`Basic NLP and Swift Feature Extraction.`

`Name : Neeraj Padarthi`

`Class ID: 20`

`LAB: 1`

Objective

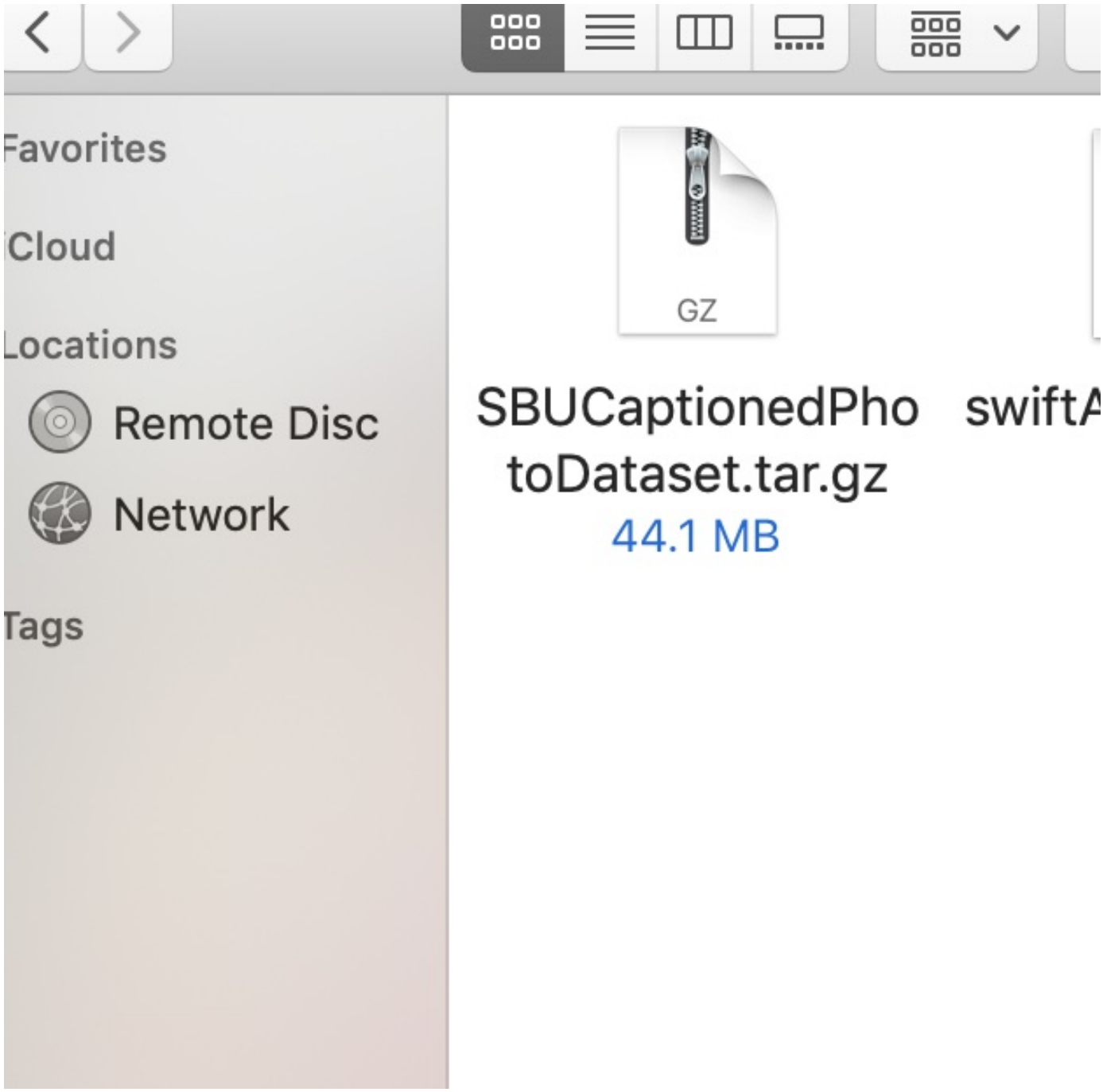
- Download the Dataset as listed in Project Dataset as provided in Project Sheet(SBU Data Set)
- Perform Basic NLP (Tokenization, Lemmatization) on top of image captions
- Report the Image Statistics based on each word after the processing
- Sample images based on words in the project theme and perform SIFT Feature Extraction.

Downloading the Data Set

- Downloaded the SBU DataSet.
- Merging the two Input files into Pandas Data Frame columns(Captions Name, Image Links)
- Each Column data is respective to the 2 text files from the SBU

Dataset.

- Performed using strip at new line and append to a list.
- Pandas Data Frame is created on the top the two lists respectively.



In [2]:
```
import pandas as pd
import nltk
import numpy as np
import cv2
import os
import matplotlib.pyplot as plot
nltk.download('punkt')
nltk.download('wordnet')
from matplotlib.pylab import rcParams
rcParams['figure.figsize'] =15,9
```

In [3]:
```
colA = []
colB = []

with open('dataset/SBU_captioned_photo_dataset_captions.txt') as f:
    for line in f:
        inner_list = [line.strip() for line in line.split('/n')]
        colA.append(inner_list)

with open('dataset/SBU_captioned_photo_dataset_urls.txt') as f:
    for line in f:
        inner_list = [line.strip() for line in line.split('/n')]
        colB.append(inner_list)
```

Tokenization and Lemmatization are performed on the image captions

- Now the DataFrame is created with columns respective to the Caption and Link
- Created a new Column called tokenized_sents, the column contains the tokenized words of the image captions
- Once the tokenized words are created, then a new column is created on it called text_lemmatized where it performers the basic Lemmatize
- Tokenization and Lemmatization is performed on the row level using Lambda Functions
- Created Function called lemmatize_text which performs the Lemmatization of the words
- Each operation has its own column, which we can be referred at

each iteration

In [6]:
```
newDF['tokenized_sents'] = newDF.apply(lambda row: nltk.word_tokenize(row['caption']),axis=1)
```

In [7]:
```
lemmatizer = nltk.stem.WordNetLemmatizer()
def lemmatize_text(text):
    return [lemmatizer.lemmatize(w) for w in nltk.tokenize.WhitespaceTokenizer().tokenize(text)]

newDF['text_lemmatized'] = newDF.caption.apply(lemmatize_text)
```

Keywords Search

- Search is performed to get the corresponding Image data.
- Logic implemented was created a DF with WRT keywords and finally all the Search DF's are merged into a new DF
- Below are the search words.
    - Stop, Road, Parking, Speed, Turn, Horn, Exit, Sign, Signal, Limit, Mile, Route
- Removed the Duplicate images as we will have the repetitive images for few of the searched Keywords.
- Search operation is performed on the column Text_Lemmatization using Lambda functions at the row level.
- Finally Concat is performed on the respective searched DF's to form a new DF called Final_DF

In [11]:
```
stopmask = newDF.text_lemmatized.apply(lambda x: ("STOP") in x)
stopdf =newDF.loc[stopmask]
roadmask = newDF.text_lemmatized.apply(lambda x: ("ROAD") in x)
roaddf =newDF.loc[roadmask]
parkingmask = newDF.text_lemmatized.apply(lambda x: ("PARKING") in x)
parkingdf =newDF.loc[parkingmask]
speedmask = newDF.text_lemmatized.apply(lambda x: ("SPEED") in x)
speeddf =newDF.loc[speedmask]
turnmask = newDF.text_lemmatized.apply(lambda x: ("TURN") in x)
turndf =newDF.loc[turnmask]
hornmask = newDF.text_lemmatized.apply(lambda x: ("HORN") in x)
horndf =newDF.loc[hornmask]
exitmask = newDF.text_lemmatized.apply(lambda x: ("EXIT") in x)
exitdf =newDF.loc[exitmask]
signmask = newDF.text_lemmatized.apply(lambda x: ("SIGN") in x)
signdf =newDF.loc[signmask]
signalmask = newDF.text_lemmatized.apply(lambda x: ("SIGNAL") in x)
signaldf =newDF.loc[signalmask]
limitmask = newDF.text_lemmatized.apply(lambda x: ("LIMIT") in x)
limitdf =newDF.loc[limitmask]
milemask = newDF.text_lemmatized.apply(lambda x: ("MILE") in x)
miledf =newDF.loc[milemask]
routemask = newDF.text_lemmatized.apply(lambda x: ("ROUTE") in x)
routedf =newDF.loc[routemask]
```

In [16]:
```
finalDF.head(5)
```

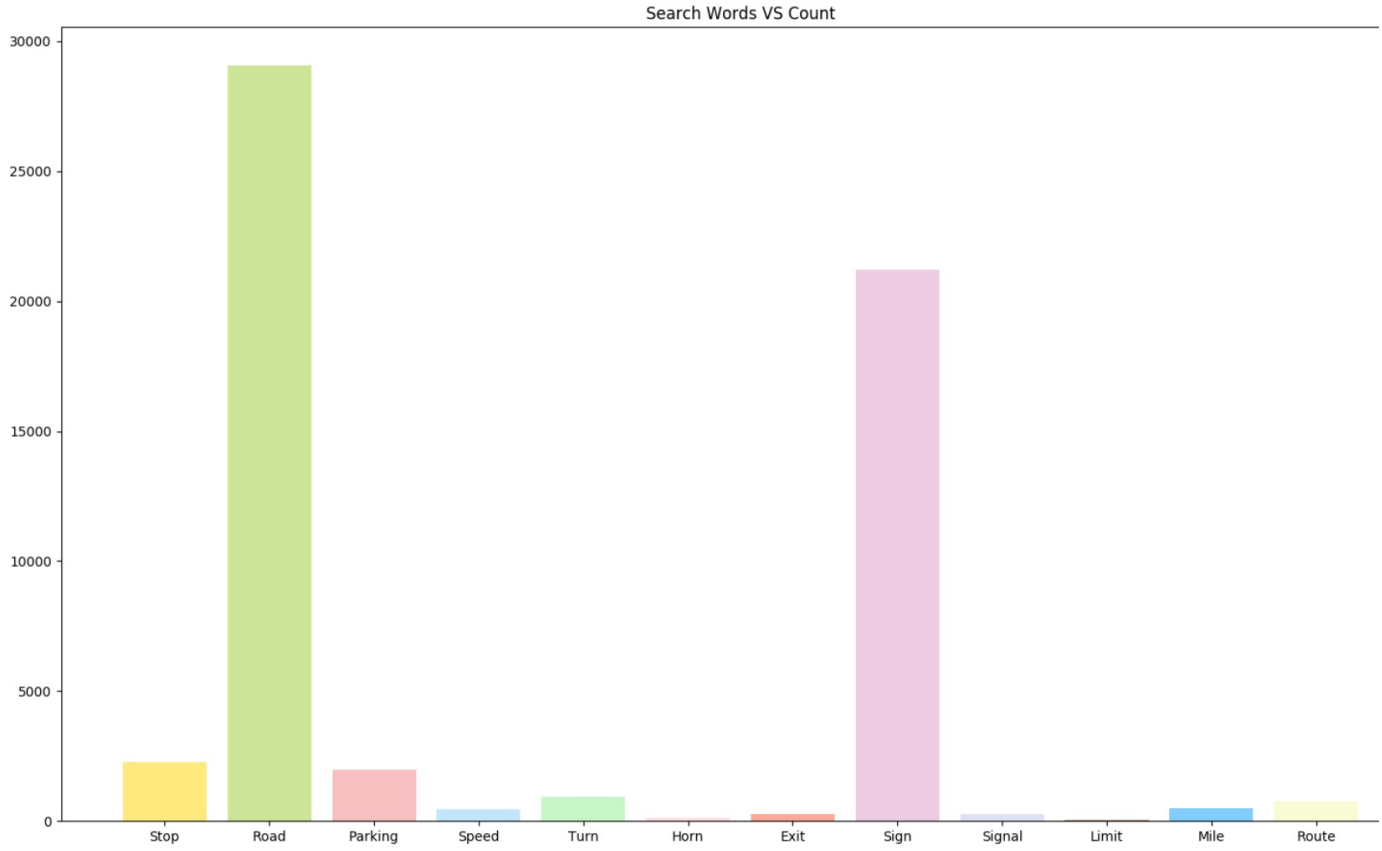
Out[16]:

	caption	link	tokenized_sents	text_lemmatized
0	MY NEXT DESTINATION, THIS IS AT THE BUS STOP N...	http://static.flickr.com/2006/2079839062_f7427...	['MY', 'NEXT', 'DESTINATION', ',', 'THIS', 'IS...	['MY', 'NEXT', 'DESTINATION,', 'THIS', 'IS', '...
1	JUST A RANDOM LITTLE SIMPSONS-ESQUE DOODLE IN ...	http://static.flickr.com/1009/942821140_c2a5b2...	['JUST', 'A', 'RANDOM', 'LITTLE', 'SIMPSONS-ES...	['JUST', 'A', 'RANDOM', 'LITTLE', 'SIMPSONS-ES...
2	SEEN ON AN EXPRESS BOX NEAR A BUS STOP ALONG 1...	http://static.flickr.com/1313/582469625_a6c5dc...	['SEEN', 'ON', 'AN', 'EXPRESS', 'BOX', 'NEAR',...	['SEEN', 'ON', 'AN', 'EXPRESS', 'BOX', 'NEAR',...
3	AT THE BUS STOP NEAR MY HOUSE	http://static.flickr.com/1292/1200574740_2fdd8...	['AT', 'THE', 'BUS', 'STOP', 'NEAR', 'MY', 'HO...	['AT', 'THE', 'BUS', 'STOP', 'NEAR', 'MY', 'HO...
6	HIBOX FROM FROM ROCK AREA LUNCH STOP NEAR HIGH...	http://static.flickr.com/3036/2835017556_6f148...	['HIBOX', 'FROM', 'FROM', 'ROCK', 'AREA', 'LUN...	['HIBOX', 'FROM', 'FROM', 'ROCK', 'AREA', 'LUN...

Image Statistics Code and Visualizations of the Search Words (Bar Chart & Pie Chart)

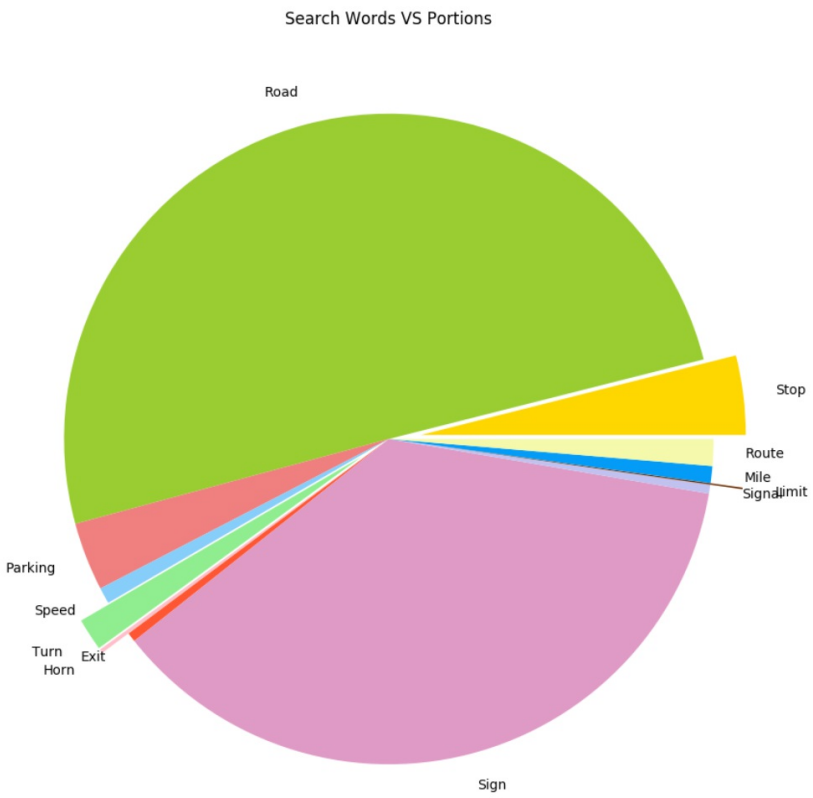
- matplotlib is used for visualization of the code.
- Each graph says the count of the searched words respectively.

In [31]:
```
performance = [stopdf['caption'].count(),\
               roaddf['caption'].count(),\
               parkingdf['caption'].count(),\
               speeddf['caption'].count(),\
               turndf['caption'].count(),\
               horndf['caption'].count(),\
               exitdf['caption'].count(),\
                  signdf['caption'].count(),\
               signaldf['caption'].count(),\
               limitdf['caption'].count(),\
               miledf['caption'].count(),\
               routedf['caption'].count()]
colors_list = ['gold', 'yellowgreen', 'lightcoral', 'lightskyblue', 'lightgreen', 'pink',\
               '#FF5733','#DF9AC6', '#c2c2f0','#6E2C00', '#059CF8','#F4F9AB']
y_pos = np.arange(12)
plot.bar(y_pos, performance, align='center', alpha=0.5,color=colors_list)
plot.xticks(y_pos, ['Stop','Road','Parking','Speed','Turn','Horn','Exit','Sign','Signal','Limit','Mile','Route'])
plot.title("Search Words VS Count")
plot.show()
```



In [32]:
```
colors_list = ['gold', 'yellowgreen', 'lightcoral', 'lightskyblue', 'lightgreen', 'pink',\
               '#FF5733','#DF9AC6', '#c2c2f0','#6E2C00', '#059CF8','#F4F9AB']
explode_list = [0.1, 0, 0, 0, 0.1, 0.1,0, 0, 0, 0.1,0, 0]
plot.pie([float(v) for v in performance], labels=['Stop','Road','Parking','Speed',\
                                                  'Turn','Horn','Exit','Sign','Signal',\
                                                  'Limit','Mile','Route'],pctdistance=1.12,
         colors=colors_list,  # add custom colors
                            explode=explode_list ,
           autopct=None)
plot.title('Search Words VS Portions')
plot.show()
```

In [ ]:



Performing Swift Feature Extraction

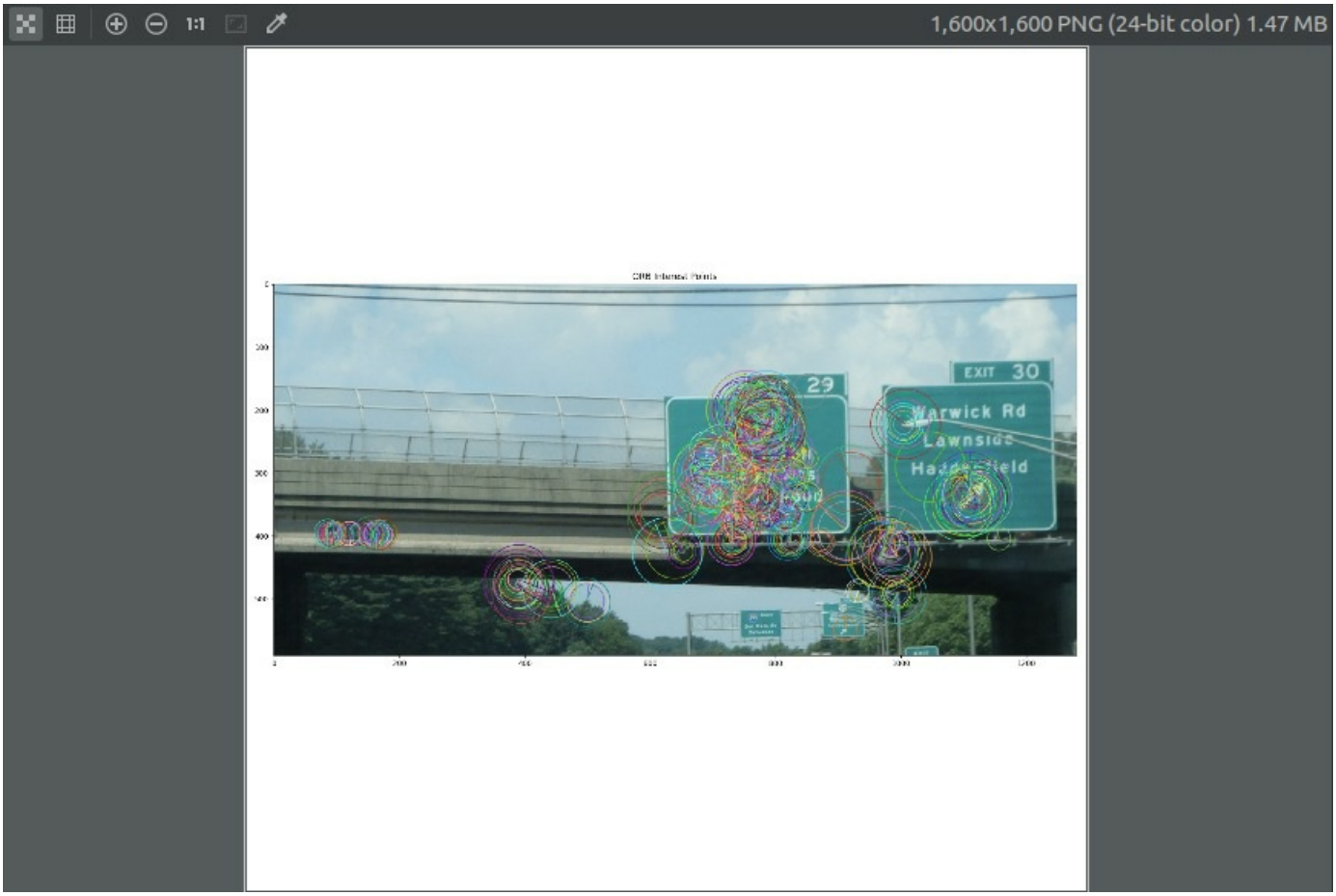
- It is mainly used for Image feature extractions.
- Modified the code to take the traffic-related images to perform the Swift Extraction.
- Below are the Referred image screenshot, code screenshots, and the respective output screenshots.
- We can see that Road and Sign has the major contribution of images

Image Used

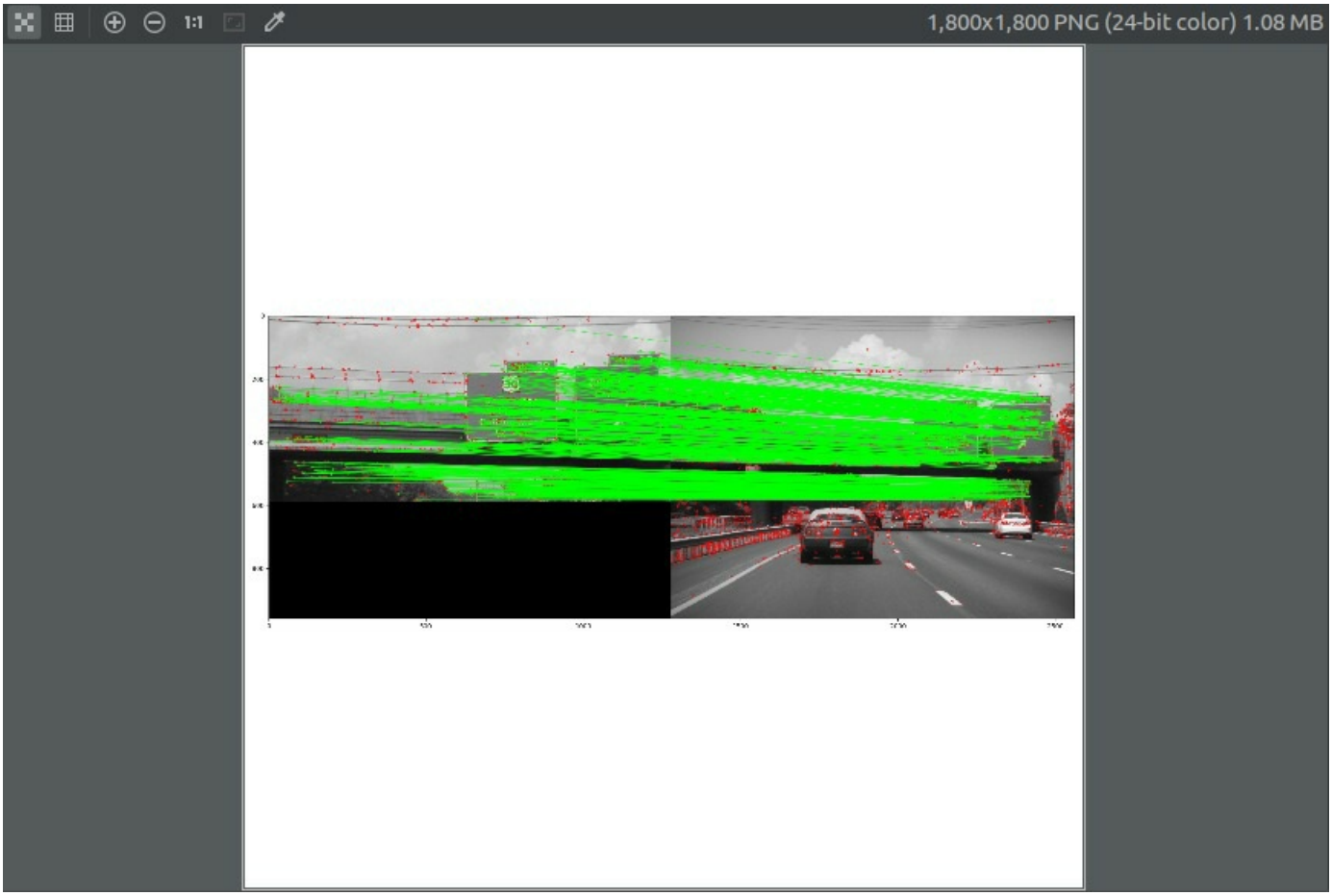
Code

```
start = int(round(time.time() * 1000))
dataset_path = '/Users/neerajpadarthi/Neeraj/2nd Sem/python'
img_building = cv2.imread(os.path.join(dataset_path, 'SwiftImage.png'))
img_building = cv2.cvtColor(img_building, cv2.COLOR_BGR2RGB)  # Convert from cv's BRG default color order to RGB

orb = cv2.ORB_create()  # OpenCV 3 backward incompatibility: Do not create a detector with `cv2.ORB()`.
key_points, description = orb.detectAndCompute(img_building, None)
img_building_keypoints = cv2.drawKeypoints(img_building,
                                           key_points,
                                           img_building,
                                           flags=cv2.DRAW_MATCHES_FLAGS_DRAW_RICH_KEYPOINTS)# Draw circles.
plt.figure(figsize=(16, 16))
plt.title('ORB Interest Points')
plt.imshow(img_building_keypoints); plt.show()

def image_detect_and_compute(detector, img_name):
    """Detect and compute interest points and their descriptors."""
    img = cv2.imread(os.path.join(dataset_path, img_name))
    img = cv2.cvtColor(img, cv2.COLOR_BGR2GRAY)
    kp, des = detector.detectAndCompute(img, None)
    return img, kp, des


def draw_image_matches(detector, img1_name, img2_name, nmatches=10):
    """Draw ORB feature matches of the given two images."""
    img1, kp1, des1 = image_detect_and_compute(detector, img1_name)
    img2, kp2, des2 = image_detect_and_compute(detector, img2_name)

    bf = cv2.BFMatcher(cv2.NORM_HAMMING, crossCheck=True)
    matches = bf.match(des1, des2)
    matches = sorted(matches, key=lambda x: x.distance)  # Sort matches by distance.  Best come first.

    img_matches = cv2.drawMatches(img1, kp1, img2, kp2, matches[:nmatches], img2, flags=2)  # Show top 10 matches
    plt.figure(figsize=(16, 16))
    plt.title(type(detector))
    plt.imshow(img_matches);
    plt.show()


orb = cv2.ORB_create()

draw_image_matches(orb, 'SwiftImage.png', 'SwiftImageCrop.jpg')

sift = cv2.xfeatures2d.SIFT_create()
kp, des = sift.detectAndCompute(img_building, None)
img_kp = cv2.drawKeypoints(img_building, kp, img_building)

plt.figure(figsize=(15, 15))
plt.imshow(img_kp); plt.show()
```

Outputs


1:1
1,600x1,600 PNG (24-bit color) 1.47 MB
EXIT 30
Warwick Rd
Lawnside


1:1
1,600x1,600 PNG (24-bit color) 668.03 kB
<class 'cv2.ORB'>


1:1
1,800x1,800 PNG (24-bit color) 1.08 MB