CMPEN/EE 454 Project 1

Due 09/27/2023 by 11:59pm (upload to Canvas)

Overview
The goal of this project will be to implement the forward pass of a Convolutional Neural Network for performing object detection. We will see how our ideas of applying image filters can be (and have been) combined to construct powerful object detection algorithms in images. You will turn in the following:

1. Your code for the entire project. Note, we will have you implement some specific function prototypes (names and interfaces) so that we can validate you implementation (details below).
2. A five minute **recorded** presentation summarizing your implementation and networks performance (details below).

Project Parameters

1. You'll need to implement your solutions in Matlab
2. You're restricted to the general routines we've been using in class thus far. **You'll want to implement what we have detailed in this project description. Do not use code off of Github or Stack Exchange.** If your implementation is some super elegant generalization of the procedures we detail, we will be suspicious of your code and may request a meeting for you to provide additional explanations of what you've implemented in person.

Project Description The following table provides the description of an 18 layer CNN which takes in $32 \times 32 \times 3$ images and classifies them into 1 of 10 categories.

layer #	layer type	input size	filterbank size	num biases	output size
1	imnormalize	$32 \times 32 \times 3$			$32 \times 32 \times 3$
2	convolve	$32 \times 32 \times 3$	$3 \times 3 \times 3 \times 10$	10	$32 \times 32 \times 10$
3	relu	$32 \times 32 \times 10$			$32 \times 32 \times 10$
4	convolve	$32 \times 32 \times 10$	$3 \times 3 \times 10 \times 10$	10	$32 \times 32 \times 10$
5	relu	$32 \times 32 \times 10$			$32 \times 32 \times 10$
6	maxpool	$32 \times 32 \times 10$			$16 \times 16 \times 10$
7	convolve	$16 \times 16 \times 10$	$3 \times 3 \times 10 \times 10$	10	$16 \times 16 \times 10$
8	relu	$16 \times 16 \times 10$			$16 \times 16 \times 10$
9	convolve	$16 \times 16 \times 10$	$3 \times 3 \times 10 \times 10$	10	$16 \times 16 \times 10$
10	relu	$16 \times 16 \times 10$			$16 \times 16 \times 10$
11	maxpool	$16 \times 16 \times 10$			$8 \times 8 \times 10$
12	convolve	$8 \times 8 \times 10$	$3 \times 3 \times 10 \times 10$	10	$8 \times 8 \times 10$
13	relu	$8 \times 8 \times 10$			$8 \times 8 \times 10$
14	convolve	$8 \times 8 \times 10$	$3 \times 3 \times 10 \times 10$	10	$8 \times 8 \times 10$
15	relu	$8 \times 8 \times 10$			$8 \times 8 \times 10$
16	maxpool	$8 \times 8 \times 10$			$4 \times 4 \times 10$
17	fullconnect	$4 \times 4 \times 10$	$4 \times 4 \times 10 \times 10$	10	$1 \times 1 \times 10$
18	softmax	$1 \times 1 \times 10$			$1 \times 1 \times 10$

You will be providing implementations for the different operations at each layer, i.e. routines for *imnormalize*, *convolve*, *relu*, *maxpool*, *fullconnect*, *softmax*. You'll then bring these together to perform

the forward pass of your network. Note that when we say "forward pass" we simply mean apply the CNN to an image to classify it.

Now, because we are only performing a forward pass, we are not going to be training the network ourselves. As a result, we have provided a set of parameters in a file named *CNNParameters.mat*. Loading this file will give the parameters for each filter bank as well as the bias vectors. We need filter parameters and bias vectors for each convolution and fully connected layer. You'll use these values in your routines which require filter banks. Code for loading this is given below:

```
%loading this file defines filterbanks and biasvectors
load 'CNNparameters.mat'

%sample code to verify which layers have filters and biases
for d = 1:length(layertypes)

    fprintf('layer %d is of type %s\n',d,layertypes{d});
    filterbank = filterbanks{d};
    if not(isempty(filterbank))
        fprintf('    filterbank size %d x %d x %d x %d\n', ...
            size(filterbank,1),size(filterbank,2), ...
            size(filterbank,3),size(filterbank,4));
        biasvec = biasvectors{d};
        fprintf('    number of biases is %d\n',length(biasvec));
    end
end
```

Format of your routines

You'll write routines for each layer of the network. They should follow the below input / output relationship because **we will have scripts which evaluate your implementations when grading. So be sure you follow the below convention**.

```
outarray = apply_imnormalize (inarray)
     inarray is an NxMx3 uint8 image and outarray is NxMx3
outarray = apply_relu(inarray)
     inarray is NxMxD and outarray is the same size
outarray = apply_maxpool(inarray)
     inarray is 2Nx2MxD and outarray is size NxMxD
outarray = apply_convolve(inarray, filterbank, biasvals)
     inarray is NxMxD1, filterbank is RxCxD1xD2,
     biasvals is a length D2 vector, and outarray is NxMxD2
outarray = apply_fullconnect(inarray, filterbank, biasvals)
     inarray is NxMxD1, filterbank is NxMxD1xD2,
     biasvals is a length D2 vector, and outarray is 1x1xD2
outarray = apply_softmax(inarray)
     inarray is 1x1xD and outarray is the same size
```

Bring it together

Once you have your routines in place, you'll glue it all together to create your network. You'll be applying your 18-layer network to the CIFAR-10 dataset which we provide in *cifar10testdata.mat*. These will consist of 10000, 32×32 color images with 1000 of each category. The dataset can be loaded using the code below.

```
%loading this file defines imageset, trueclass, and classlabels
load 'cifar10testdata.mat'

%some sample code to read and display one image from each class
for classindex = 1:10
    %get indices of all images of that class
    inds = find(trueclass==classindex);
    %take first one

    imrgb = imageset(:,:,:,inds(1));
    %display it along with ground truth text label
    figure; imagesc(imrgb); truesize(gcf,[64 64]);
    title(sprintf('\%s',classlabels{classindex}));
end
```

Evaluating Your Network

To evaluate the performance of your network, you are going to apply your network to the entire dataset of 10000 images and provide a confusion matrix C. This will be a 10×10 matrix where each entry C_{ij} logs the number of times an image of true class i is classified as class j. Clearly, the diagonal entries will correspond to the correct labeling of class i with class i. All of the number on the off diagonals will represent errors in classification. The total accuracy can be computed as

$$A = \frac{\sum_i C_{ii}}{\sum_i \sum_j C_{ij}} \tag{1}$$

Feel free to expand on any observations or insights obtained from your final confusion matrix in your presentation.

Your Presentation

Your presentation should be 5 minutes ± 30 seconds. The time is part of the grade, so please stick within this. It should contain

1. A summary of what the project is about in your own words.
2. A description with a flow chart of the flow control of the routines you implemented. Discuss any design decisions you made.
3. Experimental behavior. How was the classification performance? Were there any issues with computational speed? Did you need to rewrite anything to be more efficient?
4. Present your confusions matrix (could be a part of the previous item).
5. Describe what each person did as part of the project. Note you are welcome to distribute the implementations etc. and it doesn't have to be a precise even split of the work to get full credit. With that said, each person should contribute meaningfully to the project. If one person didn't do anything, that should be noted.

Grading

Your grade will be 50% submitting run-able Matlab code and 50% on your presentation.

Layer Routines Descriptions

A CNN is composed of a set of layers. At each layer an input array of size $N_1 \times M_1 \times D_1$ is given and the layer produces an output array of size $N_2 \times M_2 \times D_2$. The first layer will be the size of the images $32 \times 32 \times 3$ and the final output layer will be a $1 \times 1 \times 10$ array representing the probabilities of each of the 10 classes. For terminology, we'll think of N and M as the rows a columns of an image and the D values to the channel (e.g. in RGB, one channel of red, one for blue, and one for green).

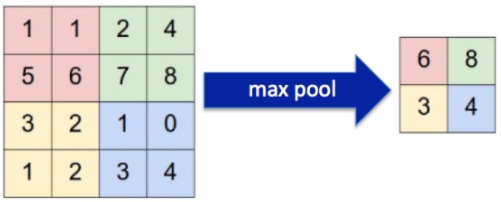
Image Normalization
Input is a color image of size $N \times M \times 3$, and output is the same size defined using

$$\text{Out}(i,j,k) = \text{In}(i,j,k)/255.0 - 0.5$$

ReLU
Input is an array of size $N \times M \times D$ and output is an array of the same size defined by

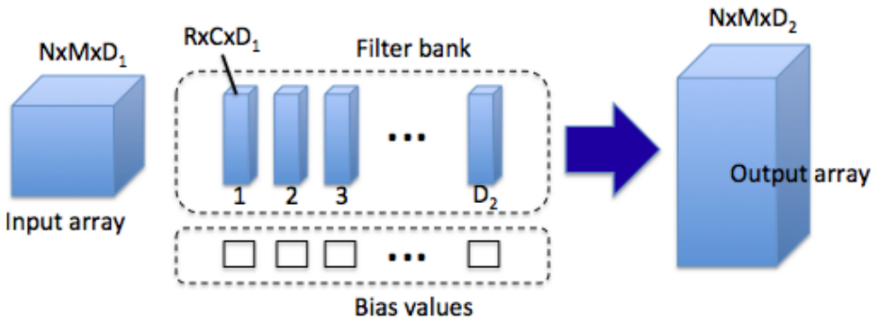
$$\text{Out}(i,j,k) = \max(\text{In}(i,j,k)\ ,\ 0)$$

Maxpool
Input is an array of size $2N \times 2M \times D$ and output is an array of size $N \times M \times D$ obtained by considering the maximum values in subsections of the rows and columns. It effectively reduces the number of total rows and columns by using the maximum to represent a patch of rows and columns. Note that it does this on each channel separately which is why the number of channels is unchanged.

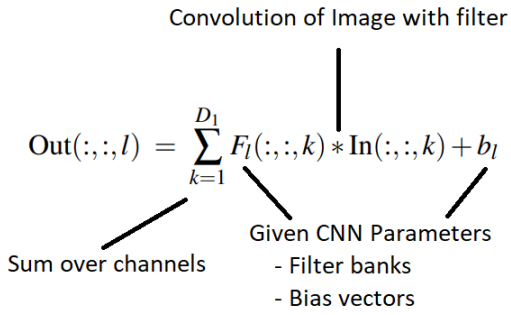


$$\text{Out}(i,j,k) = \max(\{\ \text{In}(r,c,k) \mid (2i-1) < r < 2i \ \text{ and } (2j-1) < c < 2j\ \})$$

Convolution
The input array is of size $N \times M \times D_1$ and the output is an array of size $N \times M \times D_2$. Note that the number of rows and columns is unchanged. The number of channels is what changes in the output. Below is an image describing the operation.



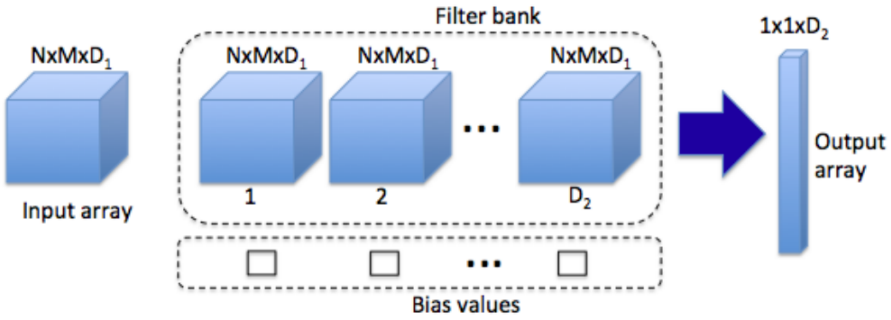
Note, that we have D_2 different filters which we apply to obtain our final output array. The equation for doing this is given below. Note that we are performance a convolution of each filter in the filter bank with the given input array and summing these together to obtain the final output array.



Unpacking this slightly, we have that for each output channel l, we use the filter and bias values associated with that channel and perform D_1 convolutions (one for each channel of the input) summing them up to produce the output for channel l

Fully Connected

Input is an array of size $N \times M \times D_1$ and the output is an array of size $1 \times 1 \times D_2$. Similarl to how the convolution layer works, the fully connected layer applies a filter bank of D_2 filters and bias values to generate an output vector (cf. below image). The primary difference is that no convolution is performed, it performs elementwise multiplication and sums over of the entries.



$$\text{Out}(1,1,l) = \sum_{i=1}^{N}\sum_{j=1}^{M}\sum_{k=1}^{D_1} F_l(i,j,k) \times \text{In}(i,j,k) + b_l$$

Softmax

Input array is of size $1 \times 1 \times D$ and the output array is an array of the same size defined by:

$$\text{Out}(1,1,k) = \frac{\exp(\text{In}(1,1,k) - \alpha)}{\sum_{k=1}^{D} \exp(\text{In}(1,1,k) - \alpha)}$$

where $\alpha = \max_k \text{In}(1,1,k)$ is the max of the input vector. This essentially turns the output of your fully connected layer into a vector of probabilities. You can then classify the category based on this vector by choosing the one with highest probability.

Debugging Help

There is a *debuggingTest.mat* file which you can use to help debug your network as you develop. This contains an input image and the expected output values at each layer of the network. This should allow you to build you routines methodically and verify their validity. Some code for loading the correct values of each layer is given below. Happy coding!

```
%loading this file defines imrgb and layerResults
load 'debuggingTest.mat'

%sample code to show image and access expected results
figure; imagesc(imrgb); truesize(gcf,[64 64]);
for d = 1:length(layerResults)
    result = layerResults{d};
    fprintf('layer %d output is size %d x %d x %d\n',...
        d,size(result,1),size(result,2), size(result,3));
end
%find most probable class
classprobvec = squeeze(layerResults{end});
[maxprob,maxclass] = max(classprobvec);
%note, classlabels is defined in  'cifar10testdata.mat'
fprintf('estimated class is %s with probability %.4f\n',...
    classlabels{maxclass},maxprob);
```