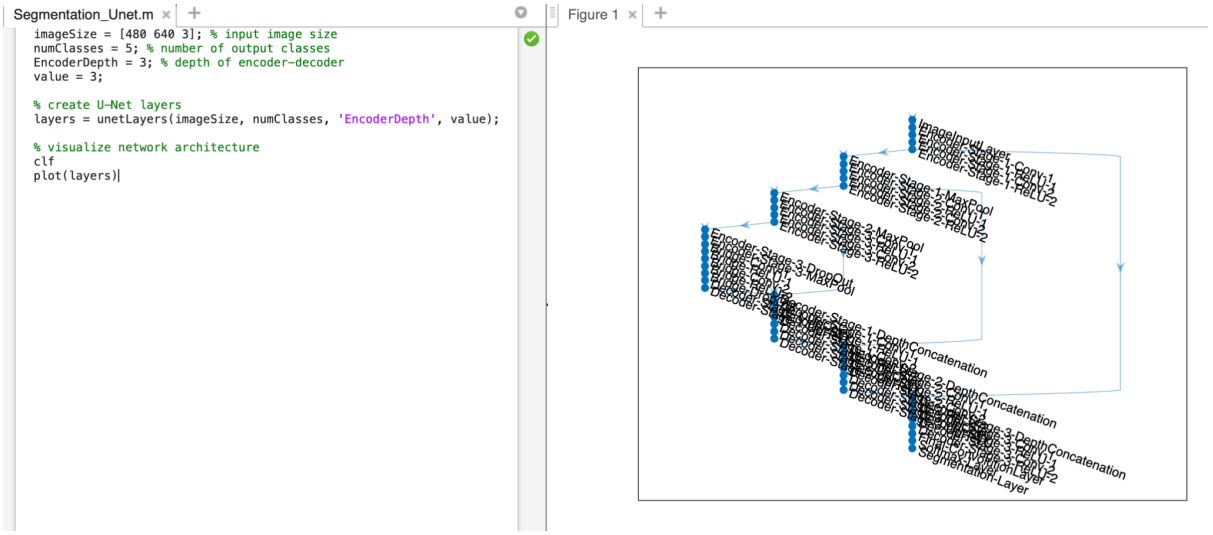
Alexa de León
A01382990

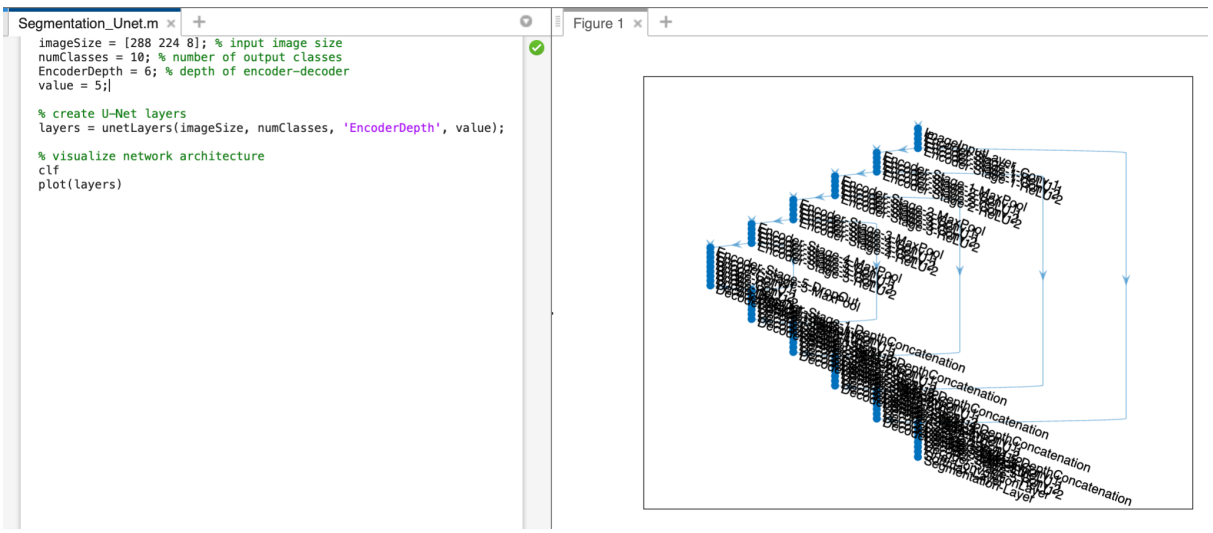
- For this activity be sure you have the Deep Learning Toolbox™ installed.

1) As a first activity you are going to learn how to create a U-Net network with an encoder-decoder depth of 3.

- Use unetLayers(imageSize,numClasses,Name,Value) with:
    - image size of 480x640x3 (introduce it as a vector)
    - five classes (it is a number)
    - Name: 'EncoderDepth'
    - Value should be the depth of the encoder-decoder (it is a number)
- To visualize the network use plot(the_output_of_uneLayers)



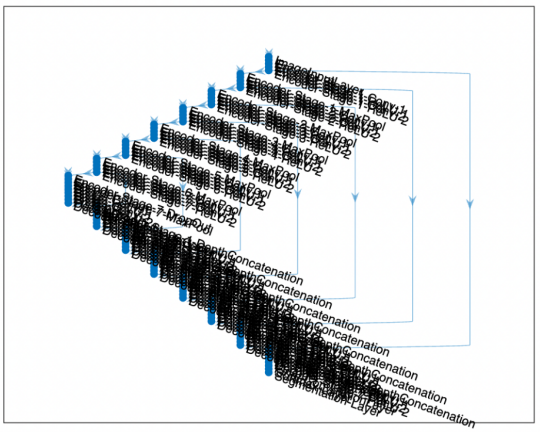
- Use different image size, number of classes, and depth of the encoder-decoder and observe how the plot changes. **Include at least three plots (using different specifications) in your report.**



Segmentation_Unet.m

```
imageSize = [256 256 8]; % input image size
numClasses = 2; % number of output classes
EncoderDepth = 1; % depth of encoder-decoder
value = 7;

% create U-Net layers
layers = unetLayers(imageSize, numClasses, 'EncoderDepth', value);

% visualize network architecture
clf
plot(layers)
```

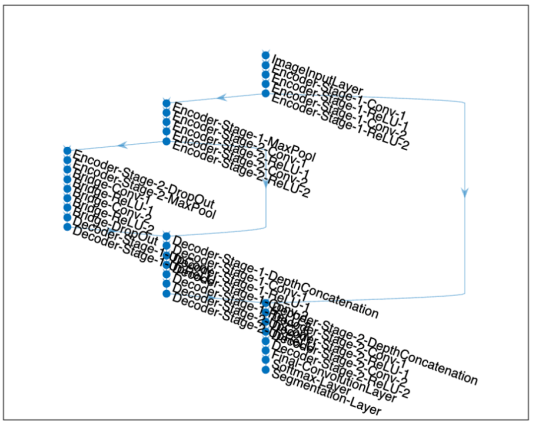
Figure 1



Segmentation_Unet.m

```
imageSize = [256 256 8]; % input image size
numClasses = 3; % number of output classes
EncoderDepth = 1; % depth of encoder-decoder
value = 2;

% create U-Net layers
layers = unetLayers(imageSize, numClasses, 'EncoderDepth', value);

% visualize network architecture
clf
plot(layers)
```

Figure 1



2) Now that you know how to create a U-Net network, you are going to learn how to train this type of network for semantic segmentation.

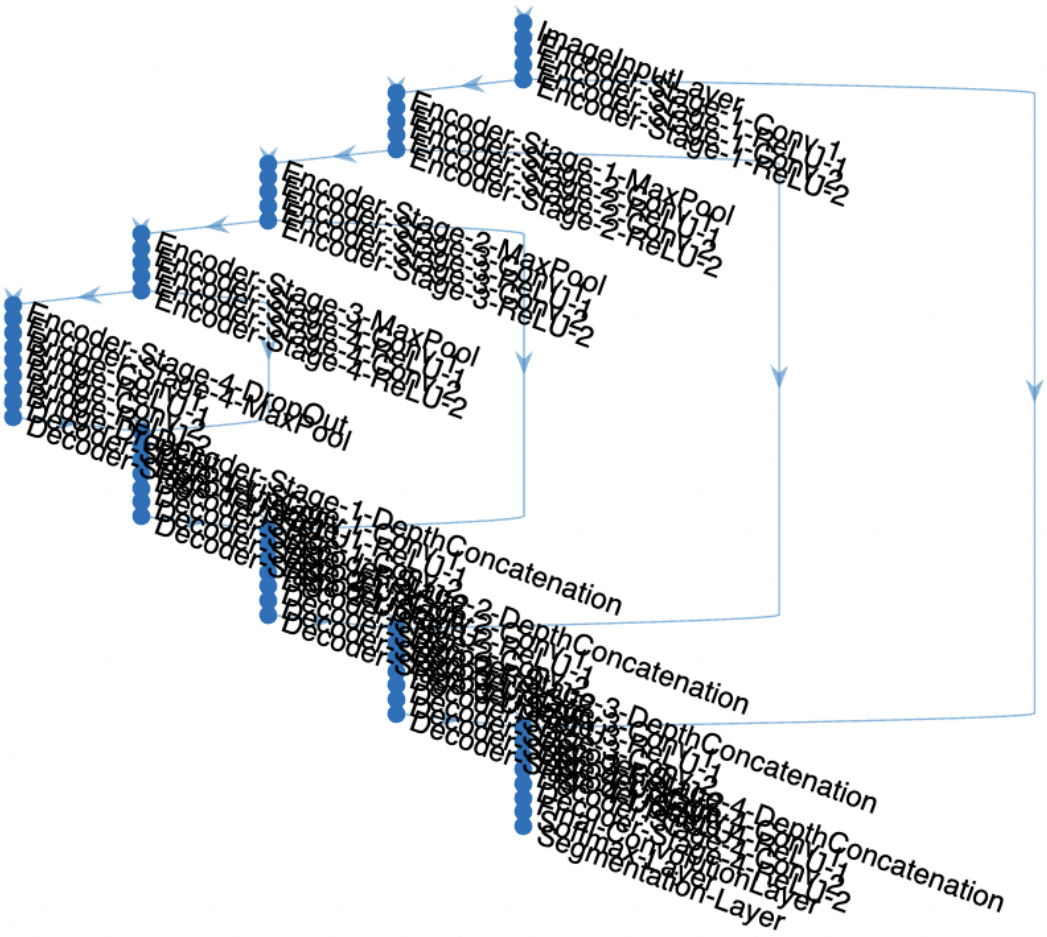
- First, you need to load the images and pixel labels that you are going to use for training in the workspace. For this excersie the images are of size 32x32 and we are working with two classes.

```
dataSetDir = fullfile(toolboxdir('vision'),'visiondata','triangleImages');
imageDir = fullfile(dataSetDir,'trainingImages');
labelDir = fullfile(dataSetDir,'trainingLabels');
```

- Now you need to create both an imageDatastore object and a pixelLabelDatastore object to store the training images and ground truth pixel labels, respectively.
    - Set the className as the vector ["triangle","b ackground"]
    - Set labelIDs as the vector [255 0]

imds = imageDatastore(imageDir);

pxds = pixelLabelDatastore(labelDir,classNames,labelIDs);



- Now you are ready to create the U-Net, use: unetLayers(imageSize, numClasses)
- Create a datastore for training the netwrok.

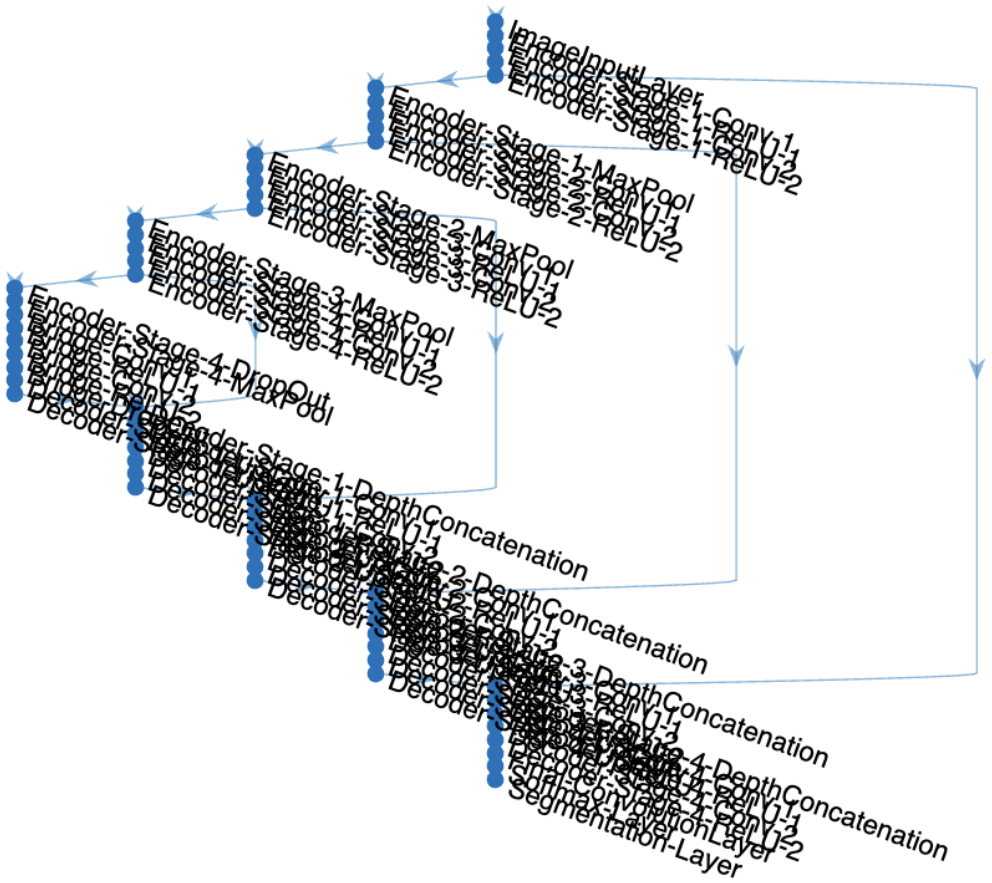
ds = combine(imds,pxds);

- Use the following training options.

options = trainingOptions('sgdm', ... 'InitialLearnRate',1e-3, ... 'MaxEpochs',20, ... 'VerboseFrequency',10);

- Finally, you are ready to train the network

net = trainNetwork(ds, the_output_of_uneLayers,options)



Command Window

```
>> Segmentation_Unet
Training on single CPU.
Initializing input data normalization.
|=========================================================================================|
|  Epoch  |  Iteration  |  Time Elapsed  |  Mini-batch  |  Mini-batch  |  Base Learning  |
|         |             |   (hh:mm:ss)   |   Accuracy   |     Loss     |      Rate       |
|=========================================================================================|
|       1 |           1 |       00:00:02 |       66.45% |       2.7845 |          0.0010 |
|      10 |          10 |       00:00:20 |       97.08% |       0.3148 |          0.0010 |
|      20 |          20 |       00:00:42 |       97.76% |       0.1491 |          0.0010 |
|=========================================================================================|
Training finished: Max epochs completed.
>>
```

- **Change the learning rate and increase the max epochs and observe the results. Report all your attempts and answer:**
  - **What is the most appropriate number of max epochs you can use? Why?**
    20 because it has more precision in mini-batch, and it has less loss.
  - **How did the learning rate affect the accuracy?**
    It affects because the larger learning rates the fewer training epochs it needs.

- First you need to remember the variable you used to store your trained network. In my case it was net, look in your previous code and find the following instructions (that is your trained network):

```
% Train the network
net = trainNetwork(ds,lgraph,options)
```

- To start your activity you need to load the testing data in your workspace.

```
% Specify test images and labels
testImagesDir = fullfile(dataSetDir,'testImages');
testimds = imageDatastore(testImagesDir);
testLabelsDir = fullfile(dataSetDir,'testLabels');
```

- Observe that you previously used these instructions but for training images.
- You need to create a pixelLabelDatastore object to hold the ground truth pixel labels for the test images.

```
pxdsTruth = pixelLabelDatastore(testLabelsDir,classNames,labelIDs);
```

- Now you're going to run your network on the test images (be patient and wait until the 100 images are processed)

```
pxdsResults = semanticseg(testimds,net,"WriteLocation",tempdir);
```

- To evaluate the quality of your prediction you are going to use the following instruction. It receives two arguments: your predictions, and the ground truth pixels.

```
metrics = evaluateSemanticSegmentation(your predictions,ground truth);
```

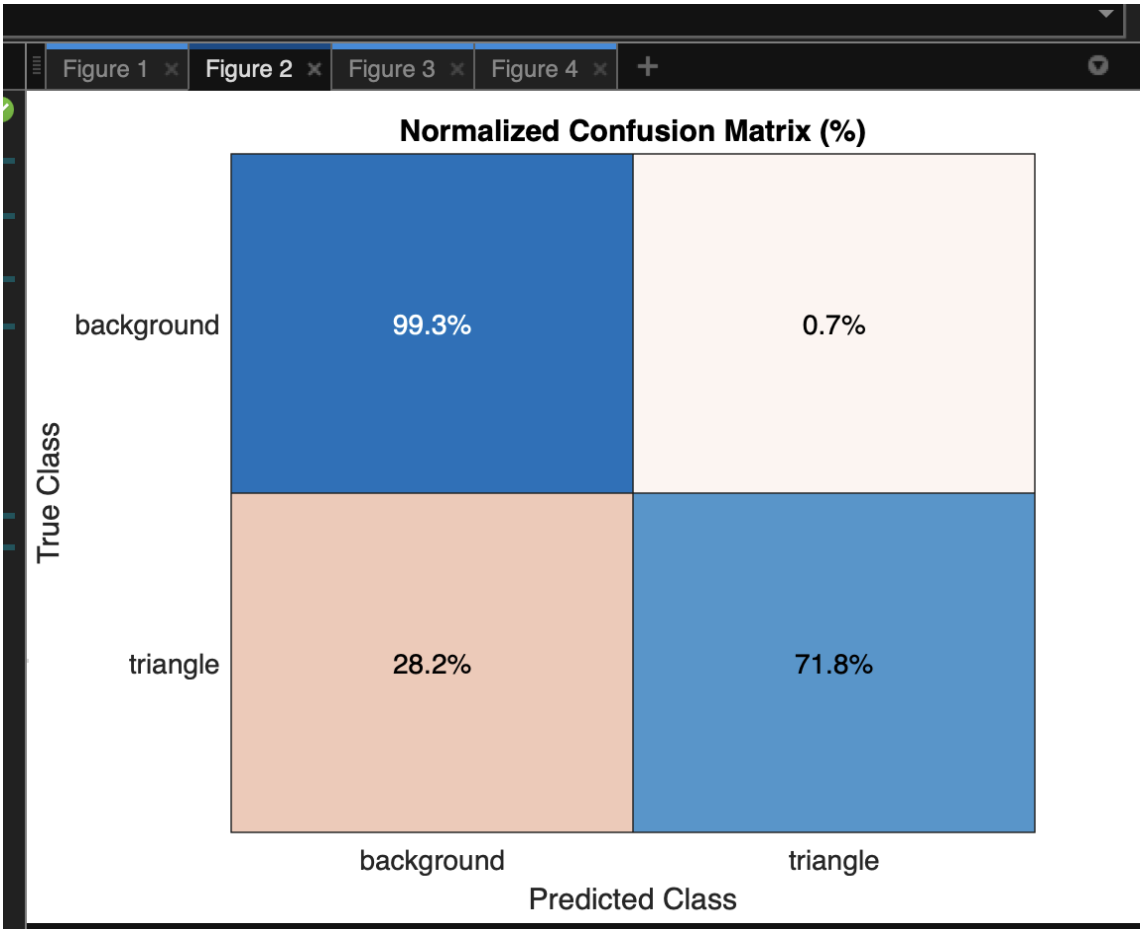
```
-------------------------------------
* Processed 1 images.
* Processed 100 images.

Evaluating semantic segmentation results
----------------------------------------
* Selected metrics: global accuracy, class accuracy, IoU, weighted IoU, BF score.
* Processed 0 images.
* Processed 40 images.
* Processed 100 images.
* Finalizing...
 Done.
* Data set metrics:

    GlobalAccuracy    MeanAccuracy    MeanIoU    WeightedIoU    MeanBFScore
    ______________    ____________    _______    ___________    ___________

       0.98072           0.85581       0.80627      0.96401        0.7778
```

- Now you are going to display your results by class, which means, how good were the regions (in this case they are triangles) and background identified separately.

```
%  Inspect class metrcis
metrics.ClassMetrics
```

```
% Display confusion matrix
metrics.ConfusionMatrix

% Visualize the normalized confusion matrix as a confusion chart in a figure window.
figure
cm = confusionchart(metrics.ConfusionMatrix.Variables, ...
classNames, Normalization='row-normalized');
cm.Title = 'Normalized Confusion Matrix (%)';
```

```
ans =

  2×3 table

                Accuracy      IoU      MeanBFScore
                ________    _______    ___________

    triangle    0.71818     0.63247      0.64752
    background  0.99344     0.98006      0.90808

ans =

  2×2 table

                triangle    background
                ________    __________

    triangle      3397         1333
    background     641        97029
```

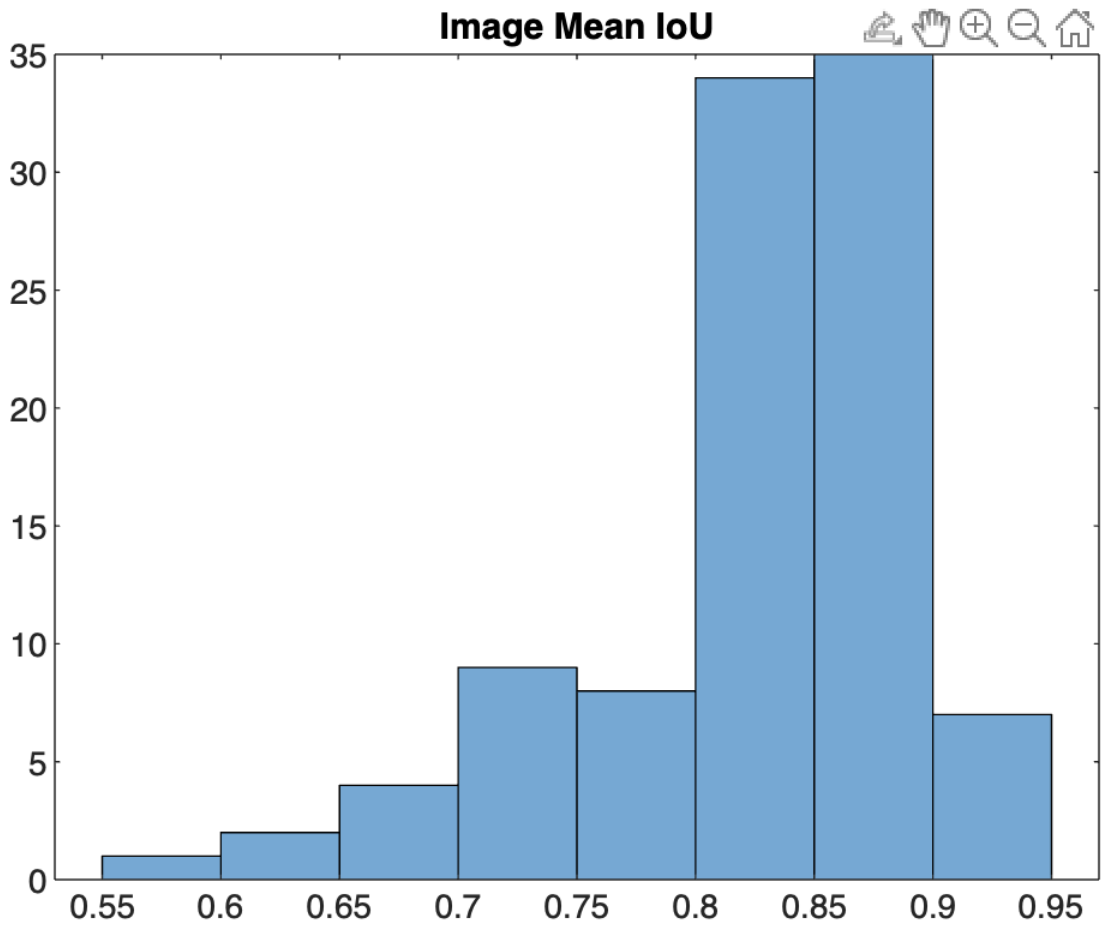


- Now visualize a histogram of the IoU per image.

```
imageIoU = metrics.ImageMetrics.MeanIoU;
figure (3)
histogram(imageIoU)
title('Image Mean IoU')
```

- **Include your histogram in the report and answer: what was the most common mean IoU through the images?**

  The most common image is 0.9



- Once you finished looking at the behavior of your segmentation through different metrics, let's visualize two examples: the images with the worst and the best mean IoU.
- Run this for the image with the lowest IoU:

```
% Find the test image with the lowest IoU.
[minIoU, worstImageIndex] = min(imageIoU);
minIoU = minIoU(1);
worstImageIndex = worstImageIndex(1);
```

```
% Read the test image with the worst IoU, its ground truth labels, and its predicted labels for
comparison.
worstTestImage = readimage(imds,worstImageIndex);
worstTrueLabels = readimage(pxdsTruth,worstImageIndex);
worstPredictedLabels = readimage(pxdsResults,worstImageIndex);


% Convert the label images to images that can be displayed in a figure window.
worstTrueLabelImage = im2uint8(worstTrueLabels == classNames(1));
worstPredictedLabelImage = im2uint8(worstPredictedLabels == classNames(1));


% Display the worst test image, the ground truth, and the prediction.
worstMontage = cat(4,worstTestImage,worstTrueLabelImage,worstPredictedLabelImage);
WorstMontage = imresize(worstMontage,4,"nearest");


figure (4)
montage(worstMontage,'Size',[1 3])
title(['Test Image vs. Truth vs. Prediction. IoU = ' num2str(minIoU)])
```

- In this last exercise you need to find the image with the highest IoU and display it by yourself.
- Note that all the instructions you need are similar to the code in the previous slide (you need to change at most two lines of code).
- **Report the images you obtained**