SANDIA REPORT
SAND2021-13776
Printed November 2021

Sandia National Laboratories

All Optical Neural Networks for Low Power Edge Computing

Raktim Sarma, Michael D. Goldflam, Jayson L. Briscoe

Prepared by
Sandia National Laboratories
Albuquerque, New Mexico
87185 and Livermore,
California 94550

Issued by Sandia National Laboratories, operated for the United States Department of Energy by National Technology & Engineering Solutions of Sandia, LLC.

Available to DOE and DOE contractors from
U.S. Department of Energy
Office of Scientific and Technical Information
P.O. Box 62
Oak Ridge, TN 37831

Telephone: (865) 576-8401
Facsimile: (865) 576-5728
E-Mail: reports@osti.gov
Online ordering: http://www.osti.gov/scitech

Available to the public from
U.S. Department of Commerce
National Technical Information Service
5301 Shawnee Rd
Alexandria, VA 22312

Telephone: (800) 553-6847
Facsimile: (703) 605-6900
E-Mail: orders@ntis.gov
Online order: https://classic.ntis.gov/help/order-methods/

ABSTRACT

We developed a simplistic physics-based model of an all-optical neural network that mimics the encoder part of an autoencoder neural network for image compression. Our approach relies on the generation of a MATLAB-based model for both data compression and decompression and utilizes MATLAB's built-in autoencoder networks in combination with simple propagation of optical fields between layers constituting phase elements via Fourier transform. We optimize the phase elements using the particle swarm optimization technique and using our model, we demonstrate a compression ratio of 25 % for 28×28-pixel input images containing numeric digits from 0 to 9.

CONTENTS

LIST OF FIGURES

LIST OF TABLES

This page left blank

ACRONYMS AND DEFINITIONS

Abbreviation	Definition
AONN	All Optical Neural Network

1. ALL OPTICAL NEURAL NETWORK (AONN) MODELING

For a first order examination of the potential for an optical autoencoder, we relied on the generation of a MATLAB-based model for both data compression and decompression. This model utilized MATLAB's built-in autoencoder networks in combination with simple propagation of optical fields between All optical neural network (AONN) layers via Fourier transform. The sections below describe the model and interaction between these two aspects of the AONN model.

1.1. Autoencoder

Autoencoder generation relied on the autoencoder functionality built into MATLAB in combination with a collection of handwritten digits that are presorted for MATLAB analysis [1]. Generally, an autoencoder is composed of two coupled feed-forward neural networks which operate serially [2].

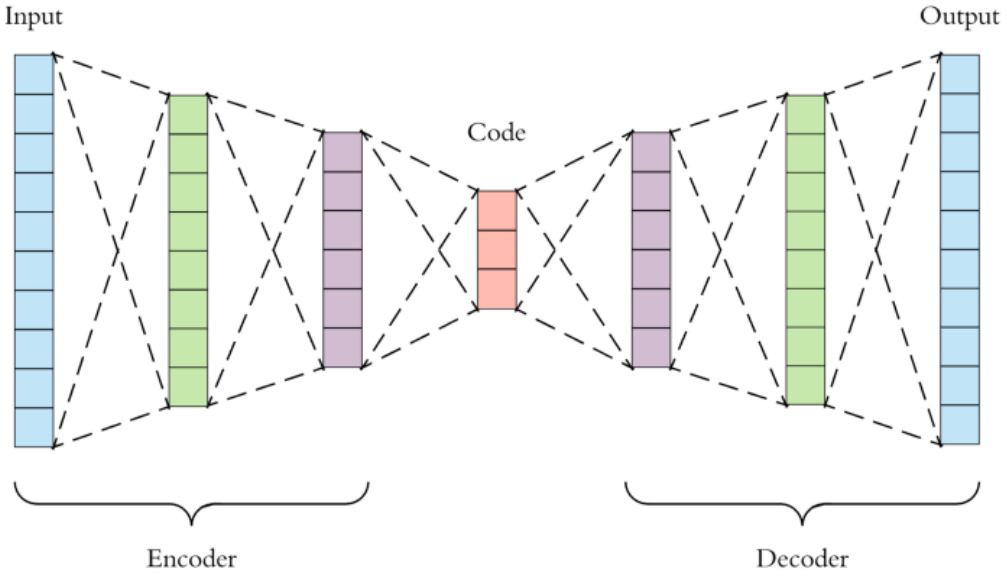

Figure 1-1. Schematic representation of an autoencoder reproduced from [2].

A schematic representation of a generic autoencoder is shown in Figure 1-1, separated into two halves defining the encoder and decoder. For this aspect of the LDRD, we followed the steps itemized below to determine the functionality of our AONN.

1. Train an entirely electronic autoencoder.
2. Utilize the encoder to calculate the code for a given set of inputs.
3. Determine transmission characteristics of a multilayer optical structure that can reproduce the code for a given input.
4. Feed the output of the optical network into the electronic decoder and compare to the input.

In this way, we essentially developed an all optical version of the encoder network that operates passively at zero-power enabling the collection of a smaller number of pixels followed by electronic decoding/decompression of the collected information to reproduce the uncompressed output.

For the simplistic demonstration here, our input consisted of 28×28 pixel images containing numeric digits from 0 to 9 with the goal of a 25% compression ratio (*i.e.* the code consists of ¼ the number of pixels of the input and output). For simplicity, default MATLAB autoencoder parameters were used during training aside those noted in Table **1-1**.

Parameter	Value
EncoderTransferFunction	logsig
DecoderTransferFunction	purelin
L2WeightRegularization	0.001
SparsityRegularization	4
SparsityProportion	0.15
useGPU	True

Table 1-1. Parameters and values chosen for the MATLAB model.

1.2. Physics-Based Network Modeling

With the trained electronic autoencoder in hand, efforts focused on the physics-based encoder network. This network takes the form of several transmissive optical elements with arbitrary position-dependent transmission phase selected such that for a given input, the optical network reproduces the output of the electronic autoencoder. This network functions analogously to that described in [3]. For the purposes of our proof-of-concept model, we utilized a three layer phase only transmissive structure (transmission amplitude of unity), designed for operation at 5 μm, which is schematically show in Figure 1-2.

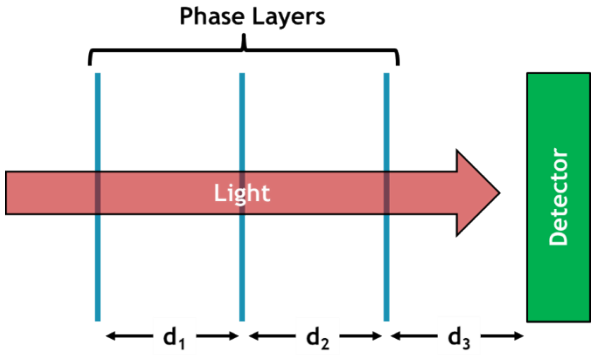

Figure 1-2. Schematic of the simplistic transmissive phase-based network. Three independent phase layers composed of 28×28 pixels each with independently variable response control the signal collected on the 14×14 pixel detector.

We fixed the distance between layers at $d_1 = d_2 = d_3 = 10\ \mu m$ and optimized the output of our structure on the detector to match the output of the encoder network. Each phase layer of the designed structure consists of 784 independent pixels (28×28) where the transmission phase can be arbitrarily set. In sum, this results in 2352 independent optimization variables. Image compression occurs at the detector. While the spatial extent of the detector is identical to that of the phase layers, it contains only 196 pixels (14×14) where each pixel spans the size of four phase layer pixels. This size matching is not a requirement. In reality, the detector could be larger or smaller than the phase layers, however, that complicates calculation of propagating fields. Optimization via backpropagation would be ideal and would more closely parallel the training of a conventional electronic neural network, however, due to the short duration of this project, we were unable to implement this for the physics-based model that includes complex values for both the field and transmission elements. As such, we relied on MATLAB's built-in particle swarm optimization routine for determining the optimum values

of each phase element [4]. Additionally, due to our use of this relatively crude optimization approach, we limited our optical network training set to only 10 input images. Ultimately, a larger training set would be required to create a network with useful capabilities, however, the results shown below point to the potential viability of this concept for image compression and reduction. For this model, we assumed coherent optical input and utilized Fourier transforms to propagate fields between layers and calculated the detector output signal as the mean signal from the four pixels that span a given detector pixel with an ultimate goal of matching the detector signal to the output of the encoder network with a root mean square error used as the metric. While we utilized this approach, an alternative error metric could be employed where the network attempts to minimize the difference between the decoder output (given the optical input rather than the electronic input) and the original input digit image.

After training our electronic autoencoder and optimizing our optical encoder, we obtained the results shown in Figure 1-3. These results point to the successful but imperfect compression and decompression of the input images from both the optical and electronic autoencoder. Specifically, significant background illumination is present in the images retrieved from optically compressed data, compounding the limitations that are already present in this simple implementation of an autoencoder.

Figure 1-3. (a) Output from our optical encoder/electronic decoder. (b) Original input image. (c) Output from the all-electronic autoencoder.

Further improvement in functionality of these types of devices could be obtained via both improved design of electronic compression schemes (*i.e.* more complex autoencoders or alternative compression networks) or via implementation of backpropagation optimization for optical element determination.

REFERENCES

[1] https://www.mathworks.com/help/deeplearning/ref/trainautoencoder.html
[2] https://towardsdatascience.com/applied-deep-learning-part-3-autoencoders-1c083af4d798
[3] X. Lin, Y. Rivenson, N. T. Yardimci, M. Veli, Y. Luo, M. Jarrahi and A. Ozcan, All-optical machine learning using diffractive deep neural networks, Science, 1004-1008, (2018), http://dx.doi.org/10.1126/science.aat8084.
[4] "MATLAB and Optimization Toolbox Release 2017b," (The MathWorks, Inc., Natick, Massachusetts, United States).

APPENDIX A. MATLAB CODE

```
function [structure] = buildAONNLayers(x,xDim,yDim,numLayers,xSize,ySize,wavelength,distance)
% figure(5)
% hold on
% plot(x)
structure = [];
structure.wavelength = wavelength;
%structure.Ein = Ein;
structure.x = xDim;
structure.y = yDim;
structure.numLayers = numLayers;
structure.xSize = xSize;
structure.ySize = ySize;
for mm = 1:numLayers
   structure.layers{mm}.distance = distance;
   TArg = reshape(x(1:xSize*ySize),ySize,xSize);
   x(1:xSize*ySize) = [];
   structure.layers{mm}.T = 1.*exp(1i*TArg*2*pi);
   structure.nX = numel(structure.x)/xSize;
   structure.nY = numel(structure.y)/ySize;
   if or(mod(structure.nX,1),mod(structure.nY,1))
      error('nX or nY is not an integer')
   end
   structure.layers{mm}.T = repelem(structure.layers{mm}.T,structure.nY,structure.nX);

end

function [error,outputSave] = calculateOutput(structure,trainingSet)
xDim = structure.x;
yDim = structure.y;
outSize = size(trainingSet{1}.output);
nX = structure.nX;
nY = structure.nY;
xSize = structure.xSize;
ySize = structure.ySize;
xSizeOut = outSize(2);
ySizeOut = outSize(1);
nXOut = nX*xSize/xSizeOut;
nYOut = nY*ySize/ySizeOut;

for nn = 1:numel(trainingSet)
   o1 = [];
   o2 = [];
   Ein = repelem(trainingSet{nn}.input,nY,nX);
   for mm = 1:structure.numLayers
      EinR = Ein.*structure.layers{mm}.T;
      [Ein] = propagateLayer(xDim,yDim,EinR,structure.layers{mm}.distance,structure.wavelength);
   end
   Eout = Ein;
   Eout = Eout.*conj(Eout);
   for ii = 1:ySizeOut
      for jj = 1:xSizeOut
         o1(:,jj)  = mean(Eout(:,((jj-1)*nXOut+1):jj*nXOut),2);
      end
      o2(ii,:) = mean(o1(((ii-1)*nYOut+1):ii*nYOut,:),1);
   end
   output = o2;
   outputSave{nn} = output./max(output(:));
   expectedOutput = trainingSet{nn}.output;
   currError = abs(expectedOutput/max(expectedOutput(:))-outputSave{nn}).^2;
```

```
    %currError = sqrt(currError.*conj(currError));
    error(nn) = mean(currError(:));
end
%error = mean(error);

%%
%trainingSet = load('training.mat');
%trainingSet = trainingSet.trainingSet;
clear trainingSet
nIn = 28;
nOut = 14;
nSet = 10;
optimizationType = 'particleswarm'; %'ga' or 'gamultiobj' 'particleswarm'
% for mm = 1:nSet
%     trainingSet{mm}.input = randi(2,nIn)-1;
%     trainingSet{mm}.output = rand(nOut);
% end
load('trainingSet.mat')
nn = randi(numel(trainingSet),nSet,1);
trainingSet = trainingSet(nn);
%%
outSize = size(trainingSet{1}.output);
inSize = size(trainingSet{1}.input);

rng default
dx = 2; %um
wavelength = 5; % um
distance = 10;
dy = dx;
nRep = 2;
numLayer = 3;
xSize = inSize(2);
ySize = inSize(1);
nX = xSize*nRep;
nY = ySize*nRep;
xDim = (-nX/2+dx/2:dx:nX/2-dx/2);
yDim = (-nY/2+dy/2:dy:nY/2-dy/2);
numelOpt = numLayer*((xSize*ySize)+1);
fun = @(x) calcError(x,xDim,yDim,numLayer,xSize,ySize,trainingSet,wavelength,distance,optimizationType);

tic

lb0 = [ones(1,xSize*ySize,1)*0];
ub0 = [ones(1,xSize*ySize)];
%%
disp('Starting optimization')
datetime
lb = repmat(lb0,1,numLayer);
ub = repmat(ub0,1,numLayer);
% options = optimoptions('particleswarm');
% options.UseParallel = true;
% options.PlotFcn = 'pswplotbestf';
% options.Display = 'iter';
% options.FunctionTolerance = 1e-10;
%x = particleswarm(fun,numelOpt,lb,ub,options);

options = optimoptions(optimizationType);
options.UseParallel = true;
options.Display = 'iter';
switch optimizationType
    case 'ga'
        options.PlotFcn = 'gaplotbestf';
```

```
        x = ga(fun,numelOpt,[],[],[],[],lb,ub,[],options);
    case 'gamultiobj'
        options.PlotFcn = 'gaplotpareto';
        [x,fval] = gamultiobj(fun,numelOpt,[],[],[],[],lb,ub,options);
        xSave = x;
            case 'particleswarm'
        options.PlotFcn = 'pswplotbestf';
        %options.SwarmSize = 100;
        [x,fval] = particleswarm(fun,numelOpt,lb,ub,options);
        xSave = x;
end

%[x,fval] = gamultiobj(fun,numelOpt,[],[],[],[],lb,ub,options);

%%
%{
%x = xSave(1,:);
[structure] = buildAONNLayers(x,xDim,yDim,numLayer,xSize,ySize,wavelength);
difference = [];
figure(5)
clf
for mm = 1:numLayer
    figure(5)
    subplot(2,numLayer,mm)
    imagesc(abs(structure.layers{mm}.T));
    caxis([0,1])
    axis off
    axis image
    subplot(2,numLayer,mm+numLayer)
    imagesc(wrapTo2Pi(angle(structure.layers{mm}.T)));
    caxis([0,2*pi])
    axis off
    axis image
end
f = gcf;
f.Color = [0,0,0];
[error,output] = calculateOutput(structure,trainingSet);
figure(6)
clf
if nSet<=10
for mm = 1:nSet
    figure(6)
    subplot(3,numel(trainingSet),mm)
    imagesc(trainingSet{mm}.input)
    caxis([0,1])
    a1 = gca;
    a1.XTickLabel = [];
    a1.YTickLabel = [];
    %axis off
    axis image
    subplot(3,numel(trainingSet),mm+numel(trainingSet))
    imagesc(trainingSet{mm}.output/max(trainingSet{mm}.output(:)))
    caxis([0,1])
    %axis off
    axis image
    caxis([0,1])
    a1 = gca;
    a1.XTickLabel = [];
    a1.YTickLabel = [];
    subplot(3,numel(trainingSet),mm+numel(trainingSet)*2)
    imagesc(output{mm})
```

```
    difference(:,:,mm) = (output{mm}-
trainingSet{mm}.output/max(trainingSet{mm}.output(:)));%./(trainingSet{mm}.output/max(trainingSet{mm}.output(:)));
    currDifference = difference(:,:,mm);
    caxis([0,1])
    %axis off
    caxis([0,1])
    a1 = gca;
    a1.XTickLabel = [];
    a1.YTickLabel = [];
    axis image
    colormap hot
    figure(7)
    subplot(2,numel(trainingSet)/2,mm)
    imagesc(abs(difference(:,:,mm)))
    caxis([0,0.15])
    a1 = gca;
    a1.XTickLabel = [];
    a1.YTickLabel = [];
    axis image
        colormap hot

    title(round(mean(abs(currDifference(:))),3))
end
colormap hot
f = gcf;
f.Color = [0.5,0.5,0.5];
end
%}
%%
load('encoder.mat')
%load('testImages.mat')
metric = 'immse';
XTest = [];
for mm = 1:numel(trainingSet)
   XTest(:,mm) = trainingSet{mm}.input(:);
end
feat1 = encode(autoenc,trainingSet);
feat2 = decode(autoenc,feat1);
feat2 = feat2(nn);

[structure] = buildAONNLayers(x,xDim,yDim,numLayer,xSize,ySize,wavelength,distance);
[error,output] = calculateOutput(structure,trainingSet);
out1 = [];
nStart = 1;
numPlot = 10;
qualityE = [];
qualityO = [];
for mm = 1:numPlot
    out1(:,mm) = output{mm}(:);
end
out2 = decode(autoenc,out1);
figure(10)
for mm = 1:numPlot
    subplot(3,numPlot,mm)
    imshow(out2{mm})
    switch lower(metric)
        case 'immse'
            qualityO(mm) = immse(out2{mm},trainingSet{mm}.input);
        case 'psnr'
            qualityO(mm) = psnr(out2{mm},trainingSet{mm}.input);
        case 'ssim'
            qualityO(mm) = ssim(out2{mm},trainingSet{mm}.input);
```

```
    end
end
for mm = 1:numPlot
    subplot(3,numPlot,mm+numPlot)
    imshow(trainingSet{mm}.input)
end
for mm = 1:numPlot
    subplot(3,numPlot,mm+numPlot*2)
    imshow(feat2{mm})
    switch lower(metric)
        case 'immse'
            qualityE(mm) = immse(feat2{mm},trainingSet{mm}.input);
        case 'psnr'
            qualityE(mm) = psnr(feat2{mm},trainingSet{mm}.input);
        case 'ssim'
            qualityE(mm) = ssim(feat2{mm},trainingSet{mm}.input);
    end
end
quality = [qualityE;qualityO];
clc
disp(quality)
colormap hot
%%
feat1 = encode(autoenc,XTest);
feat1 = feat1(:,nn);
figure(11)
for mm = 1:numPlot
    subplot(2,numPlot,mm)
    imshow(reshape(out1(:,mm),nOut,nOut))
end
for mm = 1:numPlot
    subplot(2,numPlot,mm+numPlot)
    imshow(reshape(feat1(:,mm),nOut,nOut))
end
    colormap hot
disp('Done')
%%
function [error] = calcError(x,xDim,yDim,numLayer,xSize,ySize,trainingSet,wavelength,distance,optimizationType)
[structure] = buildAONNLayers(x,xDim,yDim,numLayer,xSize,ySize,wavelength,distance);
error = calculateOutput(structure,trainingSet);
if or(strcmp(optimizationType,'ga'),strcmp(optimizationType,'particleswarm'))
    error = mean(error(:));
end
end
%%
%trainingSet = load('training.mat');
%trainingSet = trainingSet.trainingSet;
clear trainingSet
nIn = 28;
nOut = 14;
nSet = 10;
optimizationType = 'particleswarm'; %'ga' or 'gamultiobj' 'particleswarm'
% for mm = 1:nSet
%     trainingSet{mm}.input = randi(2,nIn)-1;
%     trainingSet{mm}.output = rand(nOut);
% end
load('trainingSet.mat')
nn = randi(numel(trainingSet),nSet,1);
trainingSet = trainingSet(nn);
%%
outSize = size(trainingSet{1}.output);
inSize = size(trainingSet{1}.input);
```

```
rng default
dx = 2; %um
wavelength = 5; % um
distance = 10;
dy = dx;
nRep = 2;
numLayer = 3;
xSize = inSize(2);
ySize = inSize(1);
nX = xSize*nRep;
nY = ySize*nRep;
xDim = (-nX/2+dx/2:dx:nX/2-dx/2);
yDim = (-nY/2+dy/2:dy:nY/2-dy/2);
numelOpt = numLayer*((xSize*ySize)+1);
fun = @(x) calcError(x,xDim,yDim,numLayer,xSize,ySize,trainingSet,wavelength,distance,optimizationType);

tic

lb0 = [ones(1,xSize*ySize,1)*0];
ub0 = [ones(1,xSize*ySize)];
%%
disp('Starting optimization')
datetime
lb = repmat(lb0,1,numLayer);
ub = repmat(ub0,1,numLayer);
% options = optimoptions('particleswarm');
% options.UseParallel = true;
% options.PlotFcn = 'pswplotbestf';
% options.Display = 'iter';
% options.FunctionTolerance = 1e-10;
%x = particleswarm(fun,numelOpt,lb,ub,options);

options = optimoptions(optimizationType);
options.UseParallel = true;
options.Display = 'iter';
switch optimizationType
   case 'ga'
      options.PlotFcn = 'gaplotbestf';
      x = ga(fun,numelOpt,[],[],[],[],lb,ub,[],options);
   case 'gamultiobj'
      options.PlotFcn = 'gaplotpareto';
      [x,fval] = gamultiobj(fun,numelOpt,[],[],[],[],lb,ub,options);
      xSave = x;
         case 'particleswarm'
      options.PlotFcn = 'pswplotbestf';
      %options.SwarmSize = 100;
      [x,fval] = particleswarm(fun,numelOpt,lb,ub,options);
      xSave = x;
end

%[x,fval] = gamultiobj(fun,numelOpt,[],[],[],[],lb,ub,options);

%%
%{
%x = xSave(1,:);
[structure] = buildAONNLayers(x,xDim,yDim,numLayer,xSize,ySize,wavelength);
difference = [];
figure(5)
clf
for mm = 1:numLayer
   figure(5)
```

```
    subplot(2,numLayer,mm)
    imagesc(abs(structure.layers{mm}.T));
    caxis([0,1])
    axis off
    axis image
    subplot(2,numLayer,mm+numLayer)
    imagesc(wrapTo2Pi(angle(structure.layers{mm}.T)));
    caxis([0,2*pi])
    axis off
    axis image
end
f = gcf;
f.Color = [0,0,0];
[error,output] = calculateOutput(structure,trainingSet);
figure(6)
clf
if nSet<=10
for mm = 1:nSet
    figure(6)
    subplot(3,numel(trainingSet),mm)
    imagesc(trainingSet{mm}.input)
    caxis([0,1])
    a1 = gca;
    a1.XTickLabel = [];
    a1.YTickLabel = [];
    %axis off
    axis image
    subplot(3,numel(trainingSet),mm+numel(trainingSet))
    imagesc(trainingSet{mm}.output/max(trainingSet{mm}.output(:)))
    caxis([0,1])
    %axis off
    axis image
    caxis([0,1])
    a1 = gca;
    a1.XTickLabel = [];
    a1.YTickLabel = [];
    subplot(3,numel(trainingSet),mm+numel(trainingSet)*2)
    imagesc(output{mm})
    difference(:,:,mm) = (output{mm}-
trainingSet{mm}.output/max(trainingSet{mm}.output(:)));%./(trainingSet{mm}.output/max(trainingSet{mm}.output(:)));
    currDifference = difference(:,:,mm);
    caxis([0,1])
    %axis off
    caxis([0,1])
    a1 = gca;
    a1.XTickLabel = [];
    a1.YTickLabel = [];
    axis image
    colormap hot
    figure(7)
    subplot(2,numel(trainingSet)/2,mm)
    imagesc(abs(difference(:,:,mm)))
    caxis([0,0.15])
    a1 = gca;
    a1.XTickLabel = [];
    a1.YTickLabel = [];
    axis image
       colormap hot

    title(round(mean(abs(currDifference(:))),3))
end
colormap hot
```

```
f = gcf;
f.Color = [0.5,0.5,0.5];
end
%}
%%
load('encoder.mat')
for nn = 77
%    x = xSave(nn,:);
   [structure] = buildAONNLayers(x,xDim,yDim,numLayer,xSize,ySize,wavelength);
   [error,output] = calculateOutput(structure,trainingSet);
   out1 = [];
   nStart = 5;
   numPlot = 10;
   for mm = 1:numPlot
      out1(:,mm) = output{mm}(:);
   end
   out2 = decode(autoenc,out1);
   figure(10)
   for mm = 1:numPlot
      subplot(2,numPlot,mm)
      imshow(out2{mm})
   end
   for mm = 1:numPlot
      subplot(2,numPlot,mm+numPlot)
      imshow(trainingSet{mm}.input)
   end
   %disp(nn)
   %pause
end
5
%%
function [error] = calcError(x,xDim,yDim,numLayer,xSize,ySize,trainingSet,wavelength,distance,optimizationType)
[structure] = buildAONNLayers(x,xDim,yDim,numLayer,xSize,ySize,wavelength,distance);
error = calculateOutput(structure,trainingSet);
if or(strcmp(optimizationType,'ga'),strcmp(optimizationType,'particleswarm'))
   error = mean(error(:));
end
end
function [Eout] = propagateLayer(x,y,Ein,distance,wavelength)
kz = @(kval,kx,ky) sqrt(kval.^2-kx.^2-ky.^2);
k = 2*pi/wavelength;

nPointsX = numel(x)-1;
nPointsY = numel(y)-1;
dx = x(2)-x(1);
dy = y(2)-y(1);
Fx = 1/max(x-min(x));
Fy = 1/max(y-min(y));
fx = 2*pi*Fx*(-ceil(nPointsX/2):floor(nPointsX/2));
fy = 2*pi*Fy*(-ceil(nPointsY/2):floor(nPointsY/2));
[fxg,fyg] = ndgrid(fx,fy);

currFFT = fftshift((fft2(Ein)));
kzCurr = kz(k,fxg,fyg);
currPropFFT = currFFT.*exp(1i*kzCurr*distance);
Eout = (ifft2(currPropFFT));
%%
XTrain = digitTrainCellArrayData;
% nums = randi(5000,1,5000);
% XTest = XTrain(nums(4501:end));
% save('testImages.mat','XTest')
% XTrain = XTrain(nums(1:4500));
```

```
%%
hiddenSize = 14^2;
autoenc = trainAutoencoder(XTrain,hiddenSize,...
            'EncoderTransferFunction','logsig',...
        'DecoderTransferFunction','purelin',...
        'L2WeightRegularization',0.001,...
        'SparsityRegularization',4,...
        'SparsityProportion',0.15,...
        'useGPU',true);

XTest = digitTestCellArrayData;
xReconstructed = predict(autoenc,XTest);

%%
% h1 = 20*20;
% a1 = trainAutoencoder(XTrain,h1,...
%         'L2WeightRegularization',0.001,...
%         'SparsityRegularization',4,...
%         'SparsityProportion',0.15,...
%         'useGPU',true);
% feat1 = encode(autoenc,XTest);
% h2 = 14*14;
% a2 = trainAutoencoder(feat1,h2,...
%         'L2WeightRegularization',0.001,...
%         'SparsityRegularization',4,...
%         'SparsityProportion',0.15,...
%         'useGPU',true);
%%
feat1 = encode(autoenc,XTest);
feat2 = decode(autoenc,feat1);
%%
figure(2);
nn = randi(numel(XTest),20,1);
for i = 1:20
subplot(4,5,i);
imshow(xReconstructed{nn(i)});
end
figure(3);
for i = 1:20
subplot(4,5,i);
imshow(XTest{nn(i)});
end
figure(4);
for i = 1:20
subplot(4,5,i);
imshow(reshape(feat1(:,i),sqrt(hiddenSize),sqrt(hiddenSize)));
end
%%
numSet = numel(XTest);
clear trainingSet
for mm = 1:numel(XTest)
   trainingSet{mm}.input = XTest{mm};
   trainingSet{mm}.output = reshape(feat1(:,mm),14,14);
   mm
end
save('trainingSet.mat','trainingSet')
save('encoder.mat','autoenc')

%%
```

DISTRIBUTION

Email—Internal

Name	Org.	Sandia Email Address
Raktim Sarma	01881	rsarma@sandia.gov
Dale Jackson	06772	dcjacks@sandia.gov
Jayson Briscoe	06775	jlbrisc@sandia.gov
Technical Library	01977	sanddocs@sandia.gov

Sandia National Laboratories

Sandia National Laboratories is a multimission laboratory managed and operated by National Technology & Engineering Solutions of Sandia LLC, a wholly owned subsidiary of Honeywell International Inc. for the U.S. Department of Energy's National Nuclear Security Administration under contract DE-NA0003525.