Uniform LSUN (Locallay-Structured Unitary Network)

LSUN, like PCA, is a fully linear transformation.

Please do not forget to run **setpath** in the top directory of this package, and then return to this directory.

Requirements: MATLAB R2022a

Contact address: Shogo MURAMATSU,

Faculty of Engineering, Niigata University,

8050 2-no-cho Ikarashi, Nishi-ku,

Niigata, 950-2181, JAPAN

http://msiplab.eng.niigata-u.ac.jp


```
clc, clear
isVisible = ~true;
support.fcn_download_img
```

```
kodim01.png already exists in ./data/
kodim02.png already exists in ./data/
kodim03.png already exists in ./data/
kodim04.png already exists in ./data/
kodim05.png already exists in ./data/
kodim06.png already exists in ./data/
kodim07.png already exists in ./data/
kodim08.png already exists in ./data/
kodim09.png already exists in ./data/
kodim10.png already exists in ./data/
kodim11.png already exists in ./data/
kodim12.png already exists in ./data/
kodim13.png already exists in ./data/
kodim14.png already exists in ./data/
kodim15.png already exists in ./data/
kodim16.png already exists in ./data/
kodim17.png already exists in ./data/
kodim18.png already exists in ./data/
kodim19.png already exists in ./data/
kodim20.png already exists in ./data/
kodim21.png already exists in ./data/
kodim22.png already exists in ./data/
kodim23.png already exists in ./data/
kodim24.png already exists in ./data/
See Kodak Lossless True Color Image Suite
```

```
imgfile = "./data/kodim01.png";
img = im2double(rgb2gray(imread(imgfile)));
```

```
% rblk = imd';
% nblocks = 9;
% blk = im2col(rblk,[14 14] ,'distinct')
nof = 1;
ky = 2*nof+1; % # of overlapping blocks (odd number)
kx = 2*nof+1;
disp([ky kx])
```

```
     3     3
```

```
blksz = [4 4]; % Block size
% # of coeffs
nCoefs = 4;
```

Local block PCA for reference

```
% % Create Sub-images
% s_img = createsubimg(blk);
[szy,szx] = size(img);
lbpcaimg = zeros(szy,szx);
for iBlkCol = 1:szx/blksz(2)
    for iBlkRow = 1:szy/blksz(1)
        % Extract ky x kx blocks
        subblks = fcn_extract_blks_(img,[iBlkRow,iBlkCol],blksz,[ky,kx]);
        % Reshape
        colblks = im2col(subblks,blksz,"distinct");
        % PCA
        %Vpca = pca(colblks.')
        mu = mean(colblks,2);
        colblkszm = colblks - mu;
        C = cov(colblkszm.');
        [~,S,V] = svd(C,"econ");
        [~,idxS] = sort(diag(S),"descend");
        V = V(:,idxS(1:nCoefs));
        %norm(Vpca - Vsvd,'fro')
        % Approxiamtion
        targetblk = fcn_extract_blks_(img,[iBlkRow,iBlkCol],blksz,[1 1]);
        targetblk = reshape(V*V.'*(targetblk(:)-mu)+mu,size(targetblk));
        % Place block
        lbpcaimg = fcn_place_blks_(lbpcaimg,targetblk,[iBlkRow,iBlkCol],blksz);
    end
end
```

Global block PCA for reference

```
% Reshape
```

```
colblks = im2col(img,blksz,"distinct");
% PCA
%Vpca = pca(colblks.')
mu = mean(colblks,2);
colblkszm = colblks - mu;
C = cov(colblkszm.');
[~,S,V] = svd(C,"econ");
[~,idxS] = sort(diag(S),"descend");
V = V(:,idxS(1:nCoefs));
%norm(Vpca - Vsvd,'fro')
% Approxiamtion
gbpcaimg = col2im(V*V.'*(colblks-mu)+mu,blksz,size(img),"distinct");
```

Locally Structured Unitary Network (LSUN) for 2-D Grayscale image

```
% Decimation factor (Strides)
stride = blksz; % [My Mx]

% Number of overlapping blocks (Polyphase order plus one)
ovlpFactor = [ky kx];

% Max epochs
maxEpochs = 128;

% Standard deviation of initial angles
stdInitAng = 0;

% No DC-leakage
noDcLeakage = true;
```

Bivariate lattice-structure of filter banks

As a base system for LSUN, let us adopt a multidimensional linear-phase paraunitary filter banks (MD-LPPUFB) (or non-separable oversampled lapped transform (NSOLT)) of type-I with the number of channels (the numbers of even and odd symmetric channels are identical to each other) and polyphase order (even):

$$\mathbf{E}(z_\mathrm{v}, z_\mathrm{h}) = \left(\prod_{k_\mathrm{h}=1}^{N_\mathrm{h}/2} \mathbf{V}_{2k_\mathrm{h}}^{\{\mathrm{h}\}} \bar{\mathbf{Q}}(z_\mathrm{h}) \mathbf{V}_{2k_\mathrm{h}-1}^{\{\mathrm{h}\}} \mathbf{Q}(z_\mathrm{h})\right)\left(\prod_{k_\mathrm{v}=1}^{N_\mathrm{v}/2} \mathbf{V}_{2k_\mathrm{v}}^{\{\mathrm{v}\}} \bar{\mathbf{Q}}(z_\mathrm{v}) \mathbf{V}_{2k_\mathrm{v}-1}^{\{\mathrm{v}\}} \mathbf{Q}(z_\mathrm{v})\right) \mathbf{V}_0 \mathbf{E}_0,$$

$$\mathbf{R}(z_\mathrm{v}, z_\mathrm{h}) = \mathbf{E}^T(z_\mathrm{v}^{-1}, z_\mathrm{h}^{-1}),$$

where

- $\mathbf{E}(z_\mathrm{v}, z_\mathrm{h})$: Type-I polyphase matrix of the analysis filter bank
- $\mathbf{R}(z_\mathrm{v}, z_\mathrm{h})$: Type-II polyphase matrix in the synthesis filter bank

- $z_d \in \mathbb{C}, d \in \{\mathrm{v}, \mathrm{h}\}$: The parameter of Z-transformation direction
- $N_d \in \mathbb{N}, d \in \{\mathrm{v}, \mathrm{h}\}$: Polyphase order in direction d (number of overlapping blocks)
- $\mathbf{V}_0 = \begin{pmatrix} \mathbf{W}_0 & \mathbf{O} \\ \mathbf{O} & \mathbf{U}_0 \end{pmatrix} \begin{pmatrix} \mathbf{I}_{M/2} \\ \mathbf{O} \\ \mathbf{I}_{M/2} \\ \mathbf{O} \end{pmatrix} \in \mathbb{R}^{P\times M}, \mathbf{V}_n^{\{d\}} = \begin{pmatrix} \mathbf{I}_{P/2} & \mathbf{O} \\ \mathbf{O} & \mathbf{U}_n^{\{d\}} \end{pmatrix} \in \mathbb{R}^{P\times P}, d \in \{\mathrm{v}, \mathrm{h}\}$, where $\mathbf{W}_0, \mathbf{U}_0, \mathbf{U}_n^{\{d\}} \in \mathbb{R}^{P/2\times P/2}$ are orthonromal matrices.
- $\mathbf{Q}(z) = \mathbf{B}_P \begin{pmatrix} \mathbf{I}_{P/2} & \mathbf{O} \\ \mathbf{O} & z^{-1}\mathbf{I}_{P/2} \end{pmatrix} \mathbf{B}_P,\ \bar{\mathbf{Q}}(z) = \mathbf{B}_P \begin{pmatrix} z\mathbf{I}_{P/2} & \mathbf{O} \\ \mathbf{O} & \mathbf{I}_{P/2} \end{pmatrix} \mathbf{B}_P,\ \mathbf{B}_P = \frac{1}{\sqrt{2}} \begin{pmatrix} \mathbf{I}_{P/2} & \mathbf{I}_{P/2} \\ \mathbf{I}_{P/2} & -\mathbf{I}_{P/2} \end{pmatrix}$

【Example】For $P/2 = 3$, a parametric orthonormal matrix $\mathbf{U}(\boldsymbol{\theta}, \boldsymbol{\mu})$ can be constructed by

$$\mathbf{U}(\boldsymbol{\theta}, \boldsymbol{\mu}) := \begin{pmatrix} \mu_1 & 0 & 0 \\ 0 & \mu_1 & 0 \\ 0 & 0 & \mu_2 \end{pmatrix} \begin{pmatrix} 1 & 0 & 0 \\ 0 & \cos\theta_2 & -\sin\theta_2 \\ 0 & \sin\theta_2 & \cos\theta_2 \end{pmatrix} \begin{pmatrix} \cos\theta_1 & 0 & -\sin\theta_1 \\ 0 & 1 & 0 \\ \sin\theta_1 & 0 & \cos\theta_1 \end{pmatrix} \begin{pmatrix} \cos\theta_0 & -\sin\theta_0 & 0 \\ \sin\theta_0 & \cos\theta_0 & 0 \\ 0 & 0 & 1 \end{pmatrix},$$

$$\mathbf{U}(\boldsymbol{\theta}, \boldsymbol{\mu})^T = \begin{pmatrix} \cos\theta_0 & \sin\theta_0 & 0 \\ -\sin\theta_0 & \cos\theta_0 & 0 \\ 0 & 0 & 1 \end{pmatrix} \begin{pmatrix} \cos\theta_1 & 0 & \sin\theta_1 \\ 0 & 1 & 0 \\ -\sin\theta_1 & 0 & \cos\theta_1 \end{pmatrix} \begin{pmatrix} 1 & 0 & 0 \\ 0 & \cos\theta_2 & \sin\theta_2 \\ 0 & -\sin\theta_2 & \cos\theta_2 \end{pmatrix} \begin{pmatrix} \mu_0 & 0 & 0 \\ 0 & \mu_1 & 0 \\ 0 & 0 & \mu_2 \end{pmatrix},$$

where $\boldsymbol{\theta} \in \mathbb{R}^{(P-2)P/8}$ and $\boldsymbol{\mu} = \{-1, 1\}^{P/2}$. For the sake of simplification, the sign parameters μ_k are fixed to -1 for $\mathbf{U}_n^{\{d\}}$ witn odd n, otherwise they are fixed to $+1$.

Partial differentiation can be, for examle, conducted as

$$\frac{\partial}{\partial\theta_1}\mathbf{U}(\boldsymbol{\theta}, \boldsymbol{\mu})^T = \begin{pmatrix} \cos\theta_0 & \sin\theta_0 & 0 \\ -\sin\theta_0 & \cos\theta_0 & 0 \\ 0 & 0 & 1 \end{pmatrix} \begin{pmatrix} -\sin\theta_1 & 0 & \cos\theta_1 \\ 0 & 0 & 0 \\ -\cos\theta_1 & 0 & -\sin\theta_1 \end{pmatrix} \begin{pmatrix} 1 & 0 & 0 \\ 0 & \cos\theta_2 & \sin\theta_2 \\ 0 & -\sin\theta_2 & \cos\theta_2 \end{pmatrix} \begin{pmatrix} \mu_0 & 0 & 0 \\ 0 & \mu_1 & 0 \\ 0 & 0 & \mu_2 \end{pmatrix}.$$

A locally-structured unitary network (LSUN) allows to change the parameters block by block.

【References】

- MATLAB SaivDr Package: https://github.com/msiplab/SaivDr
- S. Muramatsu, K. Furuya and N. Yuki, "Multidimensional Nonseparable Oversampled Lapped Transforms: Theory and Design," in IEEE Transactions on Signal Processing, vol. 65, no. 5, pp. 1251-1264, 1 March1, 2017, doi: 10.1109/TSP.2016.2633240.
- S. Muramatsu, T. Kobayashi, M. Hiki and H. Kikuchi, "Boundary Operation of 2-D Nonseparable Linear-Phase Paraunitary Filter Banks," in IEEE Transactions on Image Processing, vol. 21, no. 4, pp. 2314-2318, April 2012, doi: 10.1109/TIP.2011.2181527.

- S. Muramatsu, M. Ishii and Z. Chen, "Efficient parameter optimization for example-based design of nonseparable oversampled lapped transform," 2016 IEEE International Conference on Image Processing (ICIP), Phoenix, AZ, 2016, pp. 3618-3622, doi: 10.1109/ICIP.2016.7533034.
- Furuya, K., Hara, S., Seino, K., & Muramatsu, S. (2016). Boundary operation of 2D non-separable oversampled lapped transforms. *APSIPA Transactions on Signal and Information Processing, 5*, E9. doi:10.1017/ATSIP.2016.3.
- S. Muramatsu, A. Yamada and H. Kiya, "A design method of multidimensional linear-phase paraunitary filter banks with a lattice structure," in IEEE Transactions on Signal Processing, vol. 47, no. 3, pp. 690-700, March 1999,doi: 10.1109/78.747776.

Definition of custom layers and networks

Use a custom layer of Deep Learning Toolbox to implement Analysis LSUN.

Definition of layers w/ Learnable properties

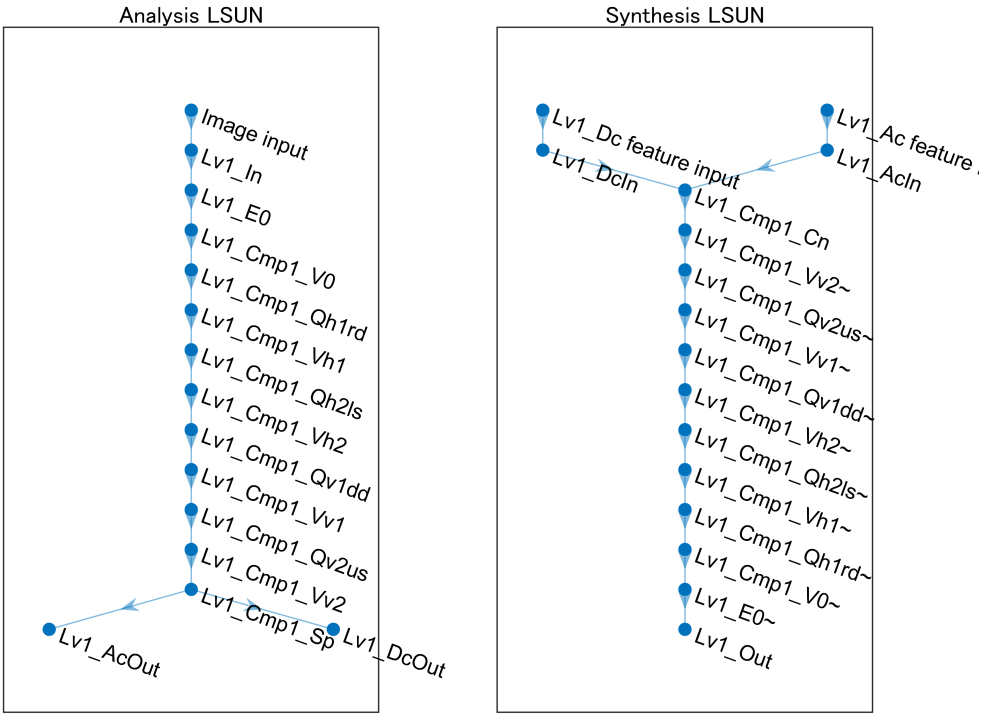
- Initial rotation: $\mathbf{V}_{0,b}$ (tansacnet.lsun.lsunInitialRotationLayer)
- Intermediate rotation: $\mathbf{V}_{n,b}^{\{d\}}$ (tansacnet.lsun.lsunIntermediateRotationLayer)

Definition of layers w/o Learnable properties

- Bivariate DCT (2-D IDCT): $\mathbf{E}_0$ (tansacnet.lsun.lsunBlockDctLayer)
- Vertical up extension: $\mathbf{Q}(z_{\mathrm{v}})$ (tansacnet.lsun.lsunAtomExtensionLayer)
- Vertical down extension: $\bar{\mathbf{Q}}(z_{\mathrm{v}})$ (tansacnet.lsun.lsunAtomExtensionLayer)
- Horizontal left extension: $\mathbf{Q}(z_{\mathrm{h}})$ (tansacnet.lsun.lsunAtomExtensionLayer)
- Horizontal right extension: $\bar{\mathbf{Q}}(z_{\mathrm{h}})$ (tansacnet.lsun.lsunAtomExtensionLayer)

```
import tansacnet.lsun.*
analysislgraph = fcn_createlsunlgraph2d([],...
    'InputSize',[szy szx],...
    'Stride',stride,...
    'OverlappingFactor',ovlpFactor,...
    'NumberOfVanishingMoments',noDcLeakage,...
    'Mode','Analyzer');
synthesislgraph = fcn_createlsunlgraph2d([],...
    'InputSize',[szy szx],...
    'Stride',stride,...
    'OverlappingFactor',ovlpFactor,...
    'NumberOfVanishingMoments',noDcLeakage,...
    'Mode','Synthesizer');
figure
subplot(1,2,1)
plot(analysislgraph)
title('Analysis LSUN')
subplot(1,2,2)
```

```
plot(synthesislgraph)
title('Synthesis LSUN')
```




```
% Construction of synthesis network.
analysisnet = dlnetwork(analysislgraph);

% Initialize
nLearnables = height(analysisnet.Learnables);
for iLearnable = 1:nLearnables
    if analysisnet.Learnables.Parameter(iLearnable)=="Angles"
        analysisnet.Learnables.Value(iLearnable) = ...
            cellfun(@(x) x+stdInitAng*randn(size(x)), ...
            analysisnet.Learnables.Value(iLearnable),'UniformOutput',false);
    end
end
```

```
import tansacnet.lsun.*
% Construction of analysis network
analysislgraph = layerGraph(analysisnet);
synthesislgraph = fcn_cpparamsana2syn(synthesislgraph,analysislgraph);
```

```
Copy angles from Lv1_Cmp1_V0 to Lv1_Cmp1_V0~
Copy angles from Lv1_Cmp1_Vh1 to Lv1_Cmp1_Vh1~
Copy angles from Lv1_Cmp1_Vh2 to Lv1_Cmp1_Vh2~
```

```
Copy angles from Lv1_Cmp1_Vv1 to Lv1_Cmp1_Vv1~
Copy angles from Lv1_Cmp1_Vv2 to Lv1_Cmp1_Vv2~
```

```
synthesisnet = dlnetwork(synthesislgraph);
```

Confirmation of the adjoint relation (perfect reconstruction)

```
x = rand([szy szx],'double');
dlx = dlarray(x,'SSCB'); % Deep learning array (SSCB: Spatial,Spatial,Channel,Batch)
[dls{1:2}] = analysisnet.predict(dlx);
dly = synthesisnet.predict(dls{:});
display("MSE: " + num2str(mse(dlx,dly)))
```

```
"MSE: 8.9731e-10"
```

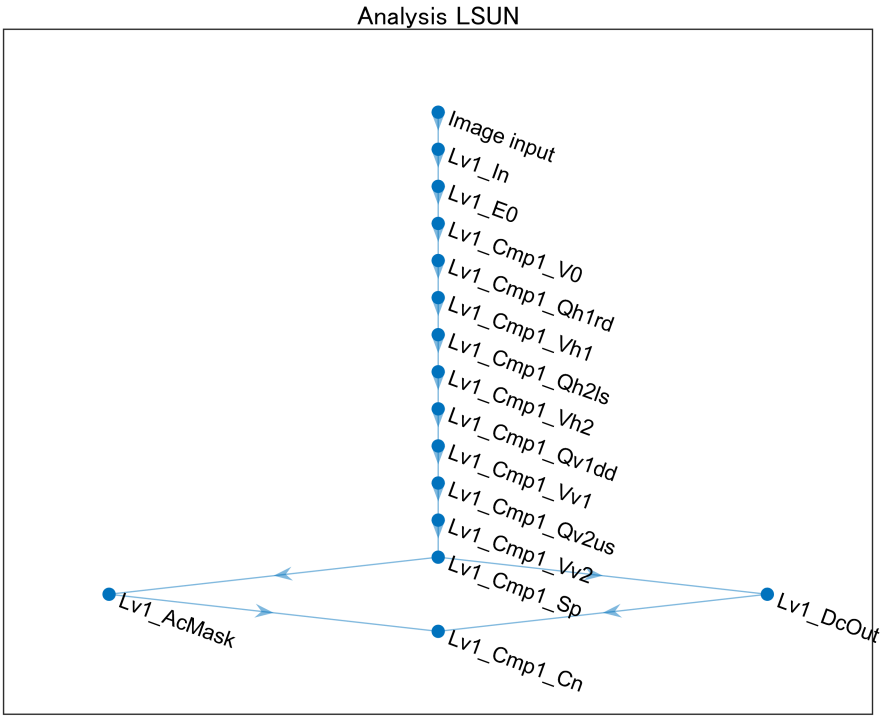
Parameter optimization and approximation

```
import tansacnet.lsun.*
analysislgraph = layerGraph(analysisnet);

% Coefficient masking
nChsTotal = prod(stride);
coefMask = [ones(nCoefs,1); zeros(nChsTotal-nCoefs,1)];
coefMask = [coefMask(1:2:end); coefMask(2:2:end)];
%nLevels = 1;
%for iLv = nLevels:-1:1
iLv = 1;
strLv = sprintf('Lv%0d_',iLv);
% For AC
analysislgraph = analysislgraph.replaceLayer([strLv 'AcOut'],...
    maskLayer('Name',[strLv 'AcMask'],'Mask',coefMask(2:end),...
    'NumberOfChannels',nChsTotal-1));
%strLvPre = strLv;
%end

% Output layer
iCmp = 1;
strCmp = sprintf('Cmp%0d_',iCmp);
%analysislgraph = analysislgraph.addLayers([...
%      lsunChannelConcatenation2dLayer('Name',[strLv strCmp 'Cn']) ...
%      lsunRegressionLayer('Coefficient output')
%     ]);
analysislgraph = analysislgraph.addLayers(...
      lsunChannelConcatenation2dLayer('Name',[strLv strCmp 'Cn']));
analysislgraph = analysislgraph.connectLayers(...
    [strLv 'AcMask' ], [strLv strCmp 'Cn/ac']);
analysislgraph = analysislgraph.connectLayers(...
    [strLv 'DcOut' ], [strLv strCmp 'Cn/dc']);
```

```
figure
plot(analysislgraph)
title('Analysis LSUN')
```



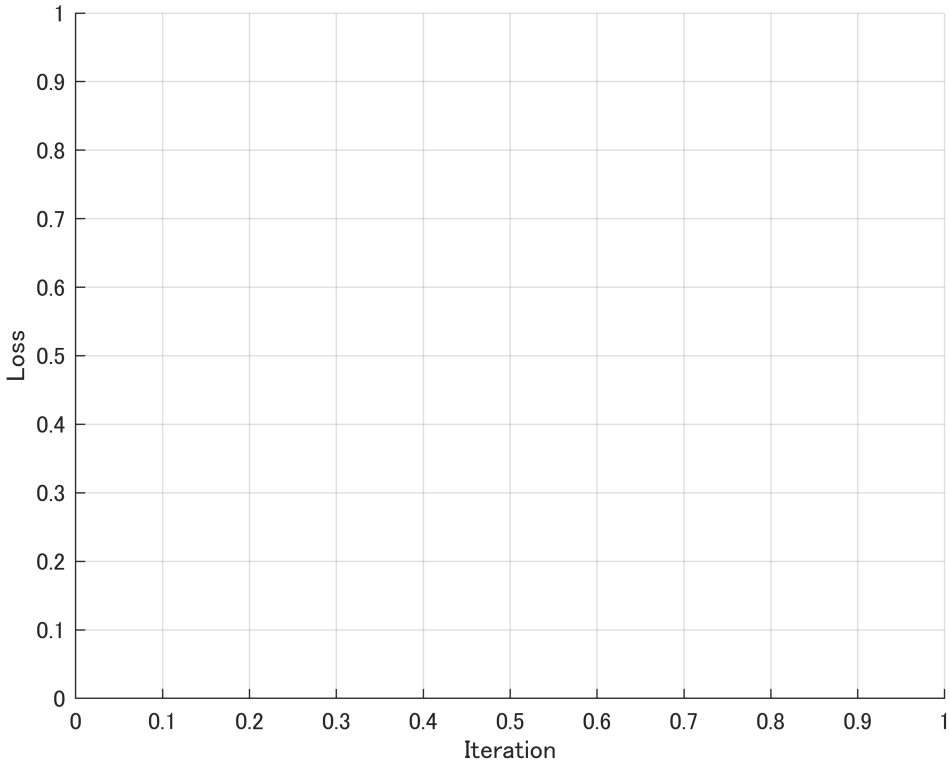

```
% Image data store
fs = matlab.io.datastore.FileSet(imgfile);
imds = imageDatastore(imgfile,"ReadFcn",@(x) im2double(rgb2gray(imread(imgfile))));
%patchds = randomPatchExtractionDatastore(imds,imds,[szy szx],'PatchesPerImage',1);
figure
%minibatch = preview(patchds);
%inputimg = minibatch.InputImage;
imshow(preview(imds));
```

```
drawnow
%figure
%responses = minibatch.ResponseImage;
%montage(responses,'Size',[2 4]);
```

```
dlX = dlarray(gpuArray(readimage(imds,1)),"SSCB");
trainnet = dlnetwork(analysislgraph,dlX);
trainnet.Initialized
```

```
ans = logical
   1
```

```
%miniBatchSize = 1;
%mbq = minibatchqueue(patchds,...
%    'MiniBatchSize',miniBatchSize,...
%    'MiniBatchFormat',{'SSBC','SSBC'});
figure
lineLossTrain = animatedline('Color',[0.85 0.325 0.098]);
ylim([0 inf])
xlabel("Iteration")
ylabel("Loss")
grid on
```




```
% Training
velocity = [];
iteration = 0.01;
momentum = 0.9;
decay = 0.01;
initialLearnRate = 1e-2;
start = tic;

% Loop over epochs.
for epoch = 1:maxEpochs
    % Shuffle data.
    %shuffle(mbq);
    shuffle(imds);

    % Loop over mini-batches.
    while hasdata(imds) % hasdata(mbq)
        iteration = iteration + 1;

        % Read mini-batch of data.
        %[dlX, T] = next(mbq);
        dlX = dlarray(gpuArray(read(imds)),"SSCB");

        % Evaluate the model gradients, state, and loss using dlfeval and the
```

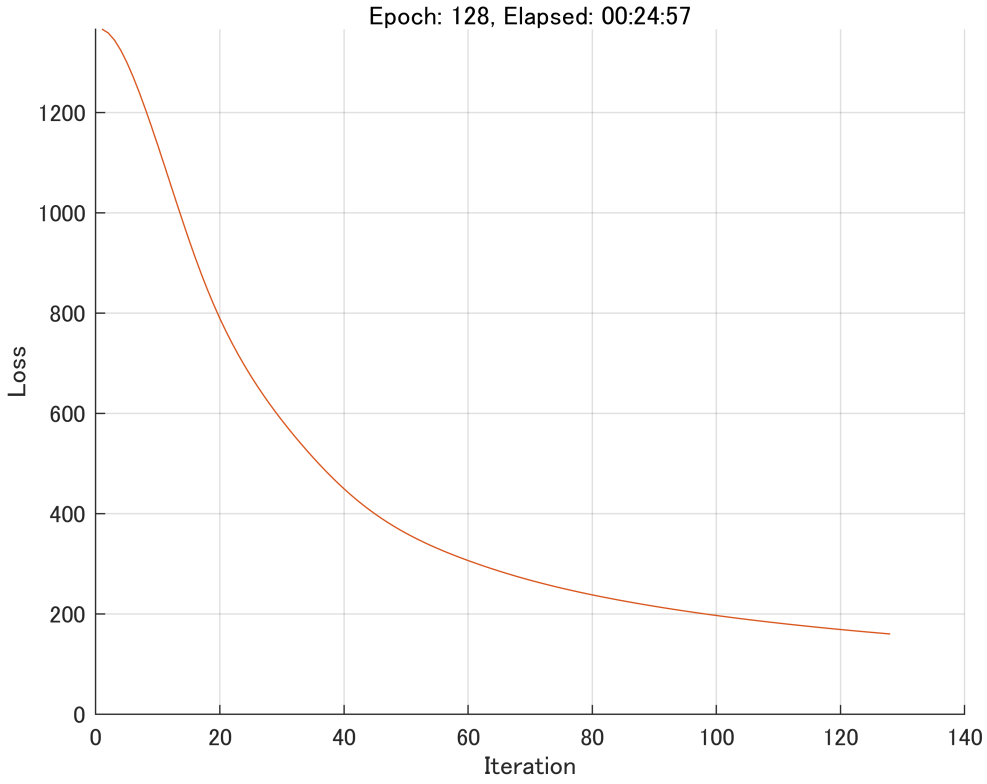
```
        % modelGradients function and update the network state.
        [gradients,loss] = dlfeval(@modelGradients,trainnet,dlX);

        % Determine learning rate for time-based decay learning rate schedule.
        learnRate = initialLearnRate/(1 + decay*iteration);

        % Update the network parameters using the SGDM optimizer.
        [trainnet,velocity] = sgdmupdate(trainnet,gradients,velocity,learnRate,momentum);

        % Display the training progress.
        D = duration(0,0,toc(start),'Format','hh:mm:ss');
        addpoints(lineLossTrain,iteration,loss)
        title("Epoch: " + epoch + ", Elapsed: " + string(D))
        drawnow
    end

    reset(imds);
end
```



Approximation by LSUN-base linear autoencoder

```
import tansacnet.lsun.*
lsunlgraph = fcn_createlsunlgraph2d([],...
    'InputSize',[szy szx],...
    'Stride',stride,...
    'OverlappingFactor',ovlpFactor,...
```

```
    'NumberOfVanishingMoments',noDcLeakage,...
    'Mode','Whole');
trainlgraph = layerGraph(trainnet);
lsunlgraph = fcn_cpparamsana2syn(lsunlgraph,trainlgraph);
```

```
Copy angles from Lv1_Cmp1_V0 to Lv1_Cmp1_V0~
Copy angles from Lv1_Cmp1_Vh1 to Lv1_Cmp1_Vh1~
Copy angles from Lv1_Cmp1_Vh2 to Lv1_Cmp1_Vh2~
Copy angles from Lv1_Cmp1_Vv1 to Lv1_Cmp1_Vv1~
Copy angles from Lv1_Cmp1_Vv2 to Lv1_Cmp1_Vv2~
```

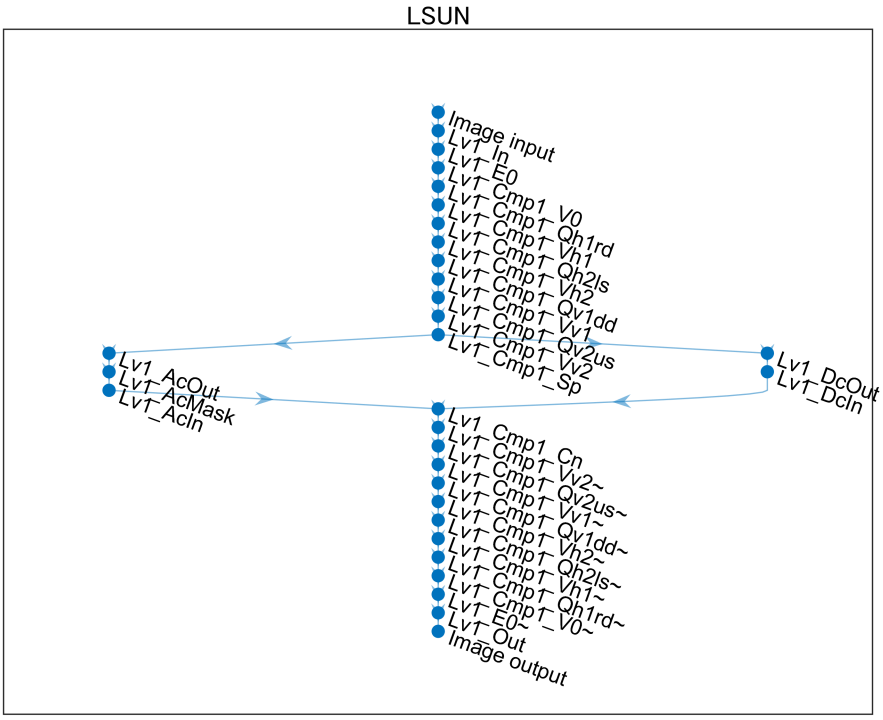
```
lsunlgraph = fcn_cpparamssyn2ana(lsunlgraph,lsunlgraph);
```

```
Copy angles from Lv1_Cmp1_V0~ to Lv1_Cmp1_V0
Copy angles from Lv1_Cmp1_Vh1~ to Lv1_Cmp1_Vh1
Copy angles from Lv1_Cmp1_Vh2~ to Lv1_Cmp1_Vh2
Copy angles from Lv1_Cmp1_Vv1~ to Lv1_Cmp1_Vv1
Copy angles from Lv1_Cmp1_Vv2~ to Lv1_Cmp1_Vv2
```

```


%nLevels = 1;
%for iLv = nLevels:-1:1
iLv = 1;
strLv = sprintf('Lv%0d_',iLv);
lsunlgraph = lsunlgraph.disconnectLayers([strLv 'AcOut'],[strLv 'AcIn']);
% For AC
lsunlgraph = lsunlgraph.addLayers(...
    maskLayer('Name',[strLv 'AcMask'],'Mask',coefMask(2:end),...
    'NumberOfChannels',nChsTotal-1));
lsunlgraph = lsunlgraph.connectLayers([strLv 'AcOut'],[strLv 'AcMask']);
lsunlgraph = lsunlgraph.connectLayers([strLv 'AcMask'],[strLv 'AcIn']);
%strLvPre = strLv;
%end

figure
plot(lsunlgraph)
title('Linear autoencoder with LSUN')
```




```
```

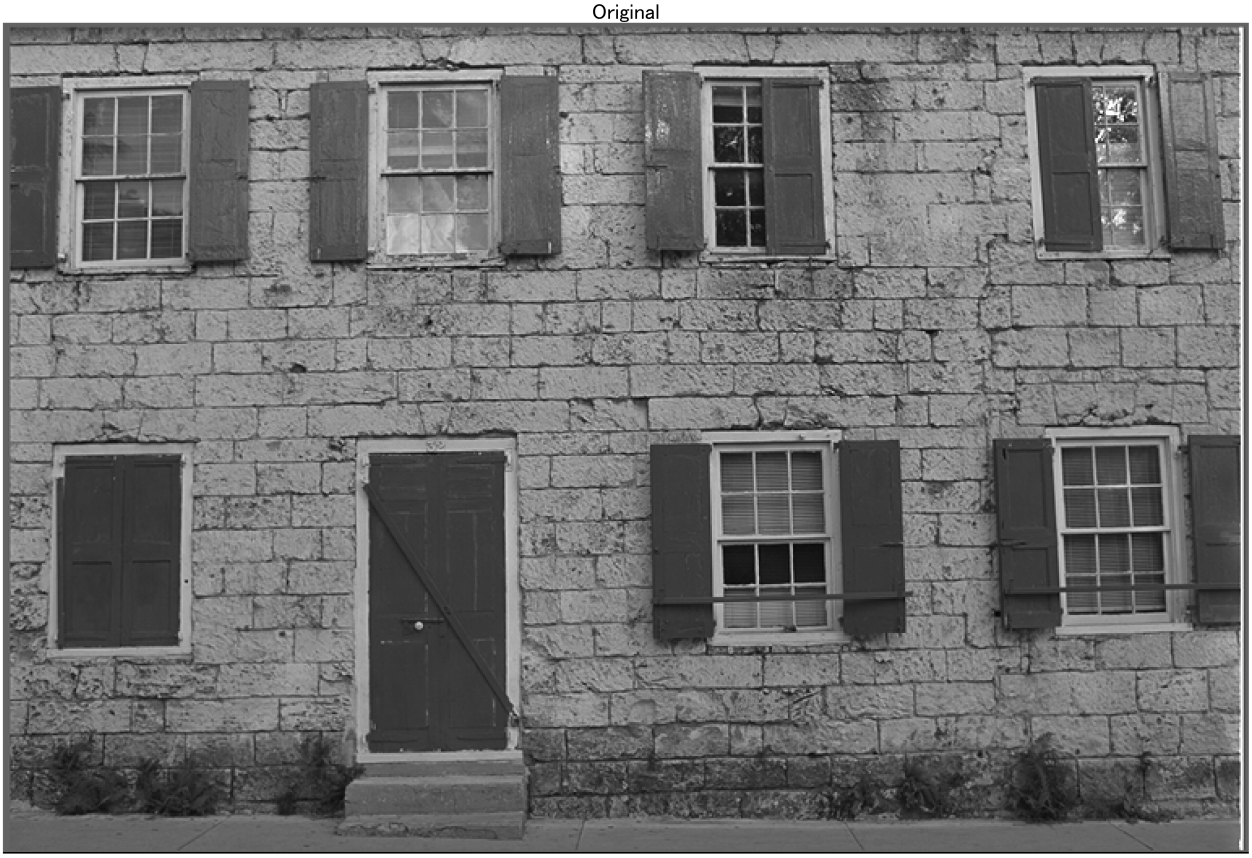
Predict

```
lsunnet = assembleNetwork(lsunlgraph);
lsunaimg = lsunnet.predict(img);
```

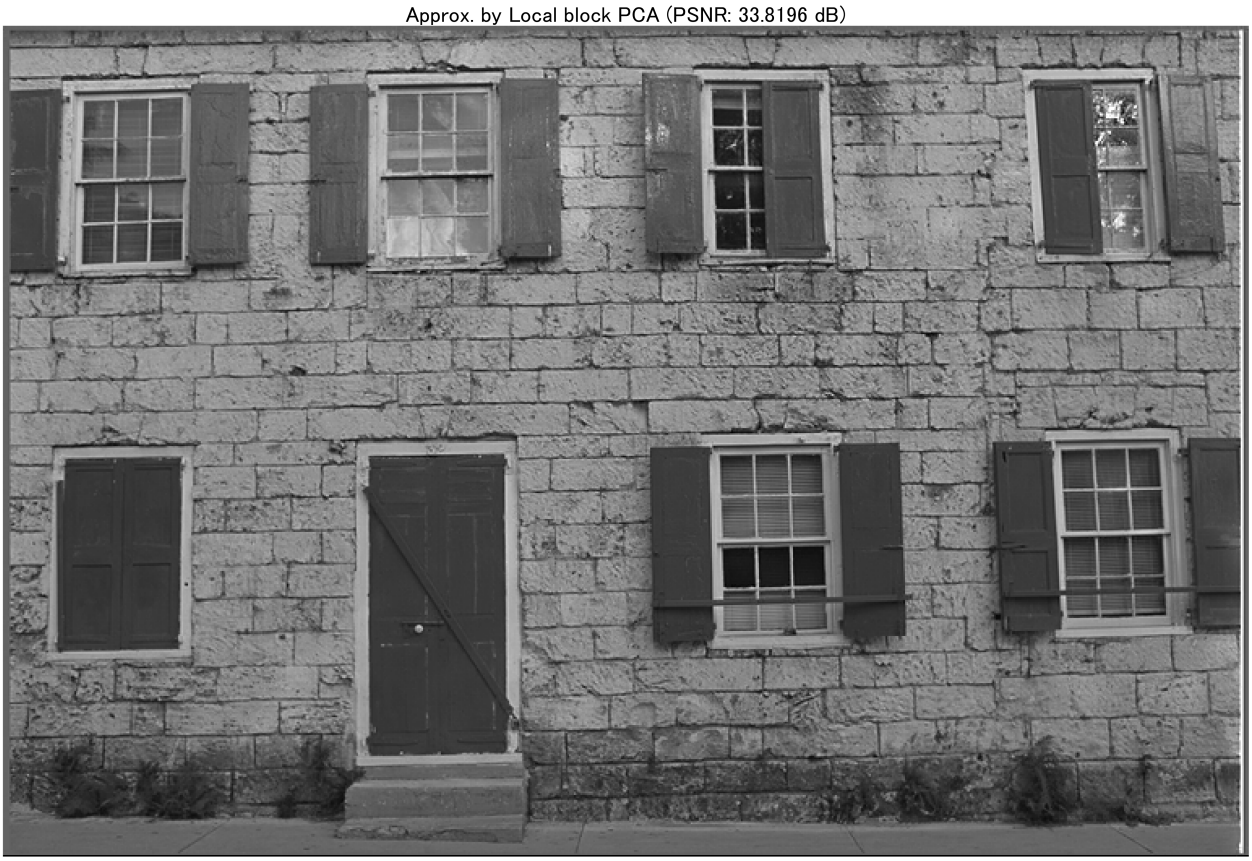
Results

```
figure
imshow(img)
title("Original")
```

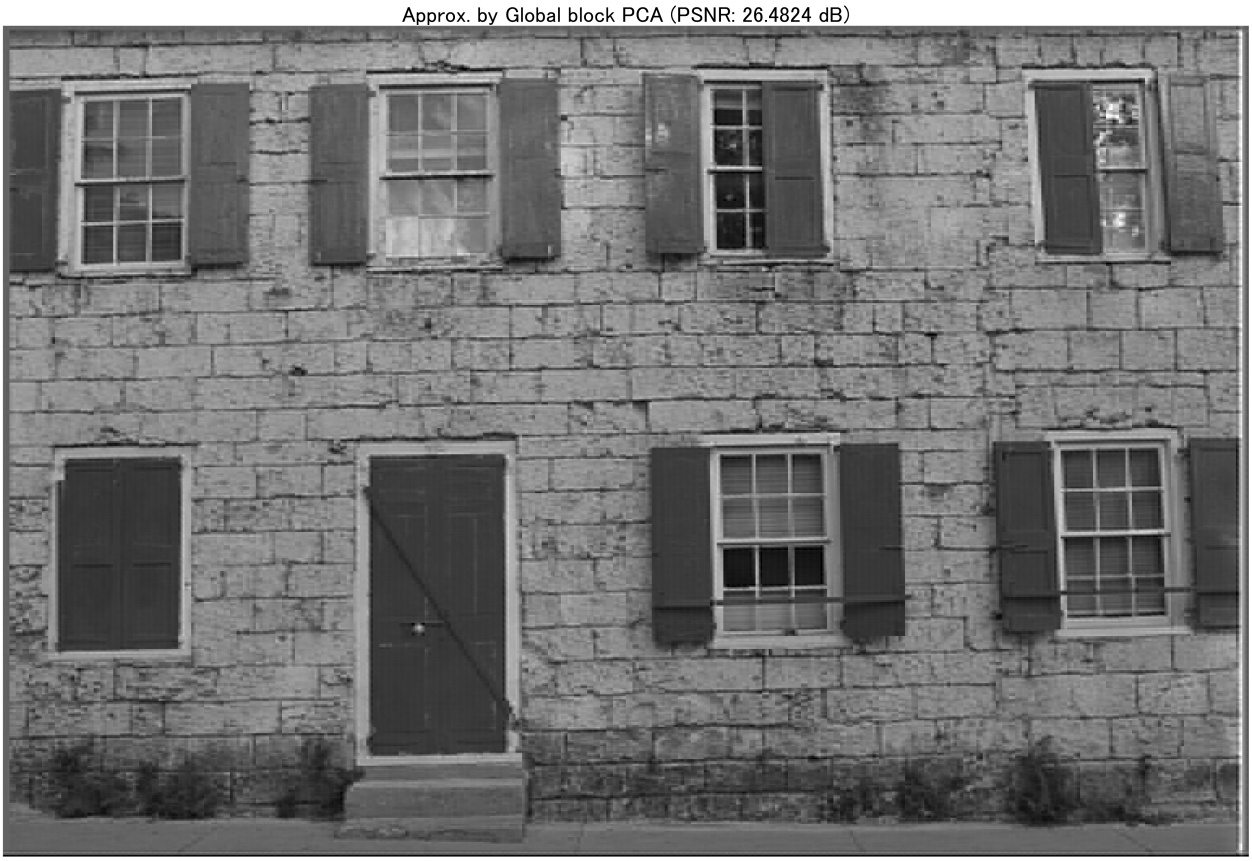
Original



```
figure
imshow(lbpcaimg)
title("Approx. by Local block PCA (PSNR: " + num2str(psnr(img,lbpcaimg))+" dB)")
```

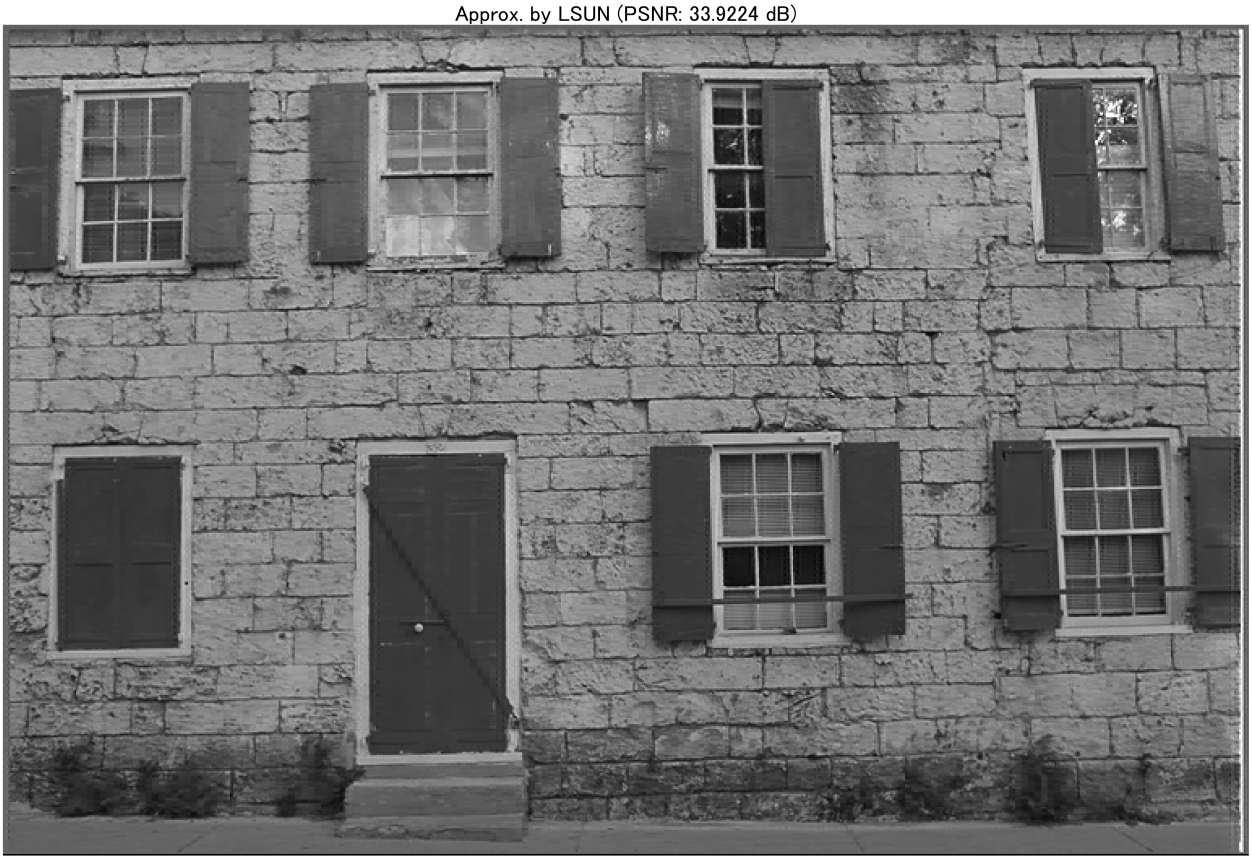



```
figure
imshow(gbpcaimg)
title("Approx. by Global block PCA (PSNR: " + num2str(psnr(img,gbpcaimg))+" dB)")
```




```
figure
imshow(lsunaimg)
title("Approx. by LSUN (PSNR: " + num2str(psnr(img,im2double(lsunaimg)))+" dB)")
```

Approx. by LSUN (PSNR: 33.9224 dB)



Definitions of local functions

Function to extract a local patch block from a global array

```
function y = fcn_extract_blks_(x,iBlk,blksz,k)
% Extend array x
ky = k(1);
kx = k(2);
iBlkRow = iBlk(1);
iBlkCol = iBlk(2);
padsz = [(ky-1)/2, (kx-1)/2].*blksz;
xx = padarray(x,padsz,"circular");
%
posy = (iBlkRow-1)*blksz(1)+1;
posx = (iBlkCol-1)*blksz(2)+1;
y = xx(posy:posy+ky*blksz(1)-1,posx:posx+kx*blksz(2)-1);
end
```

Function to place a local patch block to a global array

```
function y = fcn_place_blks_(y,blk,iBlk,blksz)
```

```
% Extend array x
iBlkRow = iBlk(1);
iBlkCol = iBlk(2);
posy = (iBlkRow-1)*blksz(1)+1;
posx = (iBlkCol-1)*blksz(2)+1;
y(posy:posy+blksz(1)-1,posx:posx+blksz(2)-1) = blk;
end
```

Loss function

$$L(\boldsymbol{\theta}) = \|\mathbf{x}\|_2^2 - \|F_{\boldsymbol{\theta}}(\mathbf{x})\|_2^2,$$

where $F_{\boldsymbol{\theta}}(\cdot)$ is a unitaly analyzer with a coefficient mask. $L(\boldsymbol{\theta}) \geq 0$ is guaranteed.

```
function [gradients, loss] = modelGradients(dlnet, dlX)
% Forward data through the dlnetwork object.
dlY = forward(dlnet,dlX); % F(x)
% Compute loss.
Nx = size(dlX,4);
Ny = size(dlY,4);
loss = sum(dlX.^2,"all")/Nx-sum(dlY.^2,"all")/Ny;
% Compute gradients.
gradients = dlgradient(loss,dlnet.Learnables);
loss = double(gather(extractdata(loss)));
end
```