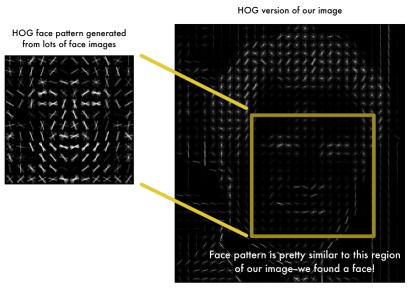
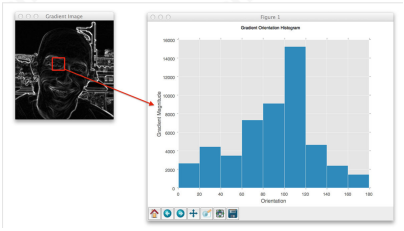
Important Terms To Know

Before starting a project with visual machine learning or facial recognition it is important to understand the basics behind how facial recognition & machine learning work.

- **Histogram of Oriented Gradients (HOG)**: When doing facial recongition, this feature descriptior allows for each image to be simplifed into a single variable. HOG breaks the image into sizes of (width * heigh & channel (colors)).



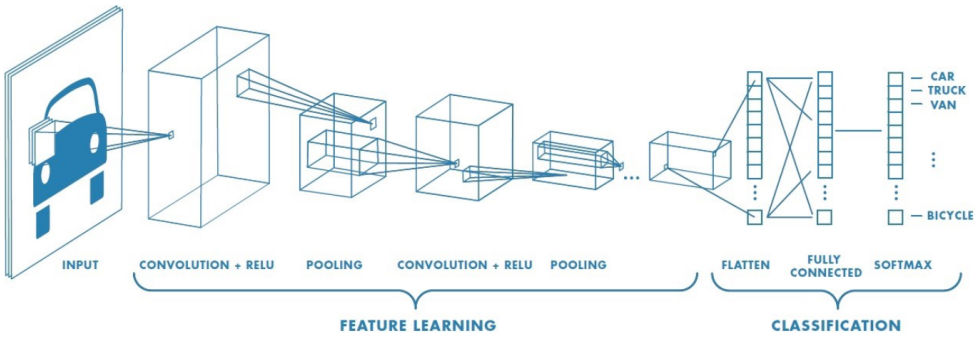
The final vectors are used to compare histograms of images and typically ran through a SVM to produce a non-linear classification.



- **Support Vector Machine (SVM)**: The classification aglorithm that the most popular image recognition library dlib uses. SVM is a supervised learning model that can be used for classification and is good for non-linear classification (which is image machine learning)
- **Convolutional Neural Networks (CNN)**: a classification that takes an input imge and is able to classify it into certain categories. This model will train and test each input (iamge) into different layers with filters - returning a single value between 0 and 1 to predict how an item should be classified.

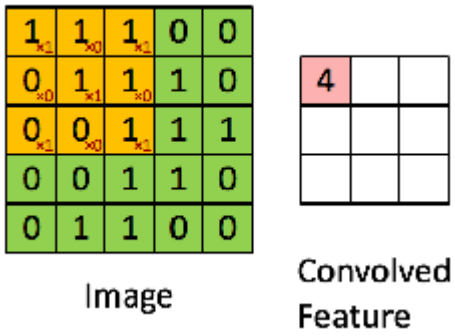
How does CNN Work?

To best understand how a CNN worked, I learned from the article Understanding of Convolutional Neural Netowrk (CNN) - Deep Learning by Prabhu. (https://medium.com/@RaghavPrabhu/understanding-of-convolutional-neural-network-cnn-deep-learning-99760835f148 (https://medium.com/@RaghavPrabhu/understanding-of-convolutional-neural-network-cnn-deep-learning-99760835f148))



Steps for CNN Modeling *This process can be repeated as many times as is necessary (or set) in the model to classify images.*

1. **Input** : Provide Image Input to model
2. **Convolution**: extract the first layer of the image, but preserve the relationship between the pixels. This will create the first "feature map" for the model.



3. **ReLU**: (Rectified Linear Unit for a non-linear operation) introduces non-linarity to the model. The more non linear the function is, the more complex of a problem it will be able to complete. This activation function $f(x)$=max(0,x) helps to increase the speed of training by removing the negative elements and setting them

Project 1: Visual Machine Learning

The first visual machine learning tutorial that I followed was from https://blog.hyperiondev.com/index.php/2019/02/18/machine-learning/ (https://blog.hyperiondev.com/index.php/2019/02/18/machine-learning/) which takes Google Street House Numbers from Stanford University to predict the number. The goal of this project is not to hone in on the best accuracy for the image recongition, but instead to learn the process of image recognition and basic steps for visual machine learning. All code snippets are directly from this tutorial.

The images are stored in a format that provides a 4D Matrix shape of 32x32x3x73257. This represents 32x32 images, in the RBG (3) format. And then there are 73257 images.

The prediction the algorithm will make is the number presented in the image between 0-9.

Set Up Process

To begin we need to make sure that the necessary dependencies are installed including: numpy, scipy and scikit-learn.

In [3]:
```
import scipy.io
import numpy as np
import matplotlib.pyplot as plt
```

Feature Processing

Next, after dependencies have been installed we can begin to process the features and train the data set. First we need to load our dataset 'train_32x32.mat'. The .mat extension are typically used in Matlab programs and are a binary data container. Scipy has a fucntion to load these filetypes just like a cvs which is ".loadmat". Then we want to split the dataset into X and Y which is already set in the .mat file.

In this preprocessed file for the tutorial the X = 4D matrix of images Y = 1D matrix of labels

so for example, X is going to be the actual image file and the Y will be the number it is trying to predict.

In [4]:
```
# load our dataset
train_data = scipy.io.loadmat('train_32x32.mat')
# extract the images and labels from the dictionary object
X = train_data['X']
y = train_data['y']
```

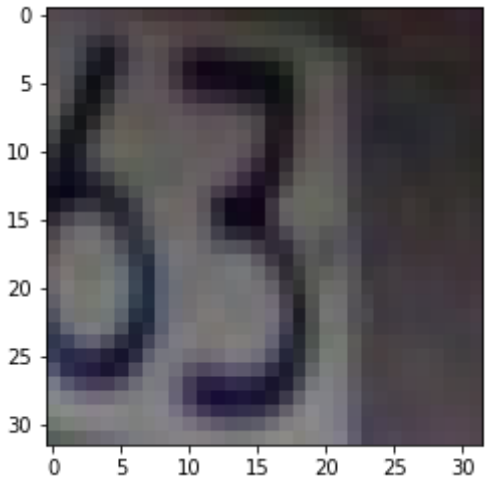
In [17]:
```
#printing the y, the associated lables
print(y[0])

#printing X the 4D matrix of the images
print(X[0,0])
```

```
[1]
[[ 33  84  19 ...  92 190 216]
 [ 30  76  54 ...  78 188 217]
 [ 38  59 110 ... 101 191 212]]
```

The 4D matrix doesn't make that much sense in the printed matrix above, so we can print out the image and its corresponding Y label - by using the plt.imshow function and looking up the image by its index:

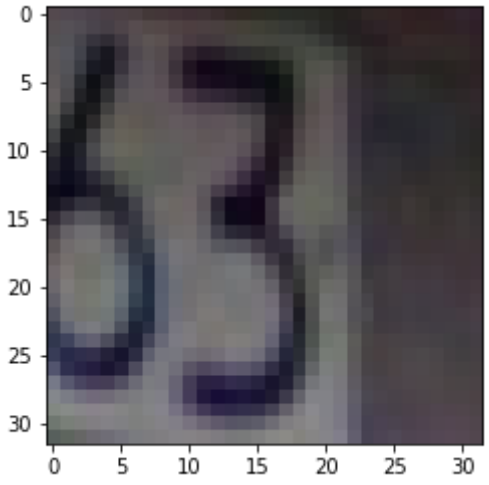
for "x" the index is : X[:,:,:,i] and for "y" the index is: y[i]

In [6]:
```
# view an image (e.g. 25) and print its corresponding label
img_index = 25
plt.imshow(X[:,:,:,img_index])
plt.figure(figsize=(1,1))
plt.show()
print(y[img_index])
```




```
[3]
```

In [7]:
```
# view an image (e.g. 25) and print its corresponding label
img_index = 25
plt.imshow(X[:,:,:,img_index])
plt.figure(figsize=(1,1))
plt.show()
print(y[img_index])
```




```
[3]
```

Vectorize the images

To perform machine learning, we need to vectorise them. Vectorization means that we will take all the dimensions we previously mentioned weidth x heigh x color channels (32x32x3) to make a 1D matrix that will be used as the feature vector.

This tutorial also had the dataste be shuffled to ensure that the data isn't distributed in a particular way in the way it was saved. This is done in the shuffle(X,y,random_state=42) which will help to eliminate accidental bias that could impact the results.

In [10]:
```
from sklearn.utils import shuffle
X = X.reshape(X.shape[0]*X.shape[1]*X.shape[2],X.shape[3]).T
y = y.reshape(y.shape[0],)
X, y = shuffle(X, y, random_state=42)
```

Machine Learning Algorithms

Now tat we have our X and y into the corrected vector and shuffled we can apply the machine learning algorithms to the dataeset.

The model we will use is Random Forest Classifier and split our data using rain_test_split from sklearn. We will set aside 80% of our data to train on.

In [11]:
```
from sklearn.ensemble import RandomForestClassifier
clf = RandomForestClassifier()
print(clf)
```

```
RandomForestClassifier(bootstrap=True, class_weight=None, criterion='gi
ni',
            max_depth=None, max_features='auto', max_leaf_nodes=None,
            min_impurity_decrease=0.0, min_impurity_split=None,
            min_samples_leaf=1, min_samples_split=2,
            min_weight_fraction_leaf=0.0, n_estimators=10, n_jobs=1,
            oob_score=False, random_state=None, verbose=0,
            warm_start=False)
```

In [15]:
```
from sklearn.model_selection import train_test_split
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.30
, random_state=42)
clf.fit(X_train, y_train)
```

Out[15]:
```
RandomForestClassifier(bootstrap=True, class_weight=None, criterion='gi
ni',
                       max_depth=None, max_features='auto', max_leaf_nodes=None,
                       min_impurity_decrease=0.0, min_impurity_split=None,
                       min_samples_leaf=1, min_samples_split=2,
                       min_weight_fraction_leaf=0.0, n_estimators=10, n_jobs=1,
                       oob_score=False, random_state=None, verbose=0,
                       warm_start=False)
```

After the clf has trained, we can use the model to predict on the test data we set aside in the previous setp. We also want to print out the Accuracy for the model.

In [14]:
```
from sklearn.metrics import accuracy_score
preds = clf.predict(X_test)
print("Accuracy:", accuracy_score(y_test,preds))
```

```
('Accuracy:', 0.5674310674310674)
```

Project 2: Facial Recognition

After mastering the basics of visual machine learning, I was interested in exploring facial recognition and the machine learning that is behind the process. The goal of this project is to allow for my own images to be recognized and classified.

For this project & learning Facial Recongition machine learning I utilized the tutorials available from https://github.com/ageitgey/face_recognition (https://github.com/ageitgey/face_recognition). This face_recongition library was built using the library *dlib* (https://github.com/davisking/dlib/blob/master/examples/dnn_metric_learning_ex.cpp (https://github.com/davisking/dlib/blob/master/examples/dnn_metric_learning_ex.cpp)) which utlitizes deep learning for facial recognition. All code snippets are used directly from this tutorial, with adjustments made based on file names, configurations or particular display changes.

Installation

The first step is to install all neccesary dependencies and packages:

- download repository https://github.com/ageitgey/face_recognition.git (https://github.com/ageitgey/face_recognition.git)
- then run: python setup.py install

(I ran into issues with the install and had to install cmake and dlib separately)

Part 1: Recongizing Faces Exist

Before I can predict if a face matches between photos, first I need to understand how the program detects faces to begin with. This was done through their face_locations function. This uses the hog model explained at the beginning of this tutorial and will return an array of bounding boxes (outline) of all human faces in the image.

This function face_locations takes the following parameters:

- image
- of of times to upsample (how far into the picture do we want to look? the more times you select this the longer it will take but is better for images with more people or smaller faces)
- model: what model do you want to use? default is hog

The goal of this step is to not identify who is in the photo but identify how many faces are in the photo. I chose to use this image since I knew it was high quality photo with the faces clearly defined.

Chosen Photo:

In [5]:
```
#If this is the first time running the program you will need to run the
 following & also git clone the repo
#! pip3 install face_recognition
##! pip3 install dlib
#! brew upgrade cmake
#! pip install dlib
#! pip install opencv-python
```

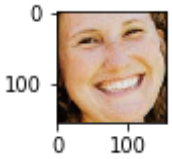
In [6]:
```
import face_recognition
image = face_recognition.load_image_file("first_test_image.jpg")
face_locations = face_recognition.face_locations(image)
print (face_locations)
```

```
[(511, 494, 666, 339), (460, 769, 614, 614)]
```

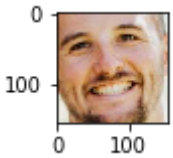
The above code just prints out the matrix of where the face outlines are, but what if we want to see what faces it identified? To do this we can use the same plt function as with the house number identification & put in the resulting image.

In [19]:
```
import face_recognition

from PIL import Image


image = face_recognition.load_image_file("first_test_image.jpg")
face_locations = face_recognition.face_locations(image)

print (face_locations)
print("I found {} face(s) in this photograph.".format(len(face_locations
)))

for face_location in face_locations:

    # Print the location of each face in this image
    top, right, bottom, left = face_location
    print("A face is located at pixel location Top: {}, Left: {}, Botto
m: {}, Right: {}".format(top, left, bottom, right))

    # You can access the actual face itself like this:
    face_image = image[top:bottom, left:right]
    pil_image = Image.fromarray(face_image)
   # pil_image.show()
    plt.figure(figsize=(1,1))

    plt.imshow(pil_image)
    plt.show()
```

```
[(511, 494, 666, 339), (460, 769, 614, 614)]
I found 2 face(s) in this photograph.
A face is located at pixel location Top: 511, Left: 339, Bottom: 666, R
ight: 494
```



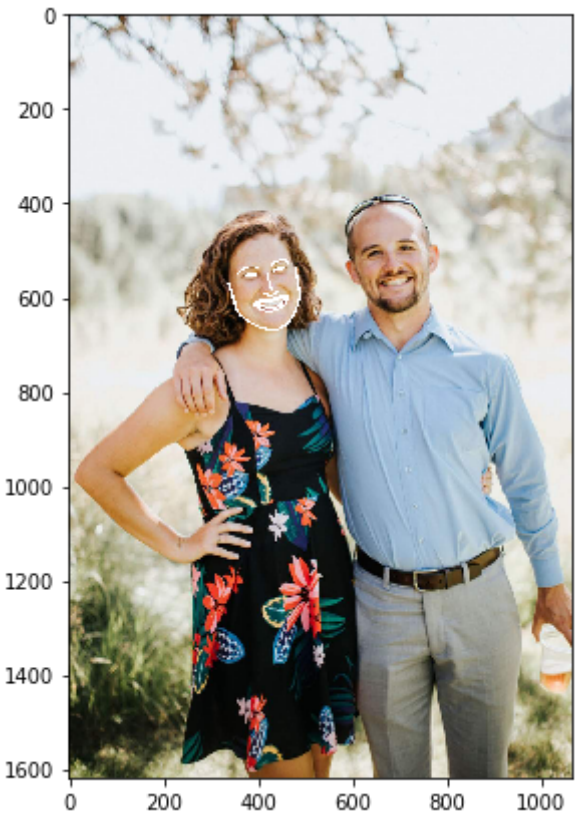

```
A face is located at pixel location Top: 460, Left: 614, Bottom: 614, R
ight: 769
```



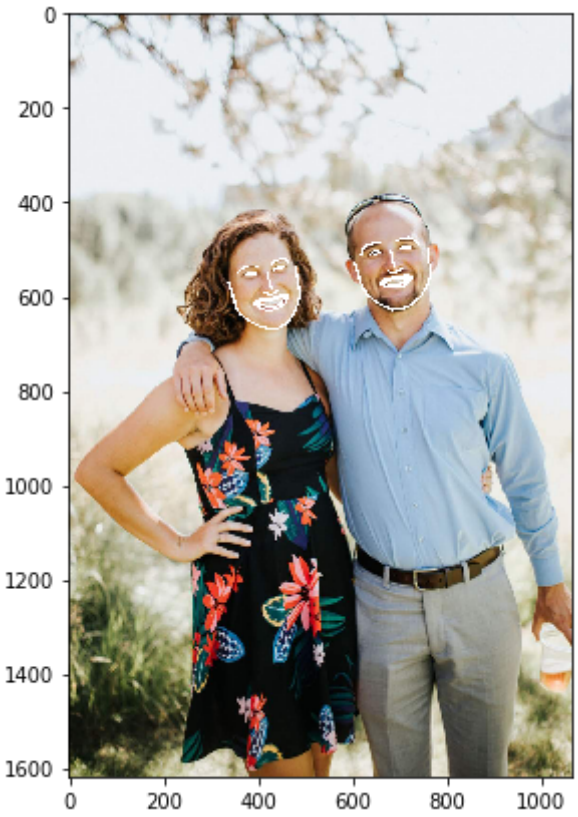
The function can also identify facial features in the face. To highlight these we can use the function “face_landmarks” , which will high light the chin, jaw line, nose, eyebrows and eyes.

In [25]:
```
# Find all facial features in all the faces in the image
face_landmarks_list = face_recognition.face_landmarks(image)

print("I found {} face(s) in this photograph.".format(len(face_landmarks
_list)))

# Create a PIL imagedraw object so we can draw on the picture
pil_image = Image.fromarray(image)
d = ImageDraw.Draw(pil_image)

for face_landmarks in face_landmarks_list:

    # Print the location of each facial feature in this image
    for facial_feature in face_landmarks.keys():
        print("The {} in this face has the following points: {}".format(
facial_feature, face_landmarks[facial_feature]))

    # Let's trace out each facial feature in the image with a line!
    for facial_feature in face_landmarks.keys():
        d.line(face_landmarks[facial_feature], width=5)

# Show the picture
#pil_image.show()
   # pil_image = Image.fromarray(face_image)
   # pil_image.show()
    plt.figure(figsize=(7,7))
    plt.imshow(pil_image)
    plt.show()
```

```
I found 2 face(s) in this photograph.
The chin in this face has the following points: [(338, 566), (343, 58
5), (349, 604), (354, 623), (365, 639), (383, 652), (402, 662), (424, 6
68), (444, 666), (459, 658), (471, 644), (480, 628), (486, 611), (489,
593), (486, 574), (483, 555), (479, 537)]
The left_eyebrow in this face has the following points: [(356, 550), (3
64, 542), (376, 536), (389, 534), (401, 536)]
The right_eyebrow in this face has the following points: [(429, 530),
(437, 524), (447, 521), (456, 521), (465, 526)]
The nose_bridge in this face has the following points: [(419, 546), (42
2, 559), (426, 572), (429, 584)]
The nose_tip in this face has the following points: [(413, 593), (421,
594), (430, 595), (436, 591), (442, 586)]
The left_eye in this face has the following points: [(374, 554), (382,
550), (389, 549), (399, 552), (390, 553), (382, 554)]
The right_eye in this face has the following points: [(433, 545), (440,
539), (447, 536), (455, 538), (449, 540), (442, 542)]
The top_lip in this face has the following points: [(390, 614), (404, 6
07), (419, 604), (430, 603), (441, 599), (453, 596), (464, 597), (460,
599), (442, 603), (431, 607), (420, 607), (394, 614)]
The bottom_lip in this face has the following points: [(464, 597), (45
8, 614), (447, 623), (436, 627), (425, 629), (407, 627), (390, 614), (3
94, 614), (423, 622), (434, 621), (445, 617), (460, 599)]
```



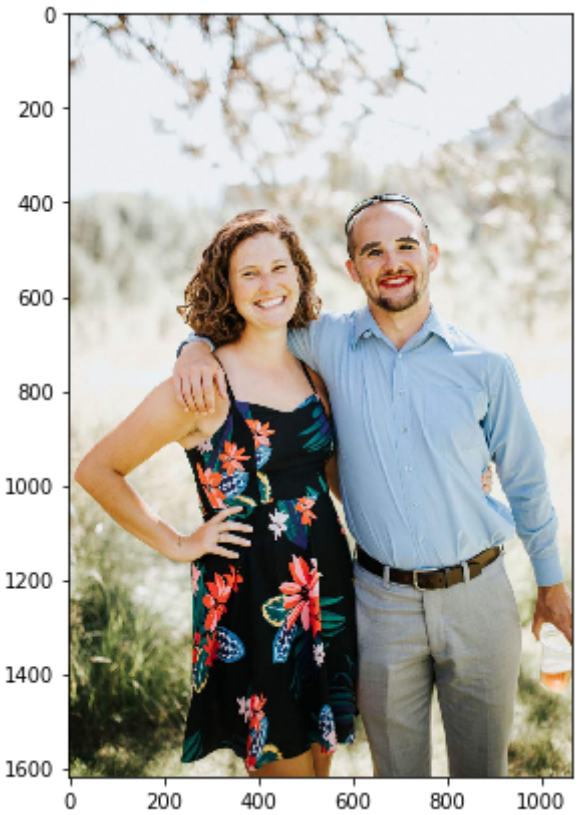
```
The chin in this face has the following points: [(604, 525), (610, 54
4), (615, 563), (622, 581), (632, 597), (647, 610), (665, 619), (684, 6
28), (703, 627), (719, 621), (733, 609), (747, 595), (757, 579), (763,
560), (763, 539), (762, 518), (761, 497)]
The left_eyebrow in this face has the following points: [(618, 512), (6
22, 498), (633, 490), (647, 487), (662, 488)]
The right_eyebrow in this face has the following points: [(692, 483),
(705, 476), (718, 475), (732, 480), (740, 490)]
The nose_bridge in this face has the following points: [(680, 500), (68
3, 512), (685, 525), (688, 538)]
The nose_tip in this face has the following points: [(673, 546), (681,
548), (690, 549), (698, 545), (705, 540)]
The left_eye in this face has the following points: [(637, 511), (644,
504), (652, 502), (661, 507), (653, 509), (645, 511)]
The right_eye in this face has the following points: [(701, 500), (708,
493), (716, 492), (724, 496), (717, 498), (709, 500)]
The top_lip in this face has the following points: [(657, 572), (668, 5
63), (681, 560), (692, 559), (702, 557), (715, 555), (727, 560), (723,
560), (703, 560), (693, 562), (682, 563), (661, 571)]
The bottom_lip in this face has the following points: [(727, 560), (71
8, 571), (707, 578), (696, 580), (685, 581), (671, 580), (657, 572), (6
61, 571), (683, 575), (694, 575), (705, 572), (723, 560)]
```



In the spirit of exploration, this tutorial had a fun side project where you could apply "digital makeup" to the photos. While not directly related to facial recogntion, it was a good exercise in thinking about what this program is capable of.

In [26]:
```
face_landmarks_list = face_recognition.face_landmarks(image)

for face_landmarks in face_landmarks_list:
    pil_image = Image.fromarray(image)
    d = ImageDraw.Draw(pil_image, 'RGBA')

    # Make the eyebrows into a nightmare
    d.polygon(face_landmarks['left_eyebrow'], fill=(68, 54, 39, 128))
    d.polygon(face_landmarks['right_eyebrow'], fill=(68, 54, 39, 128))
    d.line(face_landmarks['left_eyebrow'], fill=(68, 54, 39, 150), width
=5)
    d.line(face_landmarks['right_eyebrow'], fill=(68, 54, 39, 150), widt
h=5)

    # Gloss the lips
    d.polygon(face_landmarks['top_lip'], fill=(150, 0, 0, 128))
    d.polygon(face_landmarks['bottom_lip'], fill=(150, 0, 0, 128))
    d.line(face_landmarks['top_lip'], fill=(150, 0, 0, 64), width=8)
    d.line(face_landmarks['bottom_lip'], fill=(150, 0, 0, 64), width=8)

    # Sparkle the eyes
    d.polygon(face_landmarks['left_eye'], fill=(255, 255, 255, 30))
    d.polygon(face_landmarks['right_eye'], fill=(255, 255, 255, 30))

    # Apply some eyeliner
    d.line(face_landmarks['left_eye'] + [face_landmarks['left_eye'][0]],
fill=(0, 0, 0, 110), width=6)
    d.line(face_landmarks['right_eye'] + [face_landmarks['right_eye'][0
]], fill=(0, 0, 0, 110), width=6)

    #pil_image.show()
    plt.figure(figsize=(7,7))
    plt.imshow(pil_image)
    plt.show()
```


0
200
400
600
800
1000
1200
1400
1600
0
200
400
600
800
1000


0
200
400
600
800
1000
1200
1400
1600
0
200
400
600
800
1000

Part 2: Identifying Who Is In A Photo

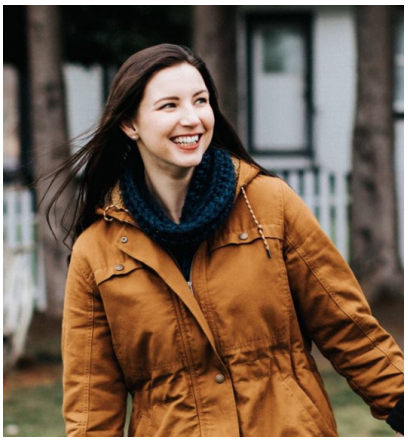
After knowing that I can identify faces with the face_recogintion program, the next step is to add in a layer of logic that will identify *who* is in each photo. This next step utilizes a training photo to let the program know who was who in each photo.

This is done through a three-step process:

1. Load both images, the known image & the unknown image
2. Run each image through the function face_encodings. This function runs through the process explained about encoding images into their feature files. This is done for both of the images because it is what will be used to compare the two files.
3. Run each encoding of images through the compare_face function. This function will take both of the encodings and compares the list of features to determine if it is a match. This will return the list of true/false results for the matched pictures. Since this round we are only doing 1 image, we will expect only a single true or false to be returned. Since it is not two pictures of myself I would expect to have ***false*** returned.

The Known Image in this case will be myself:

And the Unknown Image will be my friend Holly:



In [29]:
```
known_image = face_recognition.load_image_file("Part_2_Images/allison_known.png")
unknown_image = face_recognition.load_image_file("Part_2_Images/holly_unknown.png")

allison_encoding = face_recognition.face_encodings(known_image)[0]
unknown_encoding = face_recognition.face_encodings(unknown_image)[0]

results = face_recognition.compare_faces([allison_encoding], unknown_encoding)
print(results)
```

```
[False]
```

Now lets see what happens if I run the same code against two pictures of myself:

Known Picture:

Unknown Picture (of myself):

Since thse are both pictures of myself, I would expect the result to be ***true***.

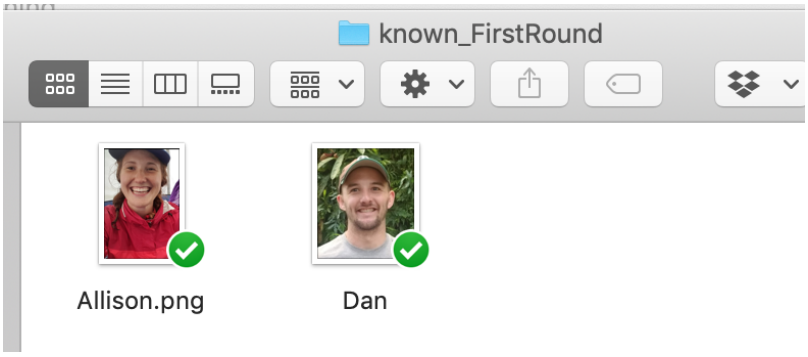
In [30]:
```
known_image = face_recognition.load_image_file("Part_2_Images/allison_known.png")
unknown_image = face_recognition.load_image_file("Part_2_Images/allisonagain.png")

allison_encoding = face_recognition.face_encodings(known_image)[0]
unknown_encoding = face_recognition.face_encodings(unknown_image)[0]

results = face_recognition.compare_faces([allison_encoding], unknown_encoding)
print(results)
```

```
[True]
```

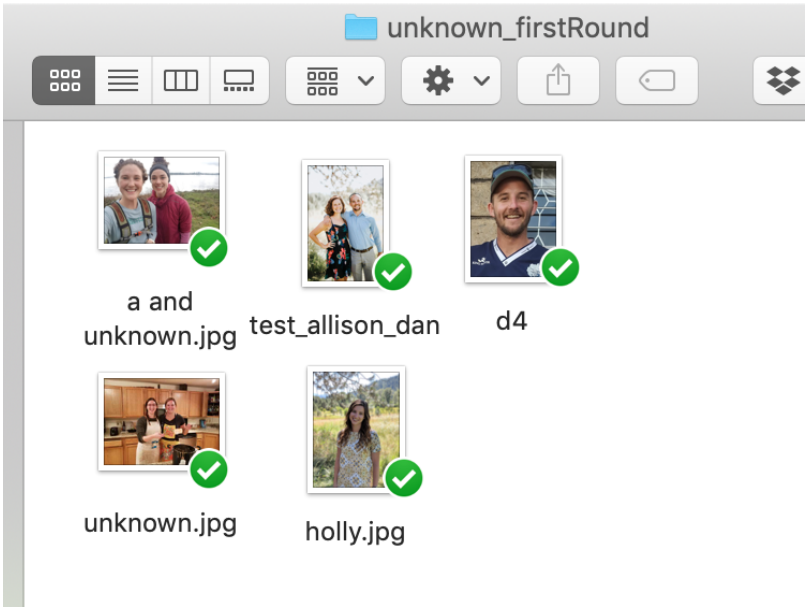
What About More Than 1 Person?

If this works well for one person, and one image, lets see if we are able to expand this to a batch of images.

First, I created a folder that was labeled “known_FirstRound” and included a picture of Dan and myself – clearly labelled. The program will pull the name from the file in “known” images.



Then, I created a folder that included all the pictures I wanted the program to identify. This folder included a variety of pictures that included pictures of myself and Dan, pictures of myself with another friend, a picture of just Dan, and two pictures of totally different people. I made sure to label each of the images with who was in the picture, because the program would return the name of the image & then who was in the image and I would get instant feedback if the program was correctly identifying the faces.



For this round, I am able to utilize the command line option for the face_recongition because we will not be adding in any other functionality. This means I can call the comparison by just passing through two parameters of the folders I created.

In a Jupyter Notebook, you must use ! in front of a command to let the notebook know you want it ran as a command line prompt (not in python).

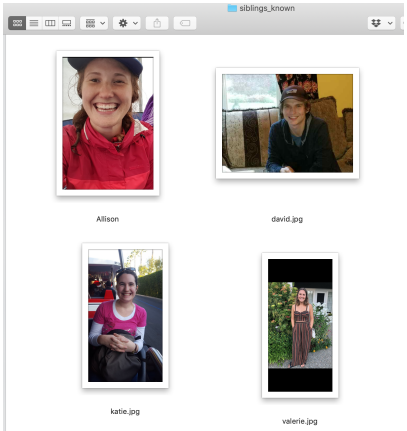
In [32]:
```
#! face_recognition ./known_FirstRound ./unknown_firstRound
```

```
./unknown_firstRound/unknown.jpg,unknown_person
./unknown_firstRound/unknown.jpg,unknown_person
./unknown_firstRound/d4.png,Dan
./unknown_firstRound/a and unknown.jpg,unknown_person
./unknown_firstRound/a and unknown.jpg,Allison
./unknown_firstRound/test_allison_dan.jpg,Dan
./unknown_firstRound/test_allison_dan.jpg,Allison
./unknown_firstRound/holly.jpg,unknown_person
```

It correctly identified all the people in the images! For each face in the image, it will return who it is. That is why for the image test_allison_dan.jpg it returns two lines, one identifying Allison and the other Identifying Dan.

What About My Siblings?

We all look alike; how would that impact the classification? I repeated the same process over with pictures of my siblings. The folder siblings_known had pictures of each of us.



Another folder was made was for "unknown images" that had a variety of groupings of my siblings, but also people who were not any of us in the photos as well.

We can run this code the same away as before, through the command line. For each person in each photo the classificaiton model will make a prediction (based on the encoded images from the "known_siblings" folder).

In [36]:
```
! face_recognition ./siblings_known ./siblings_unknown
```

```
./siblings_unknown/david.jpg,david
./siblings_unknown/david2.jpg,david
./siblings_unknown/unknown4.jpeg,unknown_person
./siblings_unknown/unknonw2.jpeg,unknown_person
./siblings_unknown/Katie_val_allison.jpg,valerie
./siblings_unknown/Katie_val_allison.jpg,Allison
./siblings_unknown/Katie_val_allison.jpg,katie
./siblings_unknown/Katie_val_allison.jpg,valerie
./siblings_unknown/Katie_val_allison.jpg,Allison
./siblings_unknown/Katie_val_allison.jpg,katie
./siblings_unknown/Katie_val_allison.jpg,valerie
./siblings_unknown/Katie_val_allison.jpg,Allison
./siblings_unknown/Katie_val_allison.jpg,katie
./siblings_unknown/val2.jpg,valerie
./siblings_unknown/val2.jpg,Allison
./siblings_unknown/val2.jpg,katie
```

Each time that it is one of my sisters, it predicts that its each of us - in the group photo of my sisters, it tries to assign each of us to each person.

To see if we can fix this issue, we are able to change the level of tolerance the program has. This can be done by adding the parameter of "sensitivity" to the function. It is set at default to be .6, so we will change it to be .50.

In [38]:
```
! face_recognition --tolerance 0.50 ./siblings_known/ ./siblings_unknow
n/
```

```
./siblings_unknown/david.jpg,david
./siblings_unknown/david2.jpg,unknown_person
./siblings_unknown/unknown4.jpeg,unknown_person
./siblings_unknown/unknonw2.jpeg,unknown_person
./siblings_unknown/Katie_val_allison.jpg,valerie
./siblings_unknown/Katie_val_allison.jpg,katie
./siblings_unknown/Katie_val_allison.jpg,Allison
./siblings_unknown/val2.jpg,valerie
./siblings_unknown/val2.jpg,Allison
```

That seemed to do the trick! It still looks like it is struggling to tell my sister Valerie and myself apart, but the Google photos algorithm also mixes us up from time to time, so I count this as a success! Changing the tolerance to a lower number makes the algorithm "stricter" which can increase the model, but also could increase the risk of overtraining the model. The stricter the tolerance, the closer the encodings must be to be considered a match.

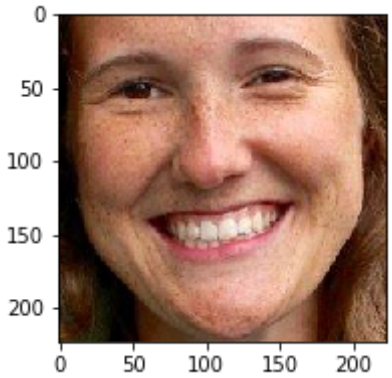
A Look Into CNN

The default model to use is HOG but now we will switch the model to use the CNN model presented in the beginning of the tutorial. The CNN is more accurate than HOG, but takes considerable more time, which is important to consideration when building a scalable model.

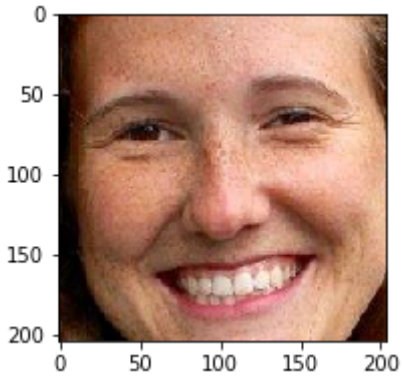
We can see the differences between these two models by running them sequentially and printing out the face locations from an image of myself. You can see that the CNN (even if just slightly) has a tighter crop on my face.

In [55]:
```
import face_recognition
from PIL import Image

image = face_recognition.load_image_file("Allison_CNN.jpg")
face_locations = face_recognition.face_locations(image, model="hog")

print (face_locations, "HOG Default Round")


for face_location in face_locations:

    # Print the location of each face in this image
    top, right, bottom, left = face_location
    # You can access the actual face itself like this:
    face_image = image[top:bottom, left:right]
    pil_image = Image.fromarray(face_image)
   # pil_image.show()
    plt.figure(figsize=(3,3))

    plt.imshow(pil_image)
    plt.show()




face_locations = face_recognition.face_locations(image, model="cnn")
# print face_locations

print (face_locations, " CNN Round")


for face_location in face_locations:

    # Print the location of each face in this image
    top, right, bottom, left = face_location
    # You can access the actual face itself like this:
    face_image = image[top:bottom, left:right]
    pil_image = Image.fromarray(face_image)
   # pil_image.show()
    plt.figure(figsize=(3,3))

    plt.imshow(pil_image)
    plt.show()
```

```
[(415, 489, 638, 266)] HOG Default Round
```



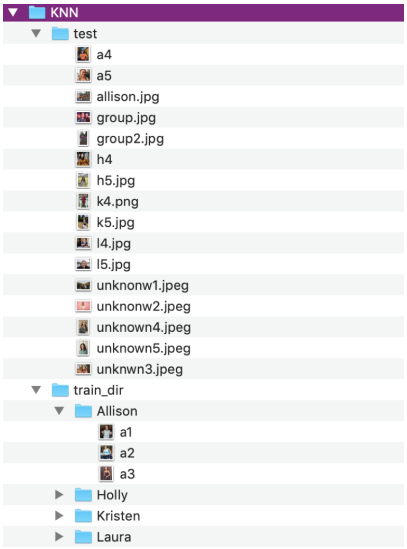

```
[(393, 470, 597, 266)]  CNN Round
```



Using KNN to Make Predictions

Now that I have understood how the base HOG model works through the command line, I took a deeper look into different models that can be used for image recognition. The supervised learning KNN (K nearest neighbor) is a classification model that can be used to run the model on a large group of known people – and then run the model against a large group of unknown images.

The model will be trained on a "train" directory that includes labeled faces of an individual in specific folders. This is much like in the first example where we created a "known_person" folder, but on a larger scale. The program assumes that all photos within the train_dir are of the same person and will be labelled by the folder name. The algorithm will then be ran on the test folder and find the images with the "k" closest facial features based on the encodings of the images. This particular KNN is weighted, so the closer the neighbor the most weight that "vote" will get.



I ran this program with training the program on 4 people, Allison, Holly, Kristen and Laura. I included 3 images for each person in the training directory. Then in the test directory I included a variety of images, some groups of us together, some individual shots and 5 pictures of unknown people altogether from stockphotos.com.

This program came with safeguards for if an image doesn't have an identifiable face, or if it's not suitable for training it will throw and error instead of reducing the accuracy of the model and keeping the image in the training set. This is done through the face_locations count, and if its less than 1 then that photo will be rejected.

The program is done in three steps:

1. Train the KNN classifier and save it to the disk. Saving it to a variable means that if this program was to be used multiple times over, it was the option to reuse the KNN classifier, saving time in the future.
2. With the classifier saved, we can now pass in the test directory and the model will predict for the unknown images.
3. Print out the results – Since the model does not know if the image its identifying is correct – it will print out the names of who is in the image as well as print the results on an overlaid image. This is done by concatenating the image name and prediction onto the image.

In [ ]:
```
import math
from sklearn import neighbors
import os
import os.path
import pickle
from PIL import Image, ImageDraw, ImageFont
import face_recognition
from face_recognition.face_recognition_cli import image_files_in_folder
import csv
ALLOWED_EXTENSIONS = {'png', 'jpg', 'jpeg'}


def train(train_dir, model_save_path=None, n_neighbors=None, knn_algo='ball_tree', verbose=False):

    X = []
    y = []

    # Loop through each person in the training set
    for class_dir in os.listdir(train_dir):
        if not os.path.isdir(os.path.join(train_dir, class_dir)):
            continue

        # Loop through each training image for the current person
        for img_path in image_files_in_folder(os.path.join(train_dir, class_dir)):
            image = face_recognition.load_image_file(img_path)
            face_bounding_boxes = face_recognition.face_locations(image)

            if len(face_bounding_boxes) != 1:
                # If there are no people (or too many people) in a training image, skip the image.
                if verbose:
                    print("Image {} not suitable for training: {}".format(img_path, "Didn't find a face" if len(face_bounding_boxes) < 1 else "Found more than one face"))
            else:
                # Add face encoding for current image to the training set
                X.append(face_recognition.face_encodings(image, known_face_locations=face_bounding_boxes)[0])
                y.append(class_dir)

    # Determine how many neighbors to use for weighting in the KNN classifier
    if n_neighbors is None:
        n_neighbors = int(round(math.sqrt(len(X))))
        if verbose:
            print("Chose n_neighbors automatically:", n_neighbors)

    # Create and train the KNN classifier
    knn_clf = neighbors.KNeighborsClassifier(n_neighbors=n_neighbors, algorithm=knn_algo, weights='distance')
    knn_clf.fit(X, y)

    # Save the trained KNN classifier
```

```
    if model_save_path is not None:
        with open(model_save_path, 'wb') as f:
            pickle.dump(knn_clf, f)

    return knn_clf


def predict(X_img_path, knn_clf=None, model_path=None, distance_threshol
d=0.6):
    """
    Recognizes faces in given image using a trained KNN classifier

    :param X_img_path: path to image to be recognized
    :param knn_clf: (optional) a knn classifier object. if not specifie
d, model_save_path must be specified.
    :param model_path: (optional) path to a pickled knn classifier. if n
ot specified, model_save_path must be knn_clf.
    :param distance_threshold: (optional) distance threshold for face cl
assification. the larger it is, the more chance
           of mis-classifying an unknown person as a known one.
    :return: a list of names and face locations for the recognized faces
in the image: [(name, bounding box), ...].
        For faces of unrecognized persons, the name 'unknown' will be re
turned.
    """
    if not os.path.isfile(X_img_path) or os.path.splitext(X_img_path)[1]
[1:] not in ALLOWED_EXTENSIONS:
        raise Exception("Invalid image path: {}".format(X_img_path))

    if knn_clf is None and model_path is None:
        raise Exception("Must supply knn classifier either thourgh knn_c
lf or model_path")

    # Load a trained KNN model (if one was passed in)
    if knn_clf is None:
        with open(model_path, 'rb') as f:
            knn_clf = pickle.load(f)

    # Load image file and find face locations
    X_img = face_recognition.load_image_file(X_img_path)
    X_face_locations = face_recognition.face_locations(X_img)

    # If no faces are found in the image, return an empty result.
    if len(X_face_locations) == 0:
        return []

    # Find encodings for faces in the test iamge
    faces_encodings = face_recognition.face_encodings(X_img, known_face_
locations=X_face_locations)

    # Use the KNN model to find the best matches for the test face
    closest_distances = knn_clf.kneighbors(faces_encodings, n_neighbors=
1)
    are_matches = [closest_distances[0][i][0] <= distance_threshold for
i in range(len(X_face_locations))]

    # Predict classes and remove classifications that aren't within the
```

```
 threshold
    return [(pred, loc) if rec else ("unknown", loc) for pred, loc, rec
in zip(knn_clf.predict(faces_encodings), X_face_locations, are_matches)]


def show_prediction_labels_on_image(img_path, predictions):
    """
    Shows the face recognition results visually.

    :param img_path: path to image to be recognized
    :param predictions: results of the predict function
    :return:
    """
    pil_image = Image.open(img_path).convert("RGB")
    draw = ImageDraw.Draw(pil_image)

    for name, (top, right, bottom, left) in predictions:
        # Draw a box around the face using the Pillow module
        draw.rectangle(((left, top), (right, bottom)), outline=(0, 0, 25
5))

        # There's a bug in Pillow where it blows up with non-UTF-8 text
        # when using the default bitmap font
        # use a bitmap font
     #   font = ImageFont.truetype('Pixel12x10.ttf', 40)

        font = ImageFont.truetype("/Library/Fonts/Tahoma Bold.ttf",48)
        name = name.encode("UTF-8")
        #print(name)
        # Draw a label with a name below the face
        name2= str(name)
        text_width, text_height = draw.textsize(name)
        draw.rectangle(((left, bottom - text_height - 10), (right, botto
m)), fill=(0, 0, 255), outline=(0, 0, 255))
       # draw.text((left + 6, bottom - text_height - 5), name,fill=(255,
255, 255, 255))
       # draw.text((left + 6, bottom - text_height - 5), name, font=fon
t)

        draw.text((left + 6, bottom - text_height - 5), name2, font=font
)


    # Remove the drawing library from memory as per the Pillow docs
    del draw

    # Display the resulting image
    #pil_image.show()
    plt.figure(figsize=(10,10))

    plt.imshow(pil_image)
    plt.show()

if __name__ == "__main__":
    # STEP 1: Train the KNN classifier and save it to disk
    # Once the model is trained and saved, you can skip this step next t
ime.
```

```
    print("Training KNN classifier...")
    classifier = train("knn_examples/train", model_save_path="trained_kn
n_model.clf", n_neighbors=2)
    print("Training complete!")

    # STEP 2: Using the trained classifier, make predictions for unknown
images
    for image_file in os.listdir("knn_examples/testing"):
        full_file_path = os.path.join("knn_examples/testing", image_file
)

        print("Looking for faces in {}".format(image_file))

        # Find all people in the image using a trained classifier model
        # Note: You can pass in either a classifier file name or a class
ifier model instance
        predictions = predict(full_file_path, model_path="trained_knn_mo
del.clf")

        # Print results on the console
        for name, (top, right, bottom, left) in predictions:
            print("- Found {} at ({}, {})".format(name, left, top))

        # Display results overlaid on an image
        show_prediction_labels_on_image(os.path.join("knn_examples/testi
ng", image_file), predictions)
```

Ages - can it predict one person throughout the years?

With and Without Glasses Does people wearing glasses or sunglasses

---

does changing the training data you give the model have an impact (age, glasses)how much can you throw in baby pictures of me? How much does it differ? `

---

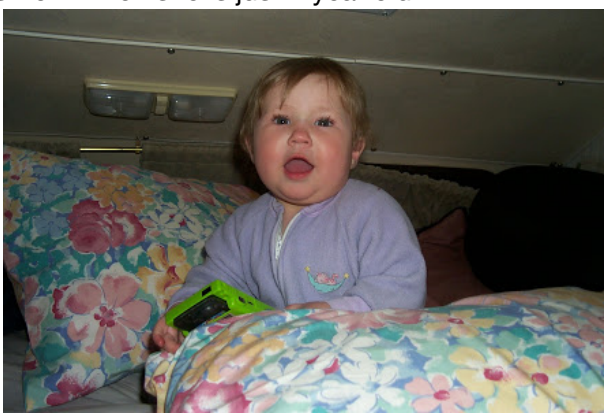
could it predict the Age of the person? make training folders of people at different ages

Introducing Age

What would happen if we introduce age into the alogrithm, would it be able to predict the correct person? Using my little sister as an example (with lots of digital photos available) We will look at the face recognition program traning on a current photo and including her photo at different ages.

The training photo that we used is her from last year:

And the oldest photo included is from when she is just 1 year old:



In [2]:
```
! face_recognition ./val_known ./val_unknown
```

```
./val_unknown/V_2012.jpg,val
./val_unknown/v_2006.JPG,val
./val_unknown/v_2004.JPG,val
./val_unknown/V_2011.jpg,val
./val_unknown/v_2001.JPG,val
./val_unknown/v_2003_1.JPG,val
./val_unknown/v_2004_1.JPG,unknown_person
./val_unknown/v_2004_1.JPG,val
./val_unknown/V-2018.jpg,val
./val_unknown/v_2004_2.JPG,val
./val_unknown/valerie.jpg,val
./val_unknown/random_3.jpeg,val
./val_unknown/v_009.JPG,val
./val_unknown/random_4.jpeg,unknown_person
./val_unknown/random_5.jpg,unknown_person
./val_unknown/V-2015-2.JPG,val
./val_unknown/v_2018.jpg,val
./val_unknown/v_2008.JPG,val
./val_unknown/random_2.jpg,unknown_person
./val_unknown/v_2009_1.JPG,val
./val_unknown/random_1.jpg,unknown_person
```

It was able to identify her if I used her most recent photo first, but what happens if I use her baby photo as the training image?

Training Image:

In [3]:
```
! face_recognition ./val_known_baby ./val_unknown_grownup
```

```
./val_unknown_grownup/V_2012.jpg,unknown_person
./val_unknown_grownup/v_2006.JPG,val
./val_unknown_grownup/v_2004.JPG,val
./val_unknown_grownup/V_2011.jpg,val
./val_unknown_grownup/v_2001.JPG,val
./val_unknown_grownup/v_2003_1.JPG,val
./val_unknown_grownup/v_2004_1.JPG,unknown_person
./val_unknown_grownup/v_2004_1.JPG,val
./val_unknown_grownup/V-2018.jpg,val
./val_unknown_grownup/v_2004_2.JPG,val
./val_unknown_grownup/valerie.jpg,val
./val_unknown_grownup/random_3.jpeg,val
./val_unknown_grownup/v_009.JPG,val
./val_unknown_grownup/random_4.jpeg,val
./val_unknown_grownup/random_5.jpg,unknown_person
./val_unknown_grownup/V-2015-2.JPG,unknown_person
./val_unknown_grownup/v_2018.jpg,unknown_person
./val_unknown_grownup/v_2008.JPG,unknown_person
./val_unknown_grownup/random_2.jpg,val
./val_unknown_grownup/v_2009_1.JPG,val
./val_unknown_grownup/random_1.jpg,unknown_person
```

It was still able to identify a lot of the images but struggled with more than when given the most recent image. Looks like this round it struggled with more recent pictures:

If you are going to do facial recongition with a variety of images that include different ages, the most succesful option is to use a more recent photo.