Peter Parianos
Dan Gluth
Final Project
Dr. Shirazi

Part 1: Installation of OpenCV onto my linux machine:

I started by installing the dependancies fo OpenCv. These are some of the encoding/decoding libraries I needed:

```
ppari@linuxcomp:~$ sudo apt-get install libavcodec-dev libavformat-dev libswscal
e-dev libv4l-dev
ppari@linuxcomp:~$ sudo apt-get install libjpeg8-dev libtiff5-dev libjasper-dev
libpng12-dev
```

After finishing the dependencies, I started to install OpenCV version 3.4.3 instead of the one listing within the tutorial. I also did the same thing for the conrib repo as well. The contrib repo is needed for the truly full installation of OpenCV 3.

```
ppari@linuxcomp:~$ wget -O opencv.zip https://github.com/Itseez/opencv/archive/3
.4.3.zip
--2018-11-10 17:05:17--  https://github.com/Itseez/opencv/archive/3.4.3.zip
Resolving github.com (github.com)... 192.30.253.113, 192.30.253.112
Connecting to github.com (github.com)|192.30.253.113|:443... connected.
HTTP request sent, awaiting response... 302 Found
Location: https://codeload.github.com/Itseez/opencv/zip/3.4.3 [following]
--2018-11-10 17:05:21--  https://codeload.github.com/Itseez/opencv/zip/3.4.3
Resolving codeload.github.com (codeload.github.com)... 192.30.253.120, 192.30.25
3.121
Connecting to codeload.github.com (codeload.github.com)|192.30.253.120|:443... c
onnected.
HTTP request sent, awaiting response... 200 OK
Length: unspecified [application/zip]
Saving to: 'opencv.zip'

opencv.zip              [               <=>  ]  74.83M  1.55MB/s
```

Next I used pip to install a virtualenvwrapper librbary for python3.6. I needed to change the .bashrc file. The error on the next page was fixed by installing pip through this way, and then reistalling virtualenv.

Ubuntu 16.04: How to install OpenCV Shell

```
1 $ cd ~
2 $ wget https://bootstrap.pypa.io/get-pip.py
3 $ sudo python get-pip.py
```

```
ppari@linuxcomp: ~
ges (16.1.0)
Requirement already satisfied: virtualenvwrapper in /usr/local/lib/python3.6/sit
e-packages (4.8.2)
Requirement already satisfied: stevedore in /usr/local/lib/python3.6/site-packag
es (from virtualenvwrapper) (1.30.0)
Requirement already satisfied: virtualenv-clone in /usr/local/lib/python3.6/site
-packages (from virtualenvwrapper) (0.4.0)
Requirement already satisfied: pbr!=2.1.0,>=2.0.0 in /usr/local/lib/python3.6/si
te-packages (from stevedore->virtualenvwrapper) (5.1.1)
Requirement already satisfied: six>=1.10.0 in /usr/local/lib/python3.6/site-pack
ages (from stevedore->virtualenvwrapper) (1.11.0)
You are using pip version 10.0.1, however version 18.1 is available.
You should consider upgrading via the 'pip install --upgrade pip' command.
ppari@linuxcomp:~$ vim .bashrc
ppari@linuxcomp:~$ source .bashrc
/usr/bin/python: No module named virtualenvwrapper
virtualenvwrapper.sh: There was a problem running the initialization hooks.

If Python could not import the module virtualenvwrapper.hook_loader,
check that virtualenvwrapper has been installed for
VIRTUALENVWRAPPER_PYTHON=/usr/bin/python and that PATH is
set properly.
ppari@linuxcomp:~$ vim /usr/local/bin/virtualenvwrapper.sh
ppari@linuxcomp:~$
```

Some other errors: I needed to use python3.6.5 instead of 3.5

```
--    Python 3:
--      Interpreter:                 /home/ppari/.virtualenvs/cv/bin/pyt
hon3 (ver 3.6.5)
--      Libraries:                   /usr/local/lib/libpython3.6m.a (ver
 3.6.5)
--      numpy:                       /home/ppari/.virtualenvs/cv/lib/pyt
hon3.6/site-packages/numpy/core/include (ver 1.15.4)
--      packages path:               lib/python3.6/site-packages
--
--    Python (for build):            /home/ppari/.virtualenvs/cv/bin/pyt
hon3
--
```

```
[100%] Linking CXX executable ../../bin/example_tutorial_Laplace_Demo
[100%] Linking CXX executable ../../bin/example_tutorial_moments_demo
[100%] Built target example_tutorial_Laplace_Demo
Scanning dependencies of target example_tutorial_planar_tracking
[100%] Building CXX object samples/cpp/CMakeFiles/example_tutorial_plan
ar_tracking.dir/tutorial_code/features2D/AKAZE_tracking/planar_tracking
.cpp.o
[100%] Built target example_tutorial_moments_demo
[100%] Linking CXX executable ../../bin/example_cpp_grabcut
[100%] Built target example_cpp_grabcut
[100%] Linking CXX executable ../../bin/example_tutorial_planar_trackin
g
[100%] Built target example_tutorial_planar_tracking
```

This is an example of it working in a virtual environment

```
(cv) ppari@linuxcomp:~/.virtualenvs/cv/lib/python3.6/site-packages$ pyt
hon
Python 3.6.5 (default, May 14 2018, 12:10:29)
[GCC 5.4.0 20160609] on linux
Type "help", "copyright", "credits" or "license" for more information.
>>> import cv2
>>> cv2.__version__
'3.4.3'
>>>
```

Part 2:

For this part of the project, we used `sklearn, matplot`, and `numpy` with a combination of different other libraries to display the confusion matrix of the trained and untrained models. The first step to this process is to find a dataset to work with. We used the given resources provided for this project in order to choose a test data set from sklearn's website. The chosen dataset was a wine data set with attributes: Alcohol, Malic acid, Ash, Alcalinity of ash, Magnesium, Total phenols, Flavanoids, Nonflavanoid phenols, Proanthocyanins, Color intensity, Hue, OD280/OD315 of diluted wines, and Proline levels. There are also three unnamed classes associated with each wine: class_0, class_1, class_2. The total number of wine instances is 178. We downloaded the raw data in .csv form.

The two different classifiers that were chosen are Naïve Bayes and Decision Tree. The version of Naïve Bayes was Gaussian, which was given as an example from one of our Homeworks. The naïve bayes classifier uses the the Bayesian Rule to determine how to classify the unknown instance. In its learning phase it will create separate tables for each of the given attributes, that determine the probabilities of which class given that attribute. A decision is made using the MAP rule which will find the largest probabilities given the unknown input. We then assign the label with the largest probability. We are using the Gaussian version because it is better suited for continuous-valued features.
The other chosen classifier is called Decision Tree. This is simple compared to Naïve Bayes classifier. This classifier automatically discretizes any continuously-valued attributes in order to make a more clear decision when traversing through each of the attributes. It will start to group each of the different attributes by each of the possible outcomes. Since all of our attributes are continuous, the classifier likely grouped them within intervals. After looking at some or all of the attributes, a decision will be made by the majority votes farthest down the tree. We can also stop the tree at a certain depth.

Description of the Code:

To read the raw data from the csv and properly format it, we needed to use numpy loadtxt to turn the .csv file into a 2-D numpy array. This array was then split into two groups: X which is the attributes for each of the wine instances, and y which is the associated classes for each of the instances. This split is required as parameters when fitting the classifier. A prediction is made using the given input data. Then the accuracy is calculated to determine how well the data is fit on the given data. We then split the data into a training set and a test set which the model will not see. The split is 25% and 75% as per directions given. A new instance of our classifier is created in order to fit the training data. We then try to predict the class in which the testing data belongs to.
Different values are calculated in order to show use the efficieny of the classifer on the data.

$$\text{Precision} = \frac{True\ Positive}{True\ Positive + False\ Positive}$$

- Precision:

$$\text{Recall} = \frac{True\ Positive}{True\ Positive + False\ Negative}$$

- Recall: $$= \frac{True\ Positive}{Total\ Actual\ Positive}$$

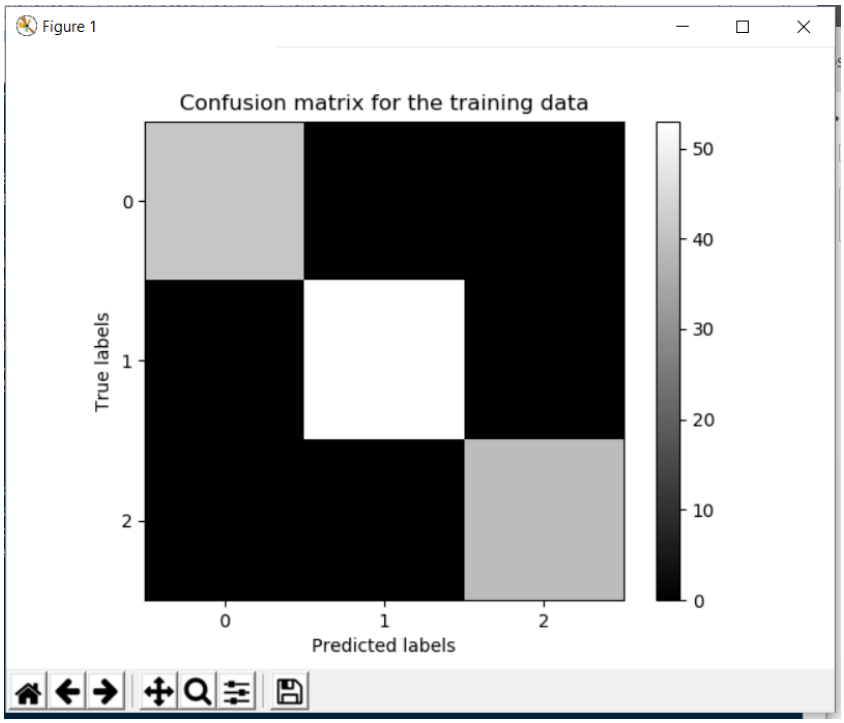
$$F1 = 2 \times \frac{Precision * Recall}{Precision + Recall}$$

- F1 Score:
- Accuracy: The equation is given in the source code.
- A true positive is when the samples for which we predicted 1 as the output and the ground truth is 1 too (or the same positive prediction).
- A true negative is where we predicted 0 as the output and the ground truth is 0 too.
- A false positive are where we predicted true as the output but the ground truth is actually false.
- A false negative is when we predicted false as the output but the ground truth is actually true.
- A visual represtaion of this is called the Confusion Matrix.

OUTPUT:
Naïve Bayes:

```
[1. 1. 1. 2. 3. 1. 1. 2. 2. 1. 2. 1. 2. 2. 1. 1. 2. 1. 3. 3. 1. 3. 1. 3.
 2. 2. 1. 1. 2. 2. 2. 1. 3. 2. 2. 3. 1. 2. 1. 3. 1. 3. 2. 2. 2.]
[1. 1. 1. 2. 3. 1. 1. 2. 2. 1. 2. 1. 2. 2. 1. 1. 2. 1. 3. 3. 1. 3. 2. 3.
 2. 2. 1. 1. 2. 3. 2. 1. 3. 2. 2. 3. 1. 2. 1. 3. 1. 3. 2. 2. 2.]
Accuracy of the new classifier = 95.56 %
Accuracy: 96.07%
Precision: 96.33%
Recall: 96.07%
F1: 96.05%
```

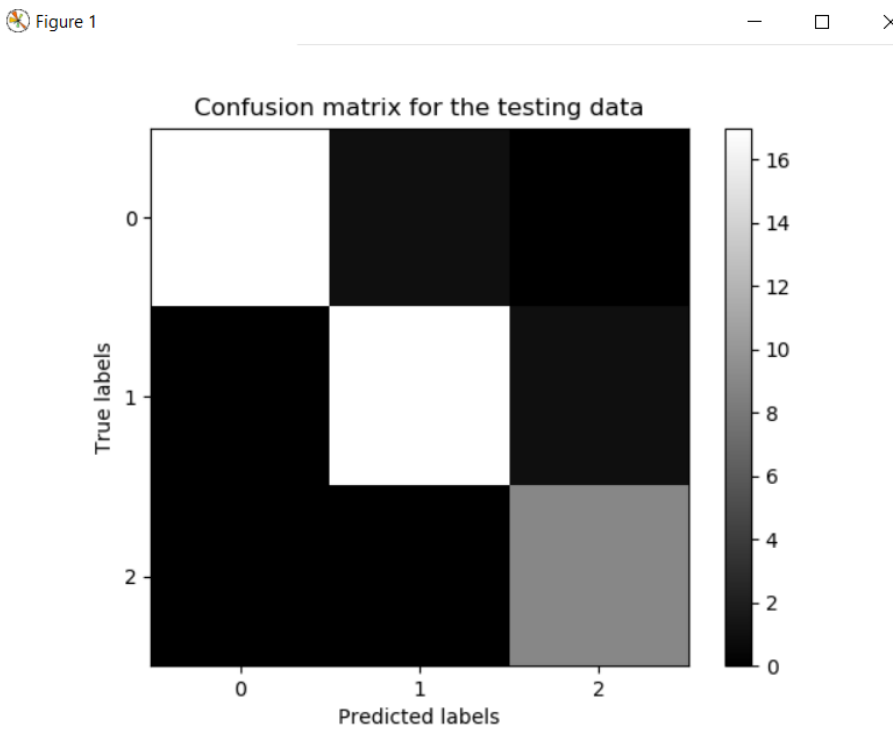
This is the beginning of the output for the naïve Bayes classifier output. Each of these values is from using the dataset for wine classification. There are three classes of wine to be chosen. The lists at the beginning of the output are the actual labels and the tested labels.



This is the confusion matrix for the training data from the wine dataset.

```
########################################

 Classifier performance on training dataset

             precision    recall  f1-score   support

    Class-0       1.00      1.00      1.00        41
    Class-1       1.00      1.00      1.00        53
    Class-2       1.00      1.00      1.00        39

avg / total       1.00      1.00      1.00       133

########################################
```

These are some other performance measures that are used to measure classifier performance on the training data. The prescision should be 100%.



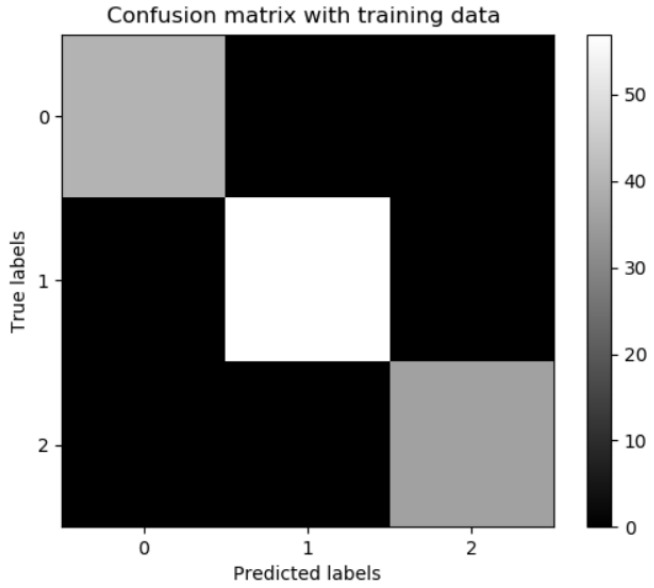
This is the confusion matrix on the testing data.

```
########################################

 Classifier performance on_test dataset

             precision    recall  f1-score   support

    Class-0       1.00      0.94      0.97        18
    Class-1       0.94      0.94      0.94        18
    Class-2       0.90      1.00      0.95         9

avg / total       0.96      0.96      0.96        45

########################################
```

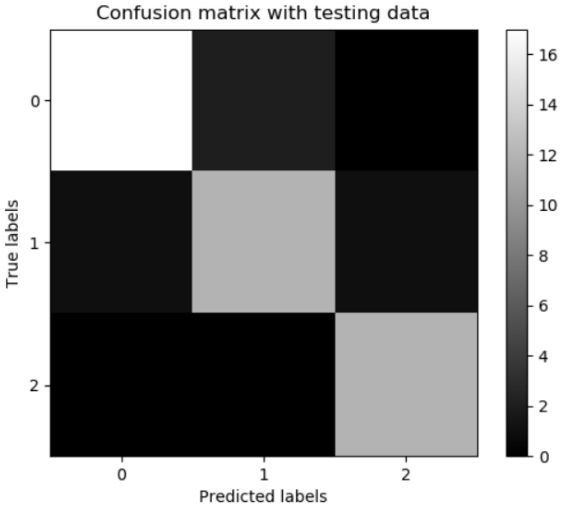
The classifier performance measures on the test data.

Decision Tree:

The second classifier I used was the Decision tree with depth of 4. We measure the same values and output almost the same information:



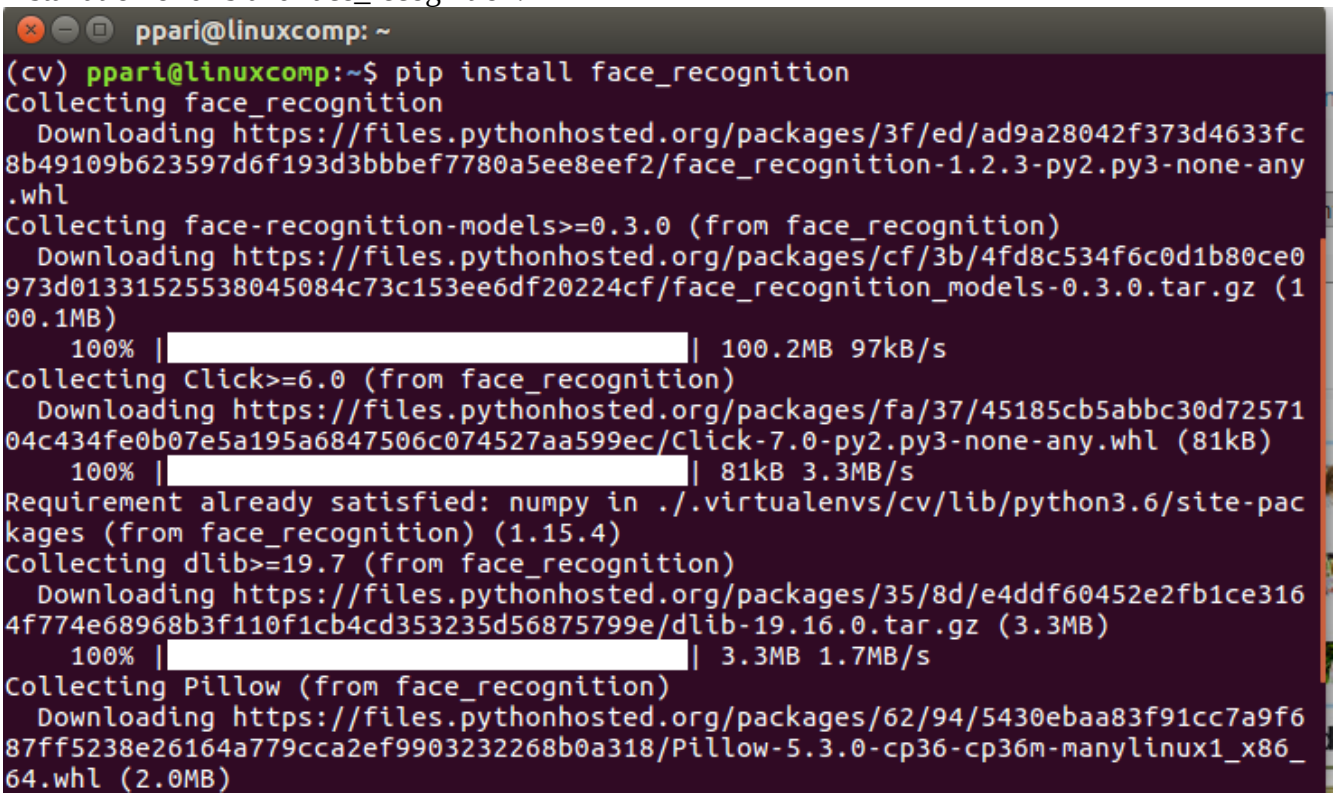

```
########################################

 Classifier performance on training dataset

             precision    recall  f1-score   support

    Class-0       1.00      1.00      1.00        40
    Class-1       1.00      1.00      1.00        57
    Class-2       1.00      1.00      1.00        36

avg / total       1.00      1.00      1.00       133

########################################
```



```
########################################

 Classifier performance on_test dataset

             precision    recall  f1-score   support

    Class-0       0.94      0.89      0.92        19
    Class-1       0.86      0.86      0.86        14
    Class-2       0.92      1.00      0.96        12

avg / total       0.91      0.91      0.91        45

########################################
```

Part 3: Face Recognition

Installation of dlib and face_recognition:



After all the required dependencies were downloaded, I needed to quickly create a dataset of images using Bing's API.
1. Sign Up for Microsoft Azure Account in order to Use Bing's Search API for a 7-day trial.

The requests package makes it super easy for us to make HTTP requests and not get bogged down in fighting with Python to gracefully handle requests.

The creation of search_bing.py:

```
#import the ncessary packages
from requests import exceptions
import argparse
import requests
import cv2
import os

#contruct the argument parser and parse the arguments
ap = argparse.ArgumentParser()
ap.add_argument("-q", "--query", required=True,
        help="search query to search Bing Image API for")
ap.add_argument("-o", "--output", required=True,
        help="path to output directory of images")
args = vars(ap.parse_args())
```

Next, we parse two command line arguments:

- --query: The image search query you're using, which could be anything such as *"pikachu"*, *"santa"* or *"jurassic park"*.
- --output: The output directory for your images. My personal preference (for the sake of organization and sanity) is to separate your images into *separate class subdirectories*, so be sure to specify the correct folder that you'd like your images to go into (shown below in the **"Downloading images for training a deep neural network"** section).

On **Lines 36-38**, we initialize the search parameters. Be sure to review the API documentation as needed.

From there, we perform the search (**Lines 42-43**) and grab the results in JSON format (**Line 47**).

We calculate and print the estimated number of results to the terminal next (**Lines 48-50**).

We'll be keeping a counter of the images downloaded as we go, so I initialize total on **Line 53**.

Here we are looping over the estimated number of results in GROUP_SIZE batches as that is what the API allows (**Line 56**).

Make the search and getting the results:
There are certain required parameters that are needed by the Bing Search API. For the headers, the API key that was generated by Microsoft is needed in the headers, which is a dictionary where the required names of parameters are keys. This is because it is paresed using the requests library which uses **params. Some other parameters is the search term, the offset (number of images to skip), and the total number of images to return.

The returned results is a large data file that is then parsed as a json file. This make it easier for Python to read because the JSON format is identical to the dictionary data structure. We can now access some data by using the name of the data as the key.

```
#make the search
print("[INFO] searching Bing API for '{}'".format(term))
search = requests.get(URL, headers=headers, params=params)
search.raise_for_status()

# grab the results from the search, including the total number of
# estimated results returned by the Bing API
results = search.json()
estNumResults = min(results["totalEstimatedMatches"], MAX_RESULTS)
print("[INFO] {} total results for '{}'".format(estNumResults,
        term))
```

Iterating through the results takes a for loop and is very intensive.

Using the premade dataset: Jurrasic Park

1. Encode the dataset to a 128-d real-valued number feature vector per face. This is not training data since the network has already been trained so that we can use it to construct 128-d embeddings for each of the 218 faces in our dataset.

2. After we encode the data, we dump to a file called encodings. Pickle.

3. This data will be used for classification, more specifically k-NN model + votes. This was studied in class. We are using a pretrained model.

4. We use encode_faces.py to construct face embeddings

5. Install facial recognition libraries to ensure you have everything you need ready.

6. Define arguments:

- Dataset: the path we created with search_bing_api.py
- Encodings: face encoding are written to both file and argument
- Detection-method: we must detect face images before we can encode them, the two face detection methods include hog and cnn, these are the only ones that will work for Detection-method

7. Initialize two lists: knowEncodings and knownNames, they will contain the face encodings and corresponding names for each person in the dataset

8. The loop will cycle 218 times (because 218 face images are in the dataset)

9. The name of the person will be extracted from each person from imagePath

10. The image is loaded while passing the imagePath to cv2.

11. Color soaces on line 37 must be swapped naming the new image rgb.

12. Lines 41 and 42 find the faces, and places boxes over them, then rgb and model(cnn or hog) are parameters that are passed to face_recognition.face_locations

13. The bounding boaxes of Ellie Sattlers face into a list of 128 numbers on line 45. The face is encoded into a vector by the method face_recognition.face_encodings.

14. Ellie Sattlers name and encoding are appended to the appropriate list.

15. This will continue for all 218 images in the dataset.

16. Line 56 constructs a dictionary with two keys encoding and names.

17. Lines 57-59 dumps the names and encodings to disk.

18. To create our facial embeddings we open the terminal and type the command:

$ python encode_faces.py –dataset dataset –encodings encodings.pickle

19. encoding.pickle is a file that is created and it contains the 128-d face embeddings for each face in our dataset

Recognizing faces in Images:

1. Open recognize_faces_images.py and insert the following code: import face_recognition, arparse, pickle, and cv2. Construct the argument parser and parse arguments with and parse the arguments:

```
ap = argparse.ArgumentParser()
ap.add_argument("-e", "--encodings", required=True,
    help="path to serialized db of facial encodings")
ap.add_argument("-i", "--image", required=True,
    help="path to input image")
ap.add_argument("-d", "--detection-method", type=str, default="cnn",
    help="face detection model to use: either `hog` or `cnn`")
args = vars(ap.parse_args())
```

Encodings is the path to the pickle file containing our face encodings. Image is the image that is undergoing facial recognition. Detection Method

2. Then we load the pre-calculated face names and then construct the 128-d face encoding for the input image: Line 19 loads our pickled encodings and face names from the disk.

3.We then load and covert the image to rgb. Then the we proceed to detect all faces in the input image and compute their 128-d encodings.

4. Facial encodings are looped until it is looped over the face encodings computer from our input image.

5. Each face is then matched to each input image to our known encodings dataset using face_recognition.compare_faces.

6.A true and false list is returned, one for each image in out dataset (218). compare_faces function is computing the Euclidean distance between the candidate embedding and all the faces in the dataset.

7.With the matches list, we can compute the number of times TRUE was associated with each name, the totals are added up and the se

```
if True in matches:
    # find the indexes of all matched faces then initialize a
    # dictionary to count the total number of times each face
    # was matched
    matchedIdxs = [i for (i, b) in enumerate(matches) if b]
    counts = {}

    # loop over the matched indexes and maintain a count for
    # each recognized face face
    for i in matchedIdxs:
        name = data["names"][i]
        counts[name] = counts.get(name, 0) + 1

    # determine the recognized face with the largest number of
    # votes (note: in the event of an unlikely tie Python will
    # select first entry in the dictionary)
    name = max(counts, key=counts.get)

# update the list of names
names.append(name)
```

8. The index of each TRUE vote from matches is constructed as a list called matchIdxs
9. We initialize a dictionary called counts which holds the characher name as a key and the number of votes as value.
10. Then matchedIdxs is looped over and set with each name while incrementing tit as necessary in counts.
11. name = max(coutns, key =counts.get) extracts the name with the most votes from counts
12. for ((top, right, bottom, left), name) in zip(boxes, names): will begin looping over the detected face bounding boxes and predicted names. To create an iterable object so we can easily loop, we call zip(boxes, names) resulting in truples that we can extract the box coordinates and name from.

13. cv2.rectangle(image, (left, top), (right, bottom), (0,255,0),2) creates a green box. These coordinates are used to calculate where the text of the persons name should go. Cv2.putText(image, name, (left, y), cv2.FONT_HERSHEY_SIMPLEX, actually places the text on the image.

14. Run the script while providing the two command line arguments at a minimum

Recognizing faces in Videos

```
# import the necessary packages
from imutils.video import VideoStream
import face_recognition
import argparse
import imutils
import pickle
import time
import cv2

# construct the argument parser and parse the arguments
ap = argparse.ArgumentParser()
ap.add_argument("-e", "--encodings", required=True,
    help="path to serialized db of facial encodings")
ap.add_argument("-o", "--output", type=str,
    help="path to output video")
ap.add_argument("-y", "--display", type=int, default=1,
    help="whether or not to display output frame to screen")
ap.add_argument("-d", "--detection-method", type=str, default="cnn",
    help="face detection model to use: either `hog` or `cnn`")
args = vars(ap.parse_args())
```

1. We import packages and then proceed to parse our command line arguments.
2. We have four command line arguments encodings, detection-method, output, display.
3. Load our encodings and start our VideoStream :
4. Face recognition with OpenCV, Python, and deep learning
5. To access the camera we will use VideoStream class from imutils.
   vs = VideoStream(src=0).start() starts the stream.
6. Start a while loop and begin to grab and process frames:
7. Face recognition with OpenCV, Python, and deep learning

8. The while loop begins and the first step we take is to grab a frame from the video stream .
9. We then read the frame , preprocess, and then detect face bounding boxes + calculate encodings for each bounding box.
10. Then loop over the facial encodings associated with the faces we have just found:
Face recognition with OpenCV, Python, and deep learning
11. Loop over each of the encodings and try to match the face.
    - If there are matches found, we count the votes for each name in the dataset.
    - We then extract the highest vote count and that is the name associated with the face.
12. The loop works over the recognized faces and proceed to draw a box around the face.

13. We're going to write the frame to disk:

```
# if the video writer is None *AND* we are supposed to write
# the output video to disk initialize the writer
if writer is None and args["output"] is not None:
        fourcc = cv2.VideoWriter_fourcc(*"MJPG")
        writer = cv2.VideoWriter(args["output"], fourcc, 20,
```

```
                (frame.shape[1], frame.shape[0]), True)

    # if the writer is not None, write the frame with recognized
    # faces to disk
    if writer is not None:
        writer.write(frame)
```

13. Initialize a video writer:

VideoWriter_fourcc

14. Pass that object into the VideoWriter along with our output file path, frames per second target, and frame dimensions.

15. If the writer exists, we can go ahead and write a frame to disk

```
    # check to see if we are supposed to display the output frame to
    # the screen
    if args["display"] > 0:
        cv2.imshow("Frame", frame)
        key = cv2.waitKey(1) & 0xFF

        # if the `q` key was pressed, break from the loop
        if key == ord("q"):
            break
```

15. Then we display the frame and check if the quit key ("q") if it has indeeed been pressed we break out of the loop .

16. Clean up and release the display, video steam, and writer.

```
cv2.destroyAllWindows()
vs.stop()
```

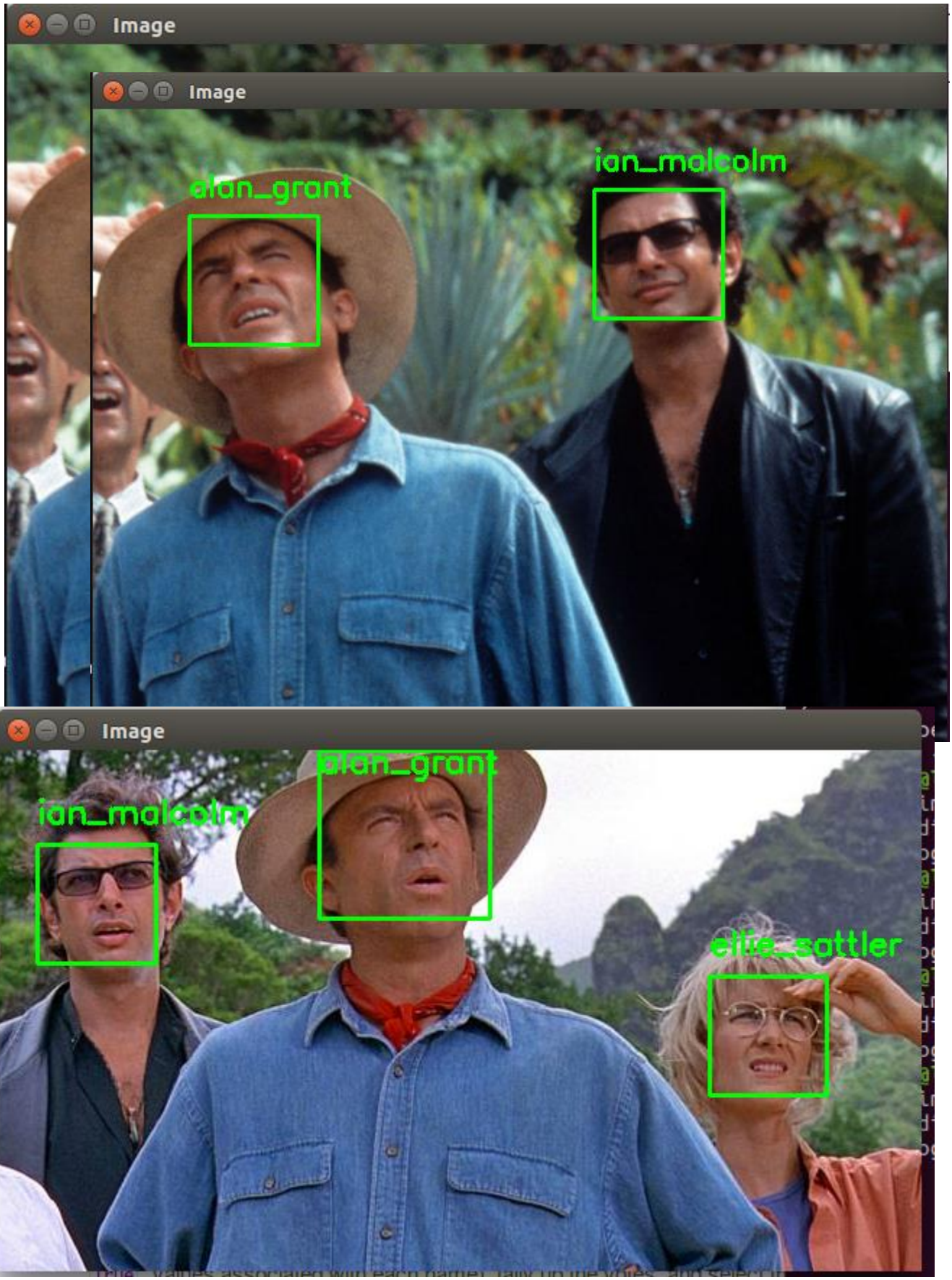


Image
Image
alan_grant
ian_malcolm
Image
ian_malcolm
alan_grant
ellie_sattler