Machine Learning Assignment 2

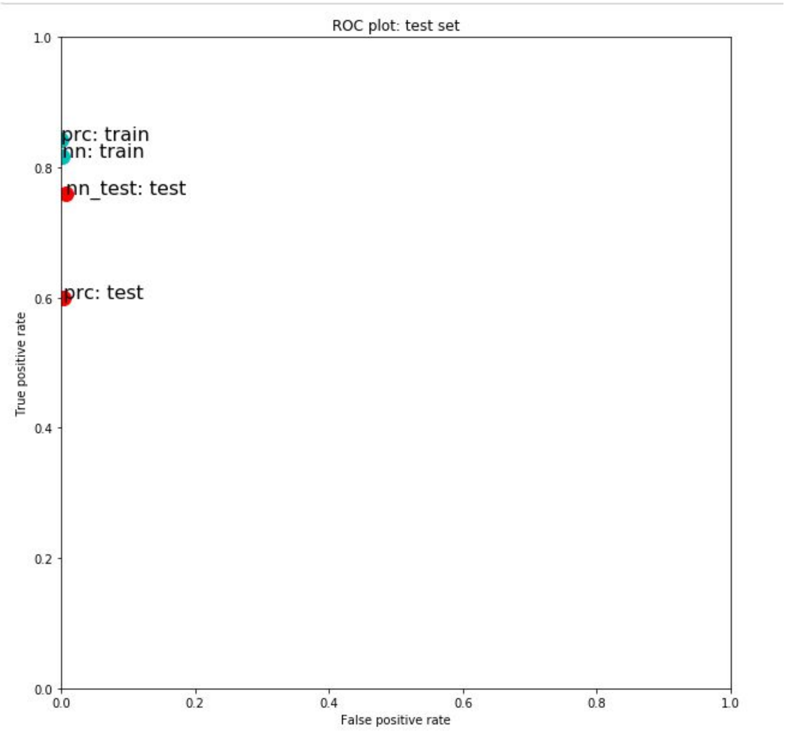
Lee Kuczewski
3.31.20 - 4.6.20
Flying Dollar Airport Assignment

Notes:

- Time of day - consider time of day as feature, as well as month, day of week, etc.
- Slope / angle of plane for take off/landing.
- Direction of the plane movement, whether inbound or outbound.
    - Landings on RWY 02
    - Departures on RWY 20
- Is the airplane in the foreground or background

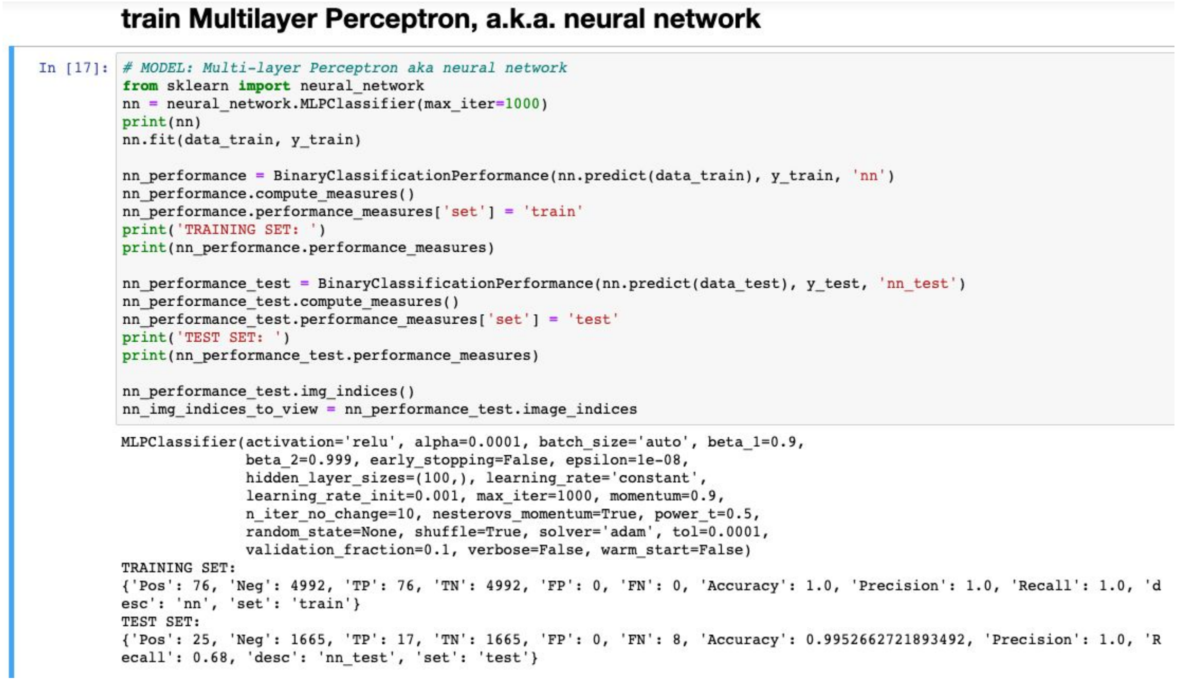
6758 and 101 contain airplanes.
1.49% of photos contain airplanes.

Starting around here after small adjustments.



Some of the many changes made

Training on Neural Network:
Default: Relu and Max_iter=1000

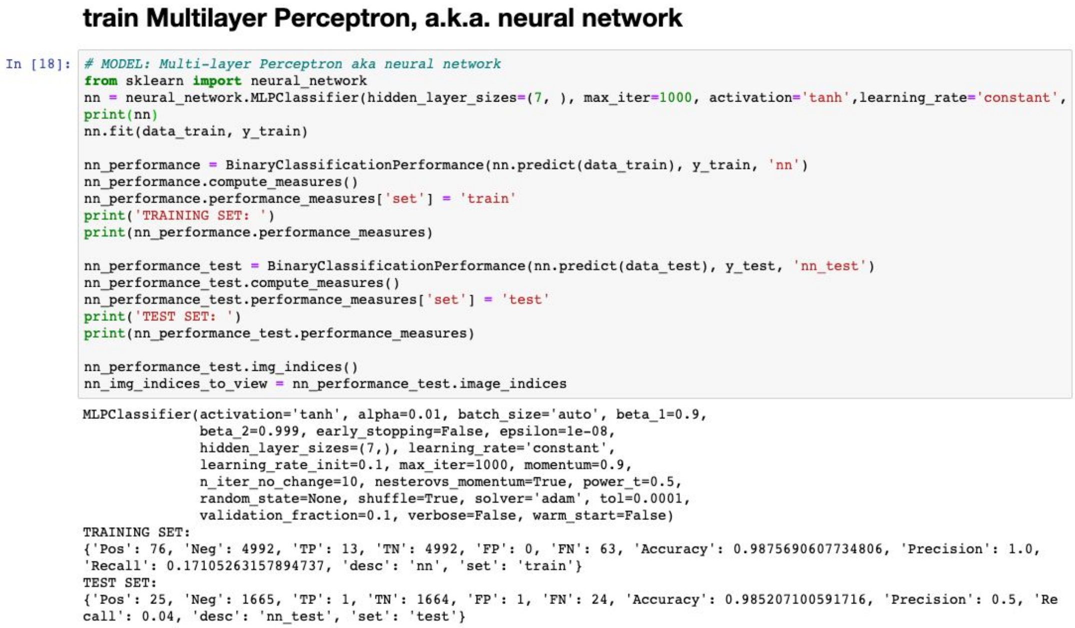


train Multilayer Perceptron, a.k.a. neural network

```
In [17]: # MODEL: Multi-layer Perceptron aka neural network
         from sklearn import neural_network
         nn = neural_network.MLPClassifier(max_iter=1000)
         print(nn)
         nn.fit(data_train, y_train)

         nn_performance = BinaryClassificationPerformance(nn.predict(data_train), y_train, 'nn')
         nn_performance.compute_measures()
         nn_performance.performance_measures['set'] = 'train'
         print('TRAINING SET: ')
         print(nn_performance.performance_measures)

         nn_performance_test = BinaryClassificationPerformance(nn.predict(data_test), y_test, 'nn_test')
         nn_performance_test.compute_measures()
         nn_performance_test.performance_measures['set'] = 'test'
         print('TEST SET: ')
         print(nn_performance_test.performance_measures)

         nn_performance_test.img_indices()
         nn_img_indices_to_view = nn_performance_test.image_indices
```

```
MLPClassifier(activation='relu', alpha=0.0001, batch_size='auto', beta_1=0.9,
              beta_2=0.999, early_stopping=False, epsilon=1e-08,
              hidden_layer_sizes=(100,), learning_rate='constant',
              learning_rate_init=0.001, max_iter=1000, momentum=0.9,
              n_iter_no_change=10, nesterovs_momentum=True, power_t=0.5,
              random_state=None, shuffle=True, solver='adam', tol=0.0001,
              validation_fraction=0.1, verbose=False, warm_start=False)
TRAINING SET:
{'Pos': 76, 'Neg': 4992, 'TP': 76, 'TN': 4992, 'FP': 0, 'FN': 0, 'Accuracy': 1.0, 'Precision': 1.0, 'Recall': 1.0, 'd
esc': 'nn', 'set': 'train'}
TEST SET:
{'Pos': 25, 'Neg': 1665, 'TP': 17, 'TN': 1665, 'FP': 0, 'FN': 8, 'Accuracy': 0.9952662721893492, 'Precision': 1.0, 'R
ecall': 0.68, 'desc': 'nn_test', 'set': 'test'}
```

Change Hidden Layer size to 7, activation function: tahn, learning rate= constant, learning rate init 0.1, alpha 0.1.

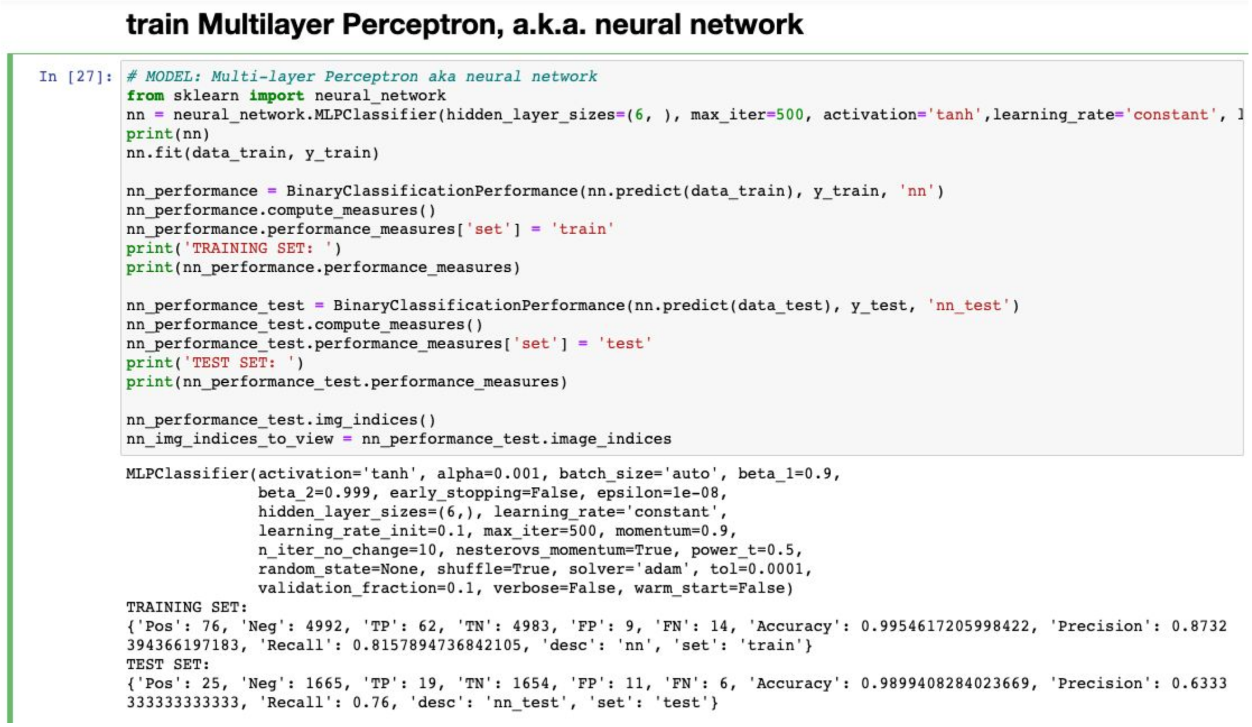


train Multilayer Perceptron, a.k.a. neural network

```
In [18]: # MODEL: Multi-layer Perceptron aka neural network
         from sklearn import neural_network
         nn = neural_network.MLPClassifier(hidden_layer_sizes=(7, ), max_iter=1000, activation='tanh',learning_rate='constant',
         print(nn)
         nn.fit(data_train, y_train)

         nn_performance = BinaryClassificationPerformance(nn.predict(data_train), y_train, 'nn')
         nn_performance.compute_measures()
         nn_performance.performance_measures['set'] = 'train'
         print('TRAINING SET: ')
         print(nn_performance.performance_measures)

         nn_performance_test = BinaryClassificationPerformance(nn.predict(data_test), y_test, 'nn_test')
         nn_performance_test.compute_measures()
         nn_performance_test.performance_measures['set'] = 'test'
         print('TEST SET: ')
         print(nn_performance_test.performance_measures)

         nn_performance_test.img_indices()
         nn_img_indices_to_view = nn_performance_test.image_indices
```

```
MLPClassifier(activation='tanh', alpha=0.01, batch_size='auto', beta_1=0.9,
              beta_2=0.999, early_stopping=False, epsilon=1e-08,
              hidden_layer_sizes=(7,), learning_rate='constant',
              learning_rate_init=0.1, max_iter=1000, momentum=0.9,
              n_iter_no_change=10, nesterovs_momentum=True, power_t=0.5,
              random_state=None, shuffle=True, solver='adam', tol=0.0001,
              validation_fraction=0.1, verbose=False, warm_start=False)
TRAINING SET:
{'Pos': 76, 'Neg': 4992, 'TP': 13, 'TN': 4992, 'FP': 0, 'FN': 63, 'Accuracy': 0.9875690607734806, 'Precision': 1.0,
'Recall': 0.17105263157894737, 'desc': 'nn', 'set': 'train'}
TEST SET:
{'Pos': 25, 'Neg': 1665, 'TP': 1, 'TN': 1664, 'FP': 1, 'FN': 24, 'Accuracy': 0.985207100591716, 'Precision': 0.5, 'Re
call': 0.04, 'desc': 'nn_test', 'set': 'test'}
```

Hidden layer size= 6, Max iter=500, activation function 'tanh'

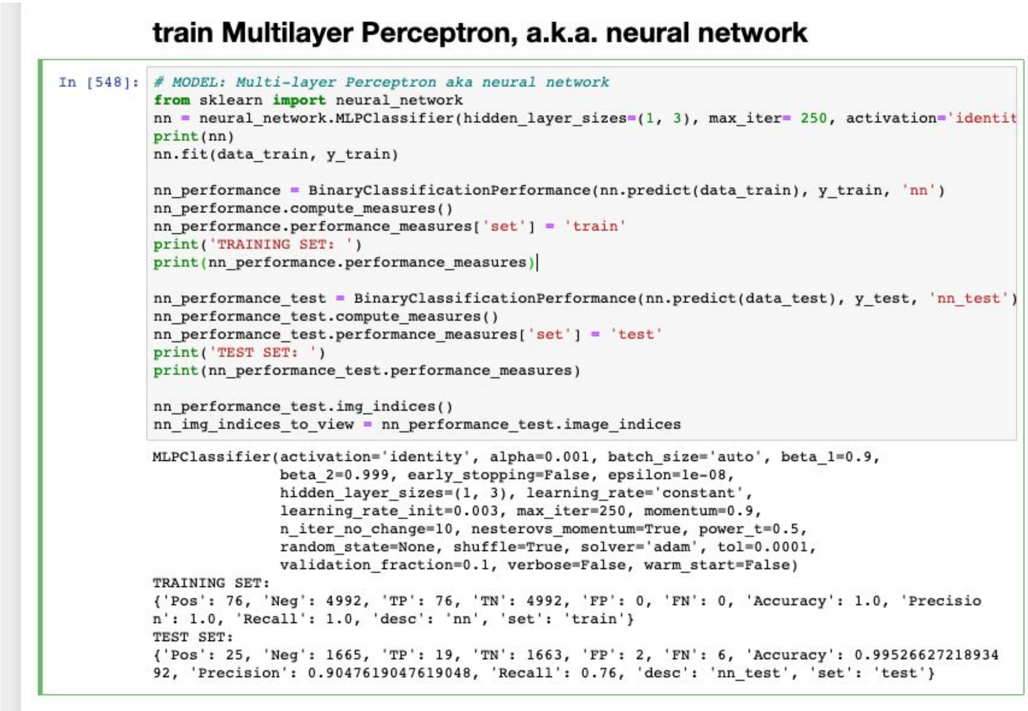


train Multilayer Perceptron, a.k.a. neural network

```
In [27]: # MODEL: Multi-layer Perceptron aka neural network
         from sklearn import neural_network
         nn = neural_network.MLPClassifier(hidden_layer_sizes=(6, ), max_iter=500, activation='tanh',learning_rate='constant', l
         print(nn)
         nn.fit(data_train, y_train)

         nn_performance = BinaryClassificationPerformance(nn.predict(data_train), y_train, 'nn')
         nn_performance.compute_measures()
         nn_performance.performance_measures['set'] = 'train'
         print('TRAINING SET: ')
         print(nn_performance.performance_measures)

         nn_performance_test = BinaryClassificationPerformance(nn.predict(data_test), y_test, 'nn_test')
         nn_performance_test.compute_measures()
         nn_performance_test.performance_measures['set'] = 'test'
         print('TEST SET: ')
         print(nn_performance_test.performance_measures)

         nn_performance_test.img_indices()
         nn_img_indices_to_view = nn_performance_test.image_indices
```

```
MLPClassifier(activation='tanh', alpha=0.001, batch_size='auto', beta_1=0.9,
              beta_2=0.999, early_stopping=False, epsilon=1e-08,
              hidden_layer_sizes=(6,), learning_rate='constant',
              learning_rate_init=0.1, max_iter=500, momentum=0.9,
              n_iter_no_change=10, nesterovs_momentum=True, power_t=0.5,
              random_state=None, shuffle=True, solver='adam', tol=0.0001,
              validation_fraction=0.1, verbose=False, warm_start=False)
TRAINING SET:
{'Pos': 76, 'Neg': 4992, 'TP': 62, 'TN': 4983, 'FP': 9, 'FN': 14, 'Accuracy': 0.9954617205998422, 'Precision': 0.8732
394366197183, 'Recall': 0.8157894736842105, 'desc': 'nn', 'set': 'train'}
TEST SET:
{'Pos': 25, 'Neg': 1665, 'TP': 19, 'TN': 1654, 'FP': 11, 'FN': 6, 'Accuracy': 0.9899408284023669, 'Precision': 0.6333
333333333333, 'Recall': 0.76, 'desc': 'nn_test', 'set': 'test'}
```

Hidden Layers to 1, 3, Max_iter to 250, activation functio to identity, etc.

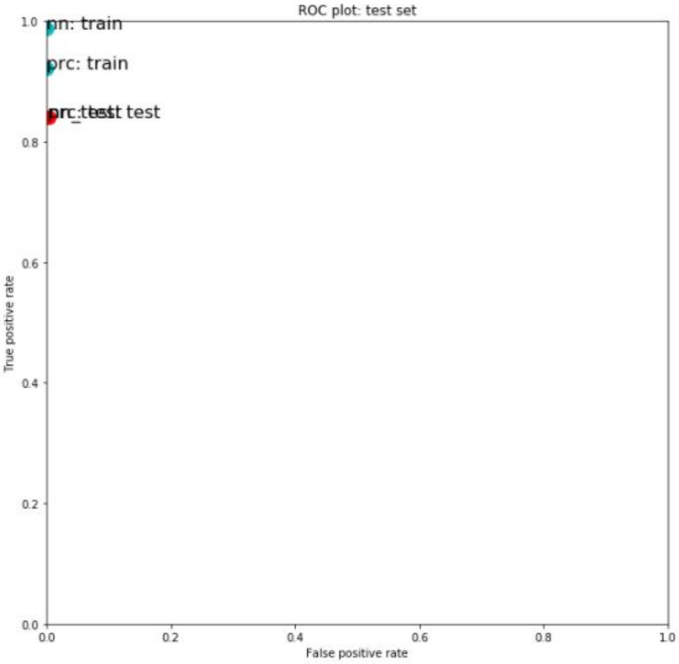


train Multilayer Perceptron, a.k.a. neural network

```
In [548]: # MODEL: Multi-layer Perceptron aka neural network
          from sklearn import neural_network
          nn = neural_network.MLPClassifier(hidden_layer_sizes=(1, 3), max_iter= 250, activation='identit
          print(nn)
          nn.fit(data_train, y_train)

          nn_performance = BinaryClassificationPerformance(nn.predict(data_train), y_train, 'nn')
          nn_performance.compute_measures()
          nn_performance.performance_measures['set'] = 'train'
          print('TRAINING SET: ')
          print(nn_performance.performance_measures)

          nn_performance_test = BinaryClassificationPerformance(nn.predict(data_test), y_test, 'nn_test')
          nn_performance_test.compute_measures()
          nn_performance_test.performance_measures['set'] = 'test'
          print('TEST SET: ')
          print(nn_performance_test.performance_measures)

          nn_performance_test.img_indices()
          nn_img_indices_to_view = nn_performance_test.image_indices
```

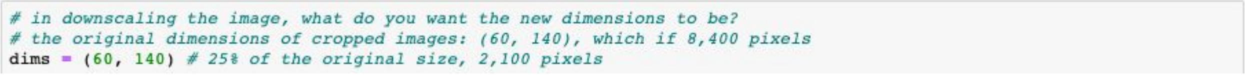
```
MLPClassifier(activation='identity', alpha=0.001, batch_size='auto', beta_1=0.9,
              beta_2=0.999, early_stopping=False, epsilon=1e-08,
              hidden_layer_sizes=(1, 3), learning_rate='constant',
              learning_rate_init=0.003, max_iter=250, momentum=0.9,
              n_iter_no_change=10, nesterovs_momentum=True, power_t=0.5,
              random_state=None, shuffle=True, solver='adam', tol=0.0001,
              validation_fraction=0.1, verbose=False, warm_start=False)
TRAINING SET:
{'Pos': 76, 'Neg': 4992, 'TP': 76, 'TN': 4992, 'FP': 0, 'FN': 0, 'Accuracy': 1.0, 'Precisio
n': 1.0, 'Recall': 1.0, 'desc': 'nn', 'set': 'train'}
TEST SET:
{'Pos': 25, 'Neg': 1665, 'TP': 19, 'TN': 1663, 'FP': 2, 'FN': 6, 'Accuracy': 0.99526627218934
92, 'Precision': 0.9047619047619048, 'Recall': 0.76, 'desc': 'nn_test', 'set': 'test'}
```

Seeing big improvements in all the adjustments:



Adding A Sobel Filter:

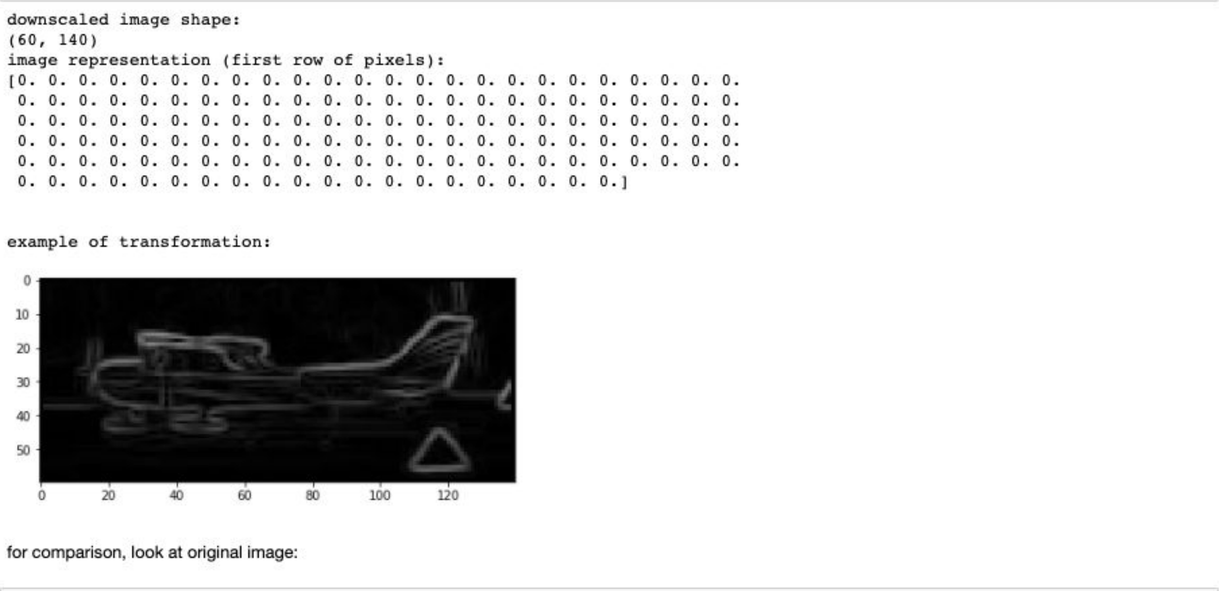
Increasing Dims back to higher resolution and adding Sobel Edge Filter



```
# in downscaling the image, what do you want the new dimensions to be?
# the original dimensions of cropped images: (60, 140), which if 8,400 pixels
dims = (60, 140) # 25% of the original size, 2,100 pixels
```

```
def image_manipulation(imname, imgs_path, imview=False):
    warnings.filterwarnings('ignore')
    imname = imgs_path + imname + '.png'
    img_raw = io.imread(imname, as_gray=True)
    downscaled = transform.resize(img_raw, (dims[0], dims[1])) # downscale image

#     gray_img = color.rgb2gray(img_raw) # remove color
    edges = filters.sobel(downscaled) # Adding Sobel Edge Filter
    final_image = edges


#     canny_image = feature.canny(downscaled) #edge filter image with Canny algorithm
#     hog_image = feature.hog(downscaled) #Extract Histogram of Oriented Gradients (HOG) for a given image.
    if imview==True:
        io.imshow(final_image)
    warnings.filterwarnings('always')
    return final_image

# test the function, look at input/output
test_image = image_manipulation('2017-08-25T23+24+13_390Z', ci_path, True)
print('downscaled image shape: ')
print(test_image.shape)
print('image representation (first row of pixels): ')
print(test_image[0])
print('\n')
print('example of transformation: ')
```

Tried Sobel filter with image shape of 60, 140 / 30,70 / 15,35 Overfitting on 60, 140, but perceptron performed well.

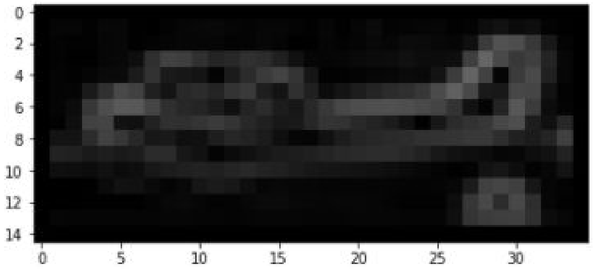


```
downscaled image shape:
(60, 140)
image representation (first row of pixels):
[0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0.
 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0.
 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0.
 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0.
 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0.
 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0.]


example of transformation:
```

for comparison, look at original image:

Low Resolution Sobel Filter at (15, 35)

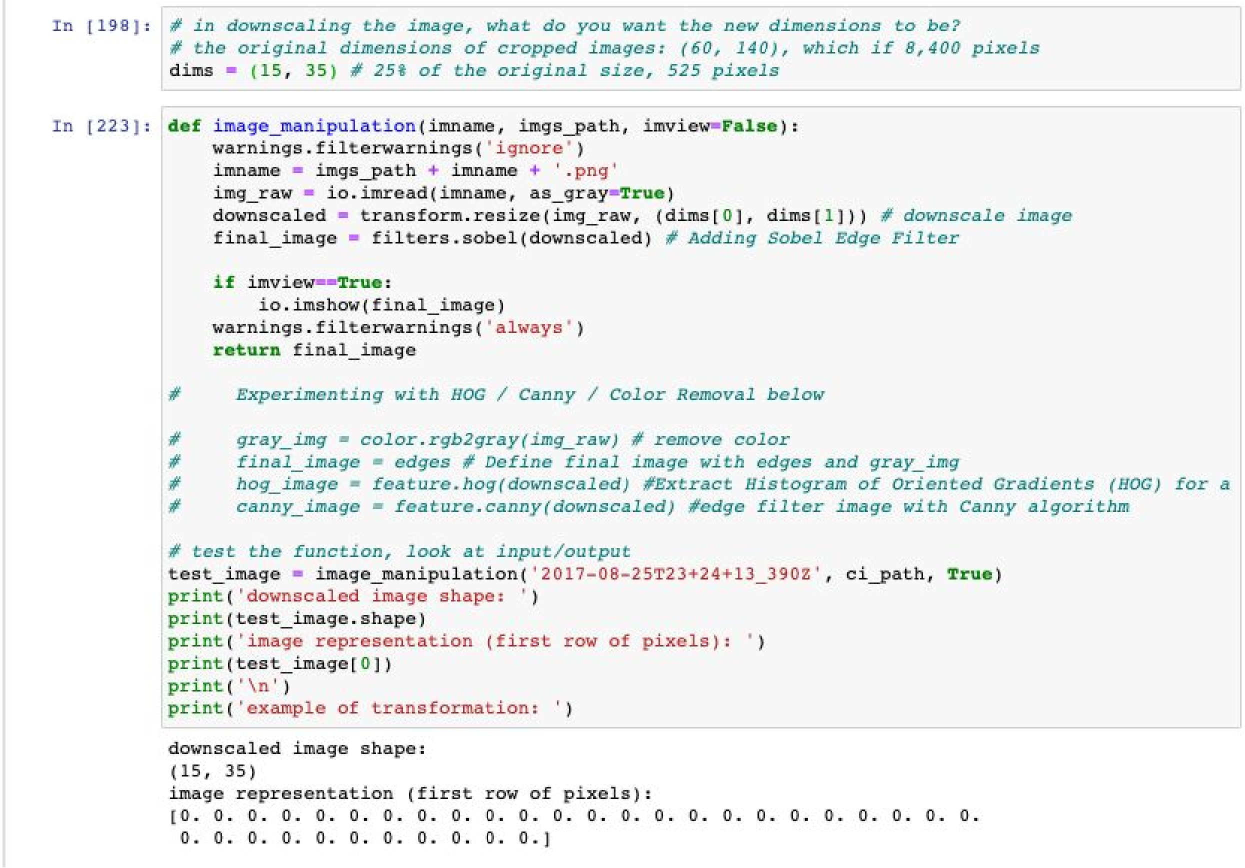
```
example of transformation:
```



for comparison, look at original image:

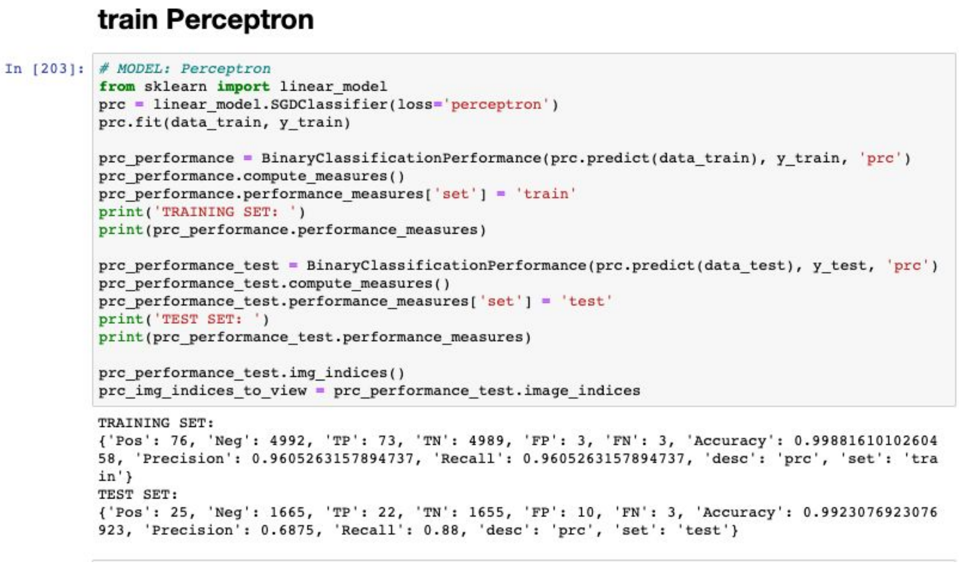
In [200]:
```python
this_imname = ci_path + '2017-08-25T23+24+13_390Z.png'
io.imshow(io.imread(this_imname))
```

Out[200]:
```
<matplotlib.image.AxesImage at 0x1c1812ab50>
```



```
In [198]: # in downscaling the image, what do you want the new dimensions to be?
          # the original dimensions of cropped images: (60, 140), which if 8,400 pixels
          dims = (15, 35) # 25% of the original size, 525 pixels

In [223]: def image_manipulation(imname, imgs_path, imview=False):
              warnings.filterwarnings('ignore')
              imname = imgs_path + imname + '.png'
              img_raw = io.imread(imname, as_gray=True)
              downscaled = transform.resize(img_raw, (dims[0], dims[1])) # downscale image
              final_image = filters.sobel(downscaled) # Adding Sobel Edge Filter

              if imview==True:
                  io.imshow(final_image)
              warnings.filterwarnings('always')
              return final_image

          #     Experimenting with HOG / Canny / Color Removal below

          #     gray_img = color.rgb2gray(img_raw) # remove color
          #     final_image = edges # Define final image with edges and gray_img
          #     hog_image = feature.hog(downscaled) #Extract Histogram of Oriented Gradients (HOG) for a
          #     canny_image = feature.canny(downscaled) #edge filter image with Canny algorithm

          # test the function, look at input/output
          test_image = image_manipulation('2017-08-25T23+24+13_390Z', ci_path, True)
          print('downscaled image shape: ')
          print(test_image.shape)
          print('image representation (first row of pixels): ')
          print(test_image[0])
          print('\n')
          print('example of transformation: ')

          downscaled image shape:
          (15, 35)
          image representation (first row of pixels):
          [0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0.
           0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0.]
```

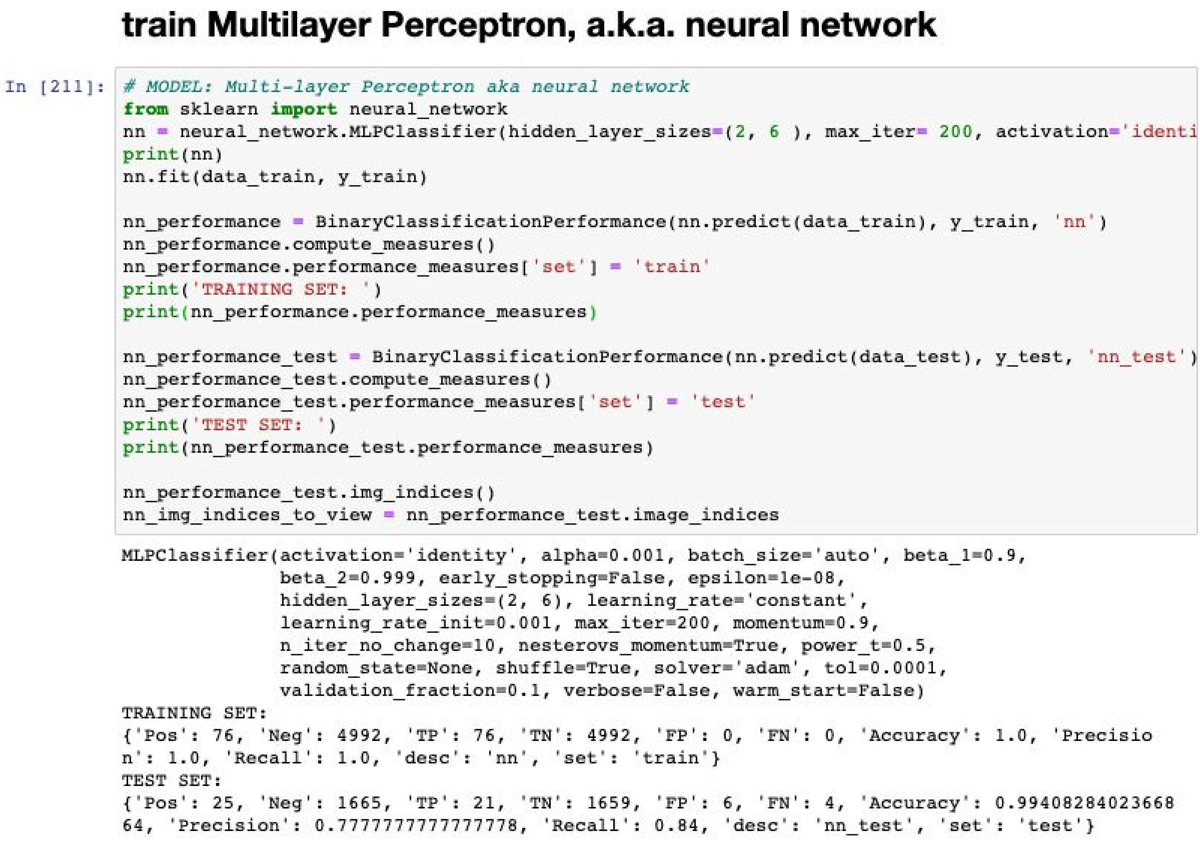
Perceptron:



train Perceptron

```
In [203]: # MODEL: Perceptron
          from sklearn import linear_model
          prc = linear_model.SGDClassifier(loss='perceptron')
          prc.fit(data_train, y_train)

          prc_performance = BinaryClassificationPerformance(prc.predict(data_train), y_train, 'prc')
          prc_performance.compute_measures()
          prc_performance.performance_measures['set'] = 'train'
          print('TRAINING SET: ')
          print(prc_performance.performance_measures)

          prc_performance_test = BinaryClassificationPerformance(prc.predict(data_test), y_test, 'prc')
          prc_performance_test.compute_measures()
          prc_performance_test.performance_measures['set'] = 'test'
          print('TEST SET: ')
          print(prc_performance_test.performance_measures)

          prc_performance_test.img_indices()
          prc_img_indices_to_view = prc_performance_test.image_indices

          TRAINING SET:
          {'Pos': 76, 'Neg': 4992, 'TP': 73, 'TN': 4989, 'FP': 3, 'FN': 3, 'Accuracy': 0.99881610102604
          58, 'Precision': 0.9605263157894737, 'Recall': 0.9605263157894737, 'desc': 'prc', 'set': 'tra
          in'}
          TEST SET:
          {'Pos': 25, 'Neg': 1665, 'TP': 22, 'TN': 1655, 'FP': 10, 'FN': 3, 'Accuracy': 0.9923076923076
          923, 'Precision': 0.6875, 'Recall': 0.88, 'desc': 'prc', 'set': 'test'}
```

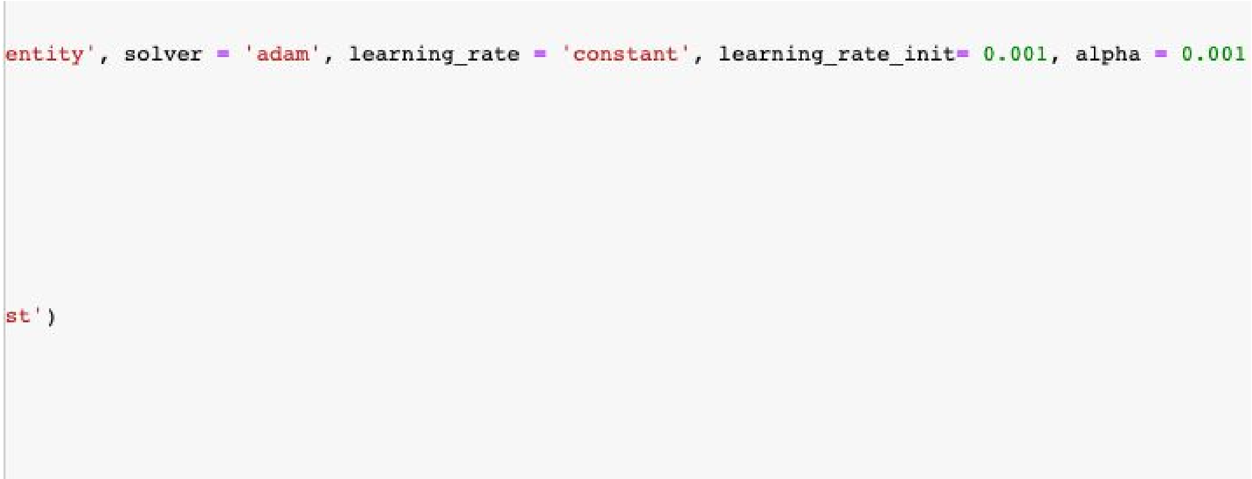
Change to Layer Sizes of NN, change to activation function, and learning rate, and solver.



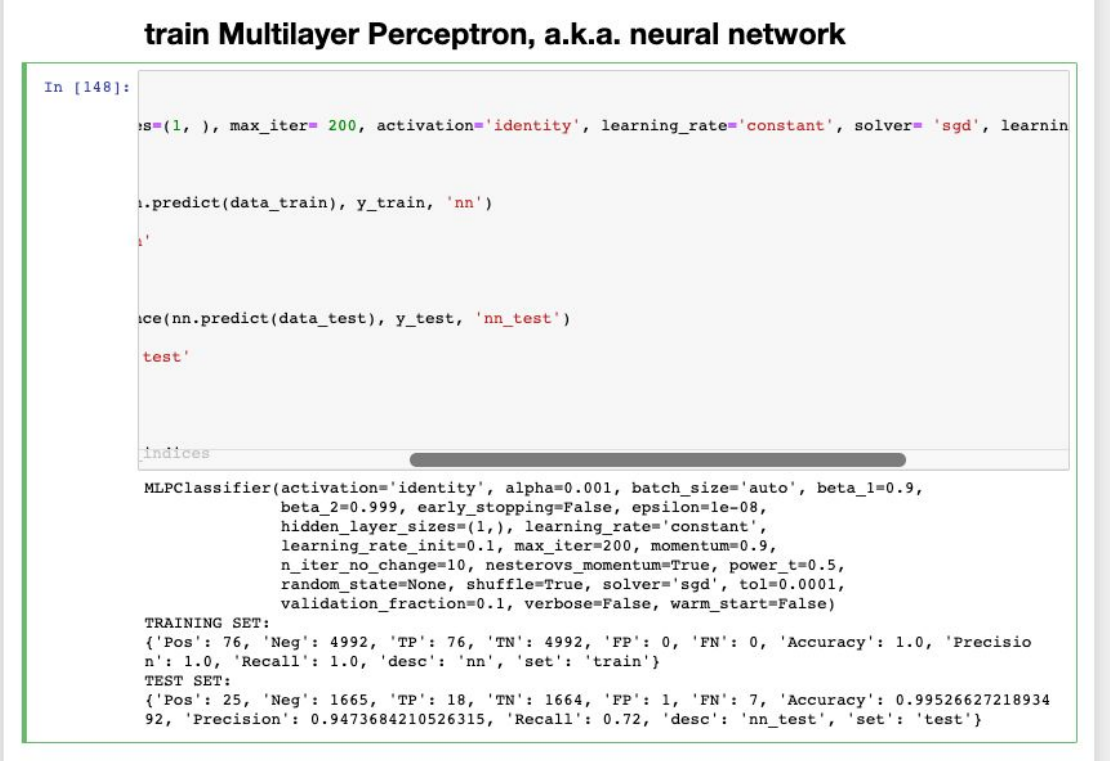
train Multilayer Perceptron, a.k.a. neural network

```
In [211]: # MODEL: Multi-layer Perceptron aka neural network
          from sklearn import neural_network
          nn = neural_network.MLPClassifier(hidden_layer_sizes=(2, 6 ), max_iter= 200, activation='identi
          print(nn)
          nn.fit(data_train, y_train)

          nn_performance = BinaryClassificationPerformance(nn.predict(data_train), y_train, 'nn')
          nn_performance.compute_measures()
          nn_performance.performance_measures['set'] = 'train'
          print('TRAINING SET: ')
          print(nn_performance.performance_measures)

          nn_performance_test = BinaryClassificationPerformance(nn.predict(data_test), y_test, 'nn_test')
          nn_performance_test.compute_measures()
          nn_performance_test.performance_measures['set'] = 'test'
          print('TEST SET: ')
          print(nn_performance_test.performance_measures)

          nn_performance_test.img_indices()
          nn_img_indices_to_view = nn_performance_test.image_indices
```

```
MLPClassifier(activation='identity', alpha=0.001, batch_size='auto', beta_1=0.9,
              beta_2=0.999, early_stopping=False, epsilon=1e-08,
              hidden_layer_sizes=(2, 6), learning_rate='constant',
              learning_rate_init=0.001, max_iter=200, momentum=0.9,
              n_iter_no_change=10, nesterovs_momentum=True, power_t=0.5,
              random_state=None, shuffle=True, solver='adam', tol=0.0001,
              validation_fraction=0.1, verbose=False, warm_start=False)
TRAINING SET:
{'Pos': 76, 'Neg': 4992, 'TP': 76, 'TN': 4992, 'FP': 0, 'FN': 0, 'Accuracy': 1.0, 'Precisio
n': 1.0, 'Recall': 1.0, 'desc': 'nn', 'set': 'train'}
TEST SET:
{'Pos': 25, 'Neg': 1665, 'TP': 21, 'TN': 1659, 'FP': 6, 'FN': 4, 'Accuracy': 0.99408284023668
64, 'Precision': 0.7777777777777778, 'Recall': 0.84, 'desc': 'nn_test', 'set': 'test'}
```



```
entity', solver = 'adam', learning_rate = 'constant', learning_rate_init= 0.001, alpha = 0.001






st')
```

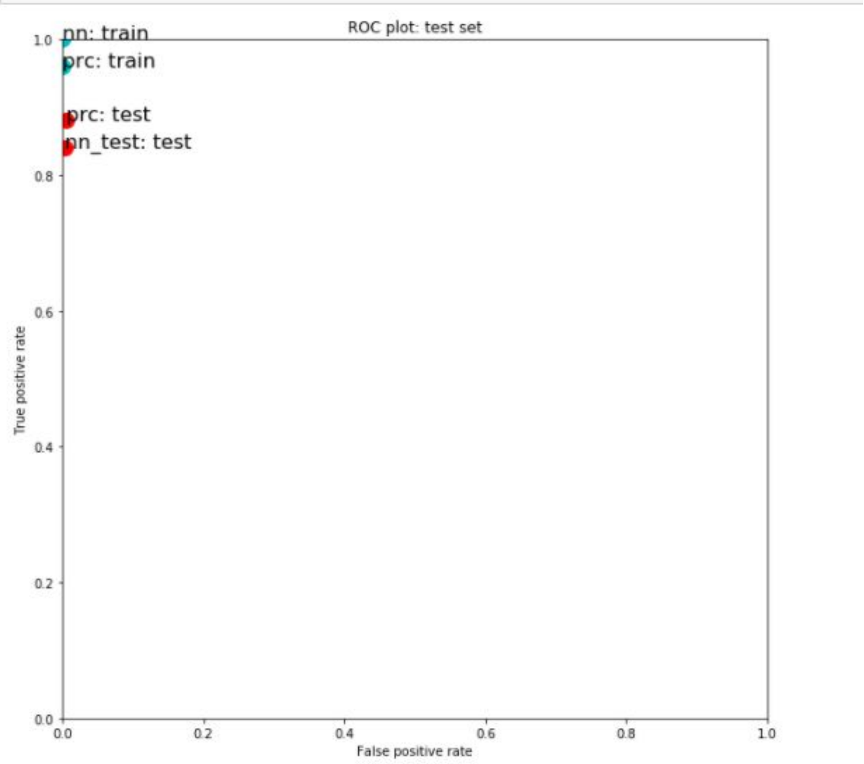


train Multilayer Perceptron, a.k.a. neural network

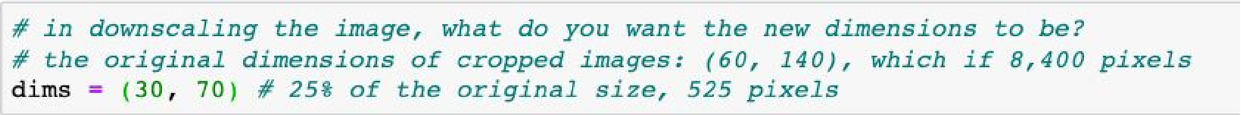
```
In [148]:
         s=(1, ), max_iter= 200, activation='identity', learning_rate='constant', solver= 'sgd', learnin

         .predict(data_train), y_train, 'nn')
         '



         ce(nn.predict(data_test), y_test, 'nn_test')
         test'



         indices
```

```
MLPClassifier(activation='identity', alpha=0.001, batch_size='auto', beta_1=0.9,
              beta_2=0.999, early_stopping=False, epsilon=1e-08,
              hidden_layer_sizes=(1,), learning_rate='constant',
              learning_rate_init=0.1, max_iter=200, momentum=0.9,
              n_iter_no_change=10, nesterovs_momentum=True, power_t=0.5,
              random_state=None, shuffle=True, solver='sgd', tol=0.0001,
              validation_fraction=0.1, verbose=False, warm_start=False)
TRAINING SET:
{'Pos': 76, 'Neg': 4992, 'TP': 76, 'TN': 4992, 'FP': 0, 'FN': 0, 'Accuracy': 1.0, 'Precisio
n': 1.0, 'Recall': 1.0, 'desc': 'nn', 'set': 'train'}
TEST SET:
{'Pos': 25, 'Neg': 1665, 'TP': 18, 'TN': 1664, 'FP': 1, 'FN': 7, 'Accuracy': 0.99526627218934
92, 'Precision': 0.9473684210526315, 'Recall': 0.72, 'desc': 'nn_test', 'set': 'test'}
```

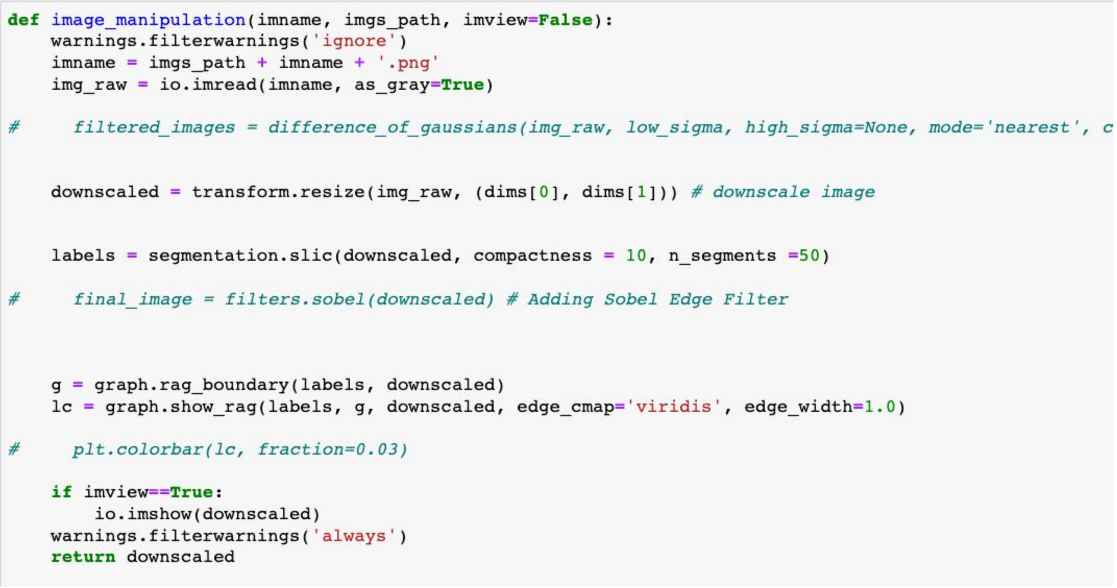
Submission on Training and Testing Sets.
4/6/20



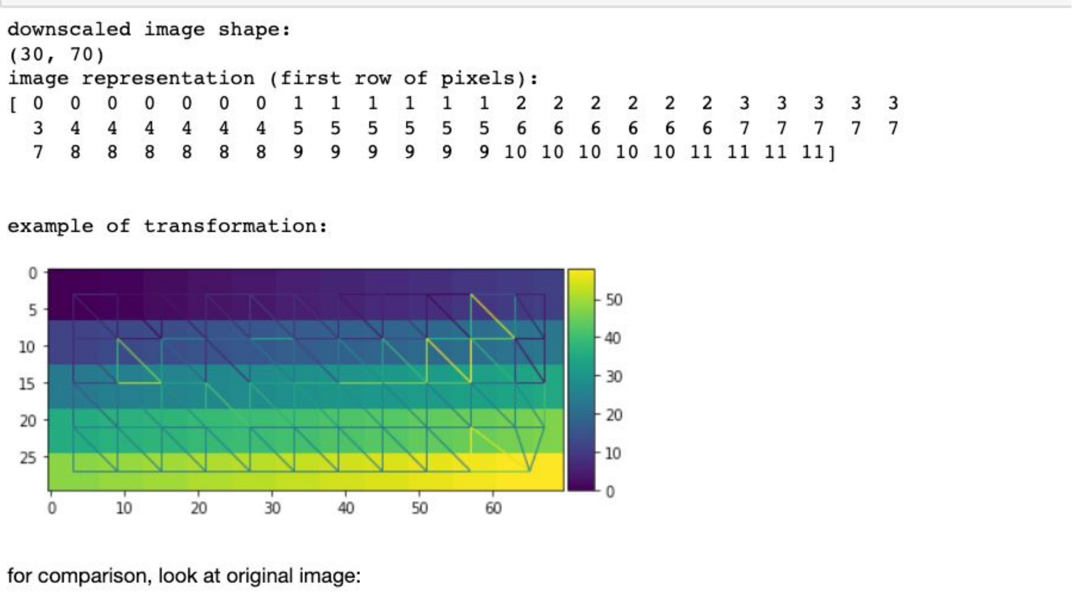
4/27/20
Updated Progress:
Experimenting with RAGs (Region Boundary Based RAGS)
Which are too intensive on the CPU of the computer -- though I have a feeling they would perform very well if run on GPUs.



```
# in downscaling the image, what do you want the new dimensions to be?
# the original dimensions of cropped images: (60, 140), which if 8,400 pixels
dims = (30, 70) # 25% of the original size, 525 pixels
```

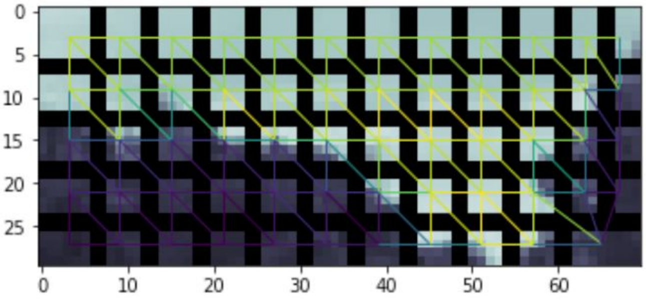


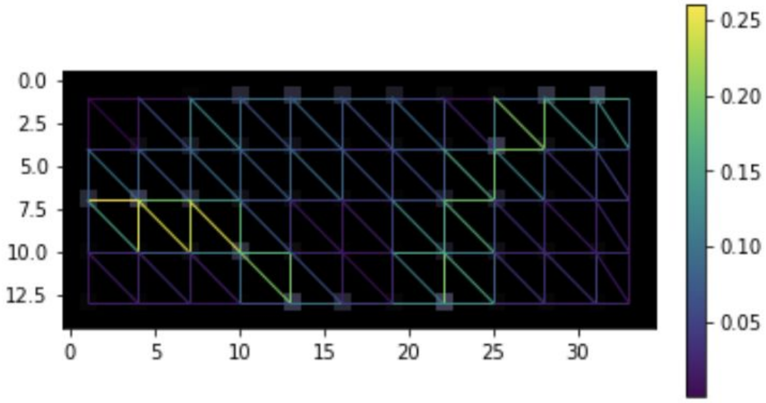
```
def image_manipulation(imname, imgs_path, imview=False):
    warnings.filterwarnings('ignore')
    imname = imgs_path + imname + '.png'
    img_raw = io.imread(imname, as_gray=True)

#     filtered_images = difference_of_gaussians(img_raw, low_sigma, high_sigma=None, mode='nearest', c


    downscaled = transform.resize(img_raw, (dims[0], dims[1])) # downscale image


    labels = segmentation.slic(downscaled, compactness = 10, n_segments =50)

#     final_image = filters.sobel(downscaled) # Adding Sobel Edge Filter



    g = graph.rag_boundary(labels, downscaled)
    lc = graph.show_rag(labels, g, downscaled, edge_cmap='viridis', edge_width=1.0)

#     plt.colorbar(lc, fraction=0.03)

    if imview==True:
        io.imshow(downscaled)
    warnings.filterwarnings('always')
    return downscaled
```

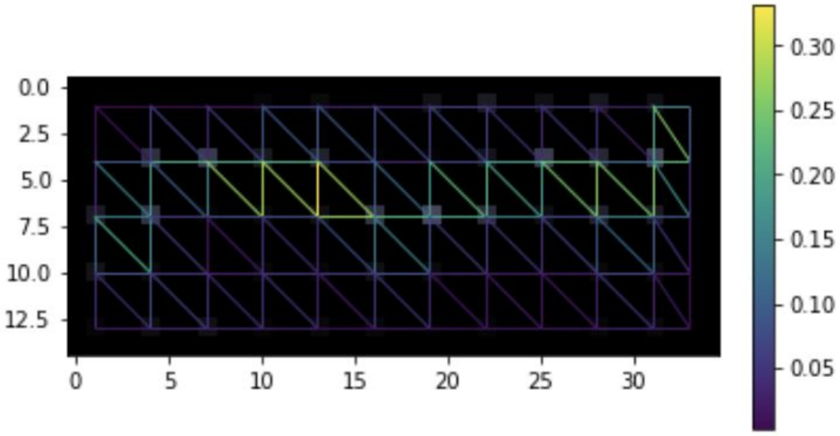


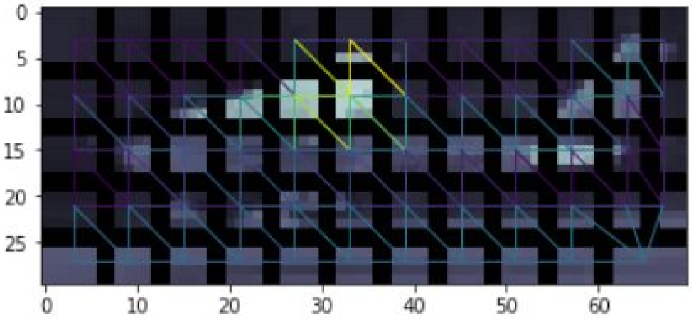
```
downscaled image shape:
(30, 70)
image representation (first row of pixels):
[ 0  0  0  0  0  0  0  1  1  1  1  1  1  2  2  2  2  2  2  3  3  3  3  3
  3  4  4  4  4  4  4  5  5  5  5  5  5  6  6  6  6  6  6  7  7  7  7  7
  7  8  8  8  8  8  8  9  9  9  9  9  9 10 10 10 10 10 10 11 11 11 11]


example of transformation:
```

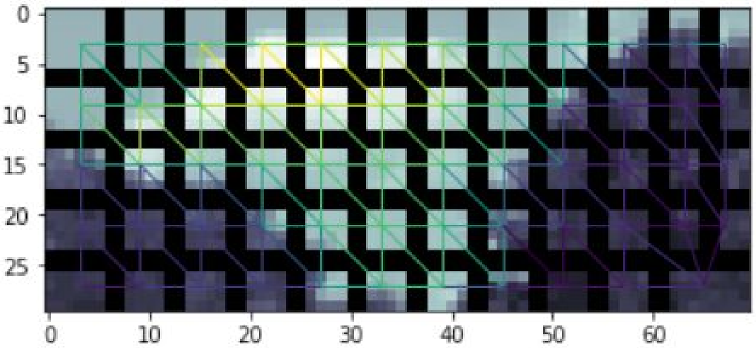
for comparison, look at original image:


0
5
10
15
20
25
0
10
20
30
40
50
60

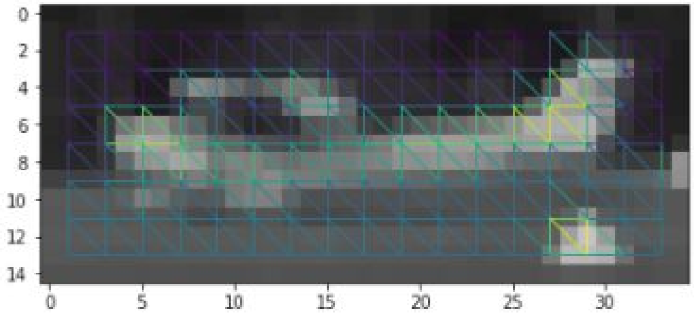

0.0
2.5
5.0
7.5
10.0
12.5
0
5
10
15
20
25
30
0.25
0.20
0.15
0.10
0.05


0.0
2.5
5.0
7.5
10.0
12.5
0
5
10
15
20
25
30
0.30
0.25
0.20
0.15
0.10
0.05



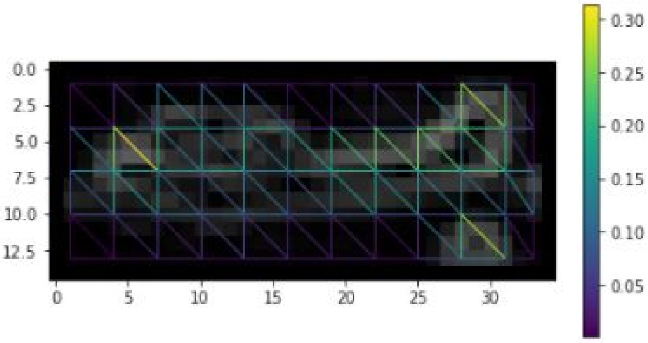


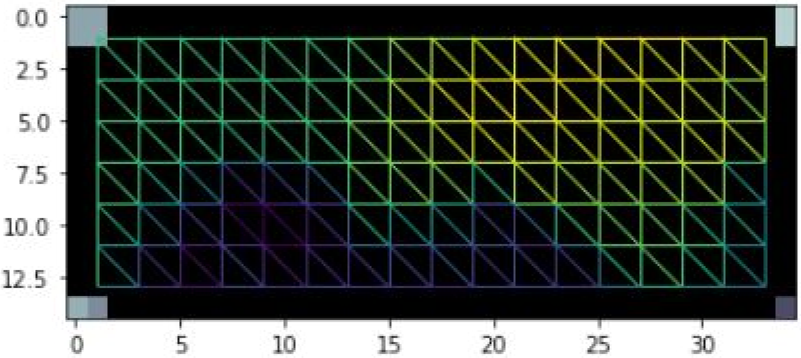
```
downscaled image shape:
(15, 35)
image representation (first row of pixels):
[0.12401573 0.11648205 0.13952649 0.16600221 0.14839046 0.11401215
 0.11715751 0.11325195 0.10247775 0.10283514 0.1154884  0.14123865
 0.14803636 0.1670785  0.1511305  0.15230284 0.15749273 0.18258377
 0.15629166 0.14333641 0.15819405 0.13206261 0.09786691 0.10152979
 0.10315129 0.09150656 0.10016536 0.11894053 0.13478119 0.1492471
 0.1982675  0.12040723 0.13799464 0.14148825 0.13607918]


example of transformation:
```

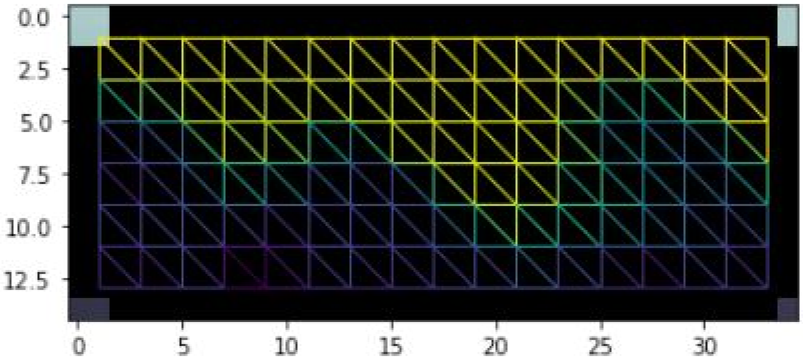


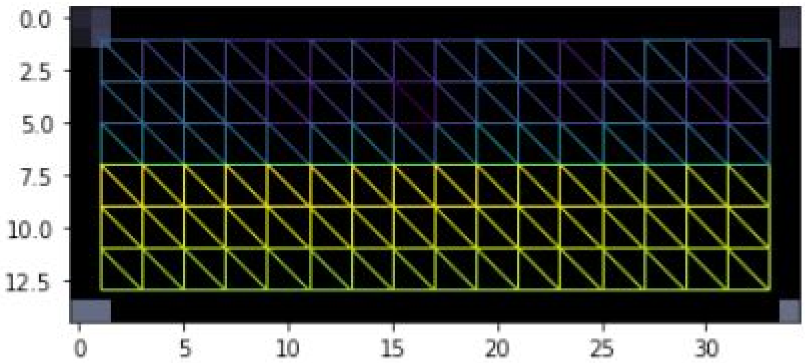
for comparison, look at original image:

```
downscaled image shape:
(15, 35)
image representation (first row of pixels):
[0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0.
 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0.]


example of transformation:
```

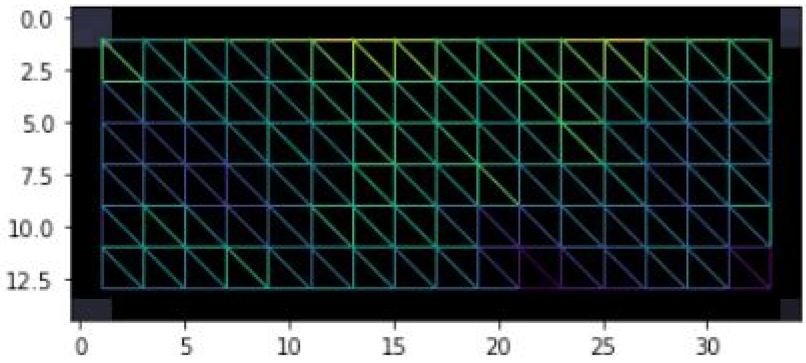


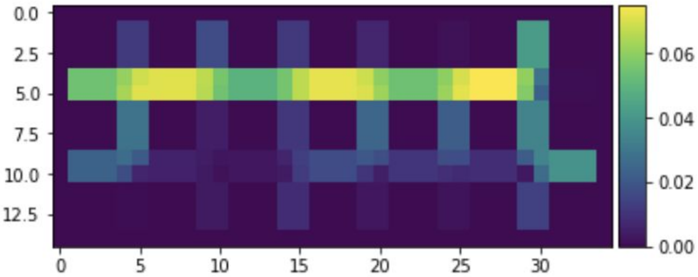


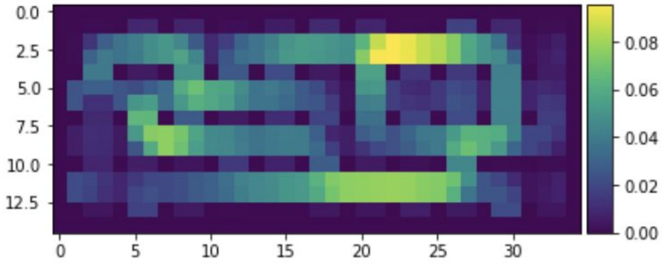




```
(1690,)
SUCCESS!
```

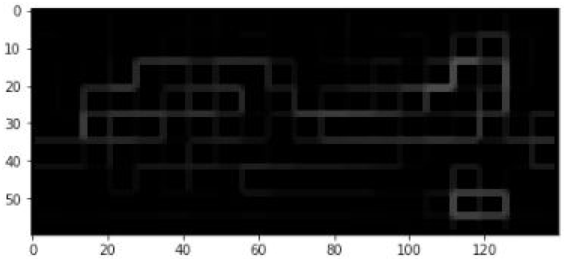






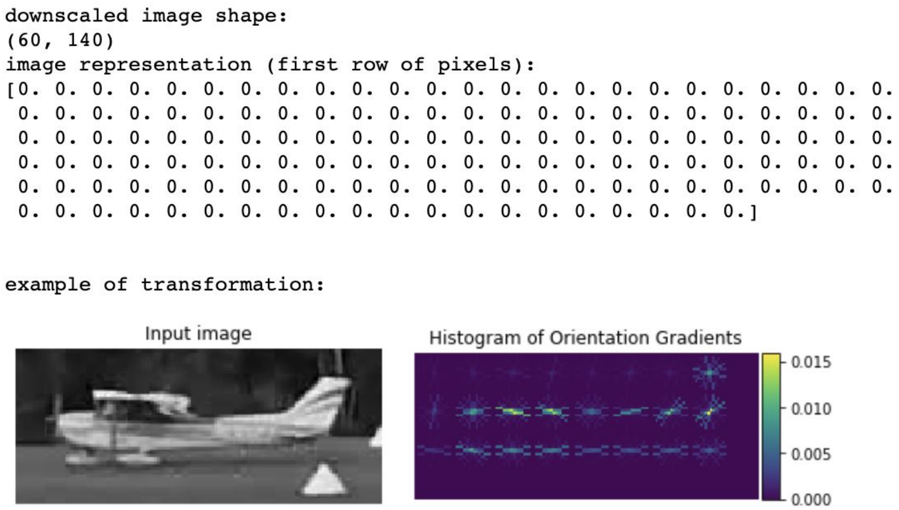
Normalization Cuts in RAGS.

```
example of transformation:
```

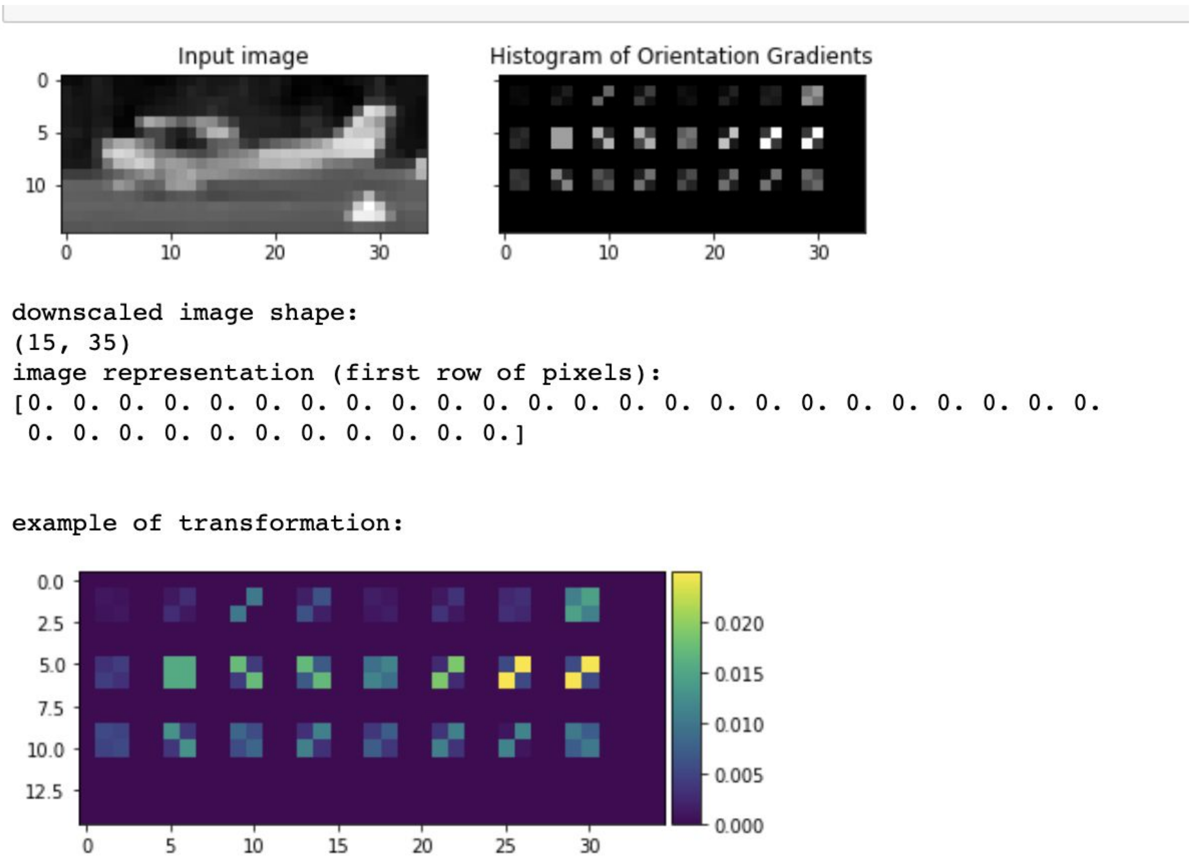


5/2/20 - Final

Histograms of Orientation Gradients:



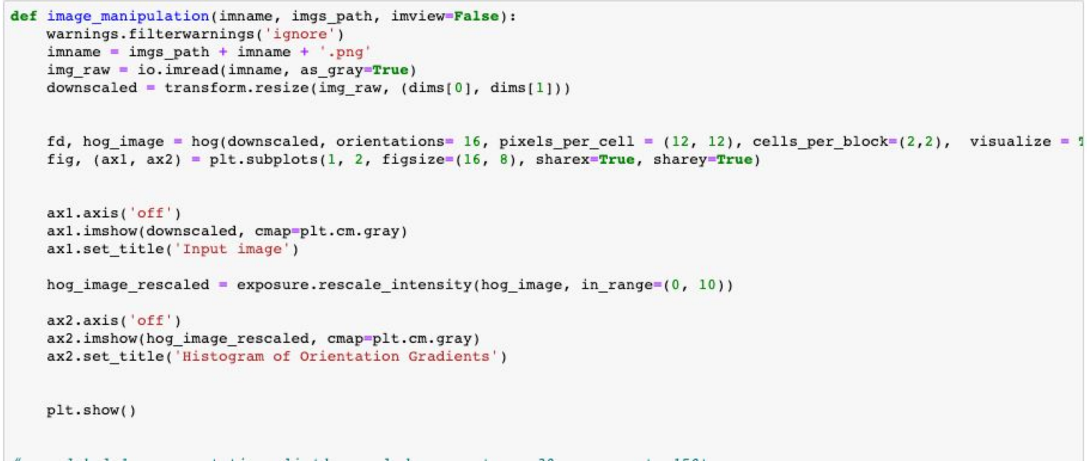
```
downscaled image shape:
(60, 140)
image representation (first row of pixels):
[0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0.
 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0.
 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0.
 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0.
 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0.
 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0.]


example of transformation:
```



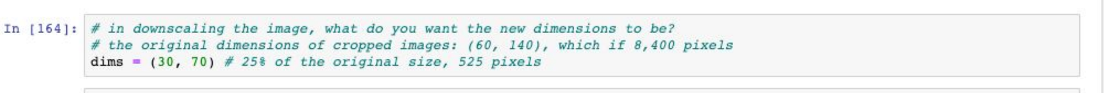
```
downscaled image shape:
(15, 35)
image representation (first row of pixels):
[0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0.
 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0.]


example of transformation:
```

for comparison, look at original image:

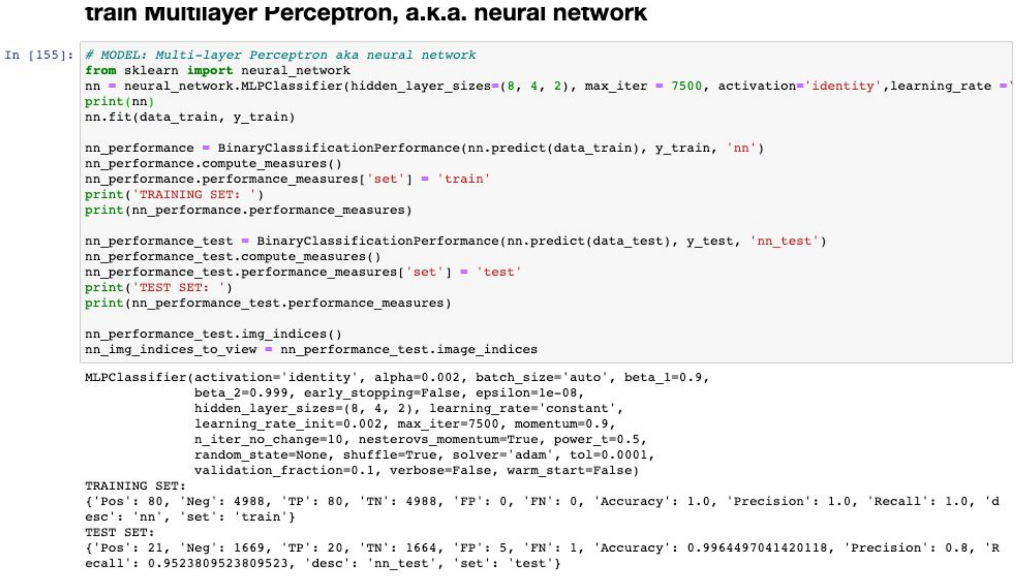
Training with HOGs



```
def image_manipulation(imname, imgs_path, imview=False):
    warnings.filterwarnings('ignore')
    imname = imgs_path + imname + '.png'
    img_raw = io.imread(imname, as_gray=True)
    downscaled = transform.resize(img_raw, (dims[0], dims[1]))


    fd, hog_image = hog(downscaled, orientations= 16, pixels_per_cell = (12, 12), cells_per_block=(2,2),  visualize = T
    fig, (ax1, ax2) = plt.subplots(1, 2, figsize=(16, 8), sharex=True, sharey=True)


    ax1.axis('off')
    ax1.imshow(downscaled, cmap=plt.cm.gray)
    ax1.set_title('Input image')

    hog_image_rescaled = exposure.rescale_intensity(hog_image, in_range=(0, 10))

    ax2.axis('off')
    ax2.imshow(hog_image_rescaled, cmap=plt.cm.gray)
    ax2.set_title('Histogram of Orientation Gradients')


    plt.show()
```



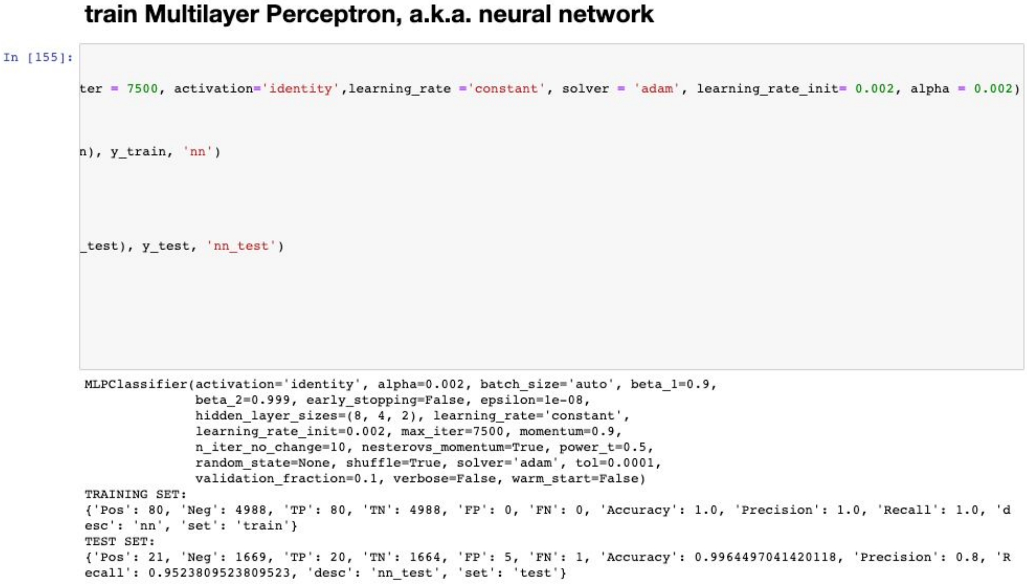
```
In [164]: # in downscaling the image, what do you want the new dimensions to be?
          # the original dimensions of cropped images: (60, 140), which if 8,400 pixels
          dims = (30, 70) # 25% of the original size, 525 pixels
```



train Multilayer Perceptron, a.k.a. neural network

```
In [155]: # MODEL: Multi-layer Perceptron aka neural network
          from sklearn import neural_network
          nn = neural_network.MLPClassifier(hidden_layer_sizes=(8, 4, 2), max_iter = 7500, activation='identity',learning_rate ='
          print(nn)
          nn.fit(data_train, y_train)

          nn_performance = BinaryClassificationPerformance(nn.predict(data_train), y_train, 'nn')
          nn_performance.compute_measures()
          nn_performance.performance_measures['set'] = 'train'
          print('TRAINING SET: ')
          print(nn_performance.performance_measures)

          nn_performance_test = BinaryClassificationPerformance(nn.predict(data_test), y_test, 'nn_test')
          nn_performance_test.compute_measures()
          nn_performance_test.performance_measures['set'] = 'test'
          print('TEST SET: ')
          print(nn_performance_test.performance_measures)

          nn_performance_test.img_indices()
          nn_img_indices_to_view = nn_performance_test.image_indices
```

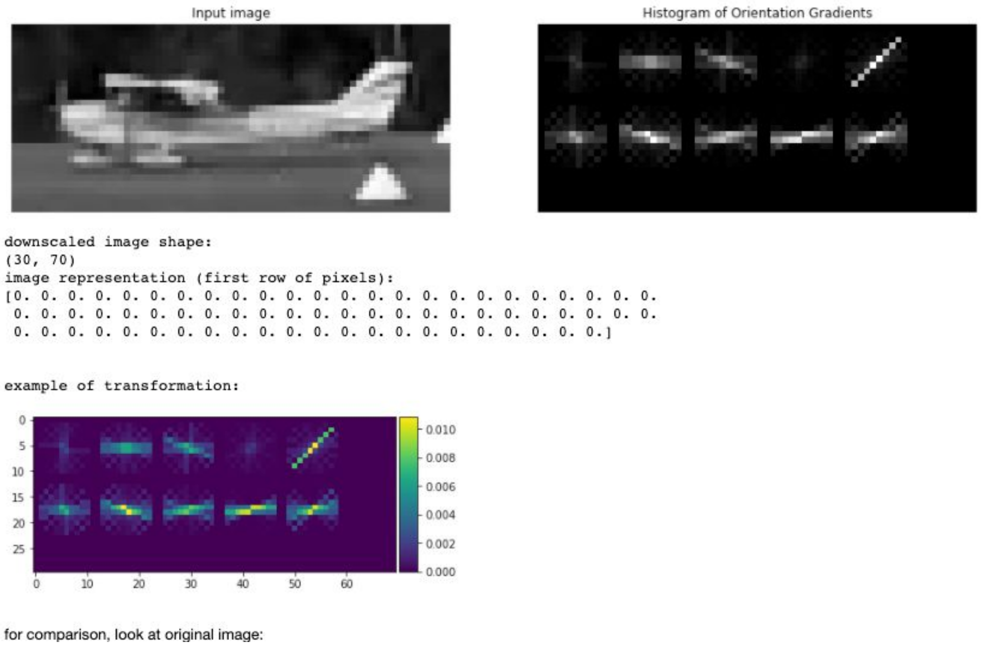
```
MLPClassifier(activation='identity', alpha=0.002, batch_size='auto', beta_1=0.9,
              beta_2=0.999, early_stopping=False, epsilon=1e-08,
              hidden_layer_sizes=(8, 4, 2), learning_rate='constant',
              learning_rate_init=0.002, max_iter=7500, momentum=0.9,
              n_iter_no_change=10, nesterovs_momentum=True, power_t=0.5,
              random_state=None, shuffle=True, solver='adam', tol=0.0001,
              validation_fraction=0.1, verbose=False, warm_start=False)
TRAINING SET:
{'Pos': 80, 'Neg': 4988, 'TP': 80, 'TN': 4988, 'FP': 0, 'FN': 0, 'Accuracy': 1.0, 'Precision': 1.0, 'Recall': 1.0, 'd
esc': 'nn', 'set': 'train'}
TEST SET:
{'Pos': 21, 'Neg': 1669, 'TP': 20, 'TN': 1664, 'FP': 5, 'FN': 1, 'Accuracy': 0.9964497041420118, 'Precision': 0.8, 'R
ecall': 0.9523809523809523, 'desc': 'nn_test', 'set': 'test'}
```



train Multilayer Perceptron, a.k.a. neural network

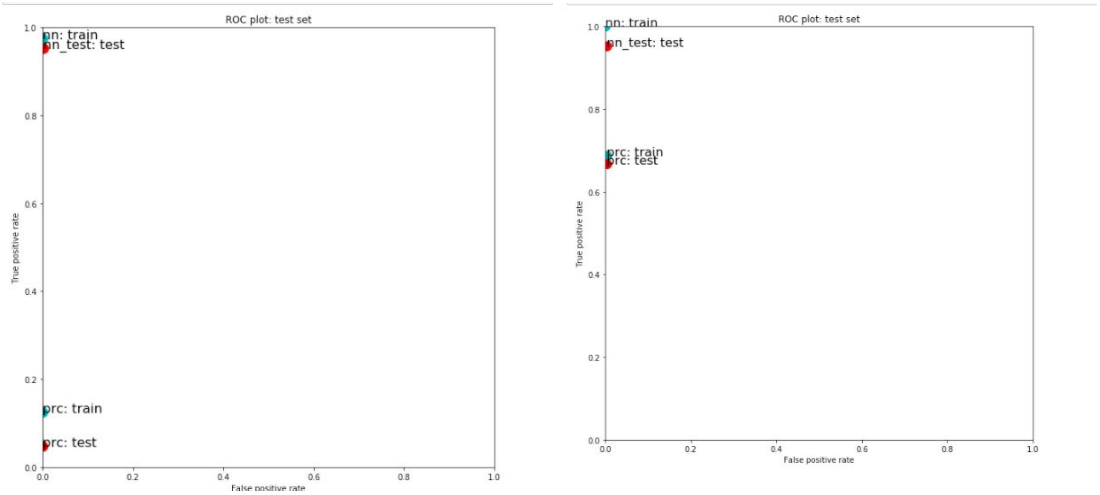
```
In [155]:

ter = 7500, activation='identity',learning_rate ='constant', solver = 'adam', learning_rate_init= 0.002, alpha = 0.002)


n), y_train, 'nn')




_test), y_test, 'nn_test')
```

```
MLPClassifier(activation='identity', alpha=0.002, batch_size='auto', beta_1=0.9,
              beta_2=0.999, early_stopping=False, epsilon=1e-08,
              hidden_layer_sizes=(8, 4, 2), learning_rate='constant',
              learning_rate_init=0.002, max_iter=7500, momentum=0.9,
              n_iter_no_change=10, nesterovs_momentum=True, power_t=0.5,
              random_state=None, shuffle=True, solver='adam', tol=0.0001,
              validation_fraction=0.1, verbose=False, warm_start=False)
TRAINING SET:
{'Pos': 80, 'Neg': 4988, 'TP': 80, 'TN': 4988, 'FP': 0, 'FN': 0, 'Accuracy': 1.0, 'Precision': 1.0, 'Recall': 1.0, 'd
esc': 'nn', 'set': 'train'}
TEST SET:
{'Pos': 21, 'Neg': 1669, 'TP': 20, 'TN': 1664, 'FP': 5, 'FN': 1, 'Accuracy': 0.9964497041420118, 'Precision': 0.8, 'R
ecall': 0.9523809523809523, 'desc': 'nn_test', 'set': 'test'}
```



```
downscaled image shape:
(30, 70)
image representation (first row of pixels):
[0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0.
 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0.
 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0.]

example of transformation:
```

for comparison, look at original image:



Final NN performance on training and Test set with Histogram of Oriented Gradients ^