Instituto Tecnológico y de Estudios Superiores de Monterrey

Procesamiento de Imágenes Médicas para el Diagnóstico (Grupo 101)

Reporte de Actividad:
Actividad 1. Segmentation U-Net

Profesores
Ing. Beatriz Alejandra Bosques Palomo

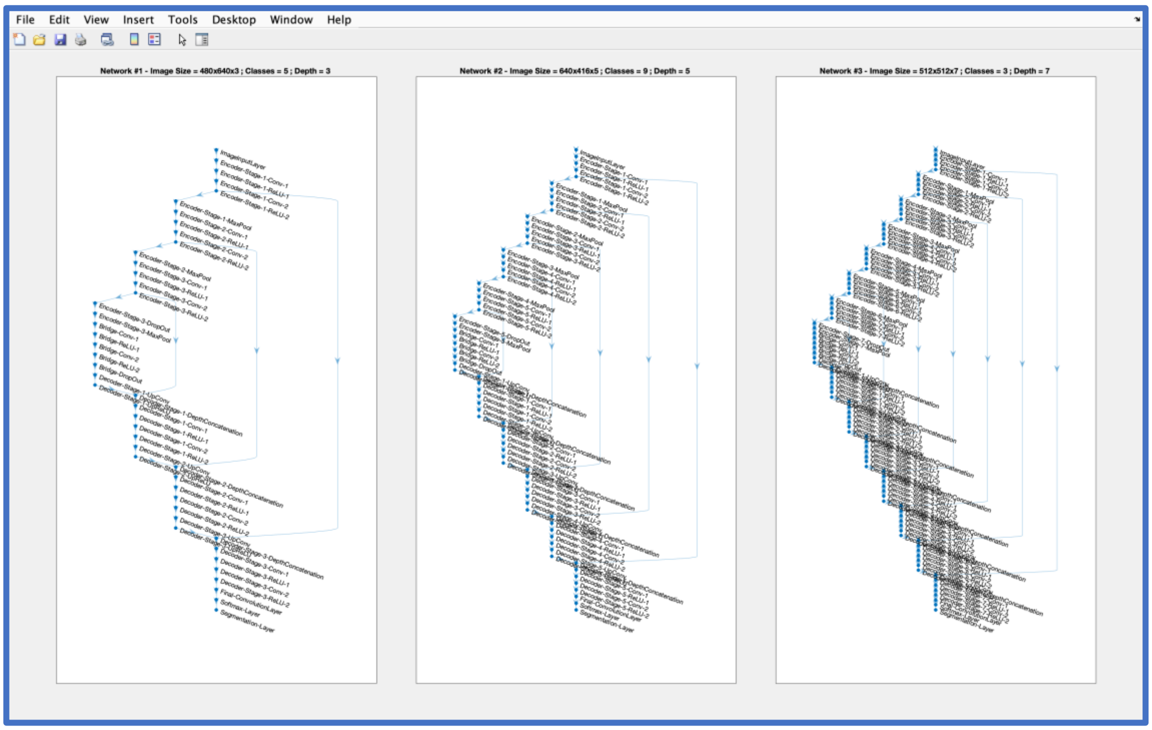
A01383088
Juan Luis Flores Sánchez

3 de marzo de 2023

Actividad 1. Segmentation U-Net

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 change. Include at least three plots (using different specifications) in your report.**



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)
```

- **Keep this last instruction without the ; to include the output in your report.**

```
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 |       00:00:05 |       88.89% |       1.2014 |          0.0010 |
|      10 |          10 |       00:00:54 |       96.11% |       0.4924 |          0.0010 |
|      20 |          20 |       00:01:48 |       97.44% |       0.1596 |          0.0010 |
|========================================================================================|
Training finished: Max epochs completed.

net1 =

  DAGNetwork with properties:

         Layers: [58×1 nnet.cnn.layer.Layer]
    Connections: [61×2 table]
     InputNames: {'ImageInputLayer'}
    OutputNames: {'Segmentation-Layer'}
```

- **Change the learning rate and increase the max epochs and observe the results.**

Try #1 – 10 Epochs

```
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 |       00:00:04 |       15.88% |      10.4025 |          0.0010 |
|      10 |          10 |       00:00:50 |       94.95% |       0.7571 |          0.0010 |
|========================================================================================|
Training finished: Max epochs completed.

net =

  DAGNetwork with properties:

         Layers: [58×1 nnet.cnn.layer.Layer]
    Connections: [61×2 table]
     InputNames: {'ImageInputLayer'}
    OutputNames: {'Segmentation-Layer'}
```

Try #2 – 20 Epochs

```
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 |       00:00:05 |       88.89% |       1.2014 |          0.0010 |
|      10 |          10 |       00:00:54 |       96.11% |       0.4924 |          0.0010 |
|      20 |          20 |       00:01:48 |       97.44% |       0.1596 |          0.0010 |
|========================================================================================|
Training finished: Max epochs completed.

net1 = 

  DAGNetwork with properties:

         Layers: [58×1 nnet.cnn.layer.Layer]
    Connections: [61×2 table]
     InputNames: {'ImageInputLayer'}
    OutputNames: {'Segmentation-Layer'}
```

Try #3 – 100 Epochs

```
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 |       00:00:05 |       41.71% |       5.8408 |          0.0010 |
|      10 |          10 |       00:00:56 |       94.95% |       0.5527 |          0.0010 |
|      20 |          20 |       00:01:47 |       95.84% |       0.2428 |          0.0010 |
|      30 |          30 |       00:02:38 |       96.97% |       0.1358 |          0.0010 |
|      40 |          40 |       00:03:30 |       97.39% |       0.0948 |          0.0010 |
|      50 |          50 |       00:04:24 |       97.69% |       0.0755 |          0.0010 |
|      60 |          60 |       00:05:17 |       97.91% |       0.0652 |          0.0010 |
|      70 |          70 |       00:06:10 |       98.09% |       0.0585 |          0.0010 |
|      80 |          80 |       00:07:04 |       98.21% |       0.0536 |          0.0010 |
|      90 |          90 |       00:07:56 |       98.32% |       0.0498 |          0.0010 |
|     100 |         100 |       00:08:54 |       98.41% |       0.0467 |          0.0010 |
|========================================================================================|
Training finished: Max epochs completed.

net2 = 

  DAGNetwork with properties:

         Layers: [58×1 nnet.cnn.layer.Layer]
    Connections: [61×2 table]
     InputNames: {'ImageInputLayer'}
    OutputNames: {'Segmentation-Layer'}
```

Try #4 – 200 Epochs

```
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 |       00:00:05 |       67.39% |       2.7954 |          0.0010 |
|      10 |          10 |       00:00:53 |       97.15% |       0.3677 |          0.0010 |
|      20 |          20 |       00:01:47 |       97.82% |       0.1698 |          0.0010 |
|      30 |          30 |       00:02:40 |       98.39% |       0.1039 |          0.0010 |
|      40 |          40 |       00:03:31 |       98.57% |       0.0813 |          0.0010 |
|      50 |          50 |       00:04:21 |       98.69% |       0.0629 |          0.0010 |
|      60 |          60 |       00:05:12 |       98.80% |       0.0520 |          0.0010 |
|      70 |          70 |       00:06:05 |       98.88% |       0.0454 |          0.0010 |
|      80 |          80 |       00:06:58 |       98.96% |       0.0403 |          0.0010 |
|      90 |          90 |       00:07:49 |       99.03% |       0.0364 |          0.0010 |
|     100 |         100 |       00:08:40 |       99.08% |       0.0334 |          0.0010 |
|     110 |         110 |       00:09:30 |       99.14% |       0.0310 |          0.0010 |
|     120 |         120 |       00:10:22 |       99.18% |       0.0290 |          0.0010 |
|     130 |         130 |       00:11:13 |       99.23% |       0.0271 |          0.0010 |
|     140 |         140 |       00:11:56 |       99.26% |       0.0256 |          0.0010 |
|     150 |         150 |       00:12:43 |       99.30% |       0.0242 |          0.0010 |
|     160 |         160 |       00:13:30 |       99.32% |       0.0230 |          0.0010 |
|     170 |         170 |       00:14:19 |       99.35% |       0.0219 |          0.0010 |
|     180 |         180 |       00:15:08 |       99.37% |       0.0210 |          0.0010 |
|     190 |         190 |       00:15:58 |       99.39% |       0.0201 |          0.0010 |
|     200 |         200 |       00:16:48 |       99.41% |       0.0194 |          0.0010 |
|========================================================================================|
Training finished: Max epochs completed.
```

Try #5 – 20 Epochs / 1 Learning Rate

```
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 |       00:00:04 |       71.55% |       2.6460 |          1.0000 |
|       3 |           3 |       00:00:13 |       18.25% |          NaN |          1.0000 |
|========================================================================================|
Training finished: Training loss is NaN.
Warning: Training stopped at iteration 3 because training loss is NaN. Predictions using the output network might contain NaN
values.

net = 

  DAGNetwork with properties:

         Layers: [58×1 nnet.cnn.layer.Layer]
    Connections: [61×2 table]
     InputNames: {'ImageInputLayer'}
    OutputNames: {'Segmentation-Layer'}
```

Try #6 – 20 Epochs / 0.1 Learning Rate

```
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 |       00:00:04 |        7.87% |       9.7986 |          0.1000 |
|      10 |          10 |       00:00:44 |       94.84% |       0.2046 |          0.1000 |
|      20 |          20 |       00:01:27 |       94.84% |       0.2344 |          0.1000 |
|========================================================================================|
Training finished: Max epochs completed.

net1 =

  DAGNetwork with properties:

         Layers: [58×1 nnet.cnn.layer.Layer]
    Connections: [61×2 table]
     InputNames: {'ImageInputLayer'}
    OutputNames: {'Segmentation-Layer'}
```

Try #7 – 20 Epochs / 0.01 Learning Rate

```
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 |       00:00:05 |       27.10% |       7.3595 |          0.0100 |
|       7 |           7 |       00:00:39 |        5.28% |          NaN |          0.0100 |
|========================================================================================|
Training finished: Training loss is NaN.
Warning: Training stopped at iteration 7 because training loss is NaN. Predictions using the output network might contain NaN
values.

net2 =

  DAGNetwork with properties:

         Layers: [58×1 nnet.cnn.layer.Layer]
    Connections: [61×2 table]
     InputNames: {'ImageInputLayer'}
    OutputNames: {'Segmentation-Layer'}
```

Try #8 – 20 Epochs / 0.001 Learning Rate

```
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 |       00:00:05 |       53.09% |       3.2176 |          0.0010 |
|      10 |          10 |       00:00:57 |       96.17% |       0.4908 |          0.0010 |
|      20 |          20 |       00:01:56 |       97.82% |       0.2266 |          0.0010 |
|========================================================================================|
Training finished: Max epochs completed.

net3 = 

  DAGNetwork with properties:

         Layers: [58×1 nnet.cnn.layer.Layer]
    Connections: [61×2 table]
     InputNames: {'ImageInputLayer'}
    OutputNames: {'Segmentation-Layer'}
```

Try #9 – 20 Epochs / 0.0001 Learning Rate

```
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 |       00:00:05 |       45.79% |       3.9943 |      1.0000e-04 |
|      10 |          10 |       00:00:58 |       95.04% |       0.6797 |      1.0000e-04 |
|      20 |          20 |       00:01:53 |       96.72% |       0.4437 |      1.0000e-04 |
|========================================================================================|
Training finished: Max epochs completed.

net4 = 

  DAGNetwork with properties:

         Layers: [58×1 nnet.cnn.layer.Layer]
    Connections: [61×2 table]
     InputNames: {'ImageInputLayer'}
    OutputNames: {'Segmentation-Layer'}
```

- **Report all your attempts and answer:**
  - **What is the most appropriate number of max epochs you can use? Why?**

The max epochs you can use is up to the context, because if you want to calculate something quick but with a 5% of uncertainty (engineering standard), the best option is to compute with 20 epochs getting the result under 2 minutes. On the other hand, if you want to calculate something with just 1% of uncertainty (medical standard), the best option is to compute with 200 epochs getting the result in 20 minutes.

- **How did the learning rate affect the accuracy?**

The learning rate have an inversely proportional affect on the accuracy, because as the learning rate decreases, the accuracy increase. But, it has a proportional affect in the time of computation, thus the 0.001 learning rate is widely suggested.

- **Previously, you trained a U-Net network, now you are going to evaluate how good was your training and you are going to look at some of your segmented images.**
- **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);
```

At this point you should obtain something similar to this (include that output in your report): The important metric for you is the IoU (intersection-over-union)

```
Running semantic segmentation network
-------------------------------------
* Processed 100 images.

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

    GlobalAccuracy    MeanAccuracy    MeanIoU    WeightedIoU    MeanBFScore
    ______________    ____________    _______    ___________    ___________

       0.96981          0.92915       0.7718       0.95035        0.61979
```

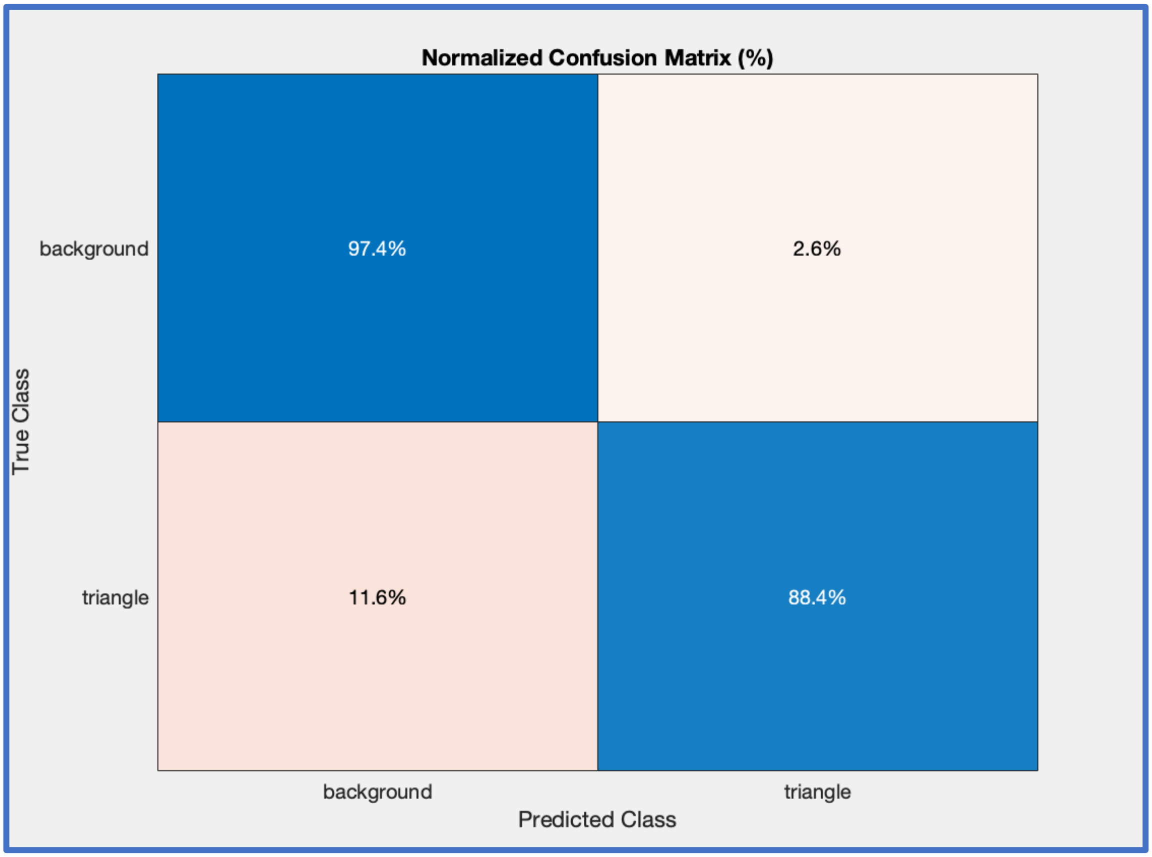
- **Now you are going to display your results by class, which means, how good were the regions (in this case they are triangles) and background identified separately.**

```
%  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 (%)';
```

At this point you should obtain something similar to this (include in your report):

- **This result is the same you obtained previously but specified by class.**
- **This result is the confusion matrix, that tells you:**
  - **how many pixels belonging to a triangle (the region for segmentation) were correctly identified (true positives, triangle-triangle),**
  - **how many pixels belonging to a triangle were incorrectly identified (false negatives, triangle-background)**
  - **the number of pixels belonging to the background that were correctly identified (true negatives, background-background)**
  - **the number of pixels belonging to the background that were incorrectly identified (false positives, background-triangle).**

```
ans =

  2×3 table

                  Accuracy      IoU      MeanBFScore
                  ________    _______    ___________

    triangle      0.88436     0.57506      0.38802
    background    0.97395     0.96853      0.85157


ans =

  2×2 table

                  triangle    background
                  ________    __________

    triangle        4183          547
    background      2544        95126
```

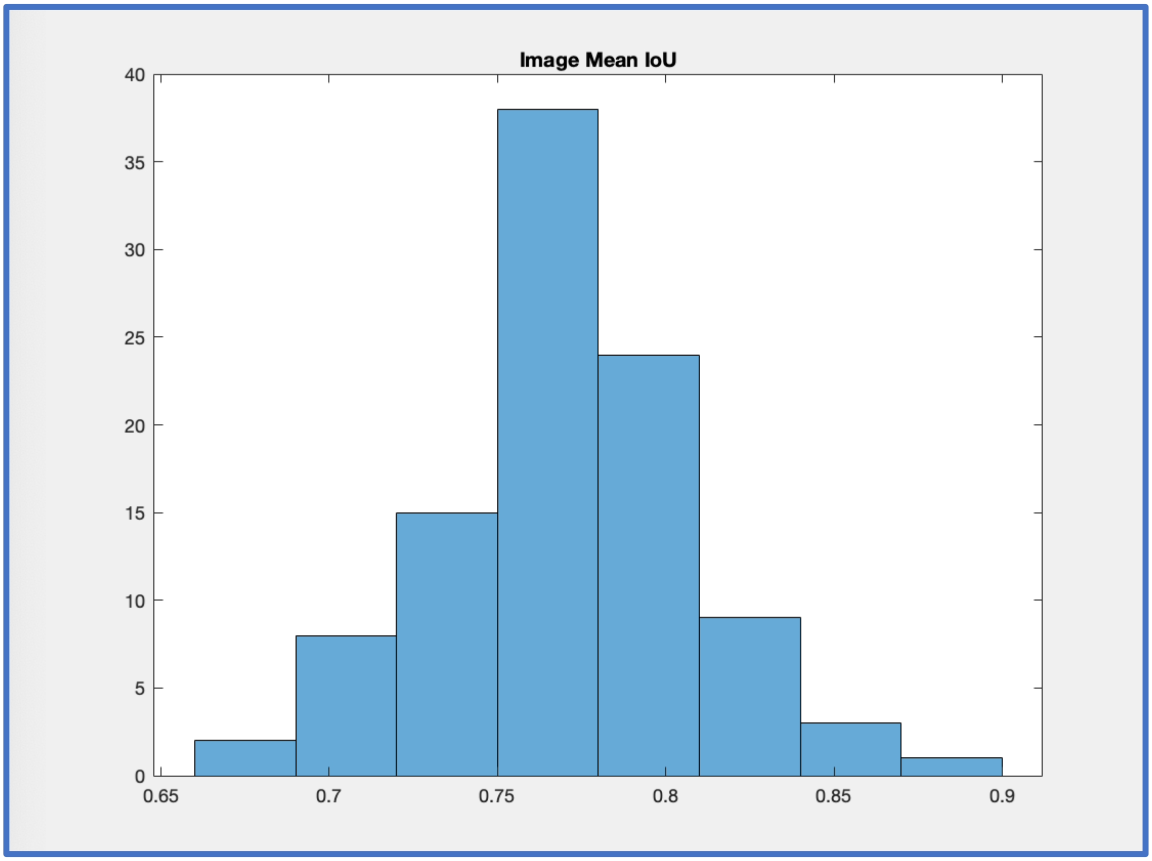


- **Now visualize a histogram of the IoU per image.**

```
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 mean IoU was between 0.78 and 0.81 values.

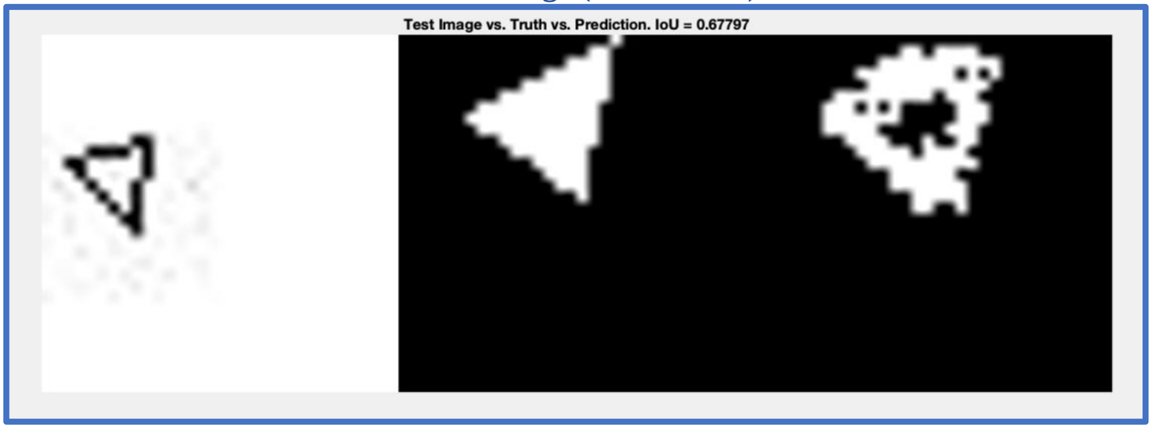


- **Once you finished looking at the behavior of your segmentation through different metrics, let's visualize two examples: the images with the worst and the best mean IoU.**
- **Run this for the image with the lowest IoU:**

```
% 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**

Worst Image (Lowest IoU)



Best Image (Highest IoU)