Cogs 118B - Unsupervised Machine Learning - FA22

Final Project - "Da Cogs Dawgs"

Raunit Kohli, Harini Adivikolanu, Moksha Poladi, Yash Potdar

This project will explore and classify images of dogs and spiders. We first clean our dataset and run a KNN algorithm with the ideal number of hyper-parameters using all dimensions. We then attempt to do PCA on our dataset of dogs and spiders to reduce the dimensionality of our data and make our classifier more accurate and computationally efficient. Finally, we discuss the limitations and pitfalls of the dataset we chose and the ideal cases in which PCA should be used.

In [1]:
```
import numpy as np
import pandas as pd
import matplotlib.pyplot as plt
from tqdm import tqdm
import fnmatch
import os
import seaborn as sns
import matplotlib.pyplot as plt
from matplotlib.pyplot import figure
import math
from PIL import Image
from sklearn.preprocessing import normalize

import torch
from sklearn.model_selection import train_test_split
```

!unzip cogs118b_data.zip

EDA

This section starts with an exploration of our data.

Find the number of animal images per category

In [3]:
```
os.listdir('raw-img/')
```

Out[3]:
```
['gallina',
 'ragno',
 'gatto',
 'farfalla',
 'mucca',
 'cavallo',
 'cane',
 'pecora',
 'scoiattolo',
 'elefante']
```

In [4]:
```
import fnmatch
species_counts_latin = {}
for subdir in os.listdir('raw-img/'):
    dir_path = f"raw-img/{subdir}"
```

```
    count = len(fnmatch.filter(os.listdir(dir_path), '*.*'))
    species_counts_latin[subdir] = count
```

In [5]:
```
species_counts_latin
```

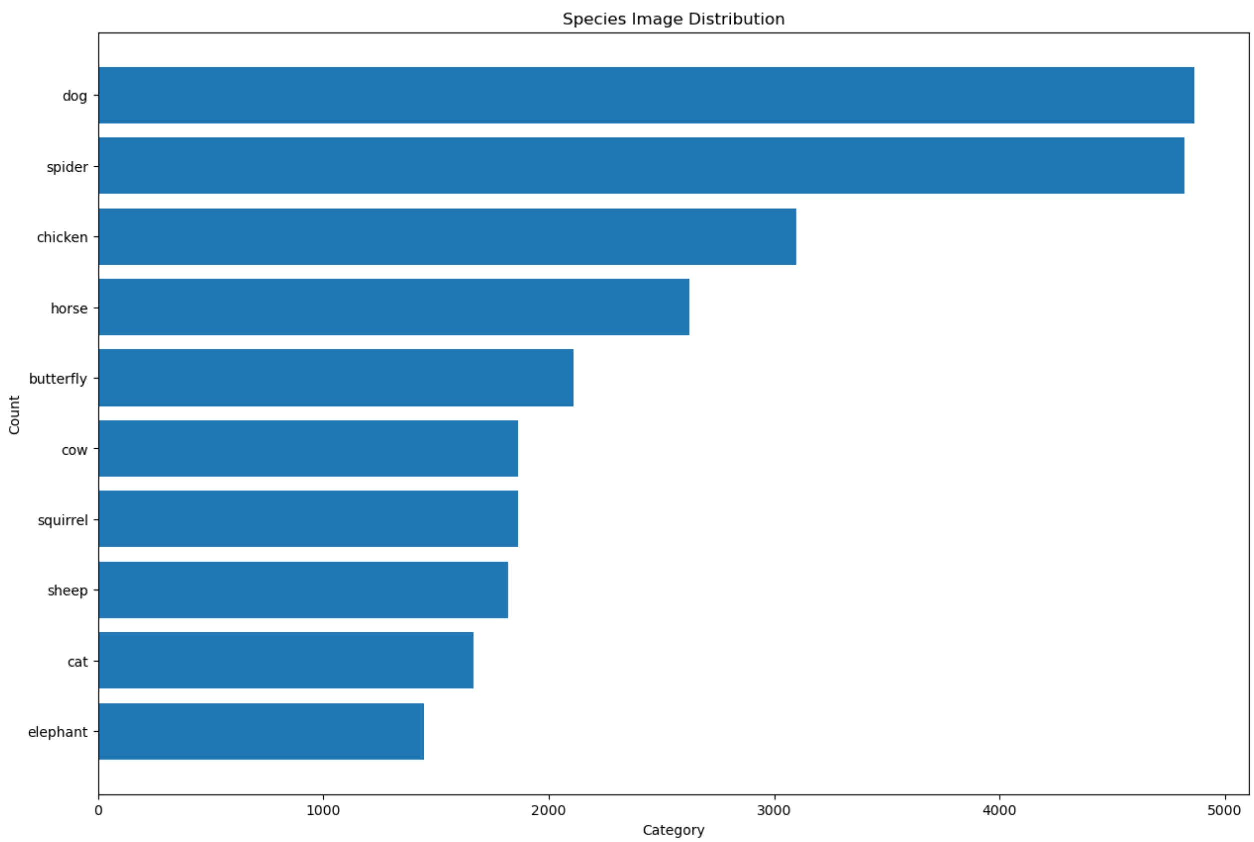
Out[5]:
```
{'gallina': 3098,
 'ragno': 4821,
 'gatto': 1668,
 'farfalla': 2112,
 'mucca': 1866,
 'cavallo': 2623,
 'cane': 4863,
 'pecora': 1820,
 'scoiattolo': 1862,
 'elefante': 1446}
```

In [6]:
```
translations = {"cane": "dog", "cavallo": "horse", "elefante": "elephant", "farfalla":
                "gallina": "chicken", "gatto": "cat", "mucca": "cow", "pecora": "sheep",
                "scoiattolo": "squirrel", "ragno": "spider"}
```

In [7]:
```
species_counts = {}
for species in species_counts_latin:
    english_species = translations[species]
    species_counts[english_species] = species_counts_latin[species]
```

In [8]:
```
species_counts = pd.Series(species_counts).sort_values()
```

In [9]:
```
figure(figsize=(15, 10))
plt.barh(species_counts.index, species_counts.values)
plt.title('Species Image Distribution')
plt.xlabel('Category')
plt.ylabel('Count');
```



From the above visualization, we can see that the dataset is imbalanced. Out of the 26179 images present, around 5000 images are for dogs and spiders, while around 1500 are for elephants and cats. Some species are overrepresented, which can have an impact when trying to make a classification or reconstructing an image of an animal using PCA.

In [10]:
```
num_images = np.sum(species_counts)
num_images
```

Out[10]:
```
26179
```

In [11]:
```
plt.imread('raw-img/cane/OIP-x4zOVsLV23fNe1uSD80MbAHaE8.jpeg').shape
```

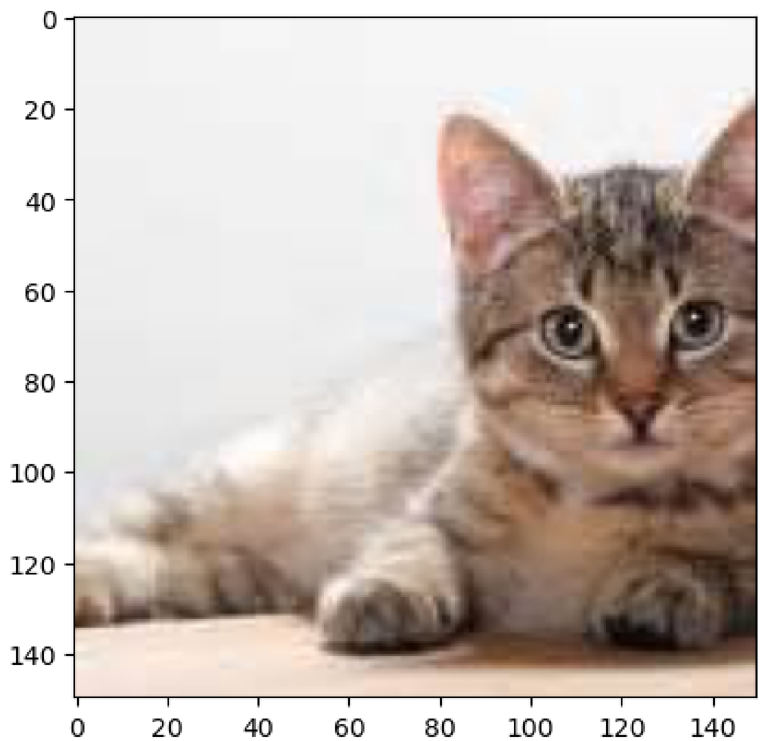
Out[11]:
```
(200, 300, 3)
```

Next, we will display a few images from our dataset. We can see that the dimensions of the images are not standard, so normalizing our images to a standard dimension will be one of our first data wrangling steps.

In [13]:
```
plt.imshow(plt.imread('raw-img/cane/OIP-x4zOVsLV23fNe1uSD80MbAHaE8.jpeg'));
```

In [15]:
```
plt.imshow(plt.imread('raw-img/cavallo/OIP-_9BjMsv3D2hGKOK4c_cvtgHaE9.jpeg'));
```

In [16]:
```
plt.imshow(plt.imread('raw-img/gatto/100.jpeg'));
```



In [17]:
```
plt.imread('raw-img/cavallo/OIP-_9BjMsv3D2hGKOK4c_cvtgHaE9.jpeg').shape
```

Out[17]:
```
(201, 300, 3)
```

In [18]:
```
plt.imread('raw-img/cavallo/OIP-_82Gy46U8XAWuoXiyps_iwHaE-.jpeg').shape
```

Out[18]:
```
(202, 300, 3)
```

In [19]:
```
img_arr = plt.imread('raw-img/cane/OIP-x4zOVsLV23fNe1uSD80MbAHaE8.jpeg')
width = img_arr.shape[1]
height = img_arr.shape[0]
```

In [20]:
```
width, height
```
Out[20]:
```
(300, 200)
```

Rescaling the image to 128x128

- Source: https://imagekit.io/blog/image-resizing-in-python/

In [21]:
```
scale = 32.0 / max(width, height)
scaledWidth = round(width * scale, 0)
scaledHeight = round(height * scale, 0)

image = Image.open('raw-img/cane/OIP-x4zOVsLV23fNe1uSD80MbAHaE8.jpeg')
res = image.resize((128, 128))

plt.imshow(np.array(res))
```
Out[21]:
```
<matplotlib.image.AxesImage at 0x7fade17fc310>
```

Data Wrangling Steps

1. Convert every image to numpy array.
2. Resize every image to 128x128.
3. Construct a numpy array for EACH species (will be used for the mean face).
4. Construct a numpy array with ALL of the resized images (will be used for training).

data = np.empty((0,128*128*3), int) image = Image.open('raw-img/cane/OIP-x4zOVsLV23fNe1uSD80MbAHaE8.jpeg') rgb_im = image.convert('RGB') res = rgb_im.resize((128, 128)) flattened = np.array(res).flatten() data = np.append(data, flattened.reshape(1,len(flattened)), axis = 0) image.save('single_image/original.jpg') res.save('single_image/rescaled.jpg')

np.array(Image.open('single_image/original.jpg')).shape

np.array(Image.open('single_image/rescaled.jpg')).shape

np.savetxt('single_image/indl_data.out', data)

```
!rm -rf rescaled_images/
!mkdir rescaled_images
!mkdir rescaled_images/dog
!mkdir rescaled_images/cat
!mkdir rescaled_images/elephant
!mkdir rescaled_images/spider
!mkdir rescaled_images/butterfly
!mkdir rescaled_images/chicken
!mkdir rescaled_images/horse
!mkdir rescaled_images/cow
!mkdir rescaled_images/sheep
!mkdir rescaled_images/squirrel
```

```
#data = np.empty((0,128*128*3), int)
count = 0

for subdir in os.listdir('raw-img/'):
    dir_path = f"raw-img/{subdir}"
    print(f"Starting with {dir_path}")
    eng_name = translations[subdir]

    for img in os.listdir(dir_path):
        img_path = dir_path + '/' + img
        image = Image.open(img_path)
        rgb_im = image.convert('RGB')
        res = rgb_im.resize((128, 128))

        res.save(f'rescaled_images/{eng_name}/{img}')

        count += 1

        if count % 100 == 0:
            print(count) # Just print the count every 100 images

    print(f'FINISHED {dir_path}')
    #np.savetxt('full_data.out', data)
    print(f'SAVED AFTER {dir_path}')
    print(f'count is {count}')
```

The code above was mostly EDA. From here, we have cleaned data. Note that most of our analysis from here on uses dog_spider_data which is stored in a npy file.

In [2]:
```
species_counts = {}
for subdir in os.listdir('rescaled_images/'):
    dir_path = f"rescaled_images/{subdir}"
    count = len(fnmatch.filter(os.listdir(dir_path), '*.*'))
    species_counts[subdir] = count
```

In [3]:
```
species_counts
```

Out[3]:
```
{'spider': 4821,
 'elephant': 1446,
 'dog': 4863,
 'chicken': 3098,
 'sheep': 1820,
 'horse': 2623,
 'butterfly': 2112,
 'cow': 1866,
 'squirrel': 1862,
 'cat': 1668}
```

The below code created a file with the joint array data for rescaled dogs and spiders...

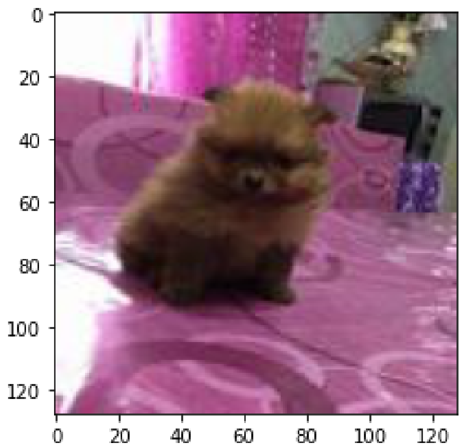
```
dog_spider_data = np.empty((0,128*128*3), int)
count = 0
directories = ['rescaled_images/dog', 'rescaled_images/spider']
for d in directories:
    print(f"Now opening: {d}")

    for img in os.listdir(d):
        img_path = d + '/' + img
        image = Image.open(img_path)
        rgb_im = image.convert('RGB')
        res = rgb_im.resize((128, 128))

        flattened = np.array(res).flatten()
        dog_spider_data = np.append(dog_spider_data,
flattened.reshape(1,len(flattened)), axis = 0)

        count += 1

        if count % 100 == 0:
            print(count) # Just print the count every 100 images

    print(f'FINISHED {d}')

print('\nDONE WITH ALL DIRECTORIES...\n')
np.save('dog_spider_data.npy', dog_spider_data)
print(f'count is {count}')

np.save('dog_spider_data_copy.npy', dog_spider_data)

import zipfile
with zipfile.ZipFile('dog_spider_data.zip','w') as zip_ref:
    zip_ref.write('dog_spider_data.npy',
compress_type=zipfile.ZIP_DEFLATED)


# If you're starting out with the dog_spider_data.zip file, unzip it
using this cell.
# This will create a new directory called dog_spider_data_dir. Inside
this directory will be a file (of size 3.81GB)
#  called dog_spider_data.npy. Move this file into the main cogs118b
directory and load it in using the following cell
# NOTE: once this cell is run and you have dog_spider_data.npy in your
local directory you don't need to run this.
import zipfile
```

```
with zipfile.ZipFile('dog_spider_data.zip',"r") as zip_ref:
    zip_ref.extractall('dog_spider_data_dir')
```

In [4]:
```
dog_spider_data = np.load('dog_spider_data.npy')
```

In [5]:
```
BREAK = 4863
res = dog_spider_data[0]
res.resize((128, 128, 3))
plt.imshow(res)
```

Out[5]:
```
<matplotlib.image.AxesImage at 0x7f530239f880>
```



In [6]:
```
dog_spider_data.shape
```

Out[6]:
```
(9684, 49152)
```

In [7]:
```
def showImg(array):
    array = array.reshape((128, 128, 3))
    plt.imshow(array)
```

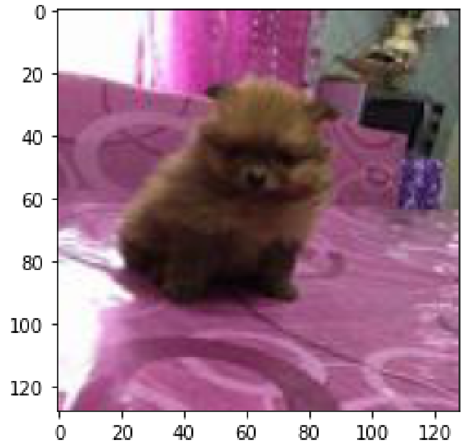
Find the Mean Face

We will be showing the mean face for dogs and spiders

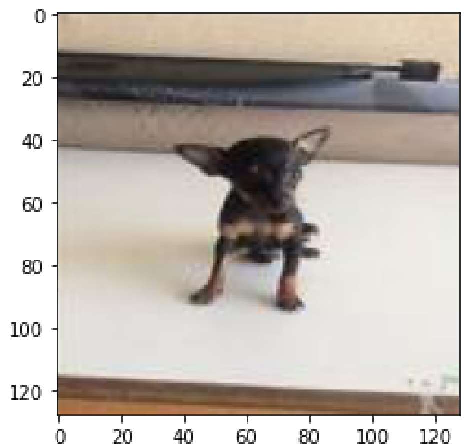
The below code manipulates the grayscale images.

```
dog_data = np.empty((0,128*128*1), int)
count = 0

for img in os.listdir('rescaled_images/dog'):
    img_path = 'rescaled_images/dog' + '/' + img
    image = Image.open(img_path)
    gray_im = image.convert('L') # convert to grayscale
    flattened = np.array(gray_im).flatten()

    dog_data = np.append(dog_data, flattened.reshape(1,len(flattened)),
axis = 0)
    count += 1
    if count % 100 == 0:
        print(count) # Just print the count every 100 images
```

In [8]:
```
showImg(dog_spider_data[0])
```



In [9]:
```
showImg(dog_spider_data[1])
```

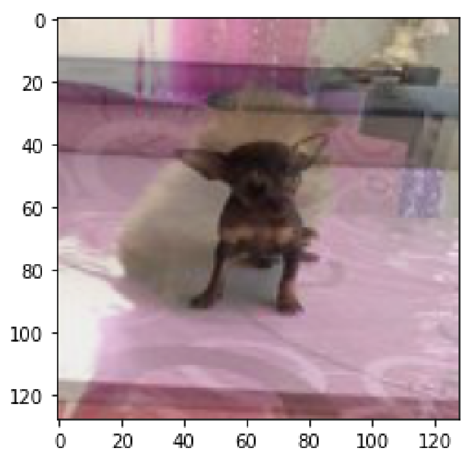


In [10]:
```
dog_data = dog_spider_data[0:2, :]
```

In [11]:
```
mean = np.mean(dog_data, axis = 0)
```

In [12]:
```
mean = mean.astype(int)
```

In [13]:
```
showImg(mean)
```



Next, we analyze around 60 hand-selected images of dogs that are similar based on pose and amount of the image they take up.

In [14]:
```
# open the zip file with 60 images...

#import zipfile
#with zipfile.ZipFile("similar_dogs-60.zip","r") as zip_ref:
#    zip_ref.extractall("60_dogs")
```

This cell opens and reshapes the 60 images into arrays and stores it in the npy file.

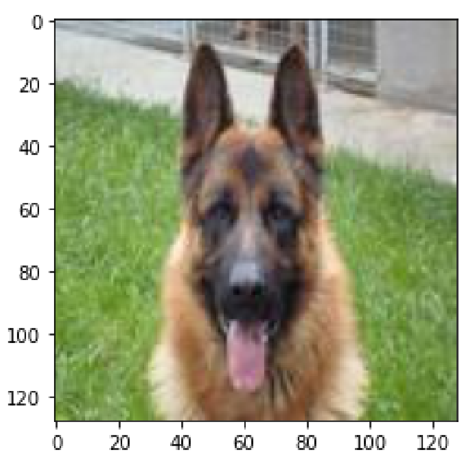
```
sixty_dogs = np.empty((0,128*128*3), int)
d = '60_dogs/similar_dogs'
count = 0
for img in os.listdir(d):
    img_path = d + '/' + img
    image = Image.open(img_path)
    rgb_im = image.convert('RGB')
    res = rgb_im.resize((128, 128))

    flattened = np.array(res).flatten()
    sixty_dogs = np.append(sixty_dogs,
flattened.reshape(1,len(flattened)), axis = 0)

    count += 1

    if count % 10 == 0:
        print(count) # Just print the count every 10 images

print('FINISHED')
np.save('sixty_dogs.npy', sixty_dogs)
```

In [15]:
```
dog_data = np.load('sixty_dogs.npy')
```

In [16]:
```
dog_data.shape
```

Out[16]:
```
(62, 49152)
```

In [17]:
```
showImg(dog_data[0])
```

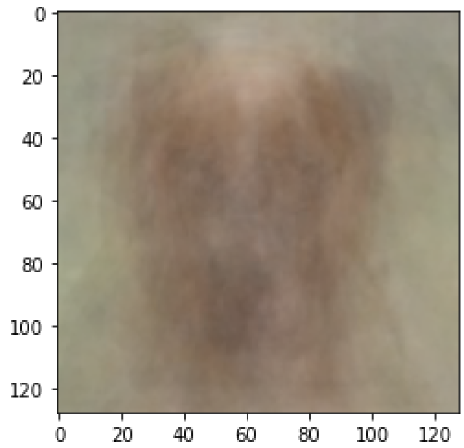


In [18]:
```
dawgs = dog_data[:, :]
```

In [19]:
```
mean = np.mean(dawgs, axis = 0)
```

In [20]:
```
mean = mean.astype(int)
```

In [21]:
```
showImg(mean)
```



Above, we can see the general silhouette of a dog, and this is due to our images having dogs that have similar poses and are focused in the image with less background. When finding the mean of all dog images, we do not see this clear silhouette because the variety of colors, dog sizes, and backgrounds will make the mean look less like a dog.

Normalizing the image means

dog_mean = np.mean(dog_data, axis = 0) dog_std = np.std(dog_data, axis = 0) norm = (dog_data - dog_mean) / dog_std norm

def find_mean_img(full_mat, animal, size = (128, 128, 3)):

```
# calculate the average
mean_img = np.mean(full_mat, axis = 0)
# reshape it back to a matrix
mean_img = mean_img.reshape(size)
plt.imshow(mean_img, )
plt.title(f'Average {animal}')
plt.axis('off')
plt.show()
return mean_img
```

find_mean_img(dog_data, 'DAWG')

find_mean_img(norm, 'DAWG')

norm2 = (dog_data - np.mean(dog_data, axis = 1).reshape((4863, 1)))/np.std(dog_data, axis = 1).reshape((4863,1)) norm2

norm2.shape

find_mean_img(norm2, 'DAWG').shape

find_mean_img(norm2, 'DAWG')

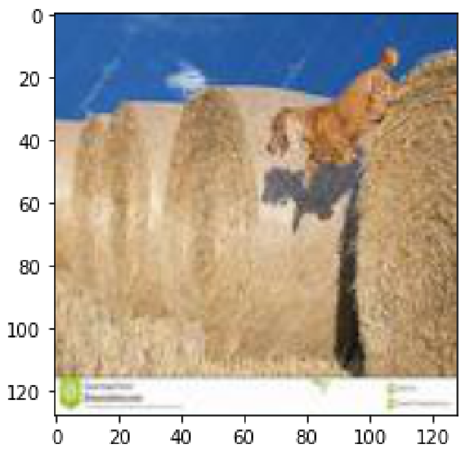
(dog_data - np.mean(dog_data, axis = 1).reshape((4863, 1)))

np.mean(dog_data, axis = 1).reshape((4863, 1))

KNN FOR DOGS and SPIDERS

In [34]:
```
from sklearn.neighbors import KNeighborsClassifier
from sklearn.model_selection import train_test_split
from sklearn.metrics import classification_report
import sklearn.metrics as skm
from sklearn.model_selection import RandomizedSearchCV
from sklearn.model_selection import GridSearchCV
```

In [23]:
```
showImg(dog_spider_data[BREAK-1])
```



In [24]:
```
# 0 = spider, 1 = dog
#
l_d = ['dog' for i in range(BREAK)]
l_s = ['spider' for i in range(len(dog_spider_data) - BREAK)]
labels = l_d + l_s
```

In [25]:
```
labels[BREAK-1]
```

Out[25]:
```
'dog'
```

In [26]:
```
(trainX, testX, trainY, testY) = train_test_split(dog_spider_data, labels, test_size=0.
```

In [27]:
```
# validation set should be 15% of entire dataset => equal to (1 - 70/85)% of training c
val_break = 1 - (70/85)
trainX, validationX, trainY, validationY = train_test_split(trainX, trainY, test_size=v
```

In [28]:
```
# model = KNeighborsClassifier()
# params = {'n_neighbors': [i for i in range(1, 26)]}
#
# clf = RandomizedSearchCV(estimator=model, param_distributions=params, cv=10, scoring=
```

In [31]:
```
# clf
```

Out[31]:
```
RandomizedSearchCV(cv=10, estimator=KNeighborsClassifier(),
                   param_distributions={'n_neighbors': [1, 2, 3, 4, 5, 6, 7, 8,
                                                        9, 10, 11, 12, 13, 14,
                                                        15, 16, 17, 18, 19, 20,
                                                        21, 22, 23, 24, 25]},
                   scoring='accuracy', verbose=3)
```

In [32]:
```
# clf.fit(trainX, trainY)
# valid_predictions = clf.predict(validationX)
```

```
Fitting 10 folds for each of 10 candidates, totalling 100 fits
[CV 1/10] END ..................n_neighbors=11;, score=0.578 total time=  35.3s
[CV 2/10] END ..................n_neighbors=11;, score=0.574 total time=  30.4s
[CV 3/10] END ..................n_neighbors=11;, score=0.588 total time=  31.8s
[CV 4/10] END ..................n_neighbors=11;, score=0.562 total time=  33.4s
[CV 5/10] END ..................n_neighbors=11;, score=0.583 total time=  28.7s
[CV 6/10] END ..................n_neighbors=11;, score=0.569 total time=  44.4s
[CV 7/10] END ..................n_neighbors=11;, score=0.552 total time=  35.0s
[CV 8/10] END ..................n_neighbors=11;, score=0.571 total time=  37.5s
[CV 9/10] END ..................n_neighbors=11;, score=0.570 total time=  35.8s
[CV 10/10] END .................n_neighbors=11;, score=0.585 total time=  40.9s
[CV 1/10] END ...................n_neighbors=4;, score=0.650 total time=  39.4s
[CV 2/10] END ...................n_neighbors=4;, score=0.631 total time=  46.1s
[CV 3/10] END ...................n_neighbors=4;, score=0.627 total time=  34.7s
[CV 4/10] END ...................n_neighbors=4;, score=0.622 total time=  43.0s
[CV 5/10] END ...................n_neighbors=4;, score=0.630 total time=  40.6s
[CV 6/10] END ...................n_neighbors=4;, score=0.615 total time=  29.6s
[CV 7/10] END ...................n_neighbors=4;, score=0.619 total time=  39.6s
[CV 8/10] END ...................n_neighbors=4;, score=0.640 total time=  41.9s
[CV 9/10] END ...................n_neighbors=4;, score=0.629 total time=  45.7s
[CV 10/10] END ..................n_neighbors=4;, score=0.635 total time=  40.1s
[CV 1/10] END ..................n_neighbors=23;, score=0.559 total time=  43.0s
[CV 2/10] END ..................n_neighbors=23;, score=0.558 total time=  34.5s
[CV 3/10] END ..................n_neighbors=23;, score=0.572 total time=  31.3s
[CV 4/10] END ..................n_neighbors=23;, score=0.555 total time=  40.1s
[CV 5/10] END ..................n_neighbors=23;, score=0.572 total time=  28.8s
[CV 6/10] END ..................n_neighbors=23;, score=0.546 total time=  30.0s
[CV 7/10] END ..................n_neighbors=23;, score=0.552 total time=  40.8s
[CV 8/10] END ..................n_neighbors=23;, score=0.556 total time=  42.8s
[CV 9/10] END ..................n_neighbors=23;, score=0.554 total time=  35.7s
[CV 10/10] END .................n_neighbors=23;, score=0.566 total time=  34.9s
[CV 1/10] END ...................n_neighbors=7;, score=0.591 total time=  40.3s
[CV 2/10] END ...................n_neighbors=7;, score=0.587 total time=  32.7s
[CV 3/10] END ...................n_neighbors=7;, score=0.583 total time=  41.4s
[CV 4/10] END ...................n_neighbors=7;, score=0.574 total time=  38.1s
[CV 5/10] END ...................n_neighbors=7;, score=0.588 total time=  35.9s
[CV 6/10] END ...................n_neighbors=7;, score=0.568 total time=  33.0s
[CV 7/10] END ...................n_neighbors=7;, score=0.565 total time=  39.6s
[CV 8/10] END ...................n_neighbors=7;, score=0.578 total time=  28.8s
[CV 9/10] END ...................n_neighbors=7;, score=0.578 total time=  34.7s
[CV 10/10] END ..................n_neighbors=7;, score=0.586 total time=  42.8s
[CV 1/10] END ..................n_neighbors=14;, score=0.586 total time=  44.9s
[CV 2/10] END ..................n_neighbors=14;, score=0.568 total time=  41.9s
[CV 3/10] END ..................n_neighbors=14;, score=0.590 total time=  40.1s
[CV 4/10] END ..................n_neighbors=14;, score=0.578 total time=  37.0s
[CV 5/10] END ..................n_neighbors=14;, score=0.587 total time=  34.3s
[CV 6/10] END ..................n_neighbors=14;, score=0.566 total time=  42.5s
[CV 7/10] END ..................n_neighbors=14;, score=0.550 total time=  42.2s
[CV 8/10] END ..................n_neighbors=14;, score=0.566 total time=  40.8s
[CV 9/10] END ..................n_neighbors=14;, score=0.579 total time=  38.6s
[CV 10/10] END .................n_neighbors=14;, score=0.588 total time=  43.4s
[CV 1/10] END ..................n_neighbors=19;, score=0.559 total time=  40.6s
[CV 2/10] END ..................n_neighbors=19;, score=0.572 total time=  38.5s
[CV 3/10] END ..................n_neighbors=19;, score=0.571 total time=  29.8s
[CV 4/10] END ..................n_neighbors=19;, score=0.556 total time=  39.3s
[CV 5/10] END ..................n_neighbors=19;, score=0.569 total time=  32.5s
[CV 6/10] END ..................n_neighbors=19;, score=0.556 total time=  33.2s
[CV 7/10] END ..................n_neighbors=19;, score=0.547 total time=  28.2s
[CV 8/10] END ..................n_neighbors=19;, score=0.560 total time=  39.0s
[CV 9/10] END ..................n_neighbors=19;, score=0.557 total time=  27.3s
[CV 10/10] END .................n_neighbors=19;, score=0.572 total time=  30.1s
[CV 1/10] END ..................n_neighbors=18;, score=0.571 total time=  38.0s
[CV 2/10] END ..................n_neighbors=18;, score=0.571 total time=  31.0s
[CV 3/10] END ..................n_neighbors=18;, score=0.581 total time=  36.0s
[CV 4/10] END ..................n_neighbors=18;, score=0.562 total time=  32.1s
```

```
[CV 5/10] END ...................n_neighbors=18;, score=0.580 total time=  36.1s
[CV 6/10] END ...................n_neighbors=18;, score=0.559 total time=  41.5s
[CV 7/10] END ...................n_neighbors=18;, score=0.552 total time=  40.6s
[CV 8/10] END ...................n_neighbors=18;, score=0.568 total time=  42.8s
[CV 9/10] END ...................n_neighbors=18;, score=0.569 total time=  45.4s
[CV 10/10] END ..................n_neighbors=18;, score=0.592 total time=  25.6s
[CV 1/10] END ...................n_neighbors=17;, score=0.562 total time=  30.0s
[CV 2/10] END ...................n_neighbors=17;, score=0.569 total time=  41.6s
[CV 3/10] END ...................n_neighbors=17;, score=0.574 total time=  41.2s
[CV 4/10] END ...................n_neighbors=17;, score=0.559 total time=  31.3s
[CV 5/10] END ...................n_neighbors=17;, score=0.568 total time=  43.3s
[CV 6/10] END ...................n_neighbors=17;, score=0.558 total time=  43.7s
[CV 7/10] END ...................n_neighbors=17;, score=0.549 total time=  36.4s
[CV 8/10] END ...................n_neighbors=17;, score=0.558 total time=  38.7s
[CV 9/10] END ...................n_neighbors=17;, score=0.557 total time=  28.2s
[CV 10/10] END ..................n_neighbors=17;, score=0.581 total time=  36.6s
[CV 1/10] END ...................n_neighbors=15;, score=0.562 total time=  28.8s
[CV 2/10] END ...................n_neighbors=15;, score=0.566 total time=  41.6s
[CV 3/10] END ...................n_neighbors=15;, score=0.580 total time=  33.6s
[CV 4/10] END ...................n_neighbors=15;, score=0.558 total time=  41.0s
[CV 5/10] END ...................n_neighbors=15;, score=0.574 total time=  36.7s
[CV 6/10] END ...................n_neighbors=15;, score=0.562 total time=  43.1s
[CV 7/10] END ...................n_neighbors=15;, score=0.549 total time=  29.4s
[CV 8/10] END ...................n_neighbors=15;, score=0.555 total time=  38.0s
[CV 9/10] END ...................n_neighbors=15;, score=0.564 total time=  43.0s
[CV 10/10] END ..................n_neighbors=15;, score=0.573 total time=  37.9s
[CV 1/10] END ...................n_neighbors=16;, score=0.571 total time=  42.1s
[CV 2/10] END ...................n_neighbors=16;, score=0.569 total time=  26.9s
[CV 3/10] END ...................n_neighbors=16;, score=0.590 total time=  35.2s
[CV 4/10] END ...................n_neighbors=16;, score=0.574 total time=  43.5s
[CV 5/10] END ...................n_neighbors=16;, score=0.580 total time=  31.8s
[CV 6/10] END ...................n_neighbors=16;, score=0.565 total time=  40.7s
[CV 7/10] END ...................n_neighbors=16;, score=0.555 total time=  38.7s
[CV 8/10] END ...................n_neighbors=16;, score=0.560 total time=  34.7s
[CV 9/10] END ...................n_neighbors=16;, score=0.573 total time=  32.5s
[CV 10/10] END ..................n_neighbors=16;, score=0.585 total time=  39.8s
```

In [34]:

```
# As we see above, 4 neighbors looks pretty good. But let's also try 2, 3, 5 neighbors

# reports = []
# for i in range(2, 6):
#     model_2 = KNeighborsClassifier(n_neighbors=i)
#     model_2.fit(trainX, trainY)
#     valid_predictions = model_2.predict(validationX)
#     print(classification_report(validationY, valid_predictions))
#     reports.append(classification_report(validationY, valid_predictions))
#     print('\n\n')
```

```
              precision    recall  f1-score   support

         dog       0.80      0.47      0.60       714
      spider       0.64      0.89      0.74       739

    accuracy                           0.68      1453
   macro avg       0.72      0.68      0.67      1453
weighted avg       0.72      0.68      0.67      1453
```

```
              precision    recall  f1-score   support

         dog       0.87      0.27      0.42       714
      spider       0.58      0.96      0.72       739

    accuracy                           0.62      1453
   macro avg       0.73      0.62      0.57      1453
weighted avg       0.72      0.62      0.57      1453
```

```
              precision    recall  f1-score   support

         dog       0.85      0.34      0.49       714
      spider       0.60      0.94      0.73       739

    accuracy                           0.65      1453
   macro avg       0.72      0.64      0.61      1453
weighted avg       0.72      0.65      0.61      1453
```

```
              precision    recall  f1-score   support

         dog       0.87      0.24      0.37       714
      spider       0.57      0.97      0.72       739

    accuracy                           0.61      1453
   macro avg       0.72      0.60      0.54      1453
weighted avg       0.72      0.61      0.55      1453
```

In [37]:
```
# 2 and 4 neighbors are weird, let's do a grid search
# model = KNeighborsClassifier()
# params = {'n_neighbors': [2, 4]}
#
# clf = GridSearchCV(estimator=model, param_grid=params, cv=10, scoring='accuracy', ver
#
# clf.fit(trainX, trainY)
# valid_predictions = clf.predict(validationX)
```

```
Fitting 10 folds for each of 2 candidates, totalling 20 fits
[CV 1/10] END ....................n_neighbors=2;, score=0.681 total time=  44.3s
[CV 2/10] END ....................n_neighbors=2;, score=0.662 total time=  44.2s
[CV 3/10] END ....................n_neighbors=2;, score=0.652 total time=  42.0s
[CV 4/10] END ....................n_neighbors=2;, score=0.642 total time=  39.6s
[CV 5/10] END ....................n_neighbors=2;, score=0.647 total time=  30.4s
[CV 6/10] END ....................n_neighbors=2;, score=0.642 total time=  39.4s
[CV 7/10] END ....................n_neighbors=2;, score=0.659 total time=  36.3s
[CV 8/10] END ....................n_neighbors=2;, score=0.668 total time=  42.5s
[CV 9/10] END ....................n_neighbors=2;, score=0.671 total time=  41.9s
[CV 10/10] END ...................n_neighbors=2;, score=0.666 total time=  38.5s
[CV 1/10] END ....................n_neighbors=4;, score=0.650 total time=  40.7s
[CV 2/10] END ....................n_neighbors=4;, score=0.631 total time=  30.9s
[CV 3/10] END ....................n_neighbors=4;, score=0.627 total time=  34.8s
[CV 4/10] END ....................n_neighbors=4;, score=0.622 total time=  47.0s
[CV 5/10] END ....................n_neighbors=4;, score=0.630 total time=  33.8s
[CV 6/10] END ....................n_neighbors=4;, score=0.615 total time=  33.2s
[CV 7/10] END ....................n_neighbors=4;, score=0.619 total time=  34.7s
[CV 8/10] END ....................n_neighbors=4;, score=0.640 total time=  47.9s
[CV 9/10] END ....................n_neighbors=4;, score=0.629 total time=  27.9s
[CV 10/10] END ...................n_neighbors=4;, score=0.635 total time=  31.1s
```

In [40]:
```
# clf.best_estimator_
```

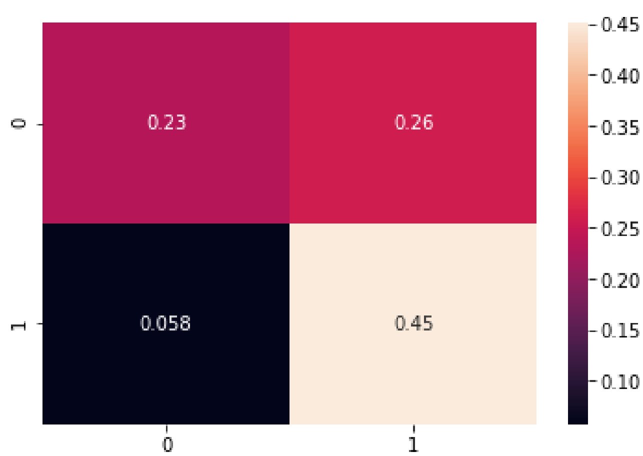
Out[40]:
```
KNeighborsClassifier(n_neighbors=2)
```

In [41]:
```
# print(classification_report(validationY, valid_predictions))
```

```
              precision    recall  f1-score   support

         dog       0.80      0.47      0.60       714
      spider       0.64      0.89      0.74       739

    accuracy                           0.68      1453
   macro avg       0.72      0.68      0.67      1453
weighted avg       0.72      0.68      0.67      1453
```

In [43]:
```
# cf_mat = skm.confusion_matrix(validationY, valid_predictions, labels=['dog', 'spider'
# sns.heatmap(cf_mat/np.sum(cf_mat), annot=True)
```

Out[43]:
```
<AxesSubplot:>
```



PCA PART

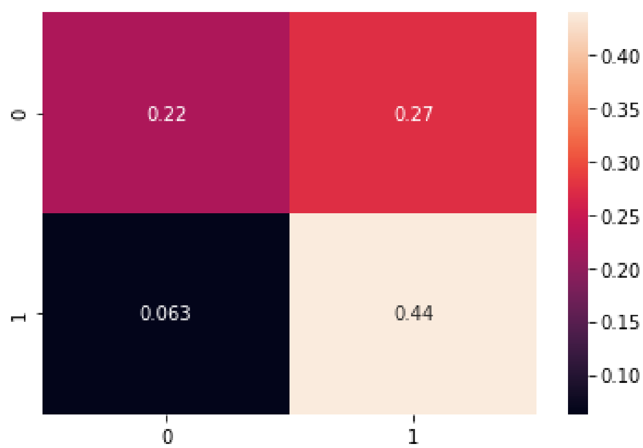
Okay so PCA will prob help cause we're removing unnecesary features.

But if we just run PCA like we did in the HW and remove one single feature, that will do like nothing.

We'll just keep like the top X amount of features... (look at HW4)

In [29]:
```
# we're gonna start by rerunning KNN on the data with the ideal params just to see what

model = KNeighborsClassifier(n_neighbors=2)
model.fit(trainX, trainY)
test_predictions = model.predict(testX)
cf_mat = skm.confusion_matrix(testY, test_predictions, labels=['dog', 'spider'])
sns.heatmap(cf_mat/np.sum(cf_mat), annot=True)
```

Out[29]: <AxesSubplot:>



In [30]:
```
print(classification_report(testY, test_predictions))
```

```
              precision    recall  f1-score   support

         dog       0.78      0.45      0.57       723
      spider       0.62      0.87      0.72       730

    accuracy                           0.66      1453
   macro avg       0.70      0.66      0.65      1453
weighted avg       0.70      0.66      0.65      1453
```

The above is our FINAL KNN thingy on the test set..

Below we start running PCA. Note that there is code commented out which was originally written to support the entire dataset by using a PyTorch GPU. However, as explained below we will instead be using a subset of the data for PCA.

In [32]:
```
# import torch
```

In [33]:
```
# torch.cuda.is_available()
```

In [34]:
```
# device = torch.device("cuda:0" if torch.cuda.is_available() else "cpu")
# print(device)
```

In [35]:
```
# torch.cuda.get_device_properties(device)
```

In [2]:
```
def eigsort(V, eigvals):

    # Sort the eigenvalues from largest to smallest. Store the sorted
    # eigenvalues in the column vector lambd.
```

```
    lohival = np.sort(eigvals)
    lohiindex = np.argsort(eigvals)
    lambd = np.flip(lohival)
    index = np.flip(lohiindex)
    Dsort = np.diag(lambd)

    # Sort eigenvectors to correspond to the ordered eigenvalues. Store sorted
    # eigenvectors as columns of the matrix vsort.
    M = np.size(lambd)
    Vsort = np.zeros((M, M))
    for i in range(M):
        Vsort[:,i] = V[:,index[i]]
    return Vsort, Dsort
```

```
def torch_eigsort(V, eigvals):

    # Sort the eigenvalues from largest to smallest. Store the sorted
    # eigenvalues in the column vector lambd.
    lohival = torch.sort(eigvals)
    lohiindex = torch.argsort(eigvals)
    lambd = torch.flip(lohival)
    index = torch.flip(lohiindex)
    Dsort = torch.diag(lambd)

    # Sort eigenvectors to correspond to the ordered eigenvalues. Store
sorted
    # eigenvectors as columns of the matrix vsort.
    M = torch.size(lambd)
    Vsort = torch.zeros((M, M))
    for i in range(M):
        Vsort[:,i] = V[:,index[i]]
    return Vsort, Dsort
```

In [22]:
```
# normc(M) normalizes the columns of M to a length of 1

def normc(Mat):
    return normalize(Mat, norm='l2', axis=0)
```

In [23]:
```
def showImg(array):
    array = array.reshape((128, 128, 3))
    plt.imshow(array)
```

In [ ]:
```
dog_spider_data = np.load('dog_spider_data.npy')
```

In [6]:
```
dog_spider_data.shape
```

Out[6]:
```
(9684, 49152)
```

In [7]:
```
BREAK = 4863
l_d = ['dog' for i in range(BREAK)]
l_s = ['spider' for i in range(len(dog_spider_data) - BREAK)]
labels = l_d + l_s

(train_data, x, train_data_Y, y) = train_test_split(dog_spider_data, labels, test_size=
```

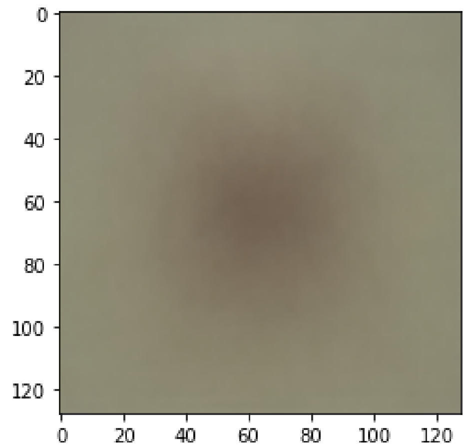
In [8]:
```
train_data = train_data.T
train_data.shape
```

Out[8]:
```
(49152, 1452)
```

In [10]:
```
meanface = np.mean(train_data, axis=1)
```

In [11]:
```
meanface.resize((49152, 1))
```

In [12]:
```
showImg(meanface.astype(int))
```



In [13]:
```
#A = dog_spider_data - np.matlib.repmat(meanface,9684,1)

Z = np.subtract(train_data, meanface)
ZT = Z.T
n = train_data.shape[1] # number of images
```

In [14]:
```
# Z = torch.from_numpy(Z)
# Z = Z.to(device)
# ZT = torch.from_numpy(ZT)
# ZT = ZT.to(device)
```

In [15]:
```
Z.shape
```

Out[15]:
```
(49152, 1452)
```

In [16]:
```
ZT.shape
```

Out[16]:
```
(1452, 49152)
```

In [17]:
```
# normally we'd calculate the covariance matrix, but for computation we're using the "t
# mul = torch.matmul(ZT, Z)
mul = ZT @ Z
```

In [18]:
```
mul.shape
```

Out[18]:
```
(1452, 1452)
```

In [19]:
```
# del ZT
# del Z
```

In [20]:
```
#torch.cuda.empty_cache()
#mul.to(device)
```

In [21]:
```
#torch.cuda.is_available()
```

In [22]:
```
eigvals, eigvecs = np.linalg.eig(mul)
```

In [23]:
```
V, Dsort = eigsort(eigvecs, eigvals) # V is the sorted eigenvectors
V.shape
```

Out[23]:
```
(1452, 1452)
```

In [24]:
```
U = np.matmul(Z, V) # U is the matrix of eigenfaces
U.shape
```

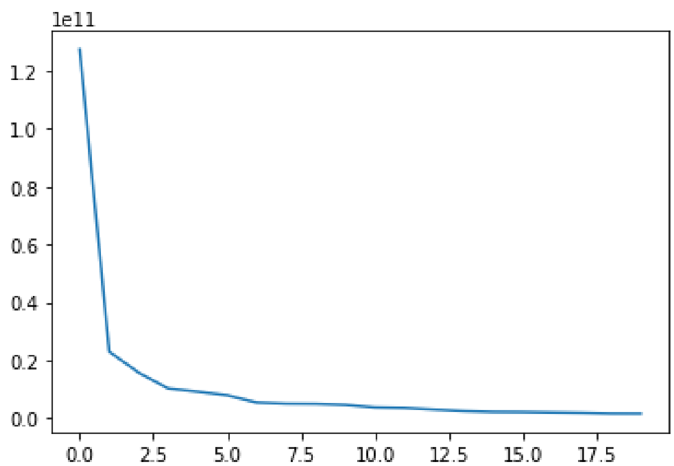
Out[24]:
```
(49152, 1452)
```

In [25]:
```
#normU = normc(U)
# i need to figure out how to get the columns to be normalized

normU = normc(U)
```

In [26]:
```
np.save('PCA_NormU.npy', normU)
```

In [27]:
```
c = np.matmul(normU.T, Z)
```

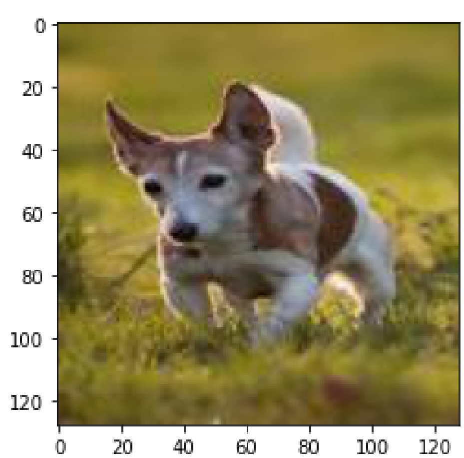
In [28]:
```
# make an elbow plot
plt.plot(range(20), Dsort.diagonal()[:20]);
```



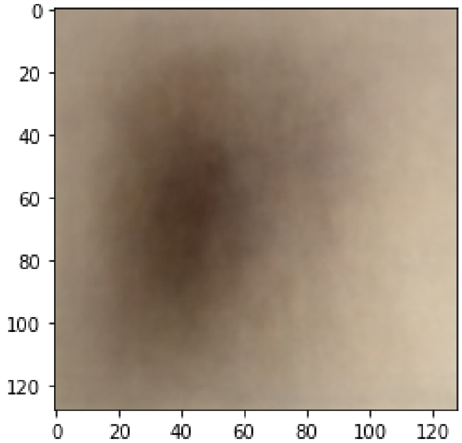
In [29]:
```
f = 3 # number of features to keep
```

In [30]:
```
training_reconstructed = meanface + (normU[:, :10] @ c[:10])
```

In [31]:
```
showImg(train_data[:, 444].astype(int))
```



In [32]:
```
showImg(training_reconstructed[:, 444].astype(int))
```



Now we (try to) generalize the PCA results to our full dataset.

In [4]:
```
import torch
```

In [5]:
```
torch.cuda.is_available()
```

Out[5]:
```
True
```

In [6]:
```
torch.cuda.empty_cache()
```

In [7]:
```
device = torch.device("cuda:0" if torch.cuda.is_available() else "cpu")
print(device)
```

```
cuda:0
```

In [6]:
```
torch.cuda.get_device_properties(device)
```

Out[6]:
```
_CudaDeviceProperties(name='GeForce RTX 2080 Ti', major=7, minor=5, total_memory=11019M
B, multi_processor_count=68)
```

In [8]:
```
dog_spider_data = np.load('dog_spider_data.npy')
normU = np.load('PCA_NormU.npy')
```

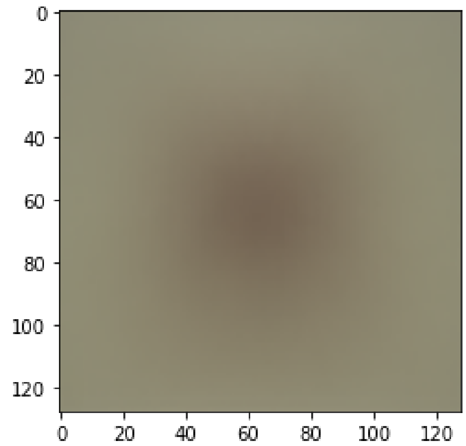
In [9]:
```
dog_spider_data = dog_spider_data.T
dog_spider_data.shape
```

Out[9]:
```
(49152, 9684)
```

In [10]:
```
meanface = np.mean(dog_spider_data, axis=1)
```

In [11]:
```
meanface.resize((49152, 1))
```

In [15]:
```
showImg(meanface.astype(int))
```



In [12]:
```
full_Z = np.subtract(dog_spider_data, meanface) # full_Z is my mean_zeroes data
```

In [13]:
```
NormUT = torch.from_numpy(normU.T)
NormUT = NormUT.to(device)
FullZ = torch.from_numpy(full_Z)
FullZ = FullZ.to(device)
```

In [14]:
```
full_C = torch.matmul(NormUT, FullZ)
```

In [15]:
```
torch.cuda.empty_cache()
```

In [16]:
```
del NormUT
del FullZ
```

In [17]:
```
full_c_cpu = full_C.cpu()
full_c = full_c_cpu.numpy()
```

In [18]:
```
# sanity check for full_c
full_c.shape
```

Out[18]:
```
(1452, 9684)
```

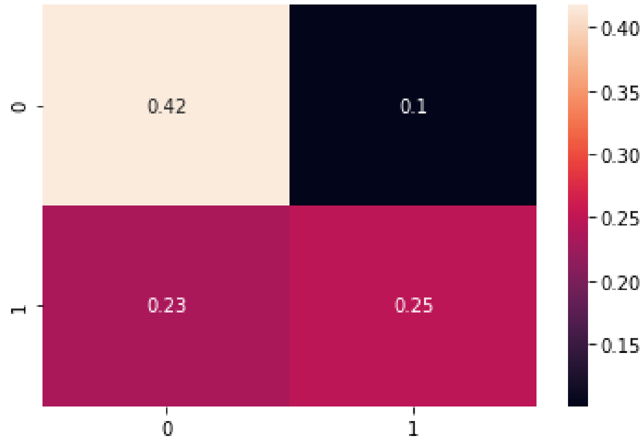
In [ ]:
```
full_reconstructed = meanface + (normU[:, :10] @ full_c[:10])
```

In [ ]:
```
showImg(full_reconstructed[:, 444].astype(int))
```

Once PCA is done, let's retrain KNN and see if it works. We'll use the same hyperparameters though.

In [35]:
```
labels = train_data_Y
(trainX, testX, trainY, testY) = train_test_split(training_reconstructed.T, labels, tes

model = KNeighborsClassifier(n_neighbors=2)
model.fit(trainX, trainY)
test_predictions = model.predict(testX)
cf_mat = skm.confusion_matrix(testY, test_predictions, labels=['dog', 'spider'])
sns.heatmap(cf_mat/np.sum(cf_mat), annot=True)
```

Out[35]:
```
<AxesSubplot:>
```



In [36]:
```
print(classification_report(testY, test_predictions))
```

```
              precision    recall  f1-score   support

         dog       0.64      0.81      0.71       113
      spider       0.71      0.51      0.60       105

    accuracy                           0.67       218
   macro avg       0.68      0.66      0.66       218
weighted avg       0.67      0.67      0.66       218
```

Literature Review

When evaluating our approach to PCA and KNN, there are a few things that could have been improved. According to Dandil, who conducted PCA on a specially generated dataset of cows, cats, dogs, goats, and rabbits, many pre-processing techniques were used to improve PCA's performance. For example, gray scaling our images could have massively improved our model. Though currently outside our scope, image improvements like elimination of roughness, removal of small debris, etc. could have also made the data easier to work with. In researching KNN based image classification, we discovered an interesting application by Amato and colleagues. In their study, KNN was used to classify images of different tourist landmarks by first classifying at the level of local features, and then extrapolating these probabilities to making a global, whole image classification. Using this technique, first classifying local features of an image and secondly classifying the whole image, could be an interesting future undertaking to improve KNN based image classification.

References

[1] Dandıl, Emre & Polattimur, Rukiye. (2018). PCA-Based Animal Classification System. 1-5. 10.1109/ISMSIT.2018.8567256. (https://www.researchgate.net/publication/329561682_PCA-Based_Animal_Classification_System)

[2] Amato, Giuseppe & Falchi, Fabrizio. (2010). KNN based image classification relying on local feature similarity. Proceedings - 3rd International Conference on SImilarity Search and APplications, SISAP 2010. 101-108. 10.1145/1862344.1862360. (https://www.researchgate.net/publication/221338231_KNN_based_image_classification_relying_on_loca