conv	relu	pool	conv	relu	pool	conv	relu	conv	Soft-max
Filter size	n/a	3	1	**2**	**3**	1	**2**	**8**	1	1	1
Filter dim.	n/a	3	n/a	n/a	**32**	n/a	n/a	**64**	n/a	**64**	n/a
No. of filters	n/a	32	n/a	n/a	**64**	n/a	n/a	**64**	n/a	**10**	n/a
Stride	n/a	1	1	**2**	**1**	1	**2**	**2**	1	1	1
Padding	n/a	1	0	0	1	0	0	0	0	0	0
Data size	**32**	**32**	**32**	**16**	**16**	**16**	**8**	1	1	**1**	1
Data depth	**3**	**32**	**32**	**32**	**64**	**64**	**64**	64	64	**10**	1

5. (10 pts) In this problem, we will use MatConvNets to visualize the weights obtained after training a CNN. We will also learn how to use a pre-trained model. We will download a pre-trained model of a popular CNN implementation from the following link-
   http://www.vlfeat.org/matconvnet/models/imagenet-googlenet-dag.mat
   This is a network from the paper-
   http://www.cv-foundation.org/openaccess/content_cvpr_2015/papers/Szegedy_Going_Deeper_With_2015_CVPR_paper.pdf

Write a Matlab code to load `imagenet-googlenet-dag.mat` from the `matconvnet-1.0-beta13\models` directory, then display the learned 64 filters in the first convolutional layer in an 8 by 8 grid. Note, this data is stored in `params(1)`. Briefly describe what you observe (color/shape/orientation/etc.) from the filters. Also attach your code in the solution.

Can you use this trained weight matrix to run with CIFAR-10 images that are 32×32×3 in size?

Ans:

```
load 'imagenet-googlenet-dag.mat' %if it's in the path

for i=1:64
    subplot(8,8,i)
    imshow(params(1).value(:,:,:,i));
end;
```

It can be observed from the filters that the base color of all of them is black with different shades of grey in different shapes and lines inside. The grey shapes and lines have different orientations, some are diagonal, some horizontal, some vertical, some are a mix of all of them.

This trained weight matrix cannot be used to run with CIFAR-10 images because it was trained on images of size 224x224x3 and the learned weights are no compatible to the size of the images in CIFAR-10.

6. (12 pts) Dimensionality reduction is the task of representing input samples in an alternate lower dimensional space. Which statements concerning dimensionality reduction are true:

a) Principal component analysis maximizes class separation in a lower dimensional space.
b) The eigenvalues are inversely proportional to their corresponding eigenvector importance towards reconstructing the original sample.
c) The eigenvector with the largest eigenvalue is called the principal component of the data.
d) If all eigenvectors are used, one can transform back and forth between original and PCA space without any loss of information (subject to floating point precision on the machine)
e) Linear Discriminant Analysis solves for projection vectors, such that when data is projected into this alternate space, the class means are as far apart as possible and within class variances are as large as possible.
f) Linear Discriminant Analysis is essentially the same thing as Fisher Linear Discriminant Analysis.
g) Manifold learning is based upon the assumption that your data is actually a lower dimensional object residing in a higher dimensional representation.
h) Locally Linear Embedding and Isomap assumes the lower dimensional representation is linear.
i) Linear extension of graph embedding techniques are linear approximations to non-linear manifold methods.
j) Linear extension of graph embedding techniques only apply to unsupervised clustering of data.
k) Linear methods such as Locality Preserving Projections can be used to increase class separability, but at the cost of higher dimensions than techniques such as PCA.
l) Linear methods such as Grassman Manifolds can be used to increase class separability, similar to methods such as Linear Discriminant Analysis.

Ans: b, c, f, i, l.

7. (7 pts) Sparse representations represent signals as a linear combination of basis functions. Which statements concerning sparse representations are true:

a) The sparsest solution is the *l*-1 (norm-1) solution.
b) The basis functions of natural images are similar to the responses found in V1 of the mammalian brain.

c) Sparse solutions are not only an efficient representation of objects, but a discriminative representation as well.
d) Coefficient contamination is an artifact in which one trait of a sample, such as object pose, strongly influences secondary object traits.
e) K-SVD is a popular method for simultaneously solving basis functions along with sparse representations of a training set.
f) Minimum reconstruction error is a common technique for determining the class of an object when the dictionary is built using K-SVD.

Ans: b, c, e.

8. (18 pts) Dimensionality reduction is a powerful technique that not only reduces the feature dimensions, but often makes the resulting unsupervised clustering or supervised classification more accurate. The following code loads image data, normalizes and vectorizes the data, then demonstrates how to perform dimensionality reduction using unsupervised PCA and supervised LPP. At the end, the top three dimensions of the dimensionality reduced data are shown in a plot. Step through line, by line, making sure to understand each step. (Note: do not step into any of the functions that are called, such as cov.m, eig.m, construct.m, or lpp.m.)

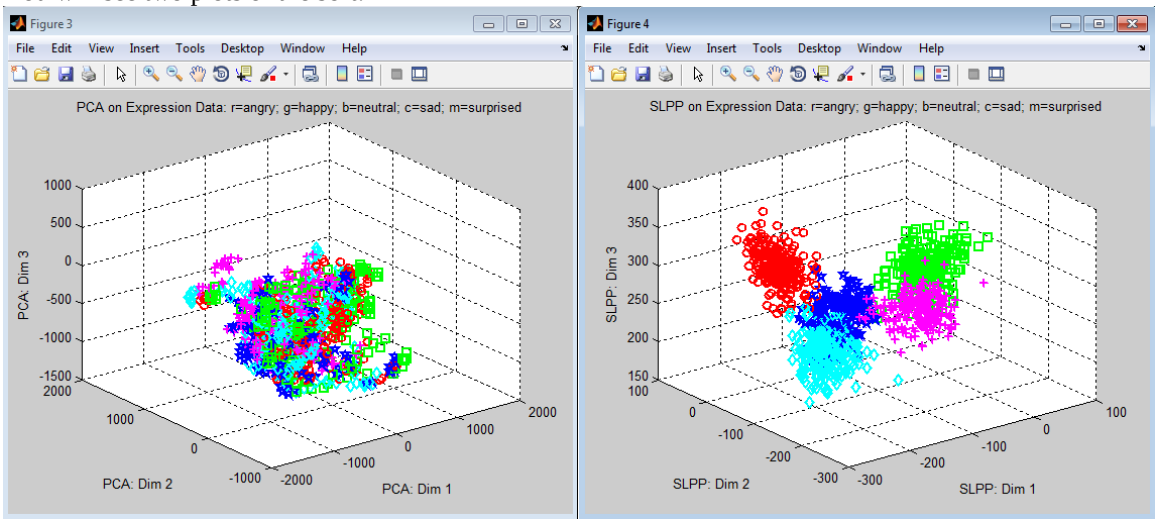
```
%% Dim Reduction
close all ; clear all;
%Download hwk8files_forMycourses.zip and place them into an appropriate
%hwk8 directory.  Update the two paths below for your machine
cd C:\Users\rwpeec\Desktop\rwpeec\rit\CMPE-677_MachineIntelligence\hwk\hwk8
addpath C:\Users\rwpeec\Desktop\rwpeec\rit\CMPE-677_MachineIntelligence\hwk\hwk6\libsvm-
3.18\windows

%  This cell will demonstrate a comparison of unsupervised PCA vs.
%  Supervised methods such as supervised LPP
load LPP_example_data.mat
% 1072 faces from Kohn-Canade (they have been cropped and resampled)
% GT- ground truth:
%   100: angry
%   200: happy
%   300: neutral
%   400: sad
%   500: surprised
% allfaces20- 26x20 images

[numfaces,height,width] = size(allfaces20);
allfacesMean128_Std100=allfaces20;  % this allocates memory, makes it faster
faceu = uint8(zeros(height,width));
for i=1:numfaces
    faceu(:) = uint8(allfaces20(i,:,:));
    %imshow(faceu);pause   %use this line if you want to display the faces

    %fix mean to be 128, std dev to be 100
    allfacesMean128_Std100(i,:,:) = uint8((double(faceu)-
mean(reshape(double(faceu),height*width,1)))*100/(std(reshape(double(faceu),height*width,1)+0.000001
))+ 128);
end
imshow(faceu); %display the last face
```

```
% Now we vectorize our data:  numSamples*numDim; numDim=26*20=520
allfacesVnorm = zeros(numfaces,height*width);
allfacesVnorm(:) = allfacesMean128_Std100;

fea_train = allfacesVnorm;  %fea_train = <num_train_samples> x <D>

%
% First do PCA Analysis
% You will have to set up training and test sets using some sort of k-fold
% cross validataion.  This is skipped here for simplicity
%
cov_mat_fea_train = cov(fea_train);
% eigvector = <DxD>, eigvalue = <DxD>
[PCA_eigvector, PCA_eigvalue] = eig(cov_mat_fea_train);
% eigvalue = <Dx1>
PCA_eigvalue = diag(PCA_eigvalue);

%sort so most important is top to bottom, left to right
PCA_eigvalue = flipud(PCA_eigvalue);
PCA_eigvector = fliplr(PCA_eigvector);

% Now apply PCA eig matrix on data and look at output
PCA_output = fea_train*PCA_eigvector;   %[nxD]*[DxD]  --> [nxD]

%
% Now call LPP code in supervised mode
%
options.Metric = 'Euclidean';
options.NeighborMode = 'Supervised';
options.gnd = GT';          %gnd = 1x<num_train_samples>
options.bLDA = 1;
% W is the Laplacian matrix:
% In supervised mode, this is from ground truth
% In unsupervised mode, this is from neighbor distances
W = constructW(fea_train,options);  % W is <num_train_samples> x <num_train_samples>
options.PCARatio = 0.99;
%options.PCARatio = 1.0;
data=fea_train;
[LPP_eigvector, LPP_eigvalue] = lpp(W, options, fea_train); % eigvector = <Dxd>, eigvalue = <dx1>

% Now apply LPP eig matrix on data and look at output
LPP_output = fea_train*LPP_eigvector;   %[nxD]*[Dxd]  --> [nxd] top d dims

%
% Plot output
%
numclasses = max(GT)/100;  %assumes 100, 200, 300, ... for each class
class_symbol{1}='ro';
class_symbol{2}='gs';
class_symbol{3}='bp';
class_symbol{4}='cd';
class_symbol{5}='m+';

method='PCA';
```

```
abc123 = PCA_output(:,1:3)';  %restrict to 3 dims
% Note: PCA needs more than 3 dimms to classify data
% Use plot(PCA_eigvalue) to see variance of each dim
[numdim,numsubjects] = size(abc123);
abc123_avg = zeros(numclasses,3);
classcount = zeros(numclasses,1);
figure
for k=1:numsubjects
    subj=zeros(1,3);
    class = GT(k)/100;
    %[i j k abc123(:,count)']
    str = class_symbol{class};

    plot3d(abc123(:,k)',str,'linewidth',2); %%this looks better if standalone plot
    hold on
    abc123_avg(class,:) = abc123_avg(class,:) + abc123(:,k)';
    classcount(class) = classcount(class)+1;
end

for i=1:numclasses
    abc123_avg(i,:) = abc123_avg(i,:)./ classcount(i);
end

xlabel(sprintf('%s: Dim 1',method));
ylabel(sprintf('%s: Dim 2',method));
zlabel(sprintf('%s: Dim 3',method));
title(sprintf('%s on Expression Data: r=angry; g=happy; b=neutral; c=sad; m=surprised',method));
grid on
% This will save plot to disk, png is best for vector graphics
% jpeg is best for images
% print -dpng sampleoutputPCA.png


method='SLPP';
abc123 = LPP_output(:,1:3)';  %restrict to 3 dims
[numdim,numsubjects] = size(abc123);
abc123_avg = zeros(numclasses,3);
classcount = zeros(numclasses,1);
figure
for k=1:numsubjects
    subj=zeros(1,3);
    class = GT(k)/100;
    %[i j k abc123(:,count)']
    str = class_symbol{class};

    plot3d(abc123(:,k)',str,'linewidth',2); %%this looks better if standalone plot
    hold on
    abc123_avg(class,:) = abc123_avg(class,:) + abc123(:,k)';
    classcount(class) = classcount(class)+1;
end

for i=1:numclasses
    abc123_avg(i,:) = abc123_avg(i,:)./ classcount(i);
end
```

```
xlabel(sprintf('%s: Dim 1',method));
ylabel(sprintf('%s: Dim 2',method));
zlabel(sprintf('%s: Dim 3',method));
title(sprintf('%s on Expression Data: r=angry; g=happy; b=neutral; c=sad; m=surprised',method));
grid on
% print -dpng sampleoutputSLPP.png
```

You will see two plots of the sort:



Note the unfair advantage SLPP has on separating the five facial expression classes of angry, happy, neutral, sad, and surprised in extremely low dimensions. Keep in mind PCA tries to find the optimal reconstruction function for input samples while LPP is hedging its bets between optimal reconstruction and maximum class separation. The function classify677_hwk8.m supports dimensionality reduction using Principal Component Analysis, Supervised Locality Preserving Projections, Spectral Regression, and Neighborhood Preserving Projections.

Load, ex4data1.mat, then call classify677_hwk8.m using (linear) SVM and nnets (1 hidden layer, 25 nodes) with 5-fold cross validation. Show results for SVM **and** nnets under the three conditions:

a) No dimensionality reduction

b) PCA – For PCA, set the dimensionality reduction fields as:

```
options.useDR=1;

options.dim_reduction='PCA';

options.PCARatio=0.9;

options.nnet_hiddenLayerSize = 25;

[PCAconfusionMatrix_nnet1,PCAaccuracy_nnet1] =  classifiy677_hwk8(X,y,options);
```

c) SLPP- for LPP, set the dimensionality reduction fields as:

```
options.useDR=1;

options.dim_reduction='SLPP';
```

```
options.SLPP_bLDA=0.7;
options.PCARatio=0.9;
 [SLPPconfusionMatrix_svm,SLPPaccuracy_svm] =  classifiy677_hwk8(X,y,options);
```

(12 pts) What are the classification results for each of the above six conditions?
(6 pts) What is the dimension of the original feature, and the approximate dimension of PCA and SLPP?

Ans:

a) **SVM accuracy = 0.9072**

   Neural Networks accuracy = 0.9108

b) **PCA SVM accuracy = 0.9124**

   PCA Neural Networks accuracy = 0.8962

c) **SLPP SVM accuracy = 0.9154**

   SLPP Neural Networks accuracy = 0.8996

The dimension of the original features is 400.

The approximate dimension of PCA is 52.

The approximate dimension of SLPP is 49.