

Convolution Neural Networks

Module 3 - Assignment 8 [100 points]

Principles of Modeling for Cyber-Physical Systems

Due Date: 12/05/2019

Instructor: Madhur Behl

madhur.behl@virginia.edu

In this assignment you will train your own neural network to identify 'X' crosses and 'O' circles in images. This is similar to the example we studied in the lectures.

In addition, we will also test your understanding of CNN hyperparameters selections, weights initializations, and optimization algorithm options.

Note: MATLAB template code is provided to you for this exercise and the template code is what we will use to describe the training and testing process. However, if you prefer not to use MATLAB for this exercise, then that is also allowed. In that case you will submit your network in the Open Neural Network Exchange (ONNX) standard format. More on this later.

Problem 1: Understanding CNNs [25 points]

The Alexnet CNN architecture is shown below along with information about each layer:

Case Study: AlexNet

[Krizhevsky et al. 2012]

Full (simplified) AlexNet architecture:
[227x227x3] INPUT

CONV: 96 11x11 filters at stride 4, pad 0

MAX POOL 1: 3x3 filters at stride 2

NORM 1: Normalization layer

CONV 2: 256 5x5 filters at stride 1, pad 2

MAX POOL 2: 3x3 filters at stride 2

NORM 2: Normalization layer

CONV 3: 384 3x3 filters at stride 1, pad 1

CONV 4: 384 3x3 filters at stride 1, pad 1

CONV 5: 256 3x3 filters at stride 1, pad 1

MAX POOL 3: 3x3 filters at stride 2

FC 6: Fully connected layer (4096 neurons)

FC 7: Fully connected layer (4096 neurons)

FC 8: 1000 neurons (logit scores)

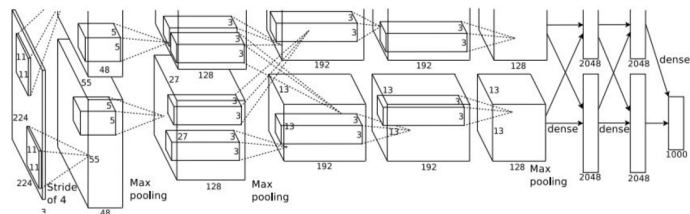


Figure copyright Alex Krizhevsky, Ilya Sutskever, and Geoffrey Hinton, 2012. Reproduced with permission.

The input to AlexNet are images of size: 227x227x3.

For each layer of the AlexNet network indicate:

- What is the size of the output images from that layer?
- How many images are produced as the output of this layer (or what will be the depth of the stack of output images)?
- What are the total number of parameters (weights) in the layer?

For E.g. your response for the first layer CONV1 should be of the form.

CONV1:

- Output image size: Y x Y px.
- Stack size: Z images
- Total number of parameters (weights) in the layer: X

Remember: Normalization layers are equivalent to ReLUs.

Problem 2: Crosses and Circles [75 points]

What is provided:

Upon uncompressing the A08.zip folder, you should see the following files and subfolders:

- training_data
 - o circles
 - o crosses
- BasicCNNtemplate.m

Each subfolder in the training data folder contains 900 images of the corresponding category. You will further split the data into training and validation sets (this is done in the template).

The overall parts of the provided template are as follows:

1. Configure the execution of the code.
2. Load and prep the data
3. Setup the CNN architecture
4. Train the Network
5. Test the performance of the CNN
6. Plotting code:
 - a. Plot wrongly classifies images
 - b. Plotting the filters
 - c. Plotting the feature maps

CNN Architecture:

Here is the suggested CNN architecture that you can implement to get you started.

1	'Input'	Image Input	116x116x1 images with 'zerocenter' normalization
2	'Conv1'	Convolution	16 10x10x1 convolutions with stride [1 1] and padding [4 4 4 4]
3	'ReLu1'	ReLU	ReLU
4	'Pool1'	Max Pooling	5x5 max pooling with stride [2 2] and padding [0 0 0 0]
5	'Conv2'	Convolution	32 10x10 convolutions with stride [1 1] and padding [4 4 4 4]
6	'ReLu2'	ReLU	ReLU
7	'Pool2'	Max Pooling	5x5 max pooling with stride [2 2] and padding [0 0 0 0]
8	'Conv3'	Convolution	32 10x10 convolutions with stride [1 1] and padding [2 2 2 2]
9	'ReLu3'	ReLU	ReLU
10	'Pool3'	Max Pooling	3x3 max pooling with stride [2 2] and padding [0 0 0 0]
11	'FC'	Fully Connected	2 fully connected layer
12	'Softmax'	Softmax	softmax
13	'Classification'	Classification Output	crossentropyex

A walkthrough of the code is presented next.

1. Configure the execution of the code.

```
doTraining          = true;|
% Set these flags to inspect and plot the network (Note: optimized for screen resolution (1920x1200))
show.wrong_classified = false;          % wrong classified images
show.filter          = false;          % filters(weights)
show.feature_maps     = true;          % feature maps
```

Here you can choose several control parameters of the execution of the template.

doTraining when set to true will train a network from scratch. You have to do this atleast once. When set to false, the training of the network is skipped and instead the code looks for a pre-trained network mat file **XONet.mat** to load instead.

Similarly, other flags show.XXXX control which of the plotting code is executed. Remember if all show options are set to true hundreds of plots will be created so you may want to choose wisely when to what and what to plot.

2. Load and prep the data.

```
IMDS = imageDatastore('training_data','IncludeSubfolders',true,'FileExtensions','.bmp','LabelSource','foldernames');
example_image = readimage(IMDS,1); % read one example image from the datastore.

% Uncomment the line below to display the example_image.
% imshow(example_image);

numChannels = size(example_image,3); % get color information - The images are single channel in th
numImageCategories = size(categories(IMDS.Labels),1); % Two image categories in our dataset.

% Create the training and testing datasets.
% Split ImageDatastore labels by proportions
training_proportion = 0.7;
[trainingDS,validationDS] = splitEachLabel(IMDS,training_proportion,'randomize');

LabelCntTr = countEachLabel(trainingDS); % load label information
LabelCntVa = countEachLabel(validationDS);
```

Here first and Image Datastore MATLAB object is created.

You point MATLAB to the training_data folder and tell that the label of the .bmp (bitmap) image file will be the name of the subfolder to which it belongs. This means that in the IMDS datastore, there will be 900 crosses, and 900 circles and they will be automatically labeled (bound) together by Matlab i.e. a random bitmap image from circles will be automatically assigned the label circle. You can view the example_image.

In addition, we are partitioning the entire data into training and validation datastores using the splitEachLabel Matlab function.

3. Setup the CNN architecture

We first need to create an input layer that can process images.

```
%% Setup of the CNN architecture.
if doTraining

    % Convolutional layer parameters
    filterSize = [10 10];
    numFilters = 16;

    % An image input layer inputs 2-D images to a network and applies data normalization.
    % The size of the layer is the same as the number of pixels in our
    % input images.
    inputLayer = imageInputLayer(size(example_image),'Name','Input'); % no data augmentation
```

The imageInputLayer function makes this straightforward.

In addition, we are initializing a few variables filterSize, and numFilters to some fixed values.

You can play with the size of the filter and the number of filters to affect the CNN performance.

Next the middle layers of the network need to be setup.

You need to add convolution, ReLu, and Max Pooling layers appropriately to build the deep network stack.

```
middleLayers = [  
    % The first convolutional layer has a bank of numFilters filters of size filterSize.  
    % A symmetric padding of 4 pixels is added.  
    convolution2dLayer(...)  
    % Next add the ReLU layer:  
    reluLayer('Name','ReLu1')  
    % Follow it with a max pooling layer that has a 5x5 spatial pooling area  
    % and a stride of 2 pixels. This down-samples the data dimensions.  
    maxPooling2dLayer(...)  
  
    % Repeat the 3 core layers to complete the middle of the network.  
    % This time use 32 filters instead of 16.  
  
    % Repeat the 3 core layers one more time  
    % This time change symmetric padding to 2 for the convolution, and  
    % the stride to 3 for the maxpoolinglayer.  
  
];
```

A template is provided but you will fill out the correct values to the function and follow the architecture included earlier in this worksheet.

Finally, the end or the final layers of the network are declared.

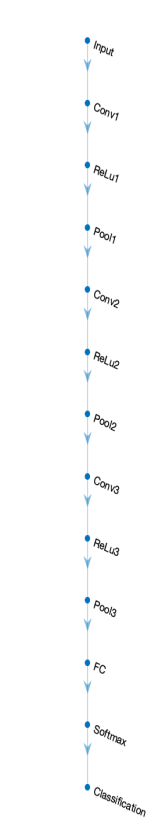
```
finalLayers = [  
  
    % % Add a fully connected layer with the same number of neurons as  
    % the number of image categories.  
    fullyConnectedLayer(numImageCategories,'Name','FC')  
  
    % Add the softmax loss layer and classification layer.  
    % The final layers use the output of the fully connected layer to compute the categorical  
    % probability distribution over the image classes. During the training  
    % process, all the network weights are tuned to minimize the loss over this  
    % categorical distribution.  
    softmaxLayer('Name','Softmax');  
    classificationLayer('Name','Classification')  
];  
  
layers = [  
    inputLayer  
    middleLayers  
    finalLayers  
];
```

You need not change these.

We have a fully connected layer with 2 output neurons, followed by a softmax layer, followed by a classificationLayer which computes the cross entropy loss for multi-class classification problems.

You then create a structure layer which has the three layers joined together in the correct order from input to middle to final.







If you created the layers in the correct order. A good sanity check is to plot the layer graph and it should look like the one shown below.









Note that in order to create the layer graph without an error, you must indicate the ‘Name’, ‘value’ option during the creation of the middle layers.

Useful Matlab functions for this part of the worksheet are:








Convolution and Fully Connected Layers

Layer	Description
 <code>convolution2dLayer</code>	A 2-D convolutional layer applies sliding convolutional filters to the input.
 <code>convolution3dLayer</code>	A 3-D convolutional layer applies sliding cuboidal convolution filters to three-dimensional input.
 <code>groupedConvolution2dLayer</code>	A 2-D grouped convolutional layer separates the input channels into groups and applies sliding convolutional filters. Use grouped convolutional layers for channel-wise separable (also known as depth-wise separable) convolution.
 <code>transposedConv2dLayer</code>	A transposed 2-D convolution layer upsamples feature maps.
 <code>transposedConv3dLayer</code>	A transposed 3-D convolution layer upsamples three-dimensional feature maps.
 <code>fullyConnectedLayer</code>	A fully connected layer multiplies the input by a weight matrix and then adds a bias vector.

Activation Layers

Layer	Description
 <code>reluLayer</code>	A ReLU layer performs a threshold operation to each element of the input, where any value less than zero is set to zero.
 <code>leakyReluLayer</code>	A leaky ReLU layer performs a threshold operation, where any input value less than zero is multiplied by a fixed scalar.
 <code>clippedReluLayer</code>	A clipped ReLU layer performs a threshold operation, where any input value less than zero is set to zero and any value above the <i>clipping ceiling</i> is set to that clipping ceiling.
 <code>eluLayer</code>	An ELU activation layer performs the identity operation on positive inputs and an exponential nonlinearity on negative inputs.
 <code>tanhLayer</code>	A hyperbolic tangent (tanh) activation layer applies the tanh function on the layer inputs.
 <code>preluLayer</code> (Custom layer example)	A PReLU layer performs a threshold operation, where for each channel, any input value less than zero is multiplied by a scalar learned at training time.

Pooling and Unpooling Layers

Layer	Description
 <code>averagePooling2dLayer</code>	An average pooling layer performs down-sampling by dividing the input into rectangular pooling regions and computing the average values of each region.
 <code>averagePooling3dLayer</code>	A 3-D average pooling layer performs down-sampling by dividing three-dimensional input into cuboidal pooling regions and computing the average values of each region.
 <code>globalAveragePooling2dLayer</code>	A global average pooling layer performs down-sampling by computing the mean of the height and width dimensions of the input.
 <code>globalAveragePooling3dLayer</code>	A global average pooling layer performs down-sampling by computing the mean of the height, width, and depth dimensions of the input.
 <code>maxPooling2dLayer</code>	A max pooling layer performs down-sampling by dividing the input into rectangular pooling regions, and computing the maximum of each region.
 <code>maxPooling3dLayer</code>	A 3-D max pooling layer performs down-sampling by dividing three-dimensional input into cuboidal pooling regions, and computing the maximum of each region.
 <code>maxUnpooling2dLayer</code>	A max unpooling layer unpool the output of a max pooling layer.

4. Train the Network

First need to specify the options for the backpropagation optimizer: SGDM – Stochastic Gradient Decent Method.

```
%% Train the Network
%Initialize the first convolutional layer weights using
% normally distributed random numbers with standard deviation of 0.0001.
% This helps improve the convergence of training.
layers(2).Weights = 0.0001 * randn([filterSize numChannels numFilters]);

% Set the network training options
% Try Momentum option 0.1 and 0.9 – Which is Better ?
% Try LearningRate 0.01, and 0.001 – What is the difference ?
% Try 10–20 Maxepochs

opts = trainingOptions('sgdm', ...
    'Momentum', 0, ...
    'InitialLearnRate', 0, ...
    'LearnRateSchedule', 'piecewise', ...
    'LearnRateDropFactor', 0.5, ...
    'LearnRateDropPeriod', 10, ...
    'L2Regularization', 0.004, ...
    'MaxEpochs', 0, ...
    'MiniBatchSize', 64, ...    % 64 for Quadro
    'Verbose', true,...
    'Plots','training-progress');|

% Train a network.
rng('default');
rng(123); % random seed

XONet = trainNetwork(trainingDS, layers, opts);
save('XONet.mat', 'XONet');
```

Our Network is not that complex so initial normally distributed random weights of 0.001 should work.

Several hyperparameters have been set to 0 by default. You need to change these.

- Try Momentum values of 0.1 and 0.9 – Which leads to better results?
- Try InitialLearnRate of 0.01 and 0.001 – What is the difference in performance (time to train, and accuracy)
- Need to specify a value of the MaxEpochs of training

Finally, we save the network as a mat file. This is important since if the doTraining was set to false, we would have skipped training and simple reloaded the saved network.

5. Test the performance of the CNN

```
%% Test the performance of the NN

% test network performance on validation set
[labels,~] = classify(XONet, validationDS, 'MiniBatchSize', 128);

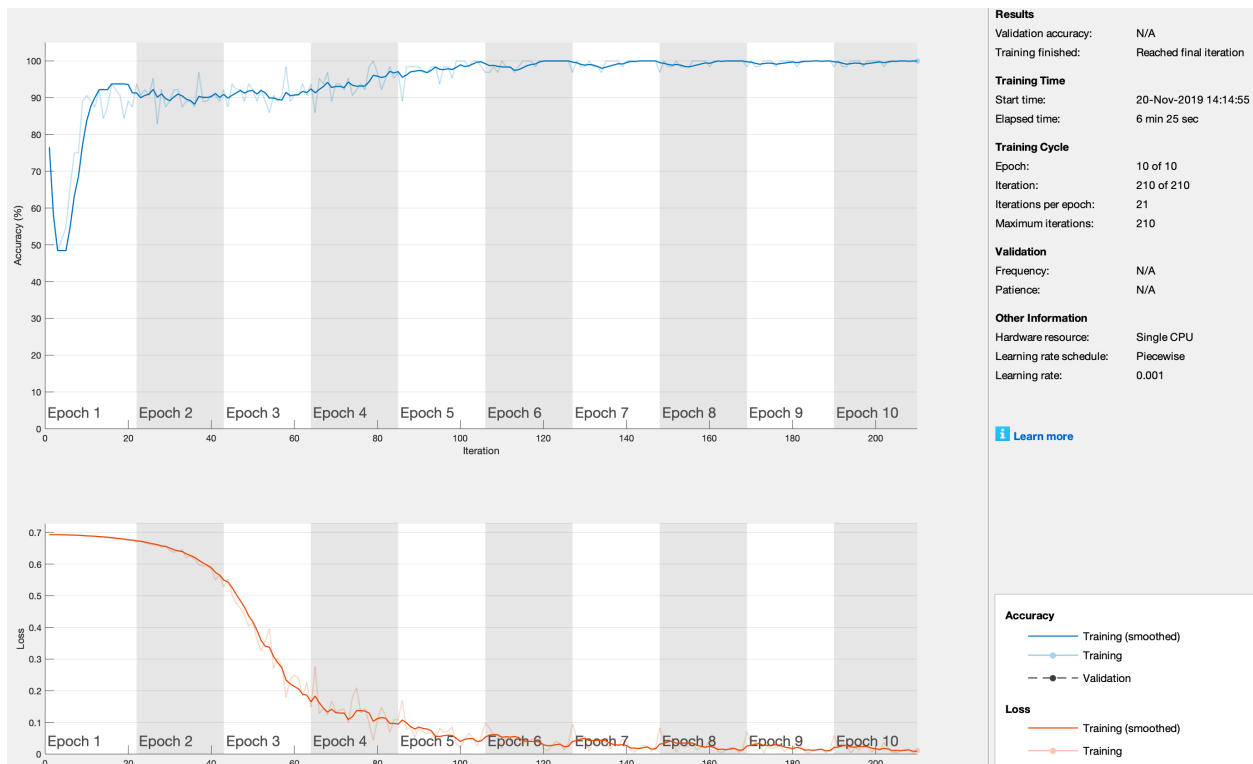
% calculate the confusion matrix. |
confMat = confusionmat(validationDS.Labels, labels);
confMat = bsxfun(@divide,confMat,sum(confMat,2));
fprintf('Performance on validation set \t\t\t%.4f\n',mean(diag(confMat)));
```

Using the `classify` function, we can obtain the predictions of the network on the `validationDS` datastore. (or the remaining 0.3 fraction) of the data.

The validation labels are compared to the true labels and a confusion matrix is produced as the output.

The mean of the diagonal of the confusion matrix is the accuracy of the XONet.

If you did everything right, you should have also seen the error reduced during the training like so:

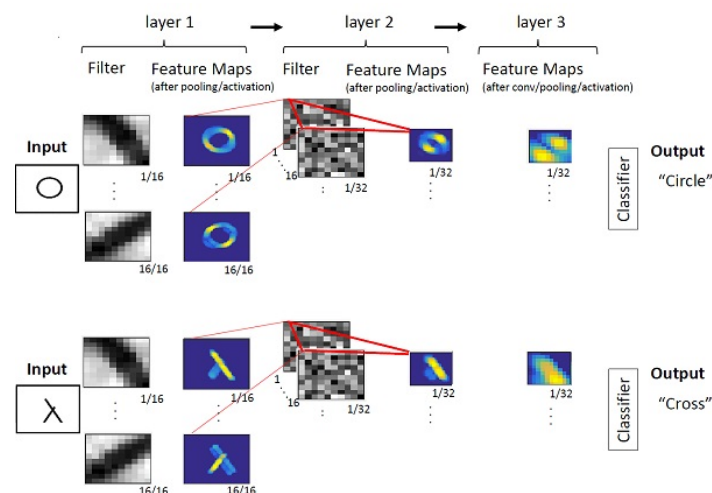
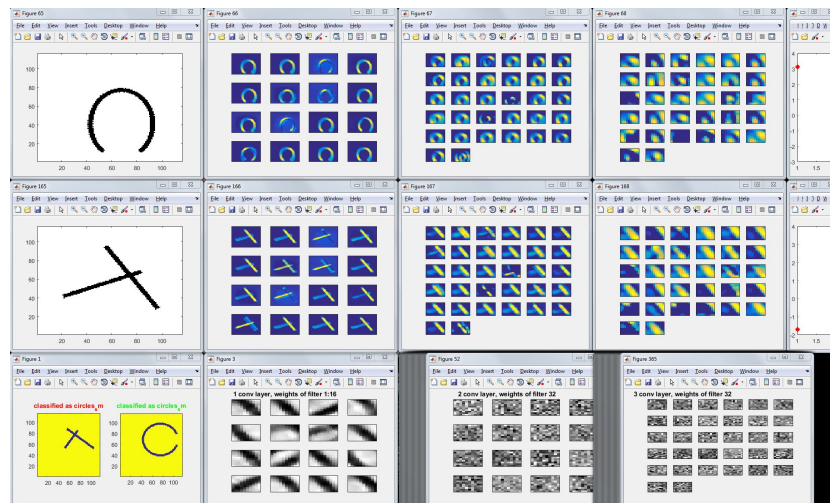


6. Plotting code:

There are three types of plots you can generate:

1. **Plot wrongly classifies images from the ValidationDS.** This is straightforward, you are searching for the index of the mismatch between ValidationDS labels and true labels and plotting the corresponding image(s)
2. **Plot the filters:** Each Convolution layer uses some number of filters, this code cycles through all of them and creates plots of the filters for visualization that indeed the learned filters are becoming complex in deeper layers. Some of the values of number of filters in the layers are hardcoded. You may want to tailor them to your network.
3. **Plot the feature maps** (outputs of the C/R/P) for the different layers in the network.

Example outputs:



Upload the following for problem 2: [60 points]

1. **Upload a single .zip file** with the filename [firstname_lastname_UVA_computing_ID].zip
2. The zip file should contain:
 - a. The original training_data folder with the subfolders circles and crosses with the images included (this is < 3.6 Mb)
 - b. Your solution to BasicCNNtemplate.m
 - c. Your best performing network in the form of a .mat file XONet (This file is automatically created when doTraining == true)
 - d. Your responses to the effect of Momentum, InitialTraingRate, and Epochs on the performance of the network – Include supporting plots and accuracy values.
 - i. This can be a PDF with the plots and inferences included.
 - e. Report (with plots) on the architecture, and accuracy of your best performing network:
 - i. Include an image of the layers of the network.
 - ii. Report accuracy (as computed by the template, using the confusion matrix) on the validationDS of your best performing model.
 - iii. Report the chosen values of the hyperparameters of your network.

Test on Unseen Data: [15 points]

In addition to running your code as indicated above.

We will also run your code on an unseen Test dataset that has not been provided to you.

This test dataset contains 100 images of circles and crosses each – very similar to the provided dataset.

We will simply use the XONet.mat file you have provided to compute the accuracy on the test set.

We will compare your network accuracy on the test set with the validation accuracy reported by you to test for overfitting. If they are close then you get the full 15 points.

Note on not using Matlab:

If you are already familiar/comfortable using a deep learning library, you can do the entire assignment in your choice of platform, but in addition to uploading the code which we can run, you also need to generate the appropriate plots requested above.

Finally, regardless of your choice of DL framework you need to provide us with a mat file of your best performing model as well.

The best way to do so is using the Open Neural Network Exchange (ONNX) open standard format for representing machine learning models. [<https://github.com/onnx/tutorials>] and using the importONNXNetwork Matlab function.