Lab 06: Image Classification

In [4]:
```
#Importing Python Modules
import matplotlib.pyplot as plt
import matplotlib.image as mpimg
import numpy as np
from PIL import Image
from sklearn import neighbors
```

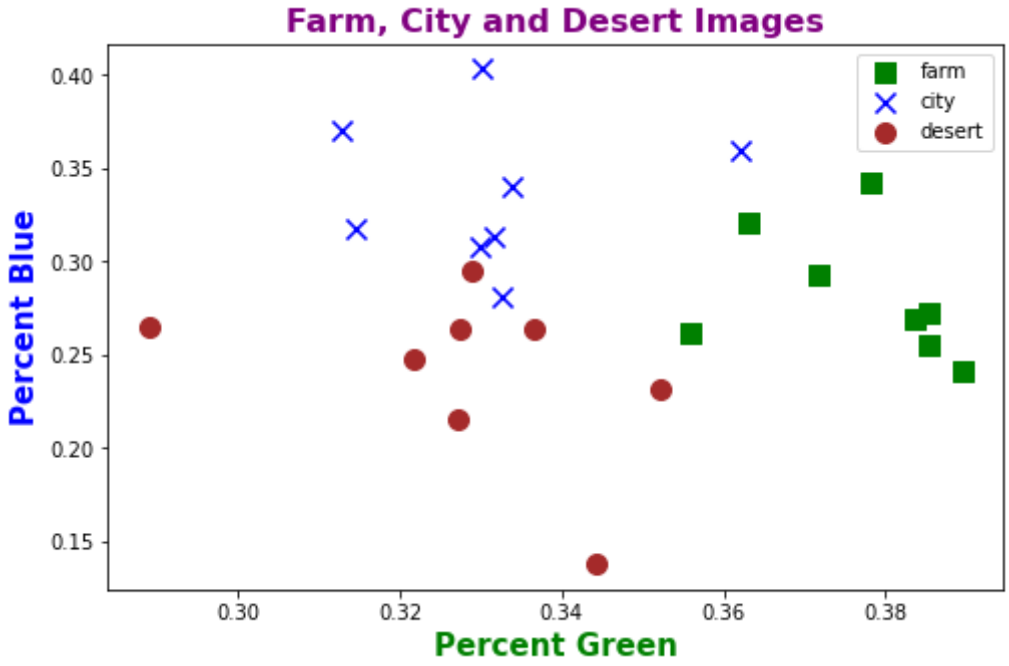
1. Create a list with the names called image_files The following list has been made for you to simplify this. Note, in practice you would likely store these files in separate folders and simply read all the files in each folder so that you could add/remove files without editing source code. ['farm1.jpg', 'farm2.jpg', 'farm3.jpg', 'farm4.jpg', 'farm5.jpg', 'farm6.jpg', 'farm7.jpg', 'farm8.jpg', 'city1.jpg', 'city2.jpg', 'city3.jpg', 'city4.jpg', 'city5.jpg', 'city6.jpg', 'city7.jpg', 'city8.jpg', 'desert1.jpg', 'desert2.jpg', 'desert3.jpg', 'desert4.jpg', 'desert5.jpg', 'desert6.jpg', 'desert7.jpg', 'desert8.jpg']

In [5]:
```
images_list = ['farm1.jpg', 'farm2.jpg', 'farm3.jpg', 'farm4.jpg',
 'farm5.jpg', 'farm6.jpg', 'farm7.jpg', 'farm8.jpg',
 'city1.jpg', 'city2.jpg', 'city3.jpg', 'city4.jpg',
 'city5.jpg', 'city6.jpg', 'city7.jpg', 'city8.jpg',
 'desert1.jpg', 'desert2.jpg', 'desert3.jpg', 'desert4.jpg',
 'desert5.jpg', 'desert6.jpg', 'desert7.jpg', 'desert8.jpg']
```

1. Create the scatter plot in the first page Create a scatter plot using the farm/city/desert image data set where each point represents an image, the x axis is the percent green in the image, and the y axis represents the percent blue. The plot should be titled with x and y axis labels, a different color for each of the 3 image classes, and a legend showing which class corresponds to which color. Note: this is with 24 images total, so you will likely want to find some way to automate the process of getting this data for the scatterplot. You can use a for loop to do this. Get the image file name by reading it from the image_files list from step 1. Then perform image processing to get the percent of blue and the percent of green.

In [33]:
```
farm_green = []
desert_green = []
city_green = []
farm_blue = []
desert_blue = []
city_blue = []

for img in images_list[:8]:
    image = mpimg.imread(img)
    tuple_rgb = np.array(image).mean(axis=(0,1))
    avg_red = tuple_rgb[0]
    avg_green = tuple_rgb[1]
    avg_blue = tuple_rgb[2]
```

```
    percent_green = avg_green/(avg_green+avg_blue+avg_red)
    percent_blue = avg_blue/(avg_green+avg_blue+avg_red)
    farm_green.append(percent_green)
    farm_blue.append(percent_blue)

for img in images_list[8:16]:
    image = mpimg.imread(img)
    tuple_rgb = np.array(image).mean(axis=(0,1))
    avg_red = tuple_rgb[0]
    avg_green = tuple_rgb[1]
    avg_blue = tuple_rgb[2]
    percent_green = avg_green/(avg_green+avg_blue+avg_red)
    percent_blue = avg_blue/(avg_green+avg_blue+avg_red)
    city_green.append(percent_green)
    city_blue.append(percent_blue)

for img in images_list[16:]:
    image = mpimg.imread(img)
    tuple_rgb = np.array(image).mean(axis=(0,1))
    avg_red = tuple_rgb[0]
    avg_green = tuple_rgb[1]
    avg_blue = tuple_rgb[2]
    percent_green = avg_green/(avg_green+avg_blue+avg_red)
    percent_blue = avg_blue/(avg_green+avg_blue+avg_red)
    desert_green.append(percent_green)
    desert_blue.append(percent_blue)

plt.figure(figsize=(8, 5))
plt.scatter(farm_green, farm_blue, s=100, marker='s', c='green')
plt.scatter(city_green, city_blue, s=100, marker='x', c='blue')
plt.scatter(desert_green, desert_blue, s=100, marker='o', c='brown')
plt.title('Farm, City and Desert Images', fontsize=16, fontweight='bold', color
plt.xlabel('Percent Green', fontsize=15, fontweight='bold', color='green')
plt.ylabel('Percent Blue', fontsize=15, fontweight='bold', color='blue')
plt.legend(['farm', 'city', 'desert'])
plt.show()
```



1. Now create an array of strings called training_target with the category of each. You can use this for convenience: ['farm', 'farm', 'farm', 'farm', 'farm', 'farm', 'farm', 'farm', 'city', 'city', 'city', 'city', 'city', 'city', 'city', 'city', 'desert', 'desert', 'desert', 'desert', 'desert', 'desert', 'desert', 'desert']

In [34]:
```
training_target = ['farm', 'farm', 'farm', 'farm',
 'farm', 'farm', 'farm', 'farm',
 'city', 'city', 'city', 'city',
 'city', 'city', 'city', 'city',
 'desert', 'desert', 'desert', 'desert',
 'desert', 'desert', 'desert', 'desert']
```

1. Create an empty array of zeros called training_data that will eventually store the percent green and percent blue values. You will be filling this soon. Given the needs of your data set (24 samples and 2 columns), it should have 24 rows and 2 columns.

In [35]:
```
training_data = np.zeros((24, 2))
```

1. Now fill the training_data array with the proper values for each image, and observe the values in the array after it is finished. You already have the percent of blue and the percent of green for all images from step 2. Make sure to put those two values in the proper place in the training_data array.

In [36]:
```
for i in range(8):
    training_data[i, 0] = farm_green[i]
    training_data[8+i, 0] = city_green[i]
    training_data[16+i, 0] = desert_green[i]
    training_data[i, 1] = farm_blue[i]
    training_data[8+i, 1] = city_blue[i]
    training_data[16+i, 1] = desert_blue[i]

training_data
```

Out[36]:
```
array([[0.38539325, 0.27247922],
       [0.38948026, 0.2416669 ],
       [0.37176578, 0.29236818],
       [0.38534953, 0.25567269],
       [0.38368975, 0.26976196],
       [0.37825091, 0.34249364],
       [0.35579943, 0.26134838],
       [0.36318337, 0.32079215],
       [0.33384693, 0.33987   ],
       [0.3145895 , 0.31742334],
       [0.32984313, 0.30757732],
       [0.33023058, 0.40331207],
       [0.31276747, 0.37058074],
       [0.36198701, 0.35924772],
       [0.33263948, 0.28122407],
       [0.33155654, 0.31387491],
       [0.28897975, 0.26477986],
       [0.32887478, 0.29461283],
       [0.32175664, 0.24744815],
       [0.35209261, 0.23171261],
       [0.32718518, 0.21564909],
       [0.33654075, 0.26392502],
       [0.34415887, 0.13753433],
       [0.32732039, 0.26438328]])
```

1. Create your classifier. This can often be done in one line. In this case, we suggest using the k-Nearest Neighbors classifier as shown in the tutorial (use k=1), but you can try others if you are interested.

In [37]:
```
classfier = neighbors.KNeighborsClassifier(n_neighbors=1, weights='distance')
```

1. Train your classifier. Again, this is often only one line of code where you provide the training data and the training target to the classifier you just created to the classifier's 'fit' or 'training' function. For such a small data set, this will be fast, but for larger data sets sometimes this is time consuming. Now you have a trained classifier... great! Now we'll set up the application for it.

In [38]:
```
classfier.fit(training_data, training_target)
neighbors.KNeighborsClassifier(algorithm='auto', leaf_size=30, metric='minkowsk
                               n_neighbors=1,
                               n_jobs=None, metric_params=None)
```

Out[38]:
```
KNeighborsClassifier
KNeighborsClassifier(n_neighbors=1, weights='distance')
```

1. Now create an empty test_data array and fill it with the proper values for each test image, and observe the filled array and consider if it matches your expectations based on your observations of the images. test_data should start with zeros and be 3 rows (for three test images) and 2 columns (for % green then % blue for each image). Loop through the three test images to fill in the values of the array.

In [39]:
```
test = ['test1.jpg', 'test2.jpg', 'test3.jpg']
test_green = []
test_blue = []

for image in test:
    img = mpimg.imread(image)
    tuple_rgb = np.array(img).mean(axis=(0,1))
    avg_red = tuple_rgb[0]
    avg_green = tuple_rgb[1]
    avg_blue = tuple_rgb[2]
    percent_green = avg_green/(avg_green+avg_blue+avg_red)
    percent_blue = avg_blue/(avg_green+avg_blue+avg_red)
    test_green.append(percent_green)
    test_blue.append(percent_blue)

test_data = np.zeros((3, 2))

for i in range(3):
    test_data[i, 0] = test_green[i]
    test_data[i, 1] = test_blue[i]

test_data
```

Out[39]:
```
array([[0.3269517 , 0.32689922],
       [0.33427965, 0.17938778],
       [0.35005005, 0.24577556]])
```

1. Predict the class of the test images. Now predict the classes given the test_data array. This should only take one line of code if the test_data array is prepared.

In [40]:
```
classfier_prediction = classfier.predict(test_data)
classfier_prediction
```

Out[40]:
```
array(['city', 'desert', 'desert'], dtype='<U6')
```

1. Print the prediction from the test images and compare with the actual images shown below. Make this comparison clear in the output of your code (e.g. prepend with 'predicted:' and 'actual:').

In [41]:
```
print('Predicted images are:', classfier_prediction)
print('Actual images are:', ['farm', 'city', 'desert'])
```

```
Predicted images are: ['city' 'desert' 'desert']
Actual images are: ['farm', 'city', 'desert']
```

In [ ]: