Neuro 120 Homework 3: Vision and Deep Networks

William Schmitt and Will Drew

Due: Thursday 8 November 2018

1 Question 1: Early Vision and Convolutions

1.1 Difference of Gaussians Filter

We created the difference of Gaussians filter via the following line of code: `dog = fspecial('gaussian', [51 51], 3)- fspecial('gaussian', [51 51], 7);` and plotted this with `surf(dog)` to produce the following image in Figure 1.

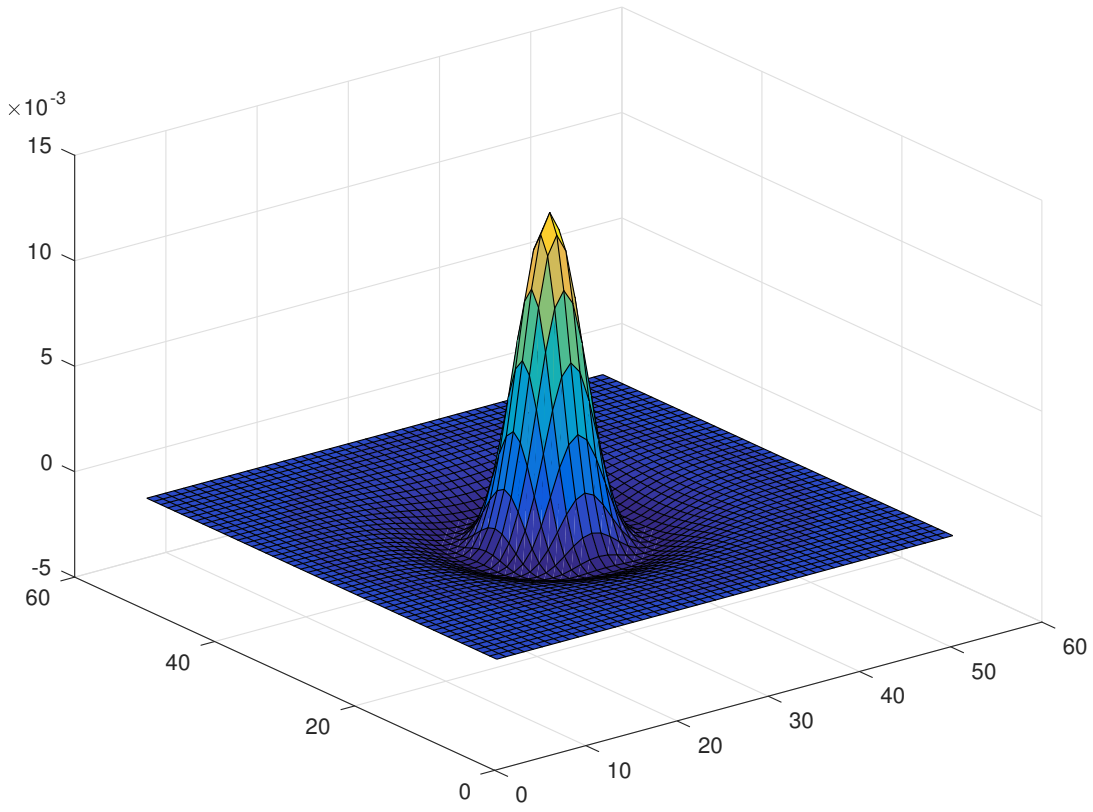

Figure 1: The coefficients of the difference of Gaussians filter.

1.2 Mach band Convolution

We convolve the Mach band image with the filter from part A via the following line of code: `res = conv2(im_mach, dog, 'valid');`. We then plot this with the command, `imagesc(res)`. This produces Figure 2. We can see from this figure that the Mach band illusion results in activity at the boundary of the lines in the images (as seen by the one white line, corresponding to an increase in the RGC responses, and by the one black line, corresponding to a decrease in the RGC responses). Thus, these RGCs are detecting two "edges" in the Mach band illusion. This illusion is due to the center-surround nature of the difference of Gaussians filter the RGCs are modeled with.

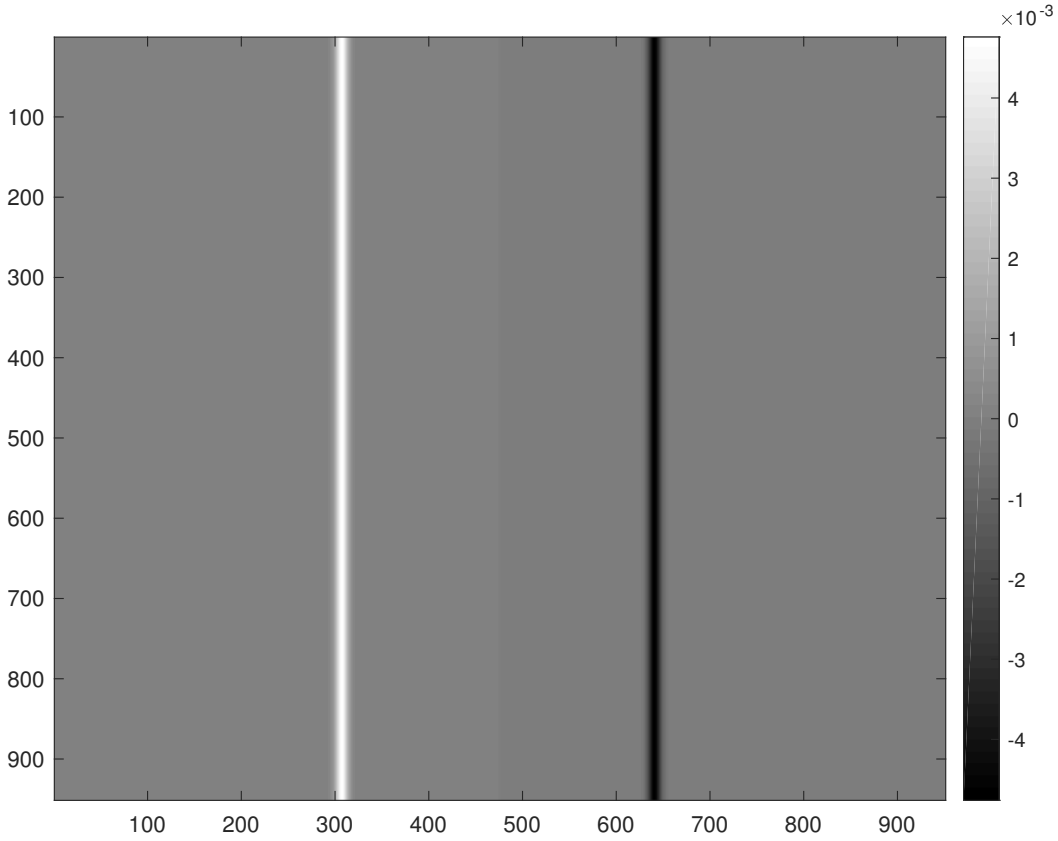

Figure 2: The result of convolving the Mach bands image with the filter from part A.

1.3 Mach band Convolution with a Small Constant

We convolve the Mach band image with the filter from part A plus a small constant of 0.00001 via the following line of code: `dog2 = dog + 0.00001; res = conv2(im_mach, dog2, 'valid');`. We then plot this with the command, `imagesc(res)`. This produces Figure 3. We can see from this command that again, the RGC model is detecting the two edges in the same locations. However, there is now a background gradient from light to dark (going left to right), with the increase (decrease) in activity being noticeably more at the first (second) edge. This relates to the Mach

band illusion in that it is this detection of an edge in an image which has a larger overall gradient of brightness, which causes the person to perceive one side of the edge to be brighter than it actually is and one side to be darker than it actually is.

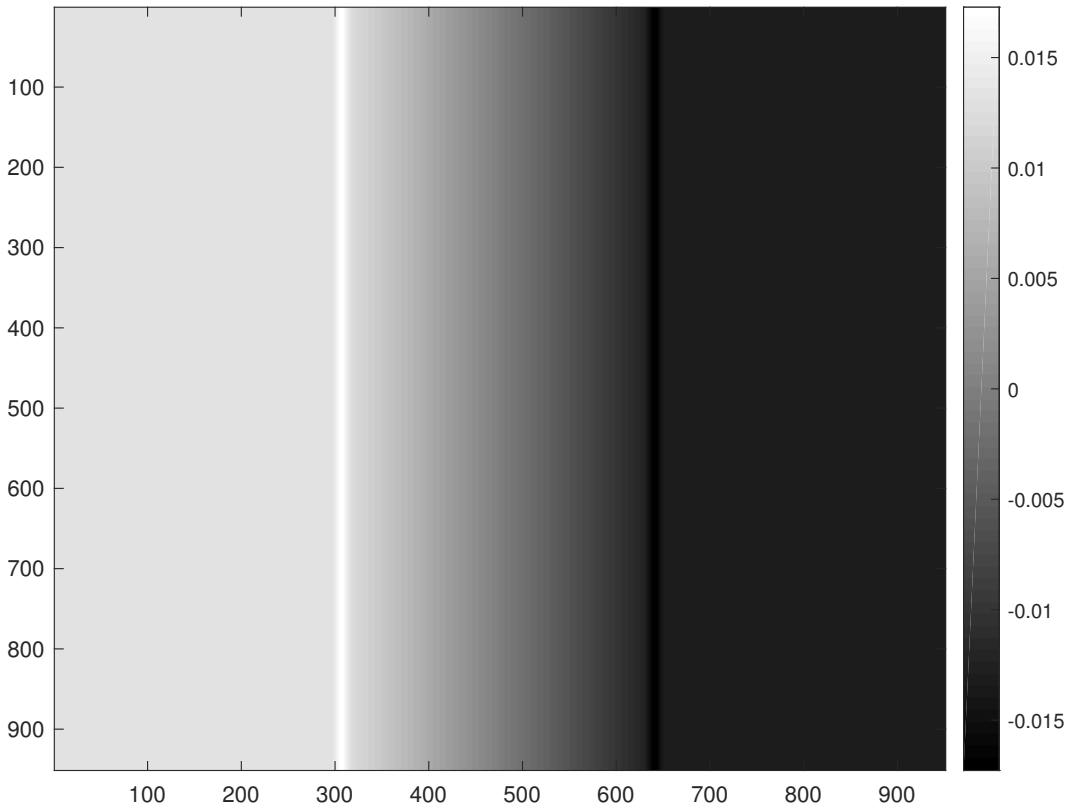

Figure 3: The result of convolving the Mach bands image with the filter from part A plus a small constant.

1.4 Checkerboard Illusion

We convolve the checkerboard illusion with the filter from part A via the following of code: `res = conv2(im_cb, dog, 'valid');`. We then plot this with the command, `imagesc(res)`. This produces Figure 4. We can see from this figure that there are slight (barely noticeable) gray circles at the intersection of the lines compared to the middle of a stripe. Thus, we see that the center-surround nature of the RGCs produce a decreased response at the intersection of these stripes, which is why it appears to our eyes that there are gray circles at the intersections when they are really just white.

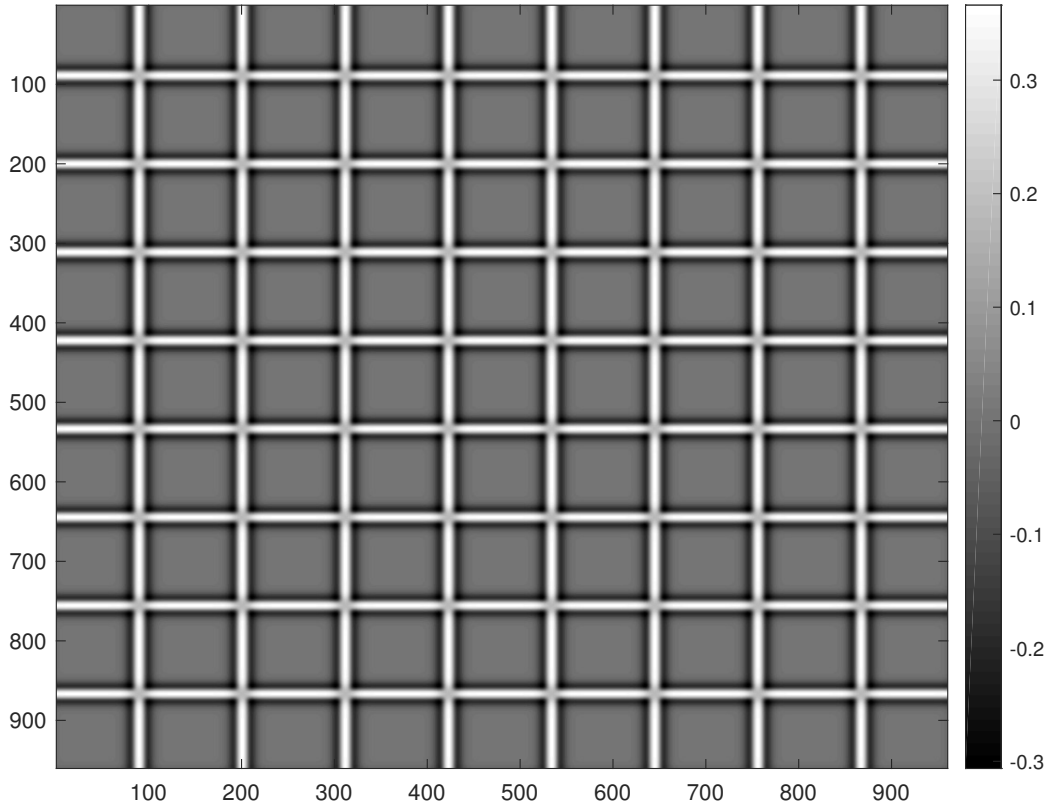

Figure 4: The result of convolving the checkerboard illusion image with the filter from part A.

1.5 Checkerboard Illusion with a Narrower Filter

We make a new difference of Gaussians filter with the following code: `dog3 = fspecial('gaussian', [51 51], 3)- fspecial('gaussian', [51 51], 7);`. We plotted this with `surf(dog3)` to produce the following image in Figure 5. We then convolve the checkerboard illusion with this filter via the following code: `res = conv2(im_cb, dog3, 'valid');`. This produces Figure 6. We can see from this figure that the intersections of the stripes are the same color as the stripes themselves, which explains why when we focus on a certain intersection, the gray circle appears to disappear. The RGCs in our fovea, which are modeled here, are not tricked by the illusion.

1.6 Peppers

We convolved the pepper image with both the filter from part A and that from part E via the following code:

```
%% Convolve and plot the result

res = conv2(im_natural, dog, 'valid')
```

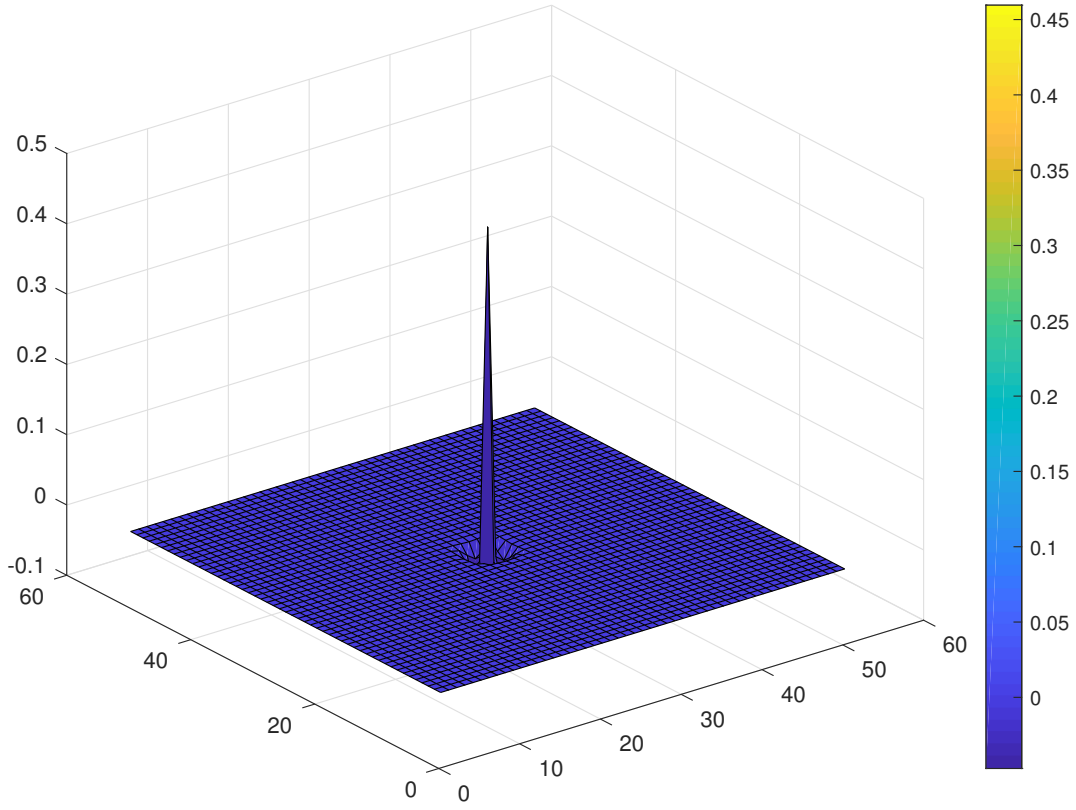

Figure 5: The coefficients of a narrower difference of Gaussians filter.

```
imagesc(res)
colormap gray
pause
res = conv2(im_natural, dog3, 'valid')
imagesc(res)
colormap gray
```

. This produces Figures 7 and 8. We can see from these figures that in the image convolved with the broader filter (from Part A), the gross gradient differences between the peppers is preserved. Further, one can distinguish (somewhat) some peppers from others, though the acuity of an individual pepper is low. However, with the narrower filter, the gradient information is lost (i.e. we can't tell which peppers are darker and which are lighter), but we can make out fine details on each pepper (i.e. the lines of the onion).

For the reader, we reproduce our entire code used for this question below before moving on to the next question:

```
clear all

%% Make mach band image
```

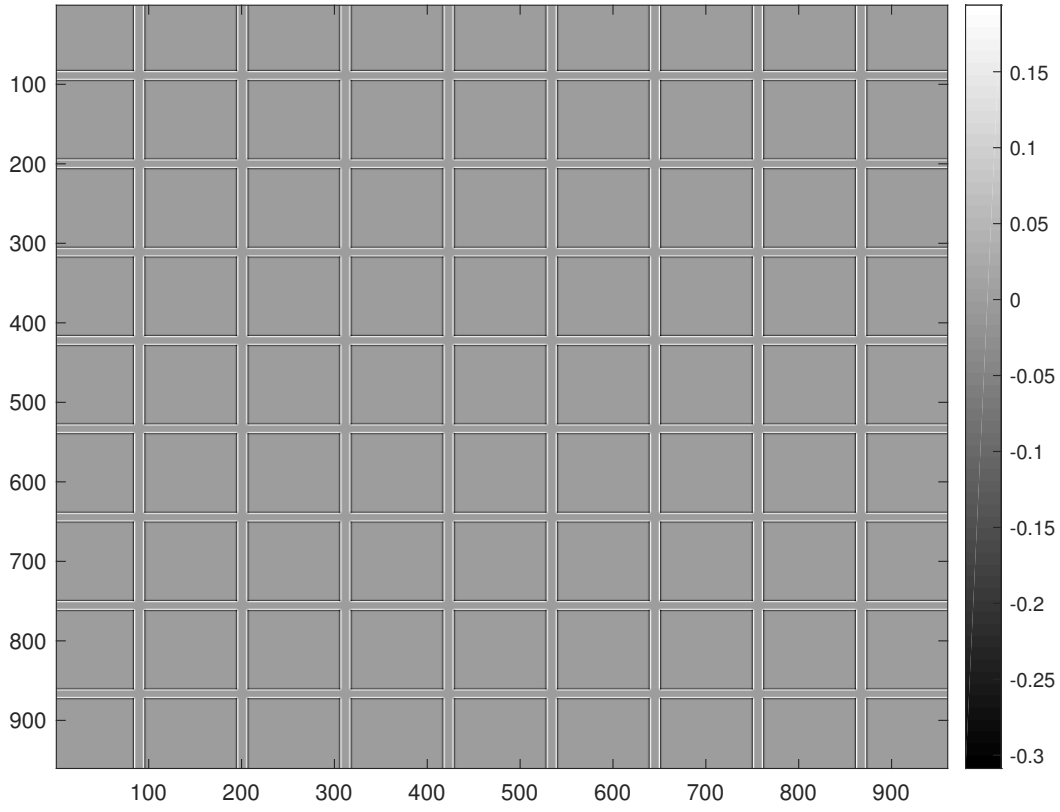

Figure 6: The result of convolving the checkerboard illusion image with a narrower filter.

```
sz = 1000;
slope = -.003;
[x,y]=meshgrid(-sz/2:sz/2,-sz/2:sz/2);
im_mach = max(-.5,min(.5,x*slope));

imagesc(im_mach)
colormap gray


%% Create and plot DoG filter

dog = fspecial('gaussian', [51 51], 3) - fspecial('gaussian', [51 51], 7);
surf(dog)

%% Convolve image and plot the result

res = conv2(im_mach, dog, 'valid');
dog2 = dog+0.00001;

imagesc(res)
```

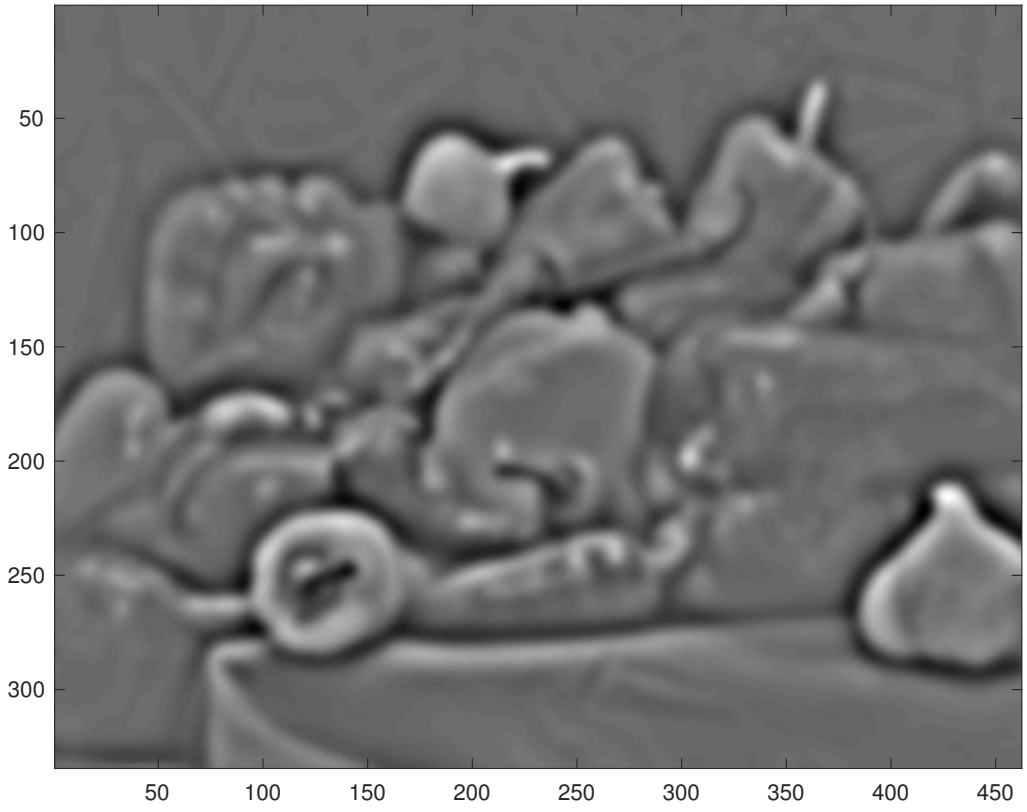

Figure 7: The result of convolving the peppers image with the original filter from part A.

```
colormap gray

res = conv2(im_mach, dog2, 'valid');
imagesc(res)
colormap gray

%% Create checkerboard image

sz = 1000;
stripe_width = 10;
num_stripes = 10;

im_cb = -.5*ones(sz,sz);

for c = round(linspace(1,sz,num_stripes));
    c
    im_cb(c:c+stripe_width,:) = .5;
    im_cb(:,c:c+stripe_width) = .5;
end

imagesc(im_cb)
```

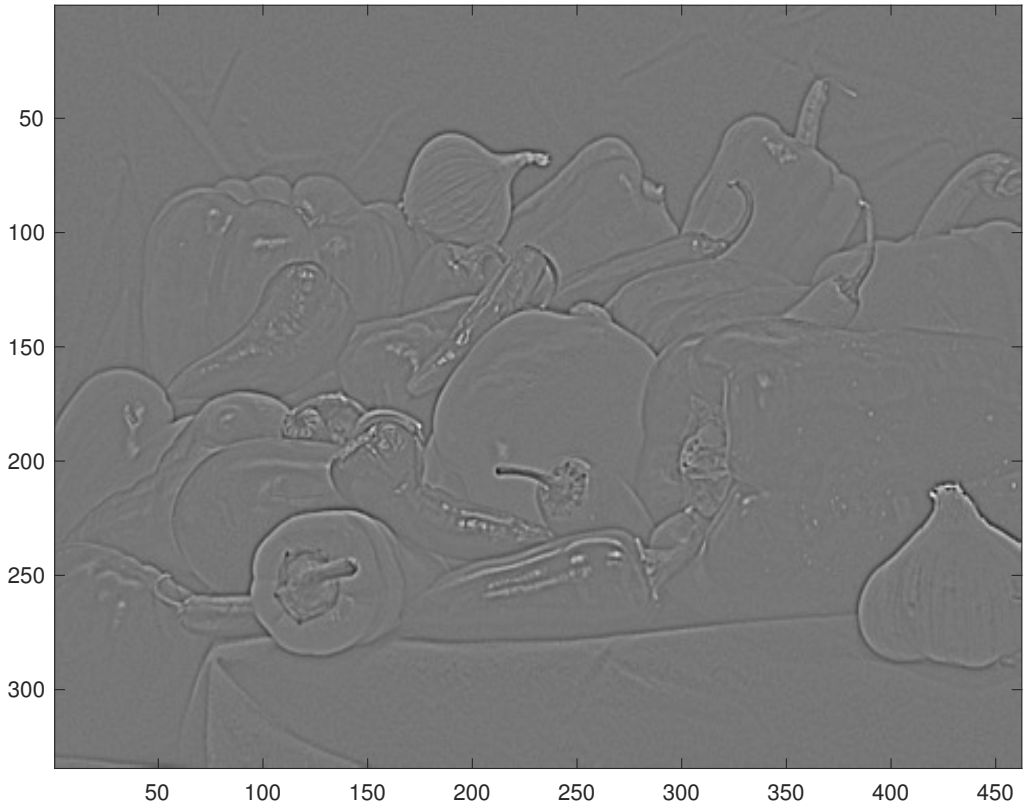

Figure 8: The result of convolving the peppers image with the narrower filter from part E.

```
colormap gray
pause
% Convolve image and plot the result

res = conv2(im_cb, dog, 'valid')

imagesc(res)
colormap gray
pause

dog3 = fspecial('gaussian', [51 51], 0.5) - fspecial('gaussian', [51 51], 1);
surf(dog3)
colormap default
pause

res = conv2(im_cb, dog3, 'valid')
imagesc(res)
colormap gray
pause

%% Load built-in natural image
```

```
im_natural = double(rgb2gray(imread('peppers.png')));
imagesc(im_natural)
colormap gray

%% Convolve and plot the result

res = conv2(im_natural, dog, 'valid')
imagesc(res)
colormap gray
pause
res = conv2(im_natural, dog3, 'valid')
imagesc(res)
colormap gray
```

2 Invariance and Representational Similarity in Deep Neural Networks

2.1 AlexNet Filters

We plot the weights of the AlexNet filters using the code below and show the result in Figure 9.

```
%% Load AlexNet
net = alexnet; % Load neural net.


%% Plot weights from the first convolution layer

% Get the network weights for the second convolutional layer
w1 = net.Layers(2).Weights;

% Scale and resize the weights for visualization
w1 = mat2gray(w1);
w1 = imresize(w1,5);

% Display a montage of network weights. There are 96 individual sets of
% weights in the first layer.
figure
montage(w1)
title('First convolutional layer weights')

% List all the layers in the network
net.Layers
```

We can see from this that there are several neurons that respond to oriented lines/bars as well as edges. These oriented bars are of various widths. There are also a few neurons that respond to circular edges in its receptive field (though these edges do not have to lie entirely within the receptive field).

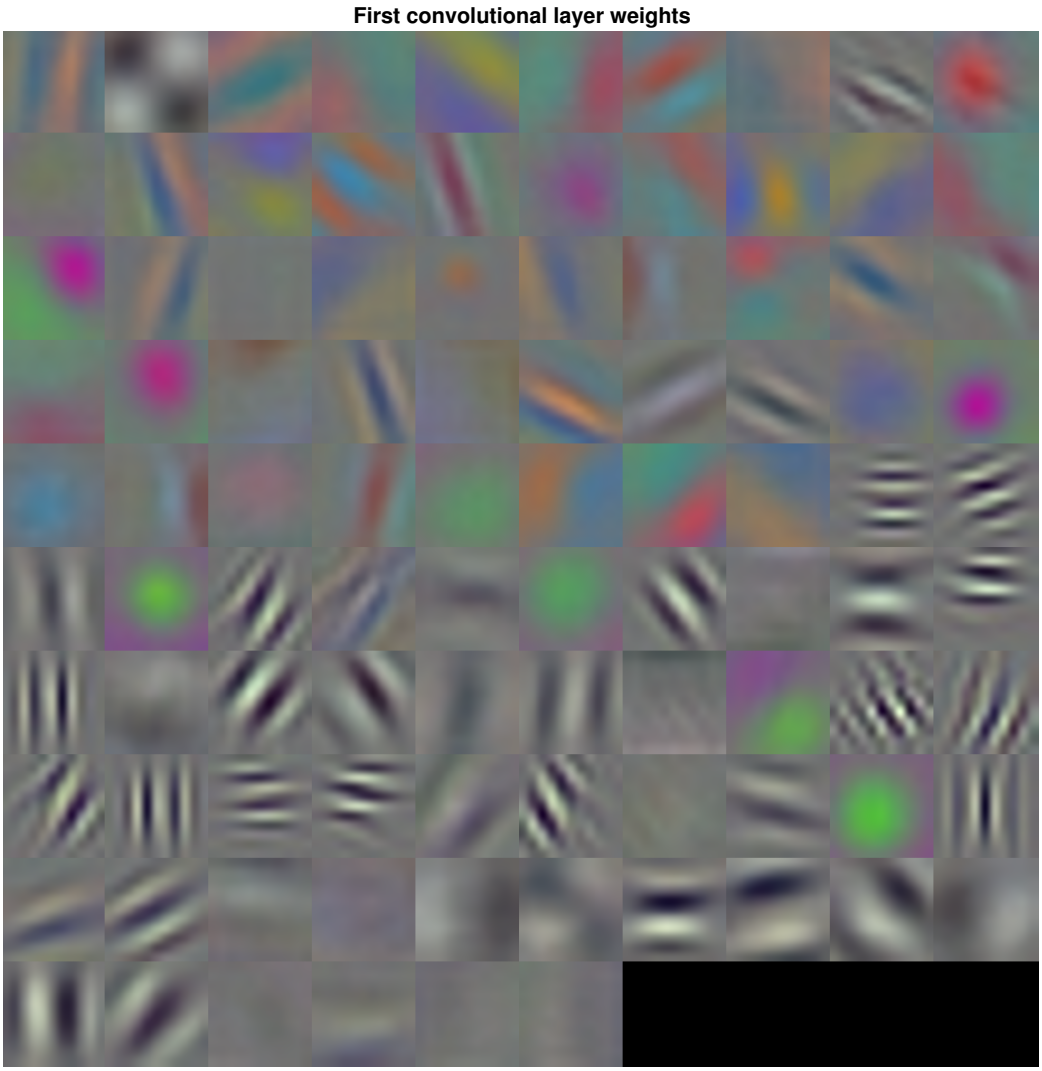

Figure 9: The weights of the first convolutional layer of AlexNet.

2.2 Applying the Network to Peppers

We apply the network to the peppers image with the following code:

```
%% Run the peppers image through AlexNet

im = imresize(imread('peppers.png'),[227 227]);
label = classify(net, im);
imshow(im)
text(10,20,char(label),'Color','white')
```

This produces Figure 10, which we can see correctly classifies the image as a bell pepper (to be fair, there are multiple bell peppers within the image, but the network wasn't trained to report

Figure 10: The result of running an image through AlexNet.

how many peppers there are). We then plot the activity of the neurons in the first convolution layer using the following code:

```
%% Plot the neural activities of different filters

f = activations(net, im, 'conv1');

for i = 1:96
    filter = i;
    subplot(131)
    imshow(w1(:,:,:,filter))
    axis square
    title(sprintf('Filter #%d',filter))

    subplot(133)
    imshow(im)
    axis square
    title('Original image')
```

```
    subplot(132)
    imagesc(f(:,:,filter))
    axis square
    title('Response')
    pause
end
```

From our investigation of the neurons in this layer, the activity of filters 3, 10, and 59 are of interest and are plotted in Figures 11, 12, and 13. From these figures, we can see that filter 3 is selective for the rough outlines of the peppers (it is certainly not 100% accurate or does so in extreme detail). Conversely, filter 10 is selectively for the bodies of the peppers, particularly those that are warm-colored. Filter 59 appears to also be selective for edges or outlines, as can be seen from the fact that it picks up the edge of the table on which the peppers lay very well.

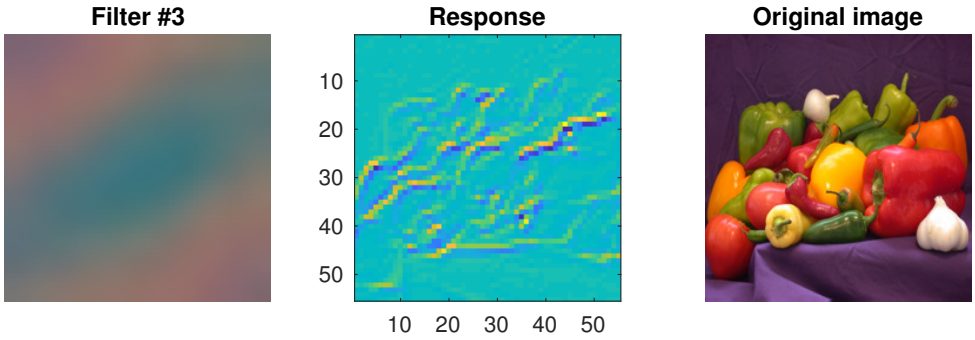

Figure 11: The activity of filter 3 in the first convolutional layer when running the pepper image through AlexNet.

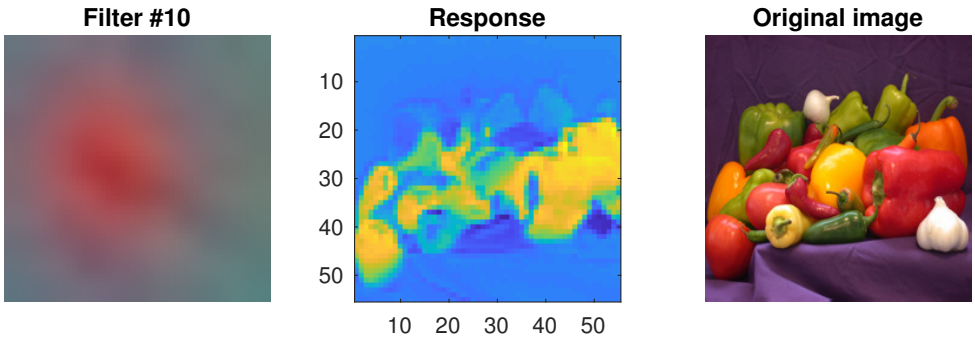

Figure 12: The activity of filter 10 in the first convolutional layer when running the pepper image through AlexNet.

2.3 Translation Invariance of AlexNet

We plot the correlation between activations for the original image and the image translated to the right by amounts

3 Back Propagation and Deep Learning

3.1 Gradient Descent

We implement gradient descent using the following code:

```
clear all;

diag=true;

%% Load MNIST dataset and plot some examples
```

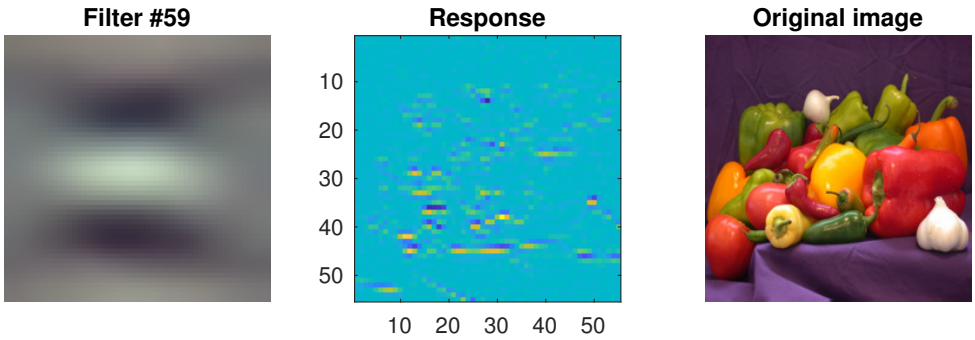

Figure 13: The activity of filter 59 in the first convolutional layer when running the pepper image through AlexNet.

```
load mnist_sevens_nines

% Sevens
figure(1)
for i = 1:16
    subplot(4,4,i)
    imshow(reshape(X_train(:,i),28,28))
end

% Nines
figure(2)
for i = 1:16
    subplot(4,4,i)
    imshow(reshape(X_train(:,i+900),28,28))
end


%% Implement gradient descent
P = size(X_train,2); % Number of examples
```

```
max_itr = 5000;
alpha = (.1/P);

Ni = size(X_train,1); % Input dimension
Nh1 = 100; % Hidden layer sizes
Nh2 = 80;

% Initialize weight matrices
weight_scale = .01;
W3 = weight_scale*randn(1,Nh2);
W2 = weight_scale*randn(Nh2,Nh1);
W1 = weight_scale*randn(Nh1,Ni);


clear L acc L_test acc_test
for i = 1:max_itr

    % Compute forward propagation for train and test data

    % Train data
    u1 = W1*X_train;
    h1 = max(u1, 0);
    u2 = W2*h1;
    h2 = max(u2, 0);
    yh = W3*h2;

    % Test data
    yh_test = W3*max(W2*max(W1*X_test, 0), 0);

    % Compute losses and accuracy on train and test data
    L(i) = (1/2)*norm(y_train-yh,'fro')^2;
    acc(i) = mean(sign(yh)==sign(y_train));

    if diag
        disp(['at epoch ',num2str(i),' loss is ',num2str(L(i))])
        if L(i)>10^200
            pause
        end
    end


    L_test(i) = (1/2)*norm(y_test-yh_test,'fro')^2;
    acc_test(i) = mean(sign(yh_test)==sign(y_test));

    % Compute backprop (just on training data)
    e = y_train - yh;

    d3 = e;
    d2 = (W3'*d3).*d_relu(u2);
    d1 = (W2'*d2).*d_relu(u1);

    % Compute gradient
    g3 = -d3*h2';
    g2 = -d2*h1';
```

```
    g1 = -d1*X_train';

    % Update weights
    W3 = W3 - alpha*g3;
    W2 = W2 - alpha*g2;
    W1 = W1 - alpha*g1;
end

% Plot the learning trajectory
t = 1:max_itr;

subplot(121)
plot(t,L/P,t,L_test/size(X_test,2),...
    'linewidth',2)
% Plot train and test loss normalized by the number of examples
legend('Train loss','Test loss')
xlabel('Epochs')
ylabel('MSE')

subplot(122)
plot(t,acc,t,acc_test,'linewidth',2)
legend('Train accuracy','Test accuracy')

function d_res = d_relu(A)
    d_res = A;
    d_res(d_res >= 0) = 1;
    d_res(d_res < 0) = 0;
end
```

This produces Figure 14, which shows the change in loss and accuracy on both the training and test data over time.

3.2 Reflections on the Neural Net

We can see from the loss and accuracy graphs that this neural net performs quite well on both training data and test data with an accuracy of 100% by the end of training. Because the model performed 100% accuracy on the test data, we can conclude that there was no overfitting, so there is no need to stop training early, though it appears we could stop at around 4000 steps as the accuracy peaked at 100% near there and maintained that level.

3.3 Larger Weight Scale

We ran the code above again changing only `weight_scale = .1;`, and plot the results in Figure 15. We can see from this that it does not substantially impact the training accuracy, but it does result in a small decrease of the test accuracy to about 99%. Stopping training early in this case, however, would not help the test accuracy as the larger weight variance removed any oscillations in the loss and accuracy that we saw before and resulted in a slow but steady increase of the accuracy that resulted in an asymptote. Thus, it seems that the smaller weight variance and not this larger weight variance might be better for the purposes of generalization based on our findings.

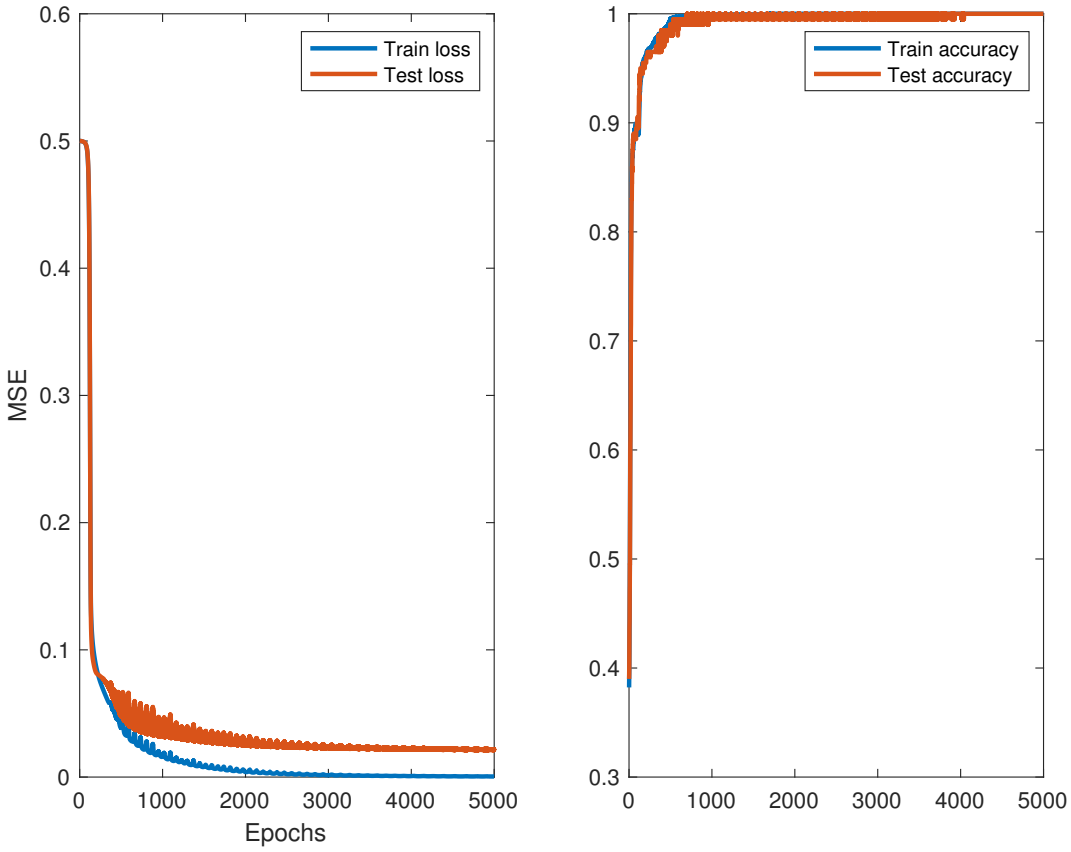

Figure 14: The loss and accuracy of both training and test data for a neural net trained on a subset of MNIST data.

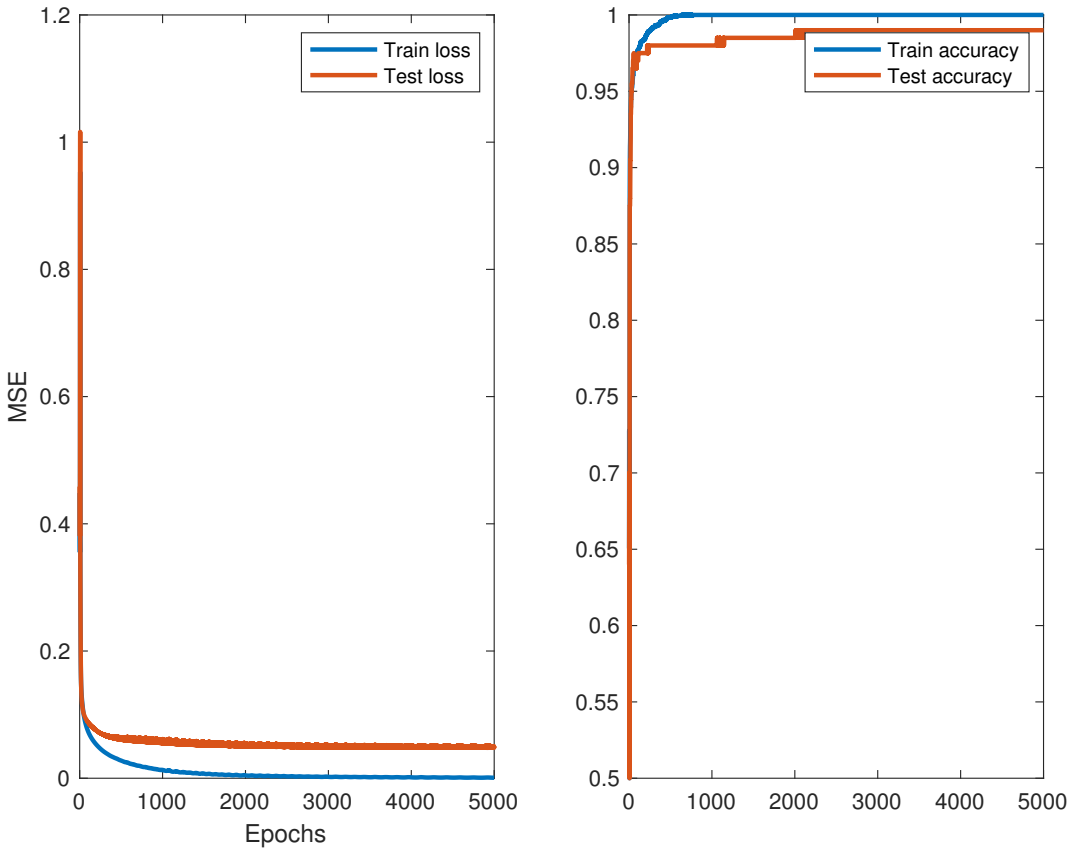

Figure 15: The loss and accuracy of both training and test data for a neural net trained on a subset of MNIST data with a larger weight scale of 0.1.