Summer Poissonnier
Ania Schulz
CAP 4401 – Digital Image Processing

Single Image Super-Resolution Using Deep Learning

Motivation

The first thing my partner and I did was look through the list of the suggested topics and then we selected the topic that we thought would work best for us. After looking through them we decided to select the topic: Single Image Super Resolution using deep learning because it appears to be the most feasible. It was also a topic that we thought looked interesting; we were excited to get started on this project and see how we could modify it.

Background

Single image super resolution is a process in which high resolution images are created from low resolution images. The purpose of SISR is to recover a single high-resolution image from a single low-resolution image. SISR does face some challenges and is an ill-posed problem. The reason for this is because there are many possible solutions for any low-resolution pixel. This example uses the VDSR network (a convolutional neural network) designed to perform single image super-resolution. Paper [1] mentions a proposed model called the SRCNN which has more appealing properties than the VDSR. The SRCNN's structure provides more accurate results, guidance for the network structure, and good quality and speed.

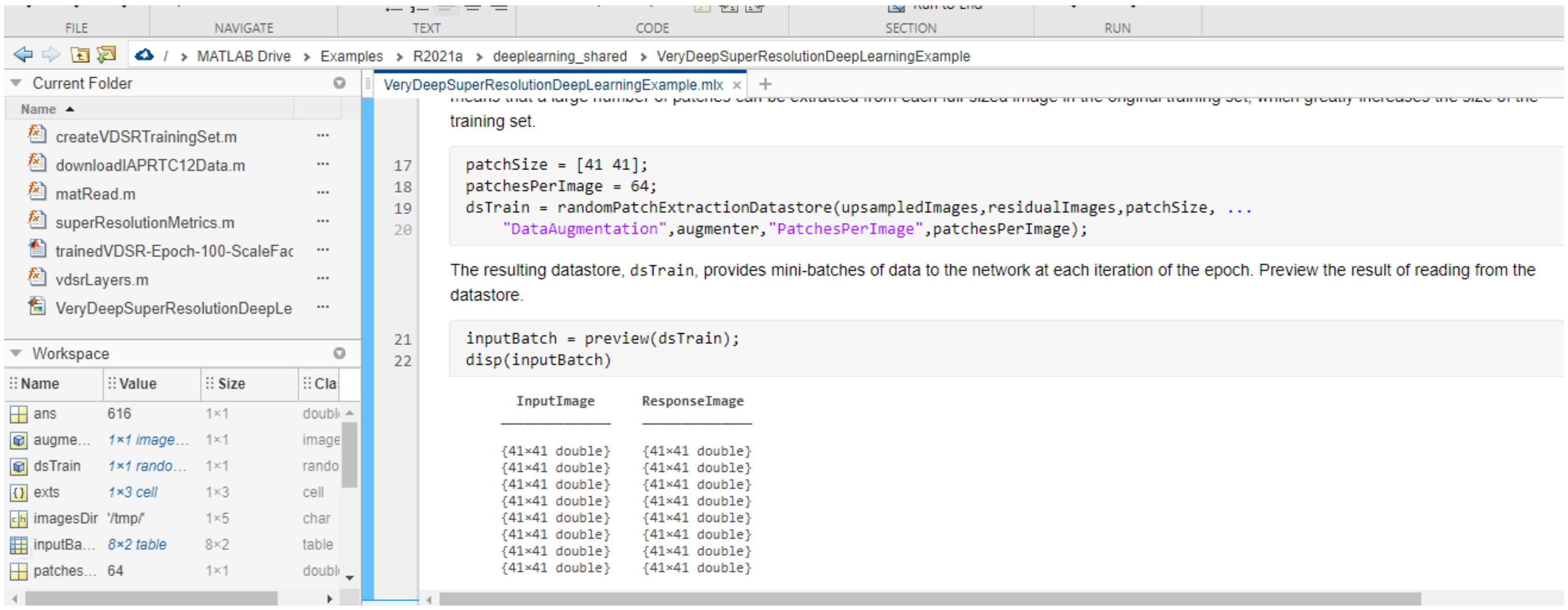
Example code

Upon first glance, there are definitely issues with the example code. The high-resolution image compared to the original reference image is definitely not as accurate as one would hope. In the modification part of the paper, we will modify the code in ways to see if we can get the accuracy of the high-resolution image to be greater.

The first part of the code in the example was downloading training and test data which consisted of photos of people, animals, cities, etc. The code then listed the number of training images which was 616.

The second part of the code was preparing the training data. To create the training set, the code generated pairs of unsampled images and corresponding residual images.

The next step in the code was to define the preprocessing pipeline for the training set. This part used an image data augmenter function and a random patch extraction datastore that performs randomized patch extraction from unsampled and residual image datastores. The resulting datastore provides mini batches of data for the network at each iteration of epoch.


FILE
NAVIGATE
TEXT
CODE
SECTION
RUN
/ > MATLAB Drive > Examples > R2021a > deeplearning_shared > VeryDeepSuperResolutionDeepLearningExample
Current Folder
Name
createVDSRTrainingSet.m
downloadIAPRTC12Data.m
matRead.m
superResolutionMetrics.m
trainedVDSR-Epoch-100-ScaleFac
vdsrLayers.m
VeryDeepSuperResolutionDeepLe
Workspace
VeryDeepSuperResolutionDeepLearningExample.mlx

Name	Value	Size	Cla
ans	616	1×1	doubl
augme…	1×1 image…	1×1	image
dsTrain	1×1 rando…	1×1	rando
exts	1×3 cell	1×3	cell
imagesDir	'/tmp/'	1×5	char
inputBa…	8×2 table	8×2	table
patches…	64	1×1	doubl

means that a large number of patches can be extracted from each full-sized image in the original training set, which greatly increases the size of the training set.

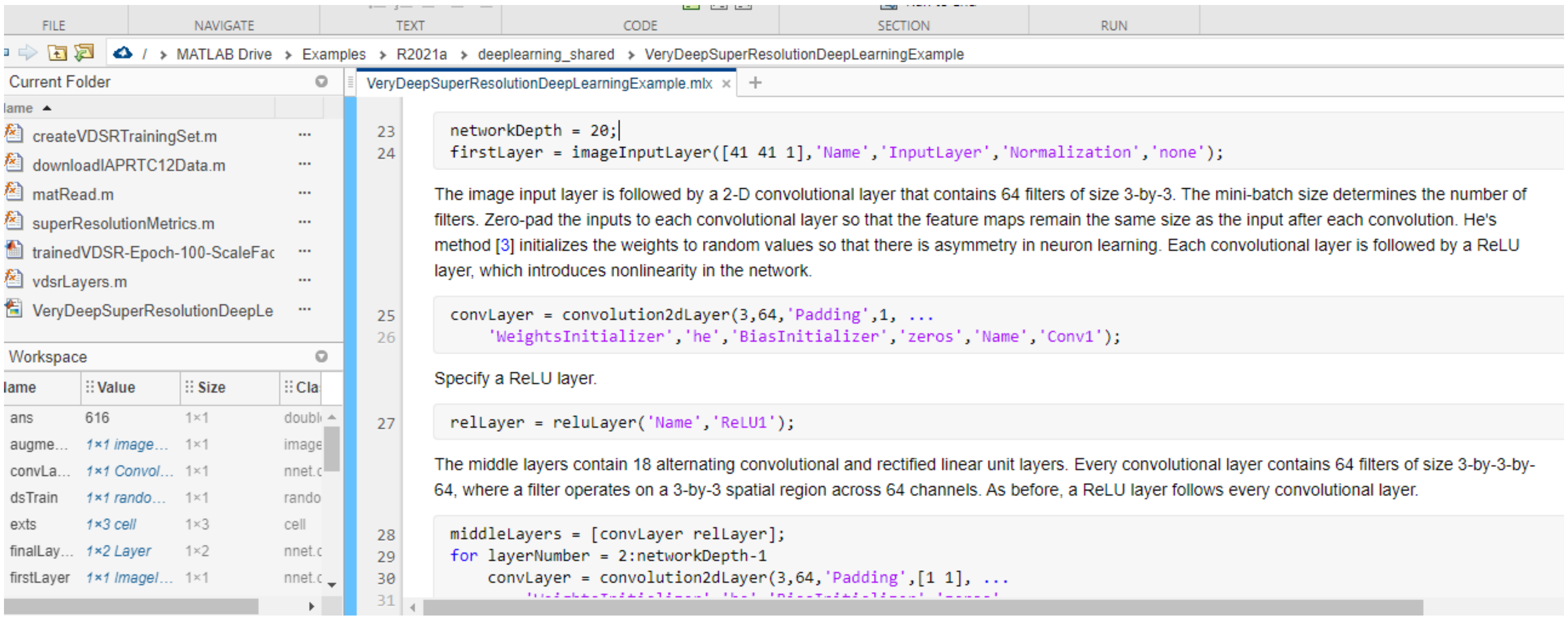
```
patchSize = [41 41];
patchesPerImage = 64;
dsTrain = randomPatchExtractionDatastore(upsampledImages,residualImages,patchSize, ...
    "DataAugmentation",augmenter,"PatchesPerImage",patchesPerImage);
```

The resulting datastore, `dsTrain`, provides mini-batches of data to the network at each iteration of the epoch. Preview the result of reading from the datastore.

```
inputBatch = preview(dsTrain);
disp(inputBatch)
```

```
     InputImage        ResponseImage
    ______________    ______________

    {41×41 double}    {41×41 double}
    {41×41 double}    {41×41 double}
    {41×41 double}    {41×41 double}
    {41×41 double}    {41×41 double}
    {41×41 double}    {41×41 double}
    {41×41 double}    {41×41 double}
    {41×41 double}    {41×41 double}
    {41×41 double}    {41×41 double}
```

Then, the set up of the VDSR layers was coded for using 41 individual layers. Some of the important layers in this section are: imageInputLayer, convolutional2dLayer, reluLayer, and regressionLayer.


FILE
NAVIGATE
TEXT
CODE
SECTION
RUN
/ > MATLAB Drive > Examples > R2021a > deeplearning_shared > VeryDeepSuperResolutionDeepLearningExample
Current Folder
lame
createVDSRTrainingSet.m
downloadIAPRTC12Data.m
matRead.m
superResolutionMetrics.m
trainedVDSR-Epoch-100-ScaleFac
vdsrLayers.m
VeryDeepSuperResolutionDeepLe
Workspace
VeryDeepSuperResolutionDeepLearningExample.mlx

lame	Value	Size	Cla
ans	616	1×1	doubl
augme…	1×1 image…	1×1	image
convLa…	1×1 Convol…	1×1	nnet.c
dsTrain	1×1 rando…	1×1	rando
exts	1×3 cell	1×3	cell
finalLay…	1×2 Layer	1×2	nnet.c
firstLayer	1×1 ImageI…	1×1	nnet.c

```
networkDepth = 20;
firstLayer = imageInputLayer([41 41 1],'Name','InputLayer','Normalization','none');
```

The image input layer is followed by a 2-D convolutional layer that contains 64 filters of size 3-by-3. The mini-batch size determines the number of filters. Zero-pad the inputs to each convolutional layer so that the feature maps remain the same size as the input after each convolution. He's method [3] initializes the weights to random values so that there is asymmetry in neuron learning. Each convolutional layer is followed by a ReLU layer, which introduces nonlinearity in the network.

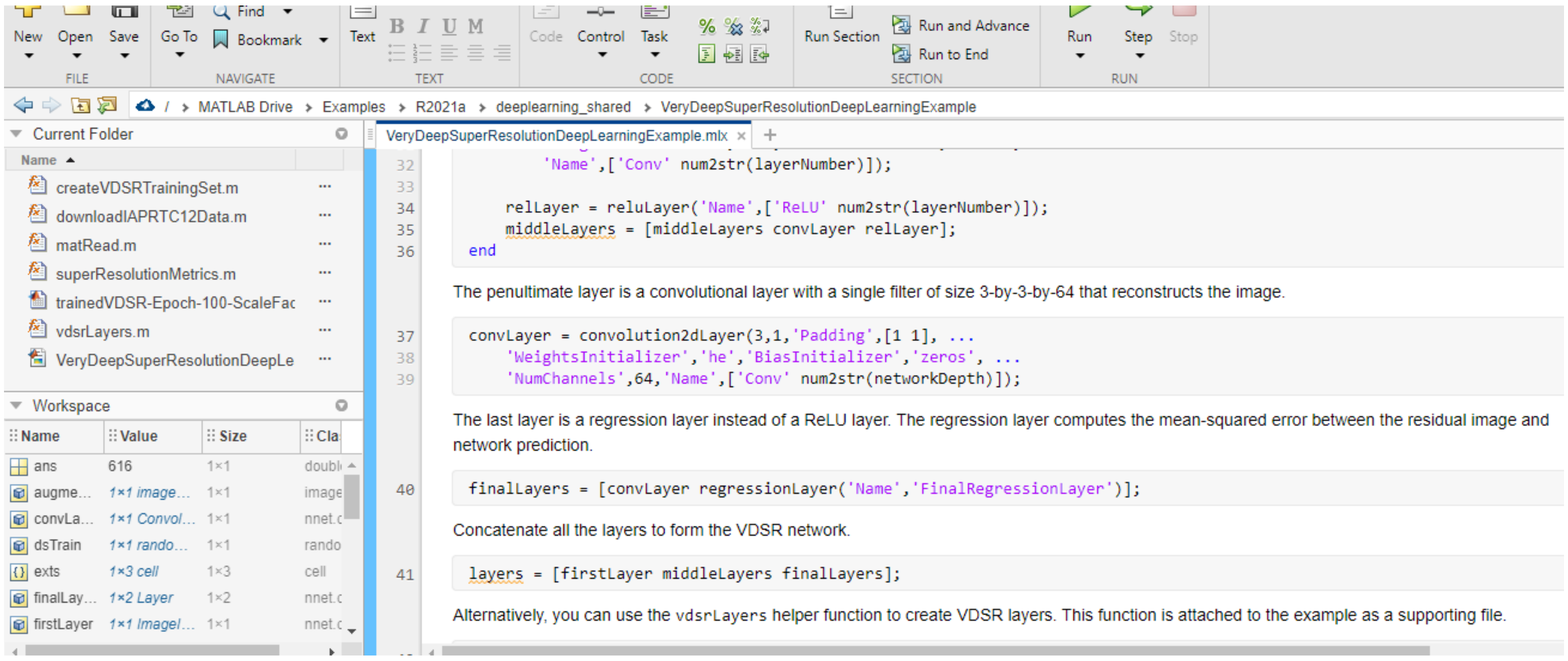
```
convLayer = convolution2dLayer(3,64,'Padding',1, ...
    'WeightsInitializer','he','BiasInitializer','zeros','Name','Conv1');
```

Specify a ReLU layer.

```
relLayer = reluLayer('Name','ReLU1');
```

The middle layers contain 18 alternating convolutional and rectified linear unit layers. Every convolutional layer contains 64 filters of size 3-by-3-by-64, where a filter operates on a 3-by-3 spatial region across 64 channels. As before, a ReLU layer follows every convolutional layer.

```
middleLayers = [convLayer relLayer];
for layerNumber = 2:networkDepth-1
    convLayer = convolution2dLayer(3,64,'Padding',[1 1], ...
        'WeightsInitializer','he','BiasInitializer','zeros',
```



Name	Value	Size	Cla
ans	616	1×1	doubl
augme...	1×1 image...	1×1	image
convLa...	1×1 Convol...	1×1	nnet.c
dsTrain	1×1 rando...	1×1	rando
exts	1×3 cell	1×3	cell
finalLay...	1×2 Layer	1×2	nnet.c
firstLayer	1×1 ImageI...	1×1	nnet.c

```
        'Name',['Conv' num2str(layerNumber)]);

    relLayer = reluLayer('Name',['ReLU' num2str(layerNumber)]);
    middleLayers = [middleLayers convLayer relLayer];
end
```

The penultimate layer is a convolutional layer with a single filter of size 3-by-3-by-64 that reconstructs the image.

```
convLayer = convolution2dLayer(3,1,'Padding',[1 1], ...
    'WeightsInitializer','he','BiasInitializer','zeros', ...
    'NumChannels',64,'Name',['Conv' num2str(networkDepth)]);
```

The last layer is a regression layer instead of a ReLU layer. The regression layer computes the mean-squared error between the residual image and network prediction.

```
finalLayers = [convLayer regressionLayer('Name','FinalRegressionLayer')];
```

Concatenate all the layers to form the VDSR network.

```
layers = [firstLayer middleLayers finalLayers];
```

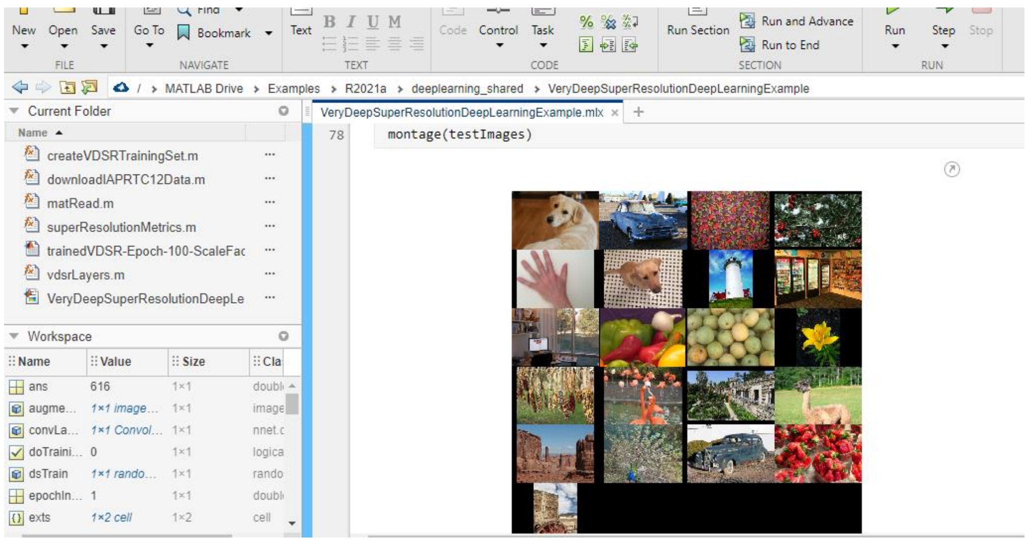
Alternatively, you can use the `vdsrLayers` helper function to create VDSR layers. This function is attached to the example as a supporting file.

The next two steps were specifying the training options and training the code. For the specifications:

```
maxEpochs = 100;
epochIntervals = 1;
initLearningRate = 0.1;
learningRateFactor = 0.1;
l2reg = 0.0001;
miniBatchSize = 64;
options = trainingOptions('sgdm', ...
    'Momentum',0.9, ...
    'InitialLearnRate',initLearningRate, ...
    'LearnRateSchedule','piecewise', ...
    'LearnRateDropPeriod',10, ...
    'LearnRateDropFactor',learningRateFactor, ...
    'L2Regularization',l2reg, ...
    'MaxEpochs',maxEpochs, ...
    'MiniBatchSize',miniBatchSize, ...
    'GradientThresholdMethod','l2norm', ...
    'GradientThreshold',0.01, ...
    'Plots','training-progress', ...
    'Verbose',false);
```

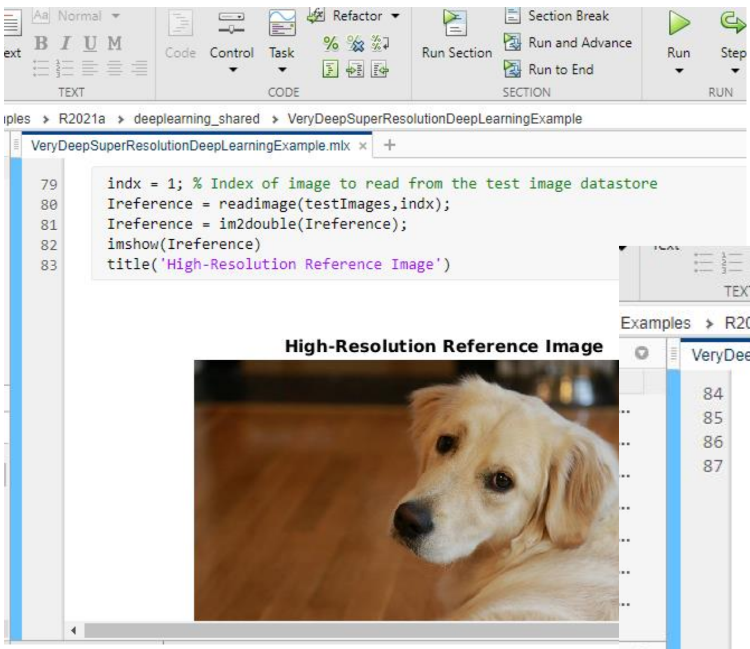
This part of the code we will modify in the next part of the paper.

Then, the code created a sample low-resolution image and then displayed the testing images as a montage. You could then choose which high-resolution image to use; the example used an image of a dog.

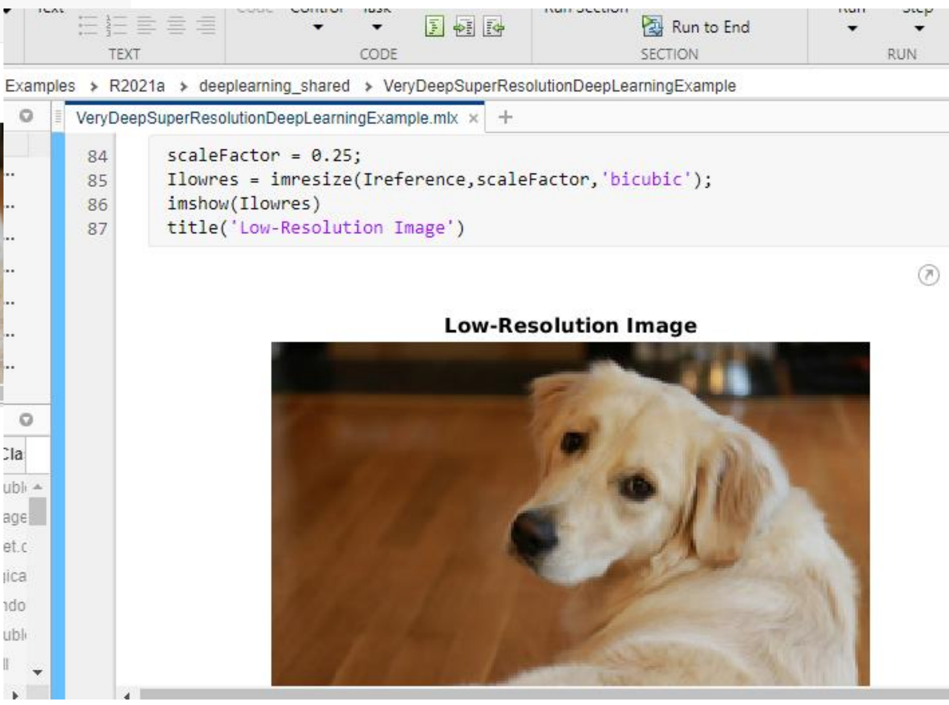

MATLAB Drive > Examples > R2021a > deeplearning_shared > VeryDeepSuperResolutionDeepLearningExample
Current Folder
createVDSRTrainingSet.m
downloadIAPRTC12Data.m
matRead.m
superResolutionMetrics.m
trainedVDSR-Epoch-100-ScaleFac
vdsrLayers.m
VeryDeepSuperResolutionDeepLe
Workspace
VeryDeepSuperResolutionDeepLearningExample.mlx

Name	Value	Size	Cla
ans	616	1×1	doubl
augme...	1×1 image...	1×1	image
convLa...	1×1 Convol...	1×1	nnet.c
doTraini...	0	1×1	logica
dsTrain	1×1 rando...	1×1	rando
epochIn...	1	1×1	doubl
exts	1×2 cell	1×2	cell

```
montage(testImages)
```

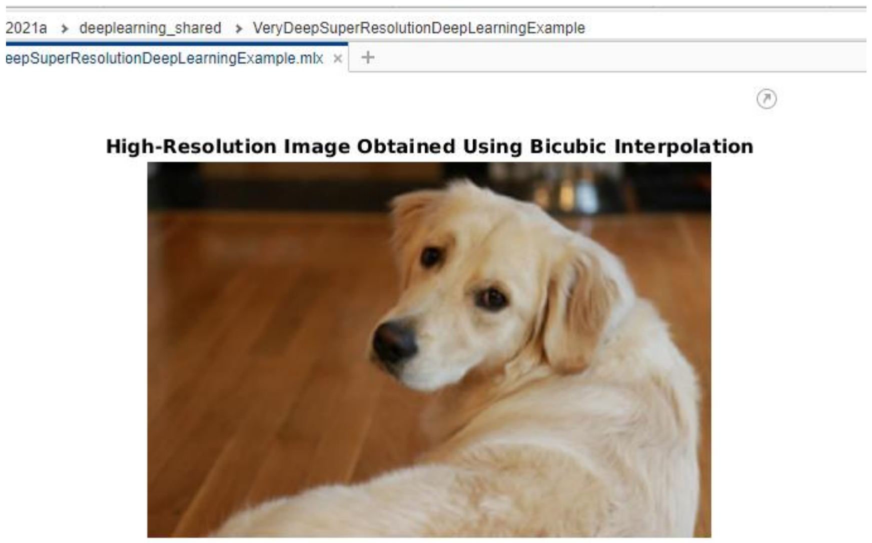

ples > R2021a > deeplearning_shared > VeryDeepSuperResolutionDeepLearningExample
VeryDeepSuperResolutionDeepLearningExample.mlx
High-Resolution Reference Image

```
indx = 1; % Index of image to read from the test image datastore
Ireference = readimage(testImages,indx);
Ireference = im2double(Ireference);
imshow(Ireference)
title('High-Resolution Reference Image')
```

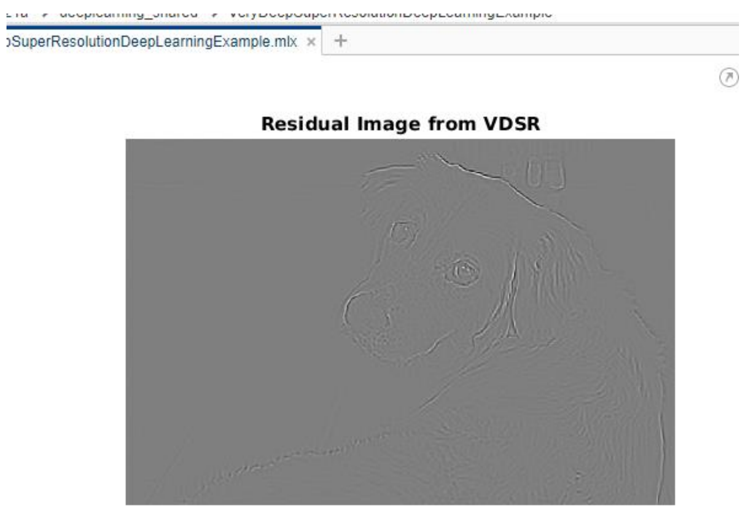

Examples > R2021a > deeplearning_shared > VeryDeepSuperResolutionDeepLearningExample
VeryDeepSuperResolutionDeepLearningExample.mlx
Low-Resolution Image

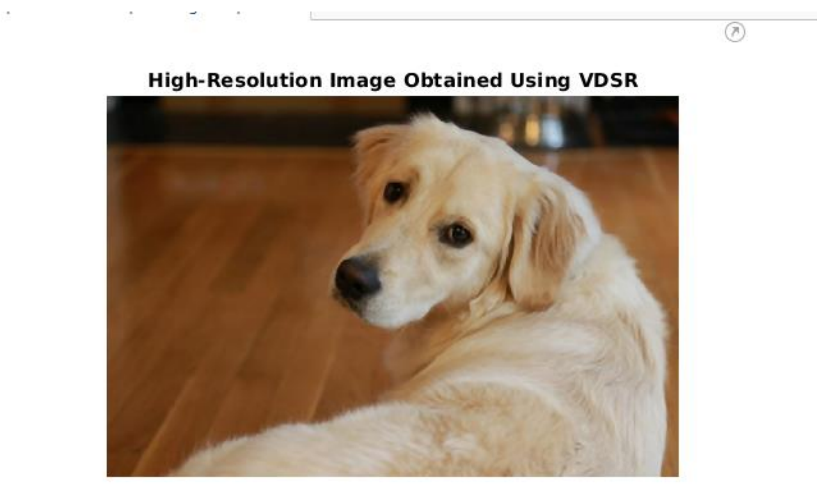
```
scaleFactor = 0.25;
Ilowres = imresize(Ireference,scaleFactor,'bicubic');
imshow(Ilowres)
title('Low-Resolution Image')
```

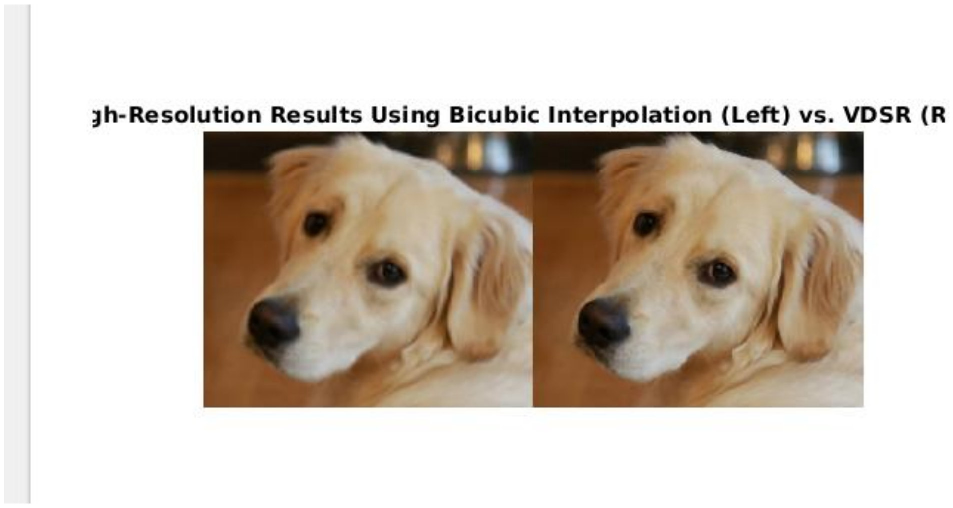
The example then used Bicubic interpolation instead of deep learning to get a high-resolution from the low-resolution image.


2021a > deeplearning_shared > VeryDeepSuperResolutionDeepLearningExample
eepSuperResolutionDeepLearningExample.mlx
High-Resolution Image Obtained Using Bicubic Interpolation

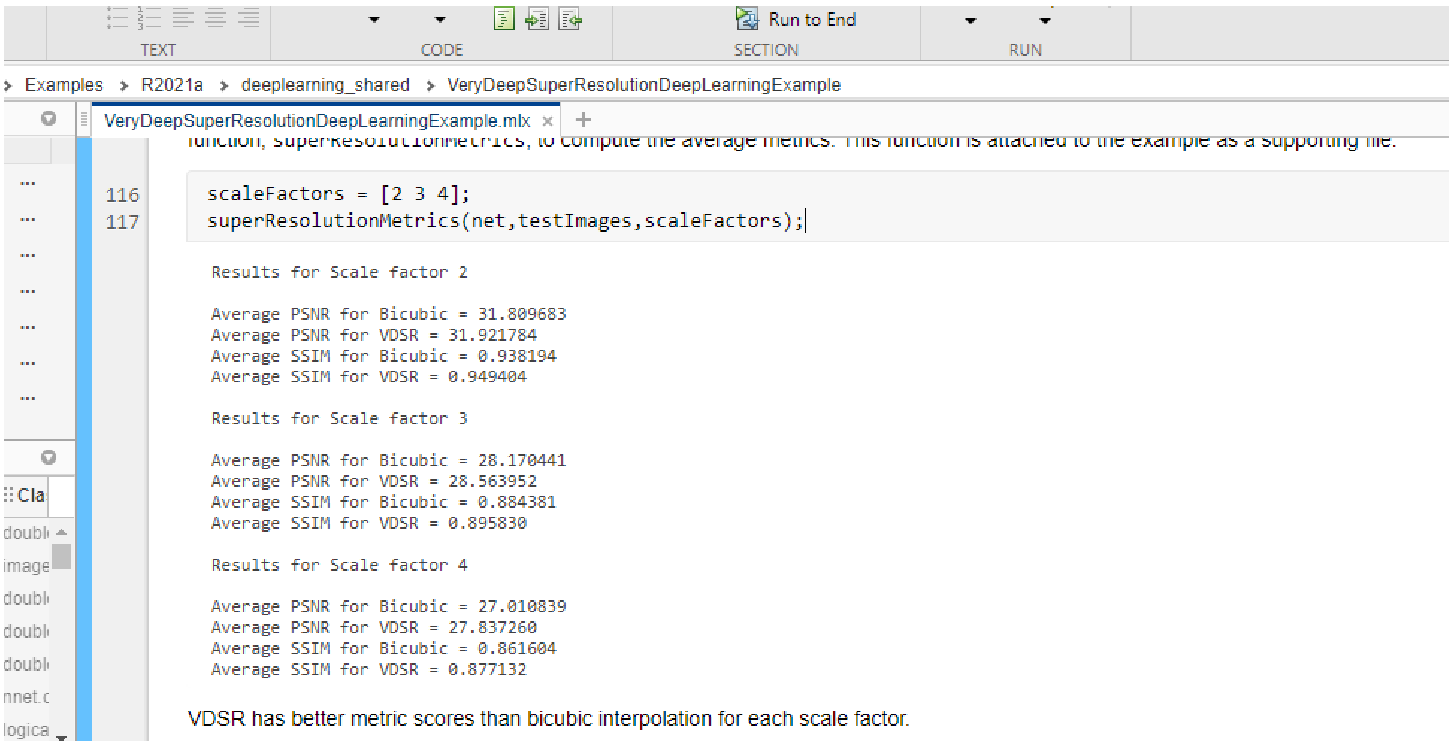
Then, the network went to improve the image resolution using VDSR network.






gh-Resolution Results Using Bicubic Interpolation (Left) vs. VDSR (R

Then, to compare the image quality, the code used image quality metrics.


TEXT
CODE
Run to End
SECTION
RUN
Examples > R2021a > deeplearning_shared > VeryDeepSuperResolutionDeepLearningExample
VeryDeepSuperResolutionDeepLearningExample.mlx

function, superResolutionMetrics, to compute the average metrics. This function is attached to the example as a supporting file.

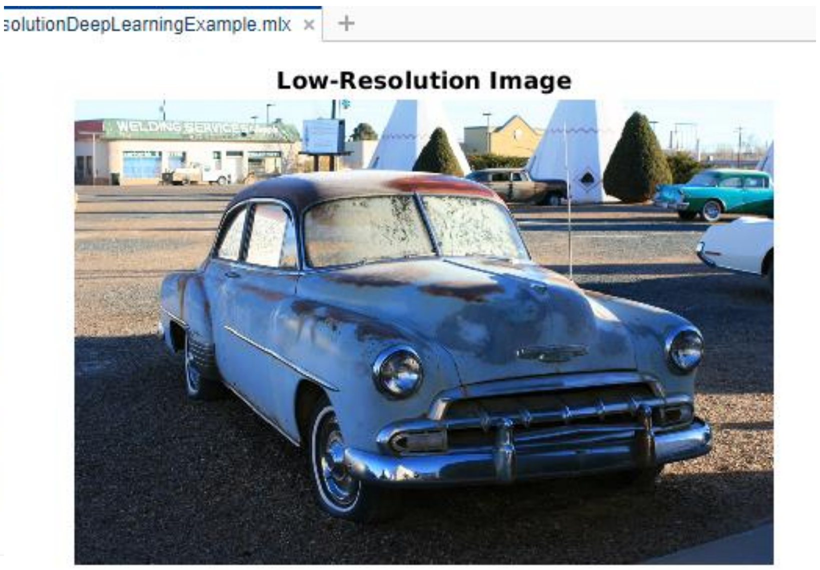
```
scaleFactors = [2 3 4];
superResolutionMetrics(net,testImages,scaleFactors);
```

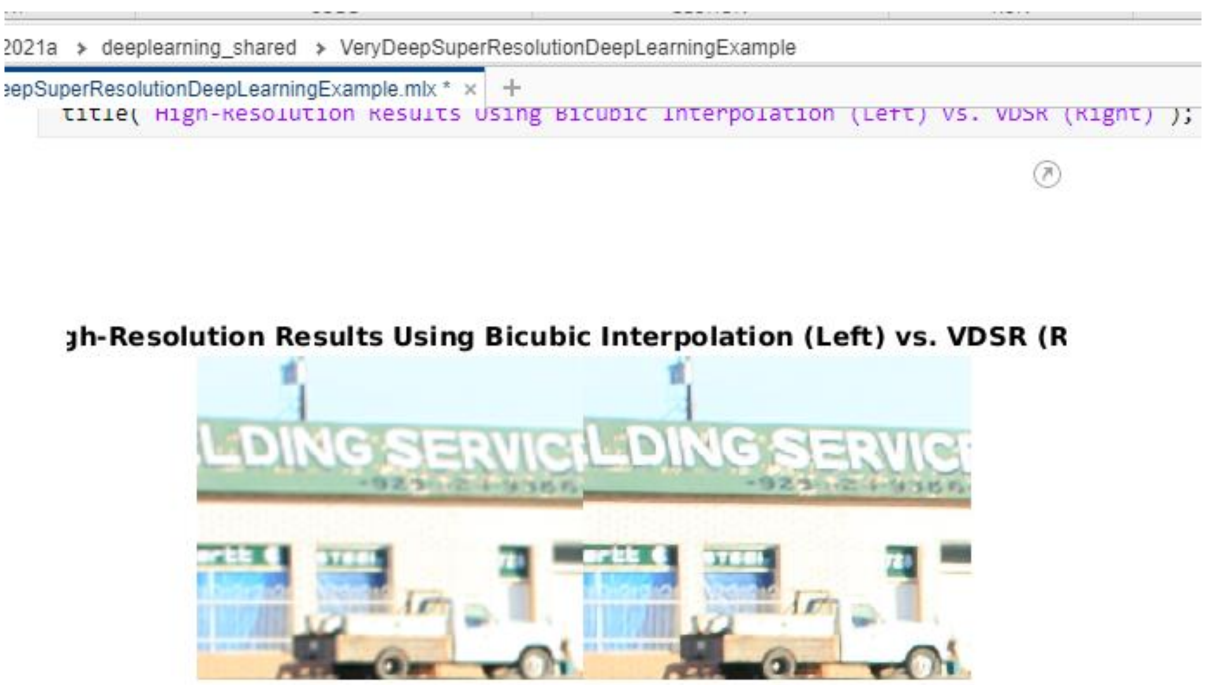
```
Results for Scale factor 2

Average PSNR for Bicubic = 31.809683
Average PSNR for VDSR = 31.921784
Average SSIM for Bicubic = 0.938194
Average SSIM for VDSR = 0.949404

Results for Scale factor 3

Average PSNR for Bicubic = 28.170441
Average PSNR for VDSR = 28.563952
Average SSIM for Bicubic = 0.884381
Average SSIM for VDSR = 0.895830

Results for Scale factor 4

Average PSNR for Bicubic = 27.010839
Average PSNR for VDSR = 27.837260
Average SSIM for Bicubic = 0.861604
Average SSIM for VDSR = 0.877132
```

VDSR has better metric scores than bicubic interpolation for each scale factor.

<u>Modification to the code</u>

There are two modifications to the code that we wanted to try. The first modification we made was: changing the specifications for the training options. We changed the number of epochs from 100 to 200 to allow the network more time to train. The second modification we made was: choosing a different image to test the accuracy of the resolution compared to the test image of the dog in the original code. The image we chose was: the image at index 2, the car.

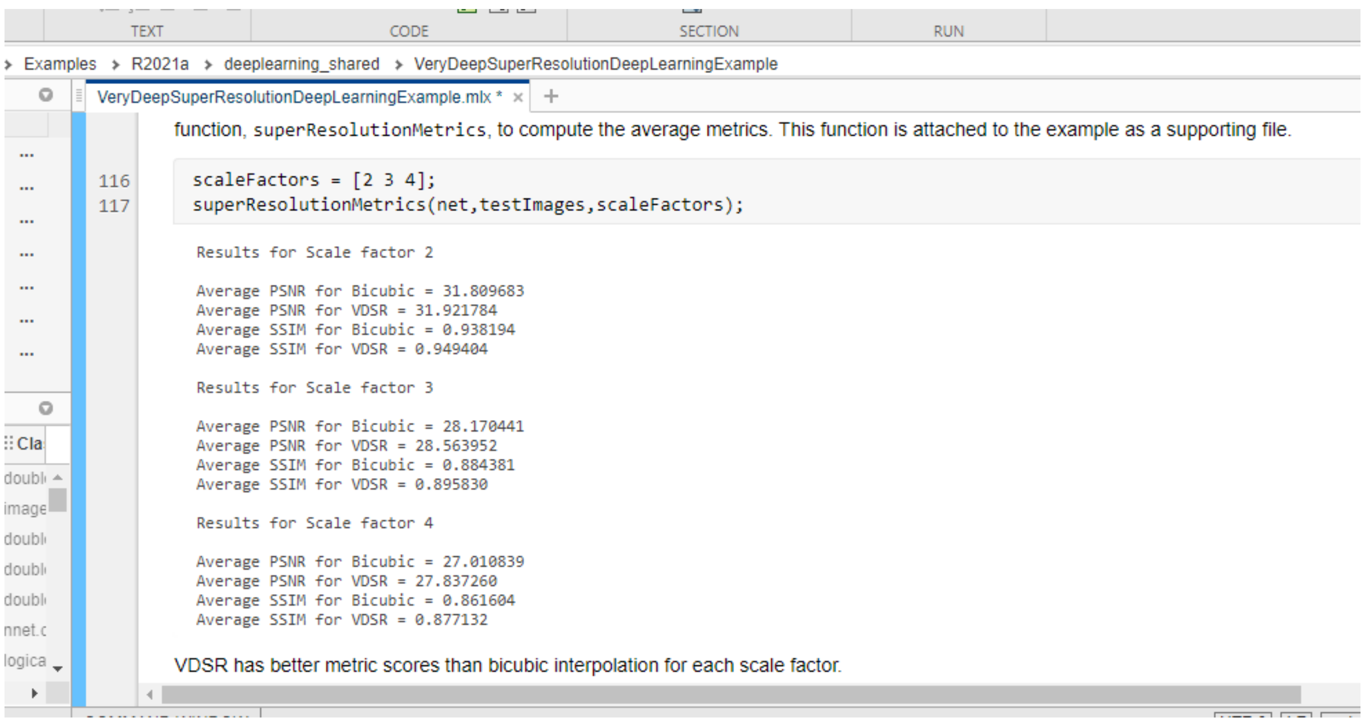



2021a › deeplearning_shared › VeryDeepSuperResolutionDeepLearningExample
epSuperResolutionDeepLearningExample.mlx *
gh-Resolution Results Using Bicubic Interpolation (Left) vs. VDSR (R

```
title('High-Resolution Results Using Bicubic Interpolation (Left) vs. VDSR (Right)');
```

Upon comparison with bicubic interpolation, we found that the quality was better for VDSR.

The image quality metrics stayed the same for the VDSR even though we increased the time the network spent training.


TEXT
CODE
SECTION
RUN
Examples > R2021a > deeplearning_shared > VeryDeepSuperResolutionDeepLearningExample
VeryDeepSuperResolutionDeepLearningExample.mlx *

function, superResolutionMetrics, to compute the average metrics. This function is attached to the example as a supporting file.

```
scaleFactors = [2 3 4];
superResolutionMetrics(net,testImages,scaleFactors);
```

```
Results for Scale factor 2

Average PSNR for Bicubic = 31.809683
Average PSNR for VDSR = 31.921784
Average SSIM for Bicubic = 0.938194
Average SSIM for VDSR = 0.949404

Results for Scale factor 3

Average PSNR for Bicubic = 28.170441
Average PSNR for VDSR = 28.563952
Average SSIM for Bicubic = 0.884381
Average SSIM for VDSR = 0.895830

Results for Scale factor 4

Average PSNR for Bicubic = 27.010839
Average PSNR for VDSR = 27.837260
Average SSIM for Bicubic = 0.861604
Average SSIM for VDSR = 0.877132
```

VDSR has better metric scores than bicubic interpolation for each scale factor.

Lessons Learned

Through this example, we learned about SISR and how a network can take low resolution images and turn them into high resolution images. We also learned that some deep networks have flaws and are not 100% accurate. We learned that the SISR method was more accurate than the bicubic interpolation method from the image quality metrics that were calculated in the code.

Ideas for the future

If we had more time, we may have tried a different deep learning method for SISR (SISR in this example used VDSR network). We could have included a graph showing the details and accuracy of the network as we have done in previous assignments. We also may have included an alternate way besides bicubic interpolation to compare how the deep learning network works.

Sources

https://arxiv.org/pdf/1501.00092.pdf

https://www.mathworks.com/help/deeplearning/ug/single-image-super-resolution-using-deep-learning.html?s_tid=mwa_osa_a