Homework 3

Due Date: April 27, 2018

Notes

-I used independent-dimension approach (spatio-temporal convolution/pooling in both 1, 2 layers)

-*main_task3.m* file and its associated files (*Feedforward.m, filterbank.m, Init_AE.m, load_data.m, normalize_image.m, pca2.m, preprocess_data.m, train_sae.m, Updata.m)* are atttached in the zip file.

-The result files *20160042_task3_sae.mat* and *20160042_task3_pca.mat* are attached in the zip file.

-Codes for visualization (*Visualization.m, draw_filter1.m, draw_filter2.m)* are also attached in the zip file.

- **I have also completed the codes for PCA implementation**, which was not mandatory.

I. main_task.m

On the header of given *main_task3.m* file, tasks 3A~3F inform us what we should do in this assignment.

```

% EE476_Audio_visual_perceptron_model
% Homework 3
%
% main_task3.m
% For video data, build the CNN architecture (2) and train weights by PCA and sparse AE
% TASK 3A : define configuration properly
% TASK 3B : cropping patches from input data
% TASK 3C : PCA, calculate 'activation_c'
% TASK 3D : PCA, calculate 'activation_p'
% TASK 3E : SAE, calculate 'activation_c'
% TASK 3F : SAE, calculate 'activation_p'
```

Figure 1 Header of main_task3.m

CNN architecture (2) is proposed on given *EE476_homework3_guideline.pdf*, page 5.

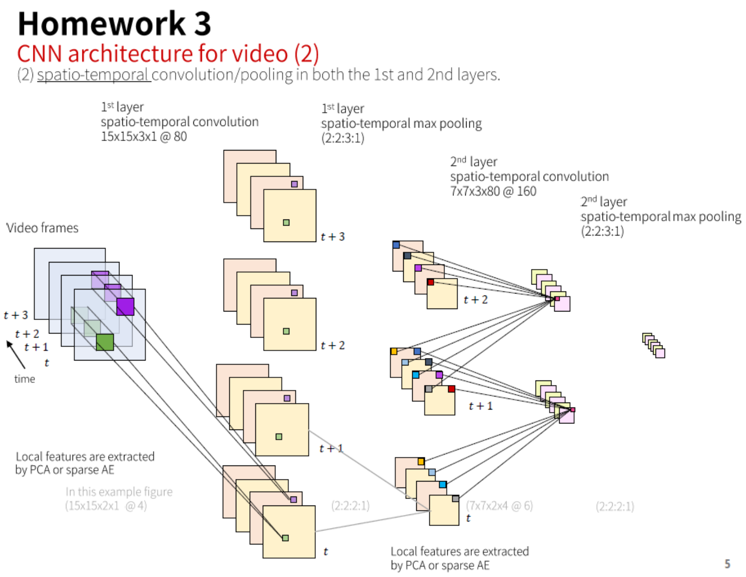

Figure 2 CNN architecture (2)

1. Setting CNN and Data Preparation

TASK 3A: define configuration properly

변수 - train_video

train_video

900x1 cell

	1
1	48x96x19 double
2	48x96x19 double
3	48x96x17 double
4	48x96x17 double
5	48x96x17 double
6	48x96x16 double
7	48x96x15 double
8	48x96x13 double
9	48x96x13 double
10	48x96x12 double
11	48x96x13 double
12	48x96x13 double
13	48x96x13 double
14	48x96x14 double
15	48x96x13 double
16	48x96x17 double
17	48x96x13 double
18	48x96x13 double
19	48x96x14 double
20	48x96x16 double
21	48x96x12 double
22	48x96x12 double
23	48x96x13 double

```
% configurations : TASK 3A - define configuration properly
config = cell(2,1);
config{1,1}.in_height=48;
config{1,1}.in_width=96;
config{1,1}.in_feat_maps=1;
config{1,1}.filter_height=15;
config{1,1}.filter_width=15;
config{1,1}.filter_frame=2;     %3
config{1,1}.out_feat_maps=80;
config{1,1}.pool_pixel=2;
config{1,1}.pool_frame=2;       %3

config{2,1}.in_height=floor((config{1,1}.in_height-config{1,1}.filter_height+1)/config{1,1}.pool_pixel);
config{2,1}.in_width=floor((config{1,1}.in_width-config{1,1}.filter_width+1)/config{1,1}.pool_pixel);
config{2,1}.in_feat_maps=config{1,1}.out_feat_maps;
config{2,1}.filter_height=7;
config{2,1}.filter_width=7;
config{2,1}.filter_frame=2;     %3
config{2,1}.out_feat_maps=160;
config{2,1}.pool_pixel=2;
config{2,1}.pool_frame=2;       %3
```

Figure 3 train_video values / Code lines for TASK 3A

I observed `train_video` values, and found out that given video size was "48 x 96 x <Frame>", where minimum value of <Frame> was 7.

On CNN architecture(2), it says filter size should be

Layer 1: "15 x 15 x 3 x 1 @80" with pooling size (2:2:3:1)

Layer 2: "7 x 7 x 3 x 1 @ 160" with pooling size (2:2:3:1).

From this information, I filled Task 3A, and defined configuration for our network.

Important thing is, that I had to fix the values for temporal dimension, since data input with size "48 x 96 x 7" would cause problem on layer 2, as size of input to layer 2 will be "17 x 41 x 1" after convolution/pooling. We cannot convolute a filter with bigger size on an input.

Thus, I modified my network as following.

Layer 1: "15 x 15 x 2 x 1 @80" with pooling size (2:2:2:1)

Layer 2: "7 x 7 x 2 x 1 @ 160" with pooling size (2:2:2:1).

Figure 3 shows how it is represented in MATLAB code.

TASK 3B: cropping patches from input data

```
% TASK 3B - cropping patches from input data
data_video = zeros(config{layer,1}.filter_height*config{layer,1}.filter_width*config{layer,1}.filter_frame*config{layer,1}.in_feat_maps, num_patch);
for idx=1:num_patch
    data_idx = ceil(rand(1,1)*num_data);
    data = train_video{data_idx,layer};     % select random data from train_video

    nheight = size(data,1);
    height = ceil(rand(1,1)*(nheight-config{layer,1}.filter_height+1));
    nwidth = size(data,2);
    width = ceil(rand(1,1)*(nwidth-config{layer,1}.filter_width+1));
    nframe = size(data,3);
    frame = ceil(rand(1,1)*(nframe-config{layer,1}.filter_frame+1));    % select random patch

    data_video(:,idx) = reshape(data(height:height+config{layer,1}.filter_height-1, ...
                                width:width+config{layer,1}.filter_width-1, ...
                                frame:frame+config{layer,1}.filter_frame-1,:), ...
        config{layer,1}.filter_height * config{layer,1}.filter_width * config{layer,1}.filter_frame * config{layer,1}.in_feat_maps, 1);
end
```

Figure 4 Code lines for TASK 3B

From raw data `train_video`, we will randomly select data and crop local patch that has same size with the filter. (The total number of patches is `num_patch`) Then we will reshape patches (3D matrices) to vectors.

Figure 4 shows how it is represented in MATLAB code.

2. PCA part (Not mandatory)

Feature Extraction on PCA

```
% feature extraction
[pc, m, v] = pca2(data_video);
weight = pc(:, 1:config{layer,1}.out_feat_maps);

params{layer,1}.weight = reshape(weight, config{layer,1}.filter_height, config{layer,1}.filter_width, config{layer,1}.filter_frame, ...
                            config{layer,1}.in_feat_maps, config{layer,1}.out_feat_maps);
params{layer,1}.bias = reshape(m, config{layer,1}.filter_height, config{layer,1}.filter_width, config{layer,1}.filter_frame, config{layer,1}.in_feat_maps);
```

Figure 5 Code lines for PCA feature extraction

To extract features using PCA, I used given function pca2. Then I resized the weight and bias as shown in Figure 5.

TASK 3C: PCA, calculate 'activation_c'

```
% convolution
%%-- TASK 3C : calculate 'activation_c' --%%
activation_c = zeros(config{layer,1}.in_height - config{layer,1}.filter_height + 1, ...
    config{layer,1}.in_width - config{layer,1}.filter_width + 1, ...
    size(data,3) - config{layer,1}.filter_frame + 1 , ...
    config{layer,1}.out_feat_maps); % convolution activation

for k=1:config{layer,1}.out_feat_maps
    for t=1:size(data,3) - config{layer,1}.filter_frame + 1
        for y=1:config{layer,1}.in_height - config{layer,1}.filter_height + 1
            for x=1:config{layer,1}.in_width - config{layer,1}.filter_width + 1
                patch = reshape(data(y:y+config{layer,1}.filter_height-1, x:x+config{layer,1}.filter_width-1, t:t+config{layer,1}.filter_frame-1, :), ...
                    config{layer,1}.filter_height * config{layer,1}.filter_width * config{layer,1}.filter_frame * config{layer,1}.in_feat_maps, 1);
                activation_c(y,x,t,k) = weight(:,k)' * (patch - m);
            end
        end
    end
end
```

Figure 6 Code lines for TASK 3C

As instructed on *EE476_homework3_guideline.pdf* page 6, Convolution of the input with the extracted features on PCA is represented as $h_{ijt}^{k} = f\left(V_k^T\left(\mathrm{x}_{[i:i+w,j:j+h,t:t+f]} - \mathrm{m}\right)\right)$, where V_k is k-th PC component, m is mean of x, f is non-linear function.

Figure 6 shows how it is represented in MATLAB code.

TASK 3D: PCA, calculate 'activation_p'

```
% pooling and non-lienar function
%%-- TASK 3D : calculate 'activation_p' --%%
activation_p = zeros(floor(size(activation_c,1)/config{layer,1}.pool_pixel), ...
    floor(size(activation_c,2)/config{layer,1}.pool_pixel), ...
    floor(size(activation_c,3)/config{layer,1}.pool_frame), ...
    size(activation_c,4)); % pooling activation
for k=1:size(activation_p,4)
    for t=1:size(activation_p,3)
        for x=1:size(activation_p,2)
            for y = 1:size(activation_p,1)
                max_pool = reshape(activation_c((y-1)*config{layer,1}.pool_pixel+1:y*config{layer,1}.pool_pixel, ...
                                          (x-1)*config{layer,1}.pool_pixel+1:x*config{layer,1}.pool_pixel, ...
                                          (t-1)*config{layer,1}.pool_frame+1:t*config{layer,1}.pool_frame, k), ...
                                      config{layer,1}.pool_pixel*config{layer,1}.pool_pixel*config{layer,1}.pool_frame,1);
                activation_p(y,x,t,k) = max(max_pool);
                activation_p(y,x,t,k) = max(0, activation_p(y,x,t,k)); %relu non-linear function
            end
        end
    end
end
train_video{data_idx, layer+1} = activation_p;
```

Figure 7 Code lines for TASK 3D

I set pooling matrices with size (2:2:2:1) with zero strides on `activation_c`. Then I got the maximum value from each matrix, to make `activation_p`. Then I deleted out values below 0, as in reLU non-linear function (no value is above 1).

Figure 7 shows how it is represented in MATLAB code. Note that I had to reshape the pooling matrices into vectors, for function 'max' gives max values for each row.

3. SAE part

Feature Extraction on SAE

```
% feature extraction
%-- you can also define 'sae_config' for nIn, nOut,
%-- nHidden, and all other training hyperparameters as you wish.
sae_config = cell(1,1);
sae_config{1,1}.nHidden = config{layer,1}.out_feat_maps;
sae_config{1,1}.lRate = 0.06;    % learning rate
sae_config{1,1}.use_sparsity = false; % option to control use of sparsity term in training autoencoder
sae_config{1,1}.AEepoch = 3000; % the number of epoch to train Autoencdoer.
sae_config{1,1}.sparsity_target = 0.01;  % Sparsity target: target activation for average hidden neuron values
sae_config{1,1}.sparsity_coeff = 10;    % Sparsity coefficients: how much do you want to weigh sparsity learning compared to reconstruction

[weight, bias, progress] = train_sae(data_video, sae_config);

params{layer,1}.weight = reshape(weight, config{layer,1}.filter_height, config{layer,1}.filter_width, config{layer,1}.filter_frame, ...
                         config{layer,1}.in_feat_maps, config{layer,1}.out_feat_maps);
params{layer,1}.bias = bias;
params{layer,1}.progress = progress;
% propagate data to the convolution and pooling layers
```

Figure 8 Code lines for SAE feature extraction

I changed the *Main.m* code I submitted for Homework2 slightly to design train_sae.

To be specific, I modified lines from *Main.m* where we load data, set configuration, and send out results.

Table 1 Modification for *train_sae.m* (Loading data)

```
%% Autoencoder training
clear all; clc; close all;

% FILL IN HERE
student_id = '20160042';
your_name = 'Inyong Koo';
audio_or_video = 'video';  % should be 'audio' or 'video'

disp(['HW#2, Your name = ' your_name ', Student ID = ' student_id ', Learning from ' audio_or_video]);

%% Part1: Load data
disp('Part1: Load data');

% Load video/audio data
if(strcmp(audio_or_video,'video'))
    load('data_video.mat'); height = 15; width = 15; % video patch
elseif(strcmp(audio_or_video,'audio'))
    load('data_audio.mat'); height = 26; width = 5; % audio patch
end
```
(Main.m)

```
function [weight, bias, progress] = train_sae(data, sae_config)
```
(train_sae.m)

Table 2 Modification for *train_sae.m* (set configuration)

```
nIn = nFeat;  nOut=nIn; nHidden = 80;  % Autoencoder size specification
lRate = 0.01; % learning rate

%[MODIFY HERE]
use_sparsity = false; % option to control use of sparsity term in training autoencoder
AEepoch = 50000; % the number of epoch to train Autoencdoer. Modify this only if you think training 50000epoch is not enough.
sparsity_target = 0.1;  % Sparsity target: target activation for average hidden neuron values
sparsity_coeff = 10;    % Sparsity coefficients: how much do you want to weigh sparsity learning compared to reconstruction
```
(Main.m)

```
nIn = nFeat;  nOut = nIn; nHidden = sae_config{1,1}.nHidden;  % Autoencoder size specification
lRate = sae_config{1,1}.lRate; % learning rate

%[MODIFY HERE]
use_sparsity = sae_config{1,1}.use_sparsity; % option to control use of sparsity term in training autoencoder
AEepoch = sae_config{1,1}.AEepoch; % the number of epoch to train Autoencdoer. Modify this only if you think training 50000epoch is not enough.
sparsity_target = sae_config{1,1}.sparsity_target;  % Sparsity target: target activation for average hidden neuron values
sparsity_coeff = sae_config{1,1}.sparsity_coeff;    % Sparsity coefficients: how much do you want to weigh sparsity learning compared to reconstruction
```
(train_sae.m)

Table 3 Modification for *train_sae.m* (sending out results)

```
weight = AE.layers{1}.w';
bias = AE.layers{1}.b;
progress = cost;
```
(train_sae.m)

`progress` is unnecessary for convolution, but I wanted to save the cost how reconstruction rate reduces.

Figure 8 shows how it is represented in MATLAB code. Note that I set `use_sparcity` false for simplicity.

TASK 3E: SAE, calculate 'activation_c'

```
        % convolution
        %%-- TASK 3E : calculate 'activation_c' --%%
        %%- you can use 'conv', 'conv2', or 'convn' functions for
        %%- efficient calcuation
        activation_c = zeros(config{layer,1}.in_height - config{layer,1}.filter_height + 1, ...
            config{layer,1}.in_width - config{layer,1}.filter_width + 1, ...
            size(data,3) - config{layer,1}.filter_frame + 1 , ...
            config{layer,1}.out_feat_maps); % convolution activation

        for k=1:size(activation_c,4)
            activation_c(:,:,:,k) = convn(data, params{layer,1}.weight(:,:,:,:,k), 'valid') + bias(k,1);
        end
```

Figure 9 Code lines for TASK 3E

As instructed on *EE476_homework3_guideline.pdf* page 6, Convolution of the input with the extracted features on SAE is represented as $h_{ijt}^{k} = f\left((W^{k} * x)_{ijk} + b_k\right)$, where W_k is k-th weight, b_k is k-th bias, f is non-linear function.

Figure 9 shows how it is represented in MATLAB code. Note that I used `convn` function with `'valid'` parameter.

TASK 3F: SAE, calculate 'activation_p'

```
        % pooling and non-lienar function
        %%-- TASK 3F : calculate 'activation_p' --%%
        activation_p = zeros(floor(size(activation_c,1)/config{layer,1}.pool_pixel), ...
            floor(size(activation_c,2)/config{layer,1}.pool_pixel), ...
            floor(size(activation_c,3)/config{layer,1}.pool_frame), ...
            size(activation_c,4)); % pooling activation
        for k=1:size(activation_p,4)
            for t=1:size(activation_p,3)
                for x=1:size(activation_p,2)
                    for y = 1:size(activation_p,1)
                        max_pool = reshape(activation_c((y-1)*config{layer,1}.pool_pixel+1:y*config{layer,1}.pool_pixel, ...
                                                        (x-1)*config{layer,1}.pool_pixel+1:x*config{layer,1}.pool_pixel, ...
                                                        (t-1)*config{layer,1}.pool_frame+1:t*config{layer,1}.pool_frame, k), ...
                                                    config{layer,1}.pool_pixel*config{layer,1}.pool_pixel*config{layer,1}.pool_frame,1);
                        activation_p(y,x,t,k) = max(max_pool);
                        activation_p(y,x,t,k) = max(0, activation_p(y,x,t,k)); %relu non-linear function
                    end
                end
            end
        end
```

Figure 10 Code lines for TASK 3F

TASK 3F is basically same with TASK 3D. Please see Figure 10.

II. Visualization

I made few files to visualize my data.

```matlab
%% Visualization

cost = params{1,1}.progress;
cost2 = params{2,1}.progress;
epoch = 3000;

% Draw cost curve
h1 = figure(1);
subplot(2,1,1);
plot(cost(1:epoch,1), 'r'); title('Reconstruction cost (Layer 1)','FontSize',18);
subplot(2,1,2);
plot(cost2(1:epoch,1), 'b'); title('Reconstruction cost (Layer 2)','FontSize',18);

% Draw filters (first layer)
draw_filters1(params);

% Draw filters (second layer)
fromNode = 1;

draw_filters2(params,fromNode);
```

Figure 11 Code lines of Visualization.m

After loading *20160042_task3_sae.mat,* executing *Visualization.m* produces total 13 figures.

Figure 1 shows cost curve of SAE result., figure 2~5 shows filters of first layer (total 80 maps x 2 frames = 160), and figure 6~13 shows filters of second layer, connected to a node `fromNode` in first layer (total 160 maps x 2 frames = 320).

To see PCA result, just enter two commands after loading *20160042_task3_pca.mat*

```
draw_filters1(params);
draw_filters2(params, <Node you want to see>);
```

II. Execution Result

The Execution time was proportional to 3 parameters:

`num_data`, `num_patch`, and `sae_config{1,1}.AEepoch`.

I also had to modify `sae_config{1,1}.lRate` or the result didn't show up. This is because input for second layer is determined by training result of layer 1, and if it didn't show meaningful result, The convolution values will diverse too much, resulting NAN reconstruction error on second layer.

I used the preset values (`num_data`: 900, `num_patch`: 30000, `sae_config{1,1}.AEepoch`: 30000, `sae_config{1,1}.lRate`: 0.1) and got invalid result.

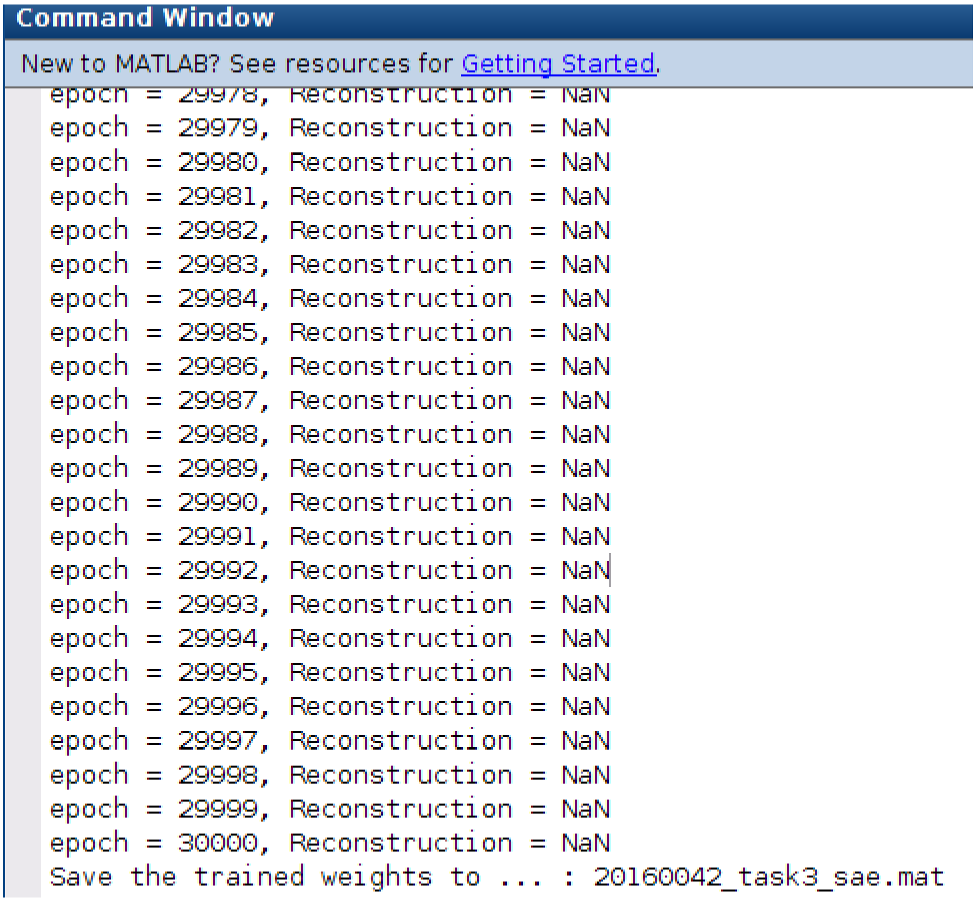

Figure 12 Invalid result

For I had limited time, and could not afford trying large values, and thus set values as following.

`num_data`: 20,

`num_patch`: 3000,

`sae_config{1,1}.AEepoch`: 3000,

`sae_config{1,1}.lRate`: 0.06

And this is the result.

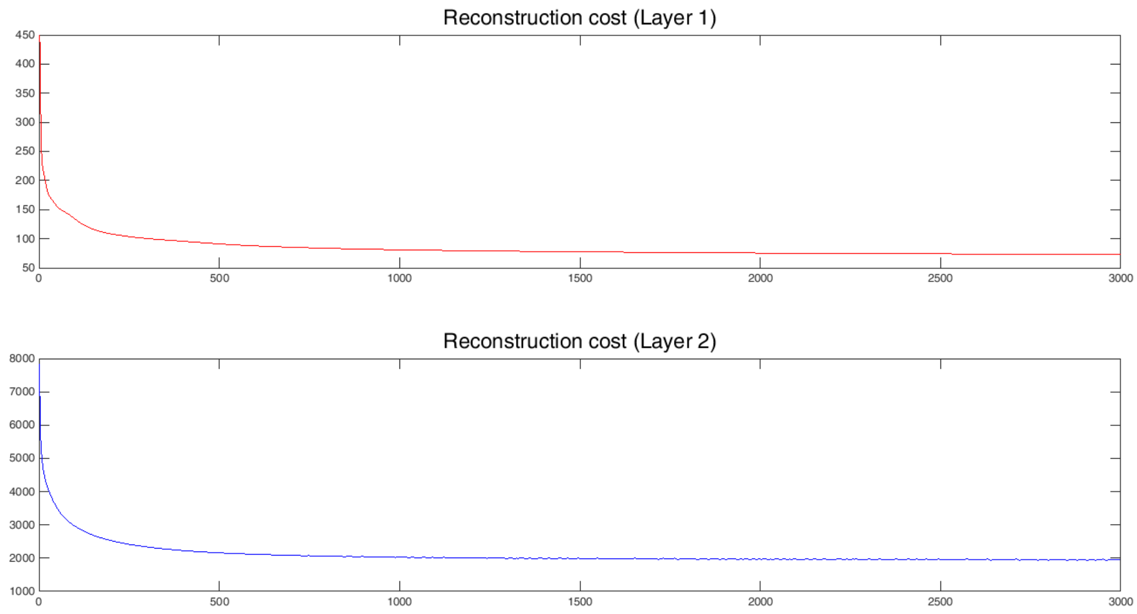

Figure 13 Reconstruction cost for SAE training

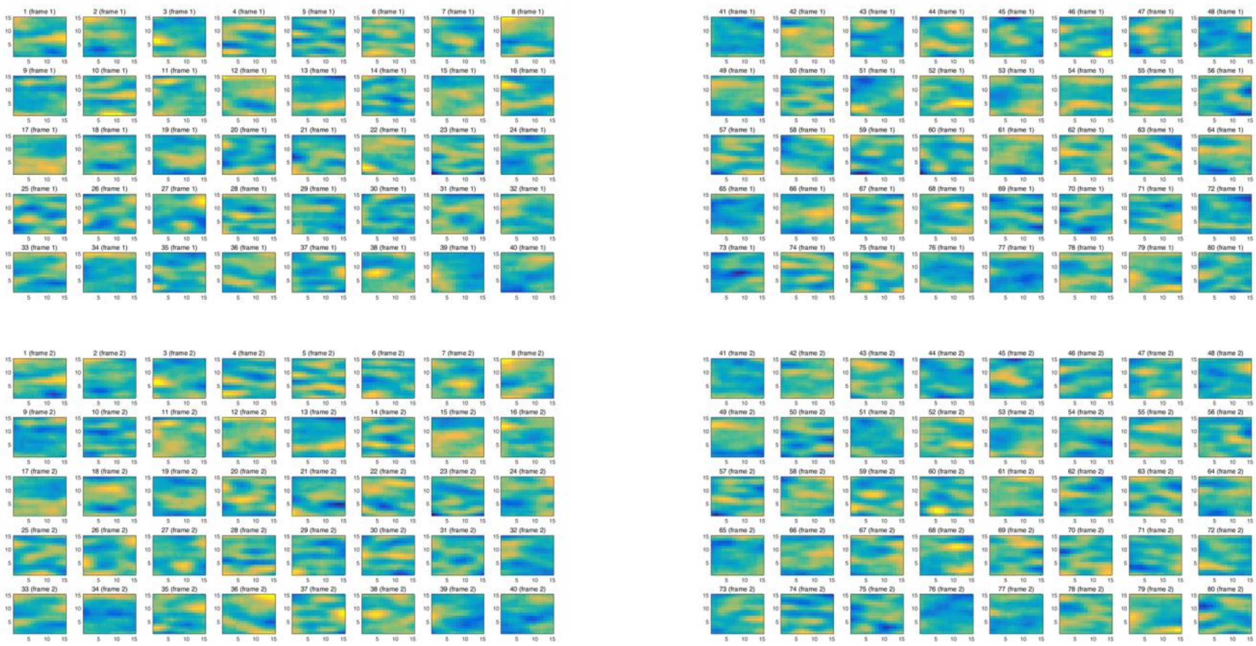

Figure 14 SAE, Filter maps (Layer 1)

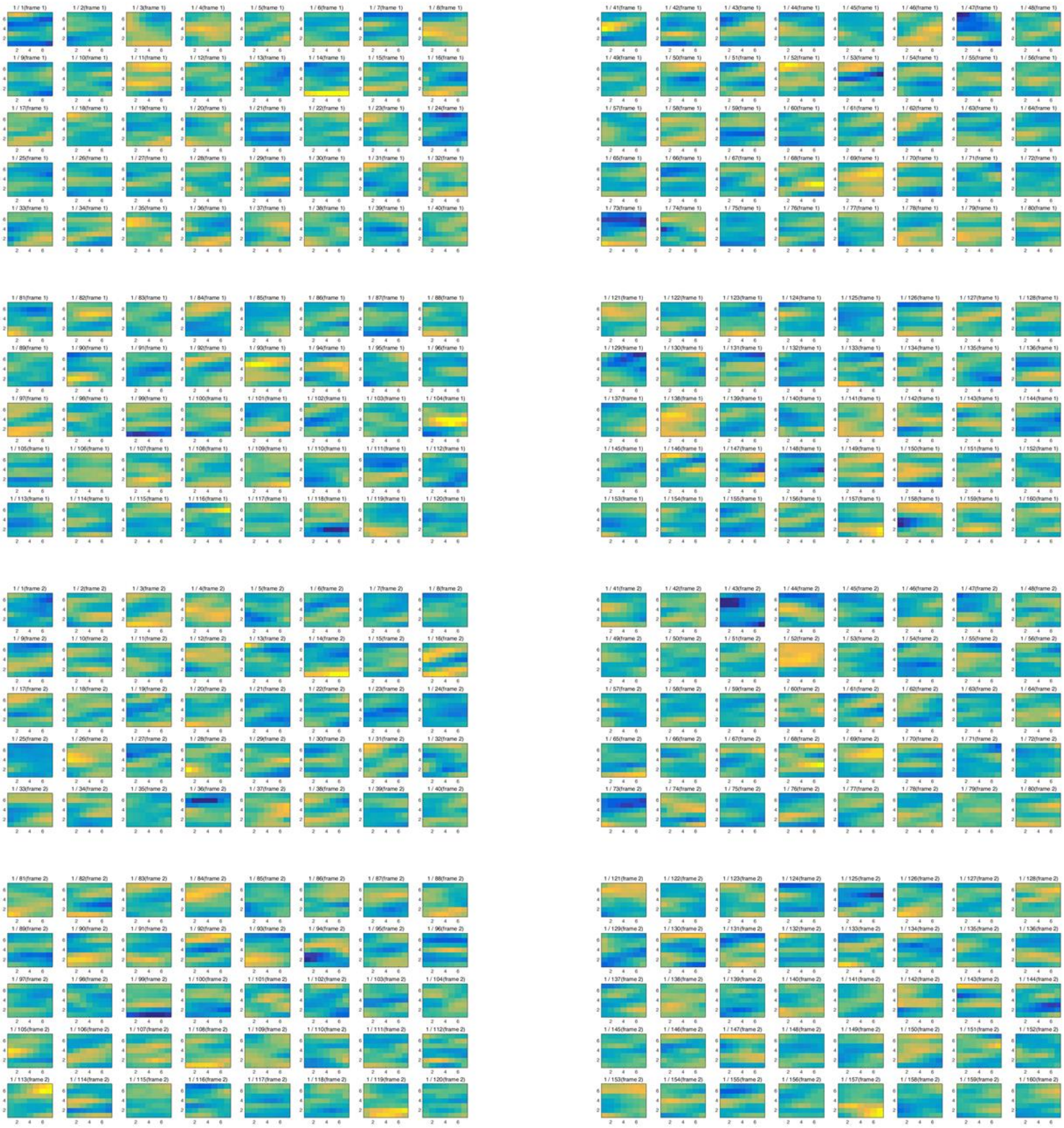

Figure 15 SAE, Filter maps (Layer 2, node 1)

Additionally, This is the result using PCA.

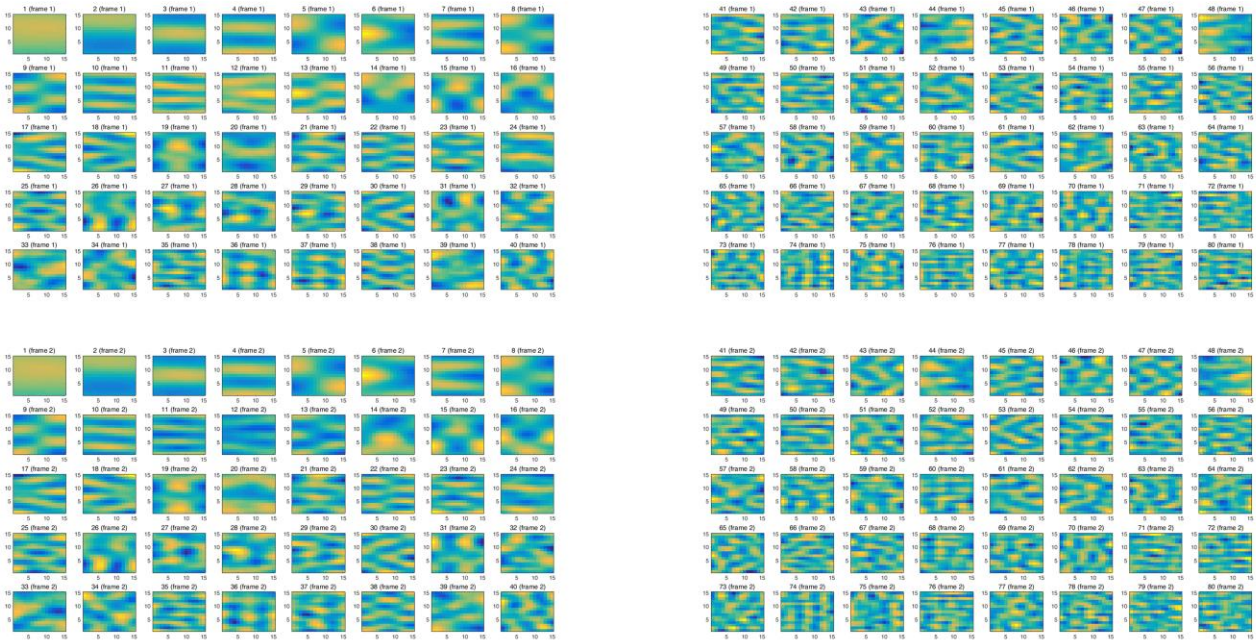

Figure 16 PCA, Filter maps (Layer 1)

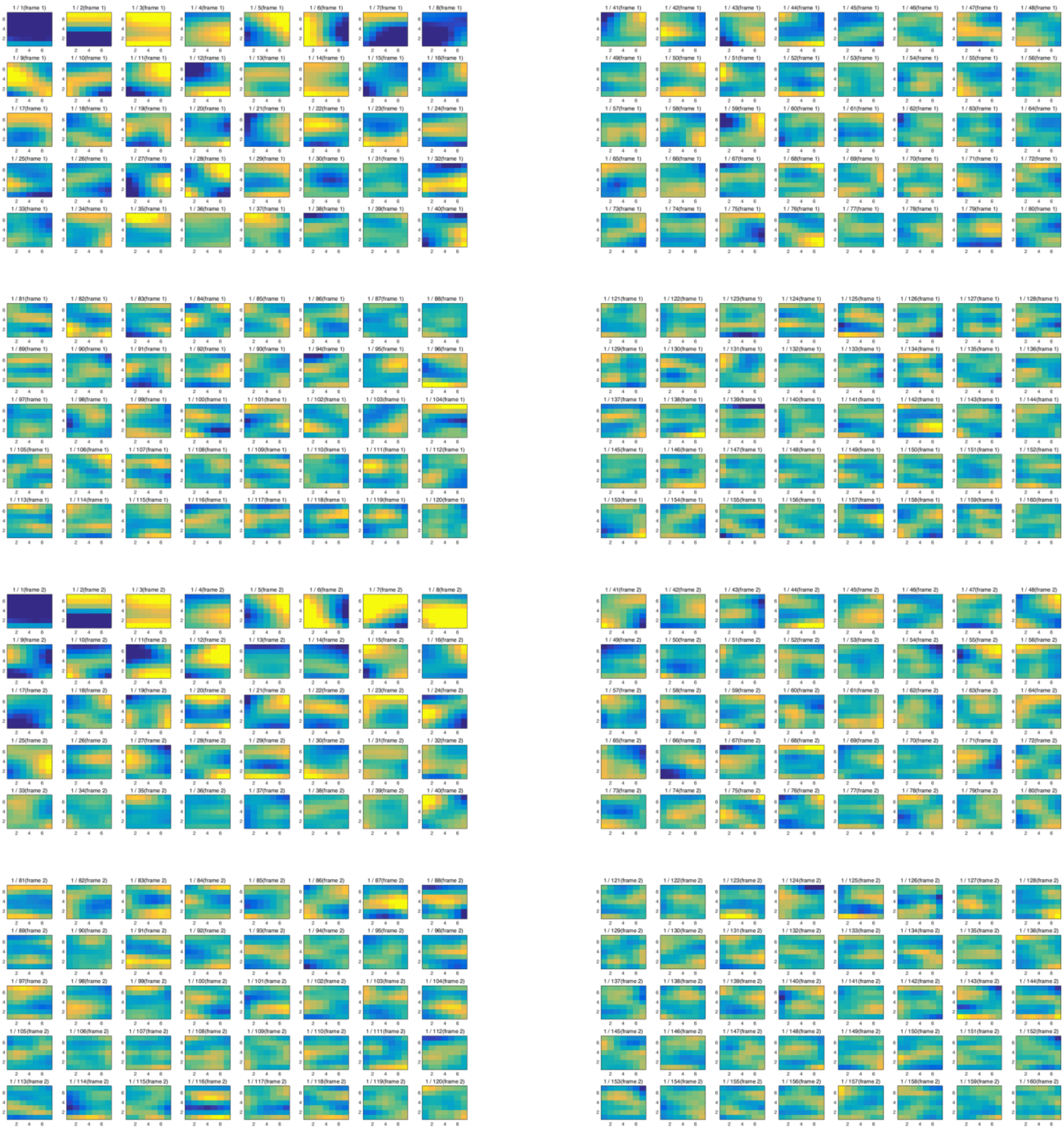

Figure 17 PCA, Filter maps (Layer 2, node 1)

Appendix. Main.m (Submission file for Homework 2)

```
%% Autoencoder training
clear all; clc; close all;

% FILL IN HERE
student_id = '20160042';
your_name = 'Inyong Koo';
audio_or_video = 'video';  % should be 'audio' or 'video'

disp(['HW#2, Your name = ' your_name ', Student ID = ' student_id ', Learning from ' audio_or_video]);

%% Part1: Load data
disp('Part1: Load data');

% Load video/audio data
if(strcmp(audio_or_video,'video'))
    load('data_video.mat'); height = 15; width = 15; % video patch
elseif(strcmp(audio_or_video,'audio'))
    load('data_audio.mat'); height = 26; width = 5; % audio patch
end

% Get data size descriptor
nData = size(data,2);
nFeat = size(data,1);

%% Part2: Feature standardization  [DO NOT MODIFY]
disp('Part2: Feature standardization');

data_mean = mean(data,2);
data_std = std(data,1,2);
data = (data - repmat(data_mean,[1 nData]))./repmat(data_std, [1 nData]);
% now each dimension of data have zero mean, and unit variance

%% Part3: Hyperparameter setting [DO NOT MODIFY except 'MODIFY HERE']
disp('Part3: Hyperparameter setting');
nIn = nFeat;  nOut=nIn; nHidden = 80;  % Autoencoder size specification
lRate = 0.01; % learning rate

%[MODIFY HERE]
use_sparsity = false; % option to control use of sparsity term in training autoencoder
AEepoch = 50000; % the number of epoch to train Autoencdoer. Modify this only if you think training 50000epoch is not enough.
sparsity_target = 0.1;  % Sparsity target: target activation for average hidden neuron values
sparsity_coeff = 10;    % Sparsity coefficients: how much do you want to weigh sparsity learning compared to reconstruction

% Variable for monitoring learning
cost = zeros(AEepoch,2);  % 1st column: reconstruction cost, 2nd column: sparsity cost
mean_hidden = zeros(AEepoch,1);

%% Part4: Initialization of Autoencoder  [DO NOT MODIFY except 'Fill in Here']
disp('Part4: Initialization of Autoencoder');

AE.nLayer = 2;
AE.layers = cell(AE.nLayer,1);
AE.memory_dim = [nIn nHidden nOut];

AE = Init_AE(AE, nData);  % Fill in Here

AE.layers{2}.w = AE.layers{1}.w';  % tied weight

%% Part5: Training Autoencoder [DO NOT MODIFY except 'Fill in Here']
disp('Part5: Training Autoencoder');

% Phase1: Feedforward
% Phase2: Error Backpropagation
% Phase3: Update weights (towards minimize cost)
for epoch = 1:AEepoch
    AE.activation{1} = data;
    %% Feed-forward
    for layerIdx=1:AE.nLayer
        AE = Feedforward(AE.activation{layerIdx}, AE, layerIdx);  % Fill in feedForward.m
    end

    avg_act_hidden = mean(AE.activation{2},2); % Average activation of hidden neurons over all data
    mean_hidden(epoch) = mean(avg_act_hidden);
```

```

    error_signal = -(AE.activation{1} - AE.activation{3}); % delta

    cost(epoch,1) = sum(sum(error_signal.^2))/nData;  % Reconstruction cost
    if(use_sparsity)
        cost(epoch,2) = sum(sum((sparsity_target.* log(sparsity_target./avg_act_hidden) + (1 - sparsity_target).* log((1-sparsity_target) ./ (1-avg_act_hidden)))))/nData; % Sparsity cost
    end

    % Monitor learning progress
    if(use_sparsity)
        disp(['epoch = ' num2str(epoch) ', Reconstruction = ' num2str(cost(epoch,1)) ', Sparsity = ' num2str(cost(epoch,2))]);
    else
        disp(['epoch = ' num2str(epoch) ', Reconstruction = ' num2str(cost(epoch,1))]);
    end

    %% Backpropagation
    % Backpropagation in 2nd layer
    AE.layers{2}.err = error_signal; % Here is hint how error_signal looks like
    AE.layers{2}.grad_w = AE.layers{2}.err * AE.activation{2}'; % FILL IN HERE. gradient for 2nd layer weight
    AE.layers{2}.grad_b = sum(AE.layers{2}.err, 2); % FILL IN HERE. gradiet for 2nd layer bias

    % Backpropagation in 1st layer
    if(use_sparsity)
        sparsity_err = repmat(- (sparsity_target./avg_act_hidden) + (1 - sparsity_target)./(1-avg_act_hidden), 1, nData); % FILL IN HERE. sparsity error
        AE.layers{1}.err = (AE.layers{2}.w'*AE.layers{2}.err - sparsity_coeff*sparsity_err) .* AE.activation{2} .* (1-AE.activation{2});    % FILL IN HERE. error for 1st layer weight: (weight * delta  + sparsity_coeff * sparsity_error) * derivative of sigmoid function
    else
        AE.layers{1}.err = (AE.layers{2}.w'*AE.layers{2}.err) .* (AE.activation{2} .* (1-AE.activation{2})); % FILL IN HERE. error for 1st layer weight: weight * delta  * derivative of sigmoid function
    end

    AE.layers{1}.grad_w = AE.layers{1}.err*AE.activation{1}' ; % Fill in here (gradient for 1st layer weight)
    AE.layers{1}.grad_b = sum(AE.layers{1}.err, 2); % Fill in here (gradient for 1st layer bias)

    %% Update [DO NOT MODIFY]
    AE = Update(AE, nData, lRate);
end
save(['Result_' student_id '_' audio_or_video],'AE','cost','mean_hidden');  % Save your
training result [DO NOT MODIFY]
```