khadgajyoth_19024_part1

April 19, 2022

1 Alli Khadga Jyoth - 19024 DSE

1.1 Q1

[ ]:
```

```

1.1.1 Sudo code for SIFT

```
Input (image)
Convert the image to grayscale
do for i range (n,N):
    image_i = Scale(image,scale(x,y) = i)
    do for j range(k,K):
        image_ij = GaussianBlur(image_i,sigma(x,y) = j)
    end for
end for

# Now doing Difference of Gaussian to find the keypoints
do for i in range(n.N):
    do for j in range(k,K):
        DOG_ij = image_ij - image_(i,j-1)
    end for
end for

# Selecting Keypoints
keypoints <- select the points which are same accross the all the DOG_ij image.

Neigh_key <- neighbourhood pixels around the key points taken from the original image and
not the DOG_ij image.
Grad_neigh <- Split the Neigh_key into 4x4 grid and compute the Gradients of each
Make a histogram for each of the Grad_neigh binned into 8 values i.e., 8 directions
spaced 45 degrees.
Hist_final <- contains a 128 valued feature vector of all the gradients
(16 splits x 8 bins for each histogram)
```

Uses for SIFT

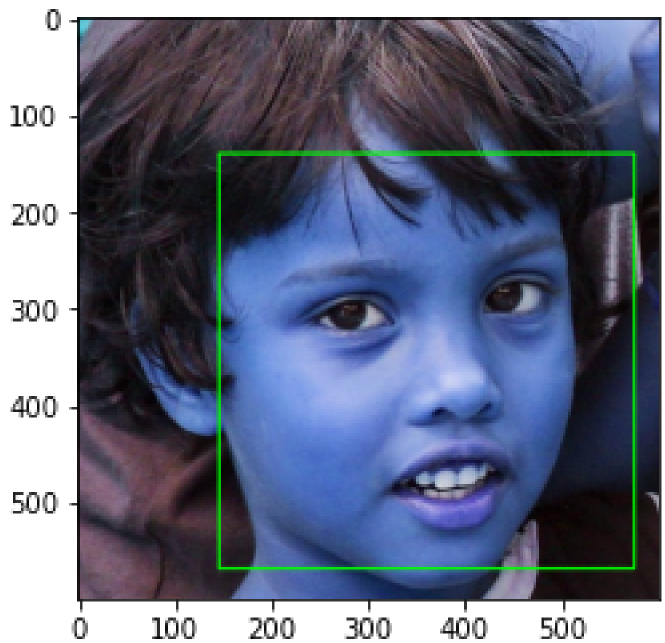
1. Used to match the images taken from different angles
2. Used for object recognition and 3d tracking

3. When the scale of the image changes other descriptors doesn't perform well.
4. SIFT feature descriptor is invariant to uniform scaling, orientation, light variations, and affine distortion. Even in the presence of clutter and partial opacity, SIFT can accurately identify objects.

1.2 Q2 Viola Jones

[1]:
```
# Importing OpenCV package
import cv2

# Reading the image
img = cv2.imread('archive/real_and_fake_face_detection/real_and_fake_face/
↪training_real/real_00001.jpg')

# Converting image to grayscale
gray_img = cv2.cvtColor(img, cv2.COLOR_BGR2GRAY)

# Loading the required haar-cascade xml classifier file
haar_cascade = cv2.CascadeClassifier('Haarcascade_frontalface_default.xml')

# Applying the face detection method on the grayscale image
faces_rect = haar_cascade.detectMultiScale(gray_img, 1.1, 9)

# Iterating through rectangles of detected faces
for (x, y, w, h) in faces_rect:
    cv2.rectangle(img, (x, y), (x+w, y+h), (0, 255, 0), 2)

# cv2.imshow('Detected faces', img)
import matplotlib.pyplot as plt
plt.imshow(img)
# cv2.waitKey(0)
```

[1]:
```
<matplotlib.image.AxesImage at 0x15f88951430>
```



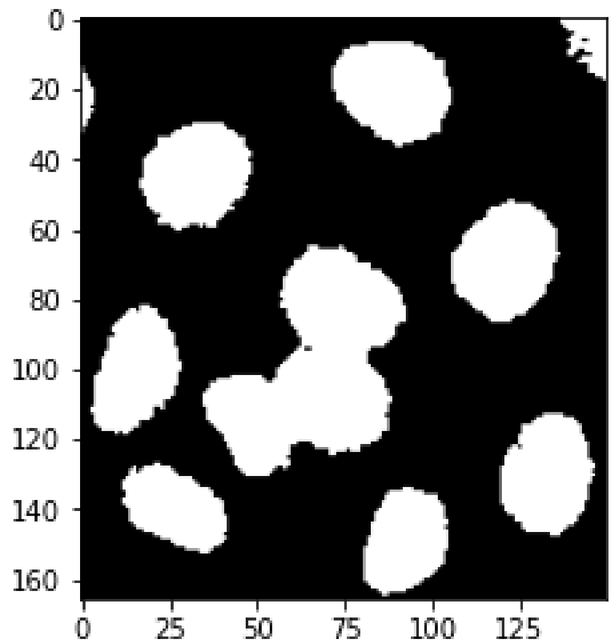
[2]:
```
import numpy as np
import matplotlib.pyplot as plt
```

1.3 Q3 Clustering

[3]:
```
import skimage
from skimage.feature import peak_local_max
from scipy import ndimage as ndi
```

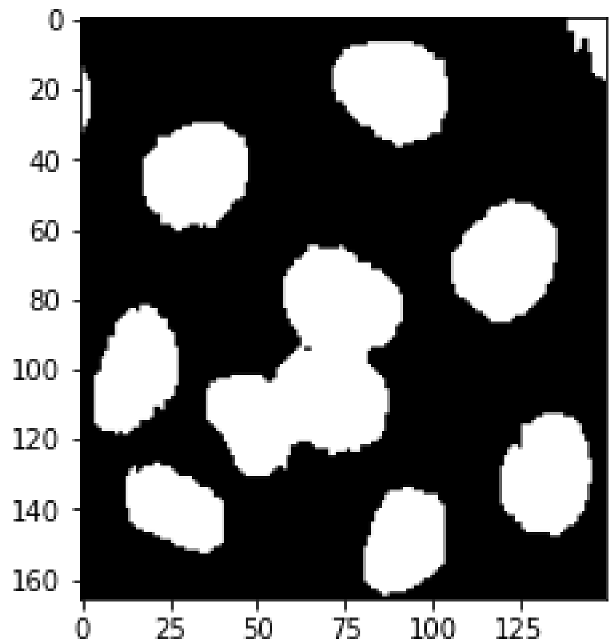
[8]:
```
image = cv2.imread('tounching_grayscale.png',cv2.IMREAD_GRAYSCALE)
plt.imshow(image,'gray')
ret, thresh1 = cv2.threshold(image, 120, 255, cv2.THRESH_OTSU+cv2.THRESH_BINARY)
plt.imshow(thresh1,'gray')
```

[8]: <matplotlib.image.AxesImage at 0x23d3086f130>



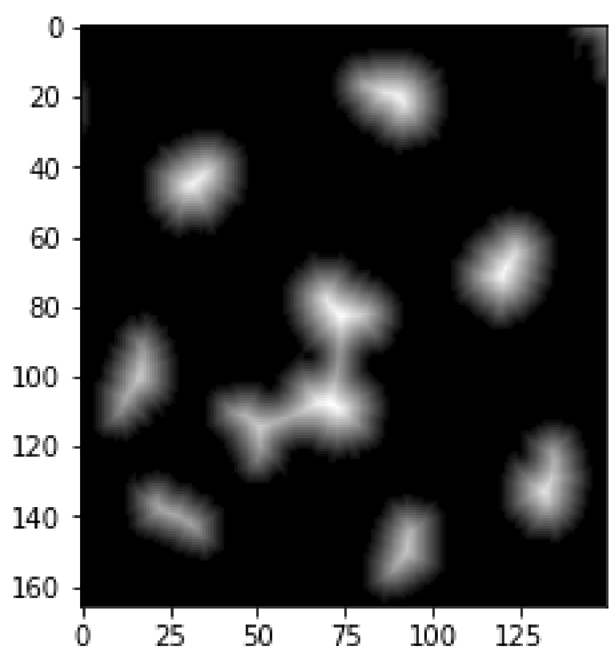
[9]:
```
# Noise removal
kernel = np.ones((3),np.uint8)
opening_img = cv2.morphologyEx(thresh1,cv2.MORPH_OPEN, kernel, iterations =3)

plt.imshow(opening_img,'gray')
```

[9]: <matplotlib.image.AxesImage at 0x23d308eb160>



[11]:
```python
dist_transform = cv2.distanceTransform(opening_img, cv2.DIST_L2,0)
plt.imshow(dist_transform,'gray')
```

[11]: <matplotlib.image.AxesImage at 0x23d309c6670>

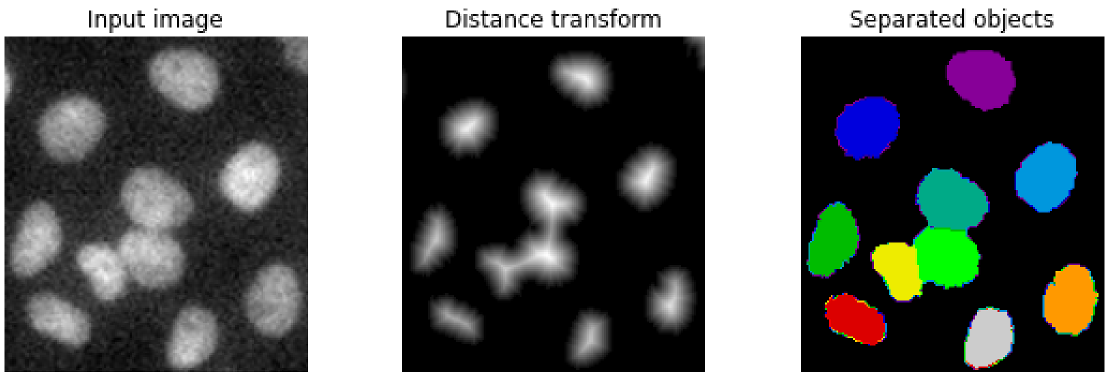


[14]:
```
local_max_location = peak_local_max(dist_transform, min_distance=6)
local_max_boolean = peak_local_max(dist_transform, min_distance=6,indices=False)

print(local_max_boolean)
```

```
[[False False False … False False False]
 [False False False … False False False]
 [False False False … False False False]
 …
 [False False False … False False False]
 [False False False … False False False]
 [False False False … False False False]]
```

```
C:\Users\KHADGA JYOTH ALLI\AppData\Local\Temp\ipykernel_14672\4016320351.py:2:
FutureWarning: indices argument is deprecated and will be removed in version
0.20. To avoid this warning, please do not use the indices argument. Please see
peak_local_max documentation for more details.
  local_max_boolean = peak_local_max(dist_transform,
min_distance=6,indices=False)
```

[15]:
```
markers, _ = ndi.label(local_max_boolean)
segmented = skimage.segmentation.watershed(255-dist_transform, markers,␣
 ↪mask=thresh1)
fig, axes = plt.subplots(ncols=3, figsize=(9, 3), sharex=True, sharey=True)
ax = axes.ravel()

ax[0].imshow(image, cmap=plt.cm.gray)
ax[0].set_title('Input image')
ax[1].imshow(dist_transform, cmap=plt.cm.gray)
ax[1].set_title('Distance transform')
ax[2].imshow(segmented, cmap=plt.cm.nipy_spectral)
ax[2].set_title('Separated objects')

for a in ax:
    a.set_axis_off()

fig.tight_layout()
plt.show()
```



1.4 Q4 shapes

[ ]:

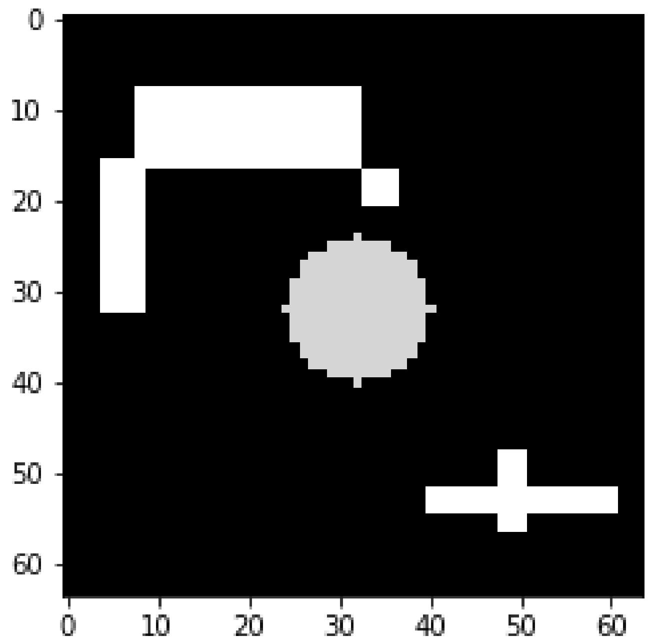
```

```

[1]:

```
from skimage.measure import label, regionprops
```

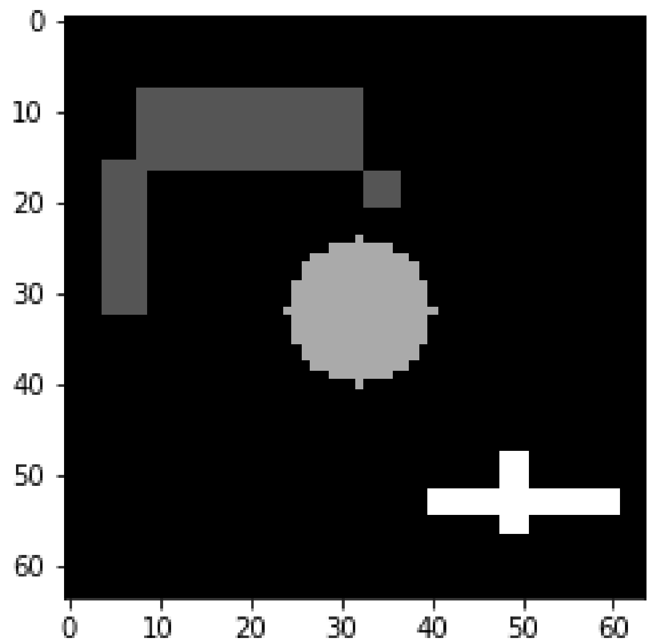
[5]:

```
img = cv2.imread('shapes.png',cv2.IMREAD_GRAYSCALE)
ret1,thresh_img = cv2.threshold(img,100,1,cv2.THRESH_BINARY)
plt.imshow(img ,cmap ='gray')
plt.show()
```



[6]:
```
labelled_img,labels = label(thresh_img,connectivity=2,return_num=True) #  for 8␣
↪Neighborhood connectivity=2
print("No. of components is : ",labels)
plt.imshow(labelled_img ,cmap ='gray')
plt.show()
```

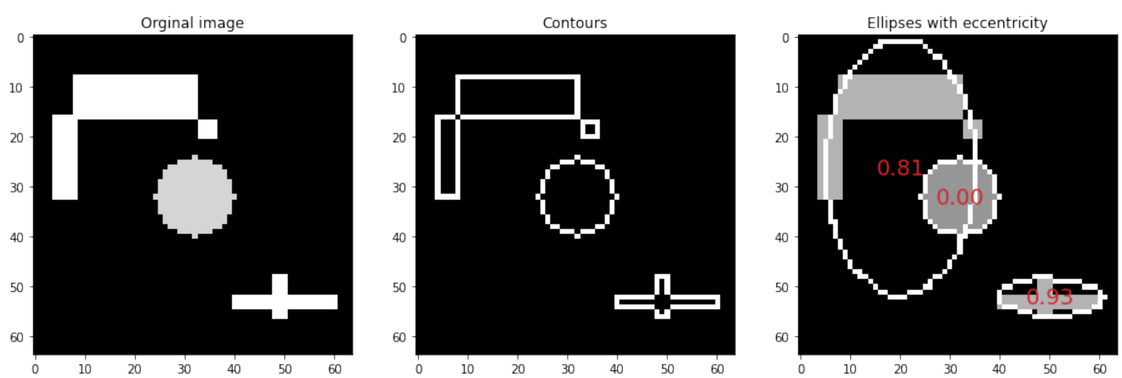
```
No. of components is :  3
```



[7]:
```
regions_propers = regionprops(labelled_img)
for region in regions_propers:
    print('Eccentricity:',region.eccentricity)
    print('BBox',region.bbox)
    print('_'*10)

plt.figure(figsize=[16,20])
plt.subplot(131)
plt.title('Orginal image')
plt.imshow(img,'gray')
# ret,thresh = cv2.threshold(img,127,255,0)
contours,hierarchy = cv2.findContours(thresh_img, 1, 2)
blank = np.zeros(thresh_img.shape[:2],dtype='uint8')
cv2.drawContours(blank, contours, -1,(255, 0, 0), 1)
plt.subplot(132)
plt.title('Contours')
plt.imshow(blank,'gray')
```

```
def eccentricity(contour):
    """Calculates the eccentricity fitting an ellipse from a contour"""

    (x, y), (MA, ma), angle = cv2.fitEllipse(cnt)

    a = ma / 2
    b = MA / 2

    ecc = np.sqrt(a ** 2 - b ** 2) / a
    return (x,y), ecc

for cnt in contours:
    ellipse = cv2.fitEllipse(cnt)
    plt.subplot(133)
    plt.title('Ellipses with eccentricity')
    _ = cv2.ellipse(img,ellipse,(255,0,0),1)
    (x,y), ecc = eccentricity(cnt)
    text_kwargs = dict(ha='center', va='center', fontsize=18, color='C3')
    plt.text(x,y,f'{ecc:.2f}',**text_kwargs)
    plt.imshow(_,'gray')
```

```
Eccentricity: 0.8669446497443333
BBox (8, 4, 33, 37)
----------
Eccentricity: 0.0
BBox (24, 24, 41, 41)
----------
Eccentricity: 0.9486451162938622
BBox (48, 40, 57, 61)
----------
```




```
[ ]:
```

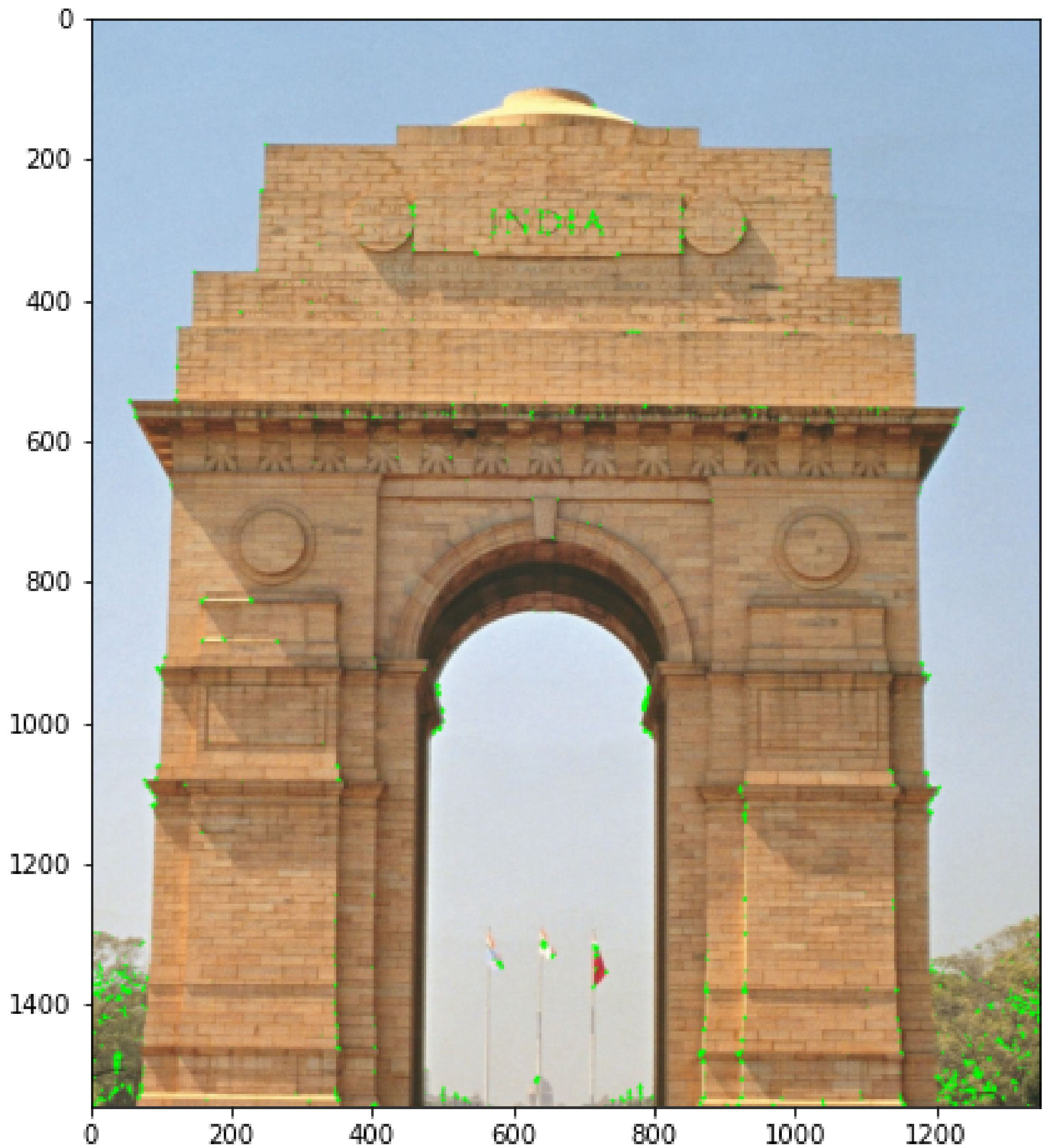
1.5 Q5

SURF – Speeded Up Robust Features SURF is the speed up version of SIFT. In SIFT, Lowe approximated Laplacian of Gaussian with Difference of Gaussian for finding scale-space. SURF goes a little further and approximates LoG with Box Filter. One big advantage of this approximation is that, convolution with box filter can be easily calculated with the help of integral images. And it can be done in parallel for different scales. Also, the SURF rely on determinant of Hessian matrix for both scale and location. For orientation assignment, SURF uses wavelet responses in horizontal and vertical direction for a neighborhood of size 6s. Adequate guassian weights are also applied to it. The dominant orientation is estimated by calculating the sum of all responses within a sliding orientation window of angle 60 degrees. wavelet response can be found out using integral images very easily at any scale. SURF provides such a functionality called Upright-SURF or U-SURF. It improves speed and is robust upto . OpenCV supports both, depending upon the flag, upright. If it is 0, orientation is calculated. If it is 1, orientation is not calculated and it is faster.

1.5.1 Harris Corner

[49]:
```
import cv2 as cv
img = cv.imread('download.jpg')
img = cv.cvtColor(img,cv.COLOR_BGR2RGB)
gray = cv.cvtColor(img,cv.COLOR_RGB2GRAY)
gray = np.float32(gray)
dst = cv.cornerHarris(gray,5,5,0.04)
#result is dilated for marking the corners, not important
# dst = cv.dilate(dst,None)
# Threshold for an optimal value, it may vary depending on the image.
img[dst>0.01*dst.max()]=[0,255,0]
plt.figure(figsize = (8,8))
plt.imshow(img,'gray')
```

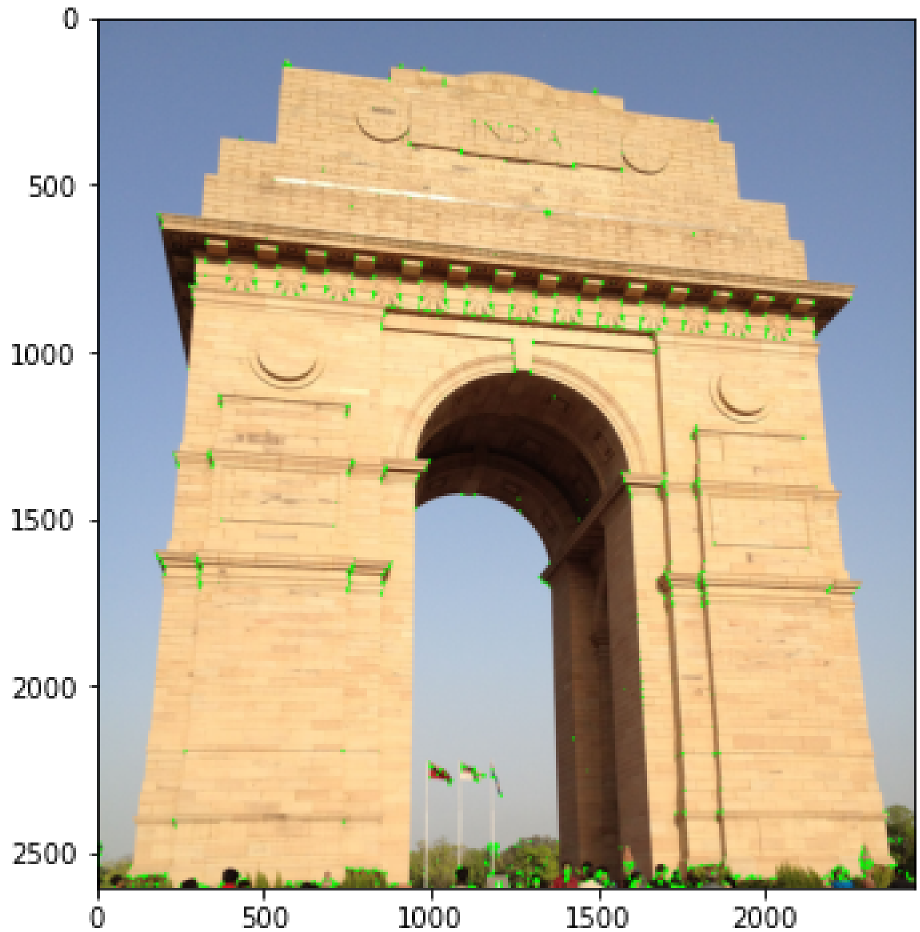
[49]:
```
<matplotlib.image.AxesImage at 0x23d331bad30>
```



[51]:
```
img = cv.imread('download (2).jpg')
img = cv.cvtColor(img,cv.COLOR_BGR2RGB)
gray = cv.cvtColor(img,cv.COLOR_RGB2GRAY)
gray = np.float32(gray)
dst = cv.cornerHarris(gray,7,9,0.04)
#result is dilated for marking the corners, not important
# dst = cv.dilate(dst,None)
# Threshold for an optimal value, it may vary depending on the image.
img[dst>0.01*dst.max()]=[0,255,0]
```

```
plt.figure(figsize = (6,6))
plt.imshow(img,'gray')
```

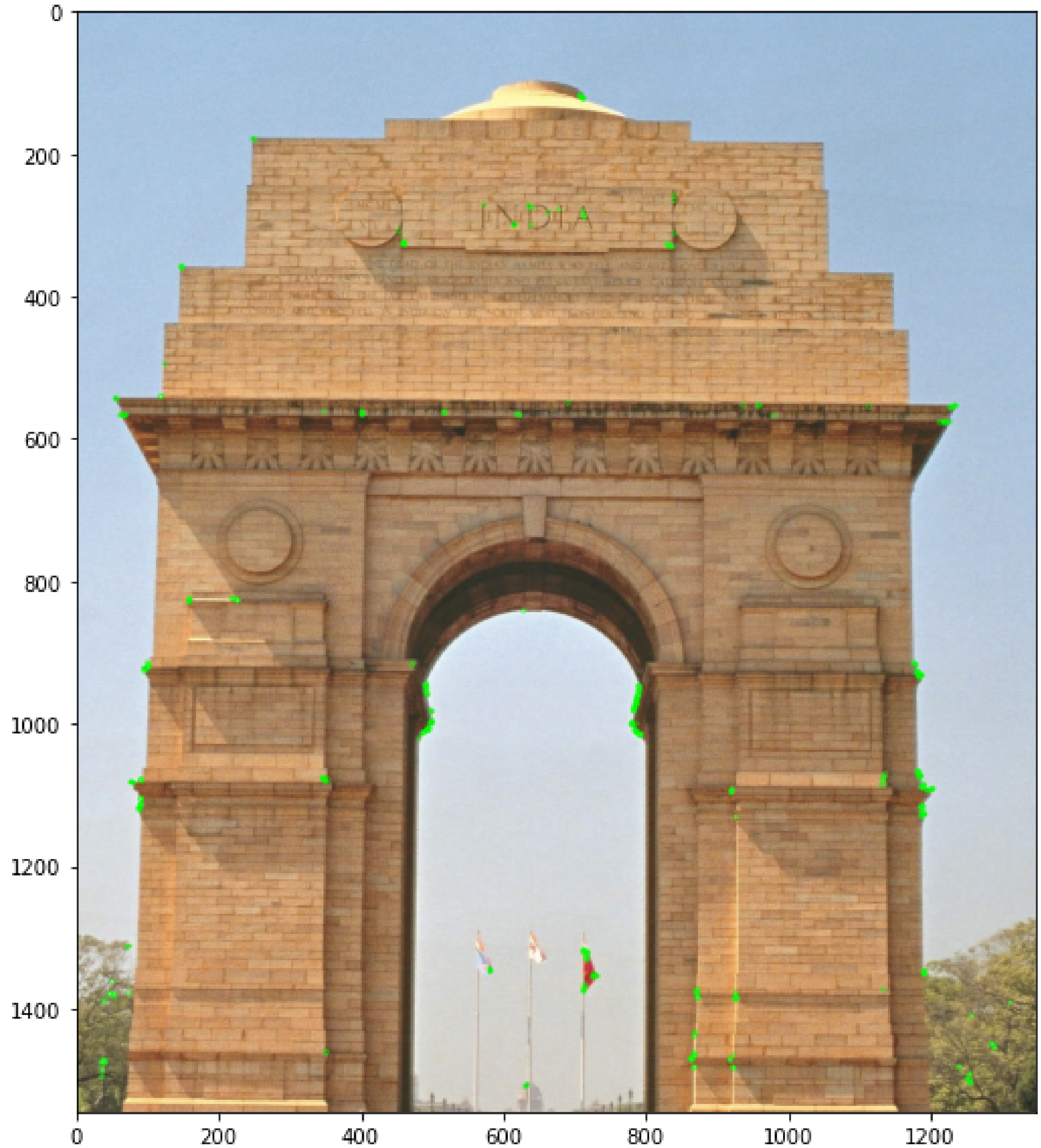
[51]: <matplotlib.image.AxesImage at 0x23d332851f0>



1.6 ORB

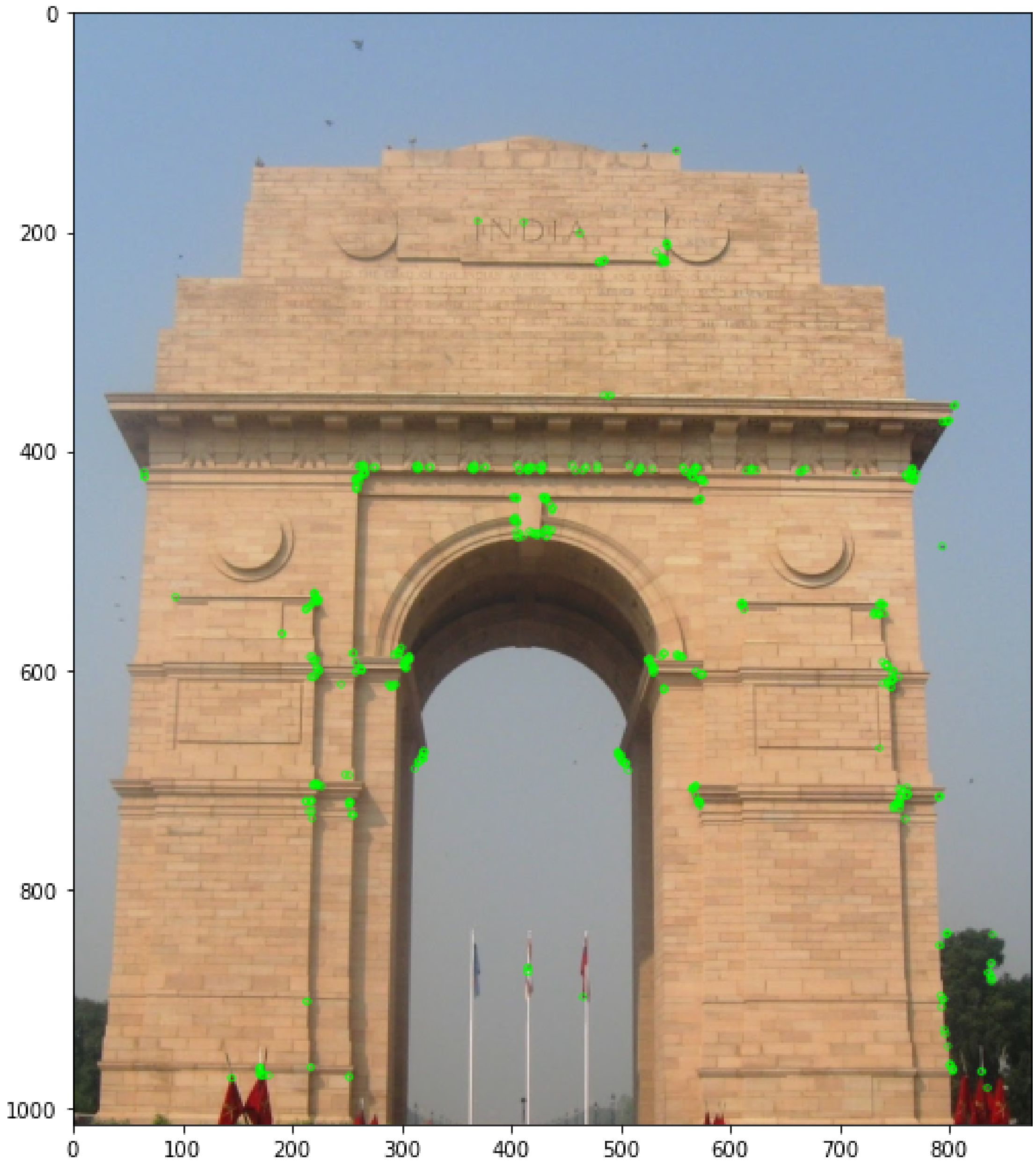
[83]:
```
import cv2
img1 = cv2.imread('download.jpg')
img1 = cv2.cvtColor(img1,cv2.COLOR_BGR2RGB)
img_g1 = cv2.cvtColor(img1,cv2.COLOR_RGB2GRAY)

# Initiate STAR detector
orb  = cv.ORB().create()

# find the keypoints with ORB and compute the descriptors with ORB
kp1, des1 = orb.detectAndCompute(img_g1,None)
```

```python
# draw only keypoints location,not size and orientation
img2 = cv2.drawKeypoints(img1,kp1,None,color=(0,255,0), flags=0)
plt.figure(figsize = (16,10))
plt.imshow(img2),plt.show()
```




```
[83]: (<matplotlib.image.AxesImage at 0x23d34ea1a60>, None)
```

[153]:
```python
import cv2
img2 = cv2.imread('download (3).jpg')
img2 = cv2.cvtColor(img2,cv2.COLOR_BGR2RGB)
img_g1 = cv2.cvtColor(img2,cv2.COLOR_RGB2GRAY)

# Initiate STAR detector
orb = cv2.ORB_create()

# find the keypoints with ORB and compute the descriptors with ORB
kp2, des2 = orb.detectAndCompute(img_g1,None)


# draw only keypoints location,not size and orientation
img21 = cv2.drawKeypoints(img2,kp2,None,color=(0,255,0), flags=0)
plt.figure(figsize = (16,10))
plt.imshow(img21),plt.show()
```

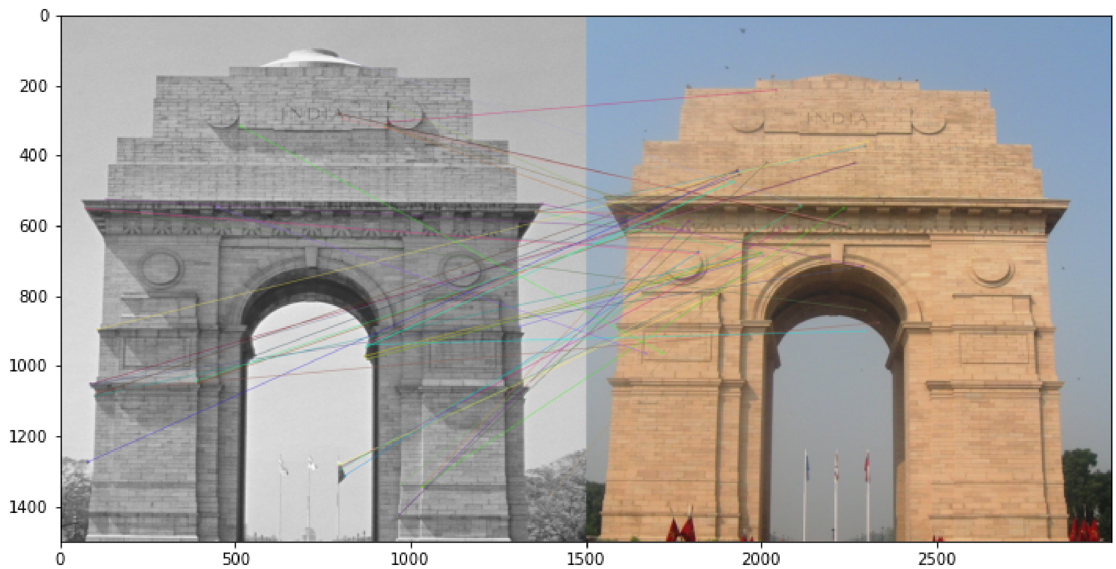


```
[153]: (<matplotlib.image.AxesImage at 0x23d001ac460>, None)
```

1.7 ORB Matching

```python
[154]: img_g1= cv2.resize(img1,(1500,1500),cv2.INTER_AREA)
img_g2= cv2.resize(img2,(1500,1500),cv2.INTER_AREA)
# create BFMatcher object
bf = cv2.BFMatcher(cv2.NORM_HAMMING, crossCheck=True)
```

```
# Match descriptors.
matches = bf.match(des1,des2)

# Sort them in the order of their distance.
matches = sorted(matches, key = lambda x:x.distance)

# Draw first 10 matches.
img3 = cv2.drawMatches(img_g1,kp1,img_g2,kp2,matches[0:50],img_g1, flags=2)
plt.figure(figsize=(12,12))
plt.imshow(img3),plt.show()
```

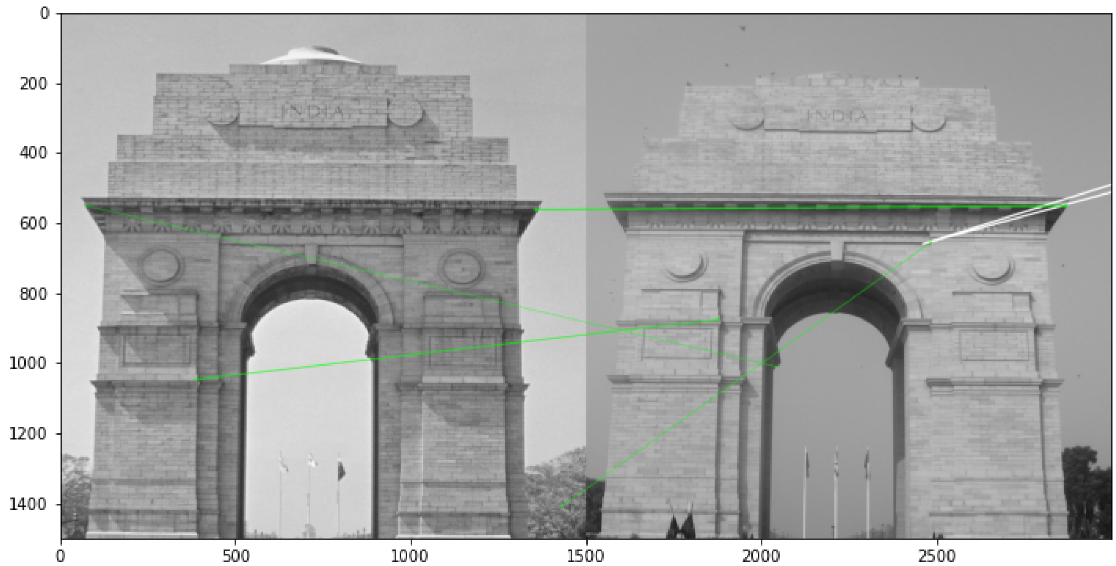


[154]: (<matplotlib.image.AxesImage at 0x23d50091970>, None)

1.8 Applying Ransac and Homography

[157]:
```
img1 = cv2.imread('download.jpg',0)          # queryImage
img2 = cv2.imread('download (3).jpg',0) # trainImage
img1= cv2.resize(img1,(1500,1500),cv2.INTER_AREA)
img2= cv2.resize(img2,(1500,1500),cv2.INTER_AREA)



orb = cv2.ORB_create()
# find the keypoints and descriptors with ORB
kp1, des1 = orb.detectAndCompute(img1,None)
kp2, des2 = orb.detectAndCompute(img2,None)



bf = cv2.BFMatcher(cv2.NORM_HAMMING, crossCheck=True)
```

```python
# Match descriptors.
matches = bf.match(des1,des2)

# Sort them in the order of their distance.
matches = sorted(matches, key = lambda x:x.distance)

# if len(good)>MIN_MATCH_COUNT:
src_pts = np.float32([ kp1[m.queryIdx].pt for m in matches ]).reshape(-1,1,2)
dst_pts = np.float32([ kp2[m.trainIdx].pt for m in matches ]).reshape(-1,1,2)

# Using RANSAC to find the good matches and avoid outliers
M, mask = cv2.findHomography(src_pts, dst_pts, cv2.RANSAC,5.0)
matchesMask = mask.ravel().tolist()

h,w = img1.shape
pts = np.float32([ [0,0],[0,h-1],[w-1,h-1],[w-1,0] ]).reshape(-1,1,2)
dst = cv2.perspectiveTransform(pts,M)

img2 = cv2.polylines(img2,[np.int32(dst)],True,255,3, cv2.LINE_AA)



draw_params = dict(matchColor = (0,255,0), # draw matches in green color
                   singlePointColor = None,
                   matchesMask = matchesMask, # draw only inliers
                   flags = 2)

img3 = cv2.drawMatches(img1,kp1,img2,kp2,matches,None,**draw_params)
plt.figure(figsize=(12,12))
plt.imshow(img3, 'gray'),plt.show()
```



```
[157]: (<matplotlib.image.AxesImage at 0x23d198a51c0>, None)
```

```
[ ]:
```

1.9 Q6 HOG

```python
[8]: from skimage import color
     from skimage.feature import hog
     from sklearn import svm
     from sklearn.metrics import classification_report,accuracy_score
     import os
     from tqdm import tqdm
     from random import shuffle
     import pandas as pd
```

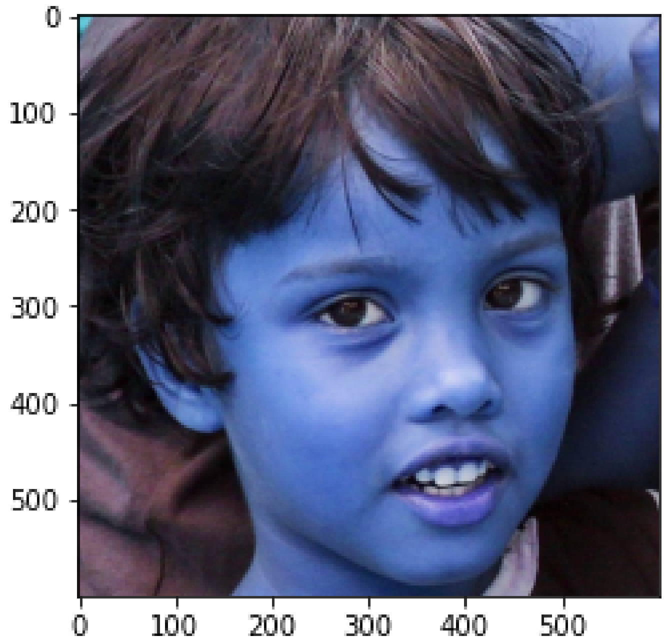
```python
[9]: # path ="archive (1)/real_and_f#renaming real and fake directories
     real = "archive/real_and_fake_face_detection/real_and_fake_face/training_real"
     fake = "archive/real_and_fake_face_detection/real_and_fake_face/training_fake"
     #we're creating a list of real and fake images
     real_path = os.listdir(real)
     fake_path = os.listdir(fake)
```

```python
[10]: print(len(real_path))
      print(len(fake_path))
```

```
1081
961
```

[11]:
```
plt.imshow(cv2.imread(os.path.join(real, real_path[0])))
```

[11]: <matplotlib.image.AxesImage at 0x23d86629370>



[12]:
```
img_size = int(128)
def create_data():
    training_data = []
    y=[]
    for img in tqdm(real_path):
        path = os.path.join(real, img)
#         label = [1]
        try:
            image = cv2.resize( cv2.imread(path,0), (img_size,img_size) )
            training_data.append(np.array(image))
            y.append(1)
        except:
            continue

    for img in tqdm(fake_path):
        path = os.path.join(fake, img)
#         label = [0]
        try:
            image = cv2.resize(cv2.imread(path,0), (img_size,img_size))
            training_data.append(np.array(image))
            y.append(0)
```

```
        except: continue

    return(training_data,y)




data_gray,labels = create_data()
```

```
100%|
  | 1081/1081 [00:10<00:00, 98.56it/s]
100%|
   | 961/961 [00:09<00:00, 99.23it/s]
```

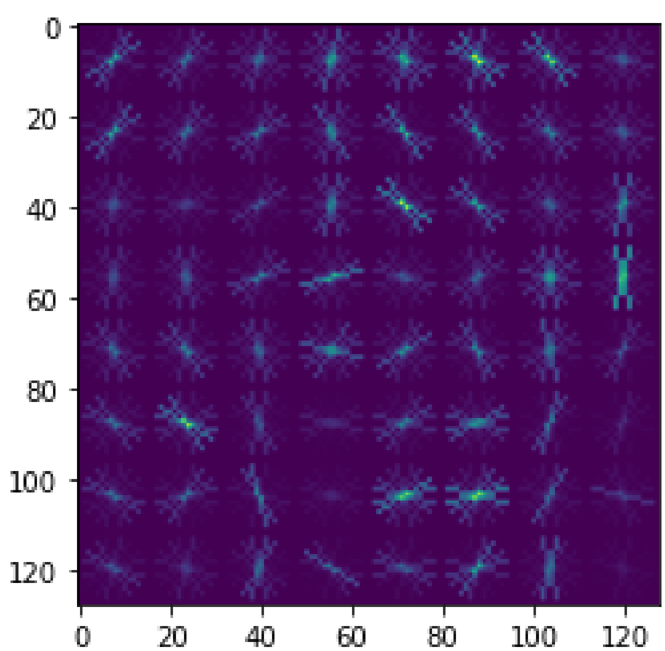
[13]:

```
ppc = 16
hog_images = []
hog_features = []
for image in data_gray:
    fd,hog_image = hog(image, orientations=8,␣
 ↪pixels_per_cell=(ppc,ppc),cells_per_block=(4, 4),block_norm=␣
 ↪'L2',visualize=True)
    hog_images.append(hog_image)
    hog_features.append(fd)
```

[14]:

```
plt.imshow(hog_images[0])
```

[14]: <matplotlib.image.AxesImage at 0x23d8683a700>



[15]:
```python
labels =  np.array(labels).reshape(len(labels),1)
```

[18]:
```python
#What percentage of data you want to keep for training
percentage = 80
partition = int(len(hog_features)*percentage/100)
```

[19]:
```python
hog_features = np.array(hog_features)
data_frame = np.hstack((hog_features,labels))
np.random.shuffle(data_frame)
x_train, x_test = data_frame[:partition,:-1],  data_frame[partition:,:-1]
y_train, y_test = data_frame[:partition,-1:].ravel() , data_frame[partition:,-1:
↪].ravel()
```

[20]:
```python
clf = svm.SVC()
clf.fit(x_train,y_train)
y_pred = clf.predict(x_test)
```

[ ]:
```python

```

[21]:
```python
print("Accuracy: "+str(accuracy_score(y_test, y_pred)))
print('\n')
print(classification_report(y_test,y_pred))
```

```
Accuracy: 0.6234718826405868


              precision    recall  f1-score   support

         0.0       0.61      0.55      0.58       191
         1.0       0.64      0.68      0.66       218

    accuracy                           0.62       409
   macro avg       0.62      0.62      0.62       409
weighted avg       0.62      0.62      0.62       409
```

1.10 Q7 MNIST Classification

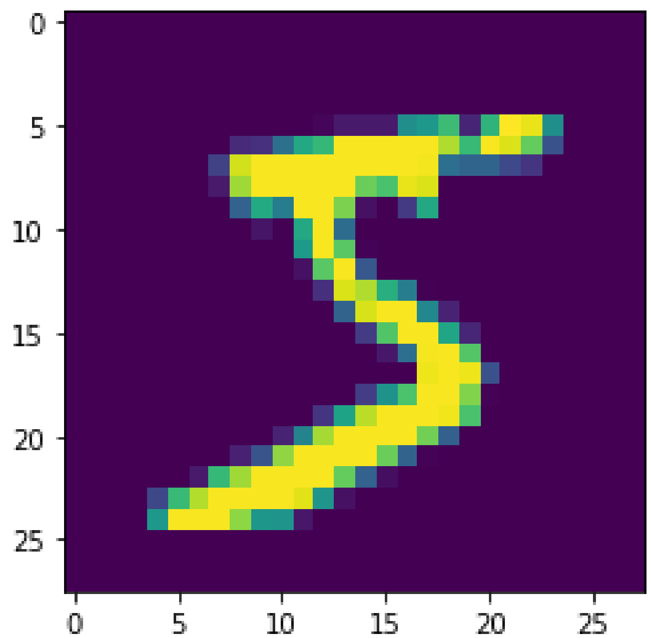
[22]:
```python
from sklearn.model_selection import train_test_split
from sklearn.neighbors import KNeighborsClassifier
from sklearn.preprocessing import StandardScaler
import struct
```

[23]:
```python
with open('train-images-idx3-ubyte/train-images.idx3-ubyte', 'rb') as f:
    magic, size = struct.unpack('>II', f.read(8))
    nrows, ncols = struct.unpack('>II', f.read(8))
```

```
    data = np.fromfile(f, dtype=np.dtype(np.uint8)).newbyteorder(">")
    data = data.reshape((size,nrows,ncols))
with open('train-labels-idx1-ubyte/train-labels.idx1-ubyte', 'rb') as i:
    magic, size = struct.unpack('>II', i.read(8))
    data_1 = np.fromfile(i, dtype=np.dtype(np.uint8)).newbyteorder(">")

data, labels = data, data_1
# len(x_train), len(y_train)
```

[24]:
```
plt.imshow(data[0])
labels[0]
```

[24]: 5



[127]:
```
##HOG Descriptor
#Returns a 1D vector for an image
ppcr = 8
ppcc = 8
hog_images = []
hog_features = []
for image in tqdm(data):
#  blur = cv.GaussianBlur(image,(5,5),0) #Gaussian Filtering
 fd,hog_image = hog(image, orientations=8,␣
 ↪pixels_per_cell=(ppcr,ppcc),cells_per_block=(2,2),block_norm=␣
 ↪'L2',visualize=True)
```

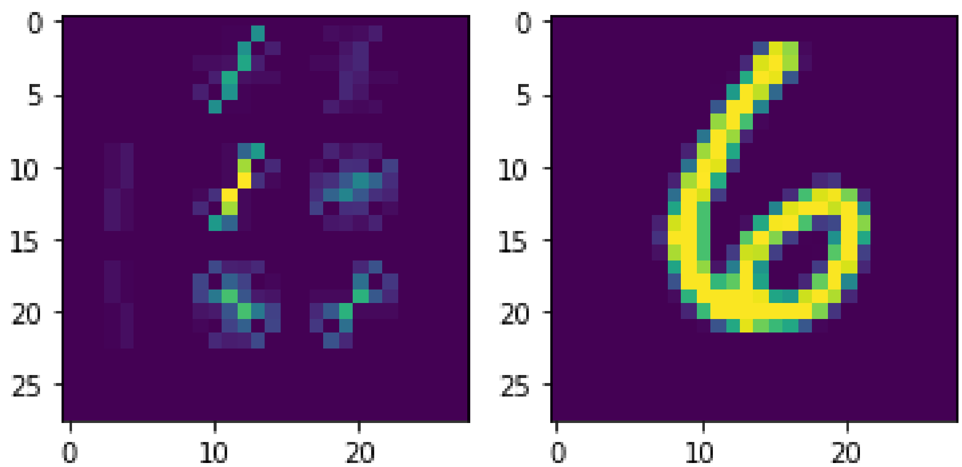
```
 hog_images.append(hog_image)
 hog_features.append(fd)
hog_features = np.array(hog_features)
hog_features.shape
```

```
100%|
| 60000/60000 [01:03<00:00, 951.96it/s]
```

[127]:
```
(60000, 128)
```

[128]:
```
plt.subplot(121)
plt.imshow(hog_images[8911])
plt.subplot(122)
plt.imshow(data[8911])
```

[128]:
```
<matplotlib.image.AxesImage at 0x23d02200220>
```



[129]:
```
labels = np.asarray(labels)
```

[131]:
```
X_train, X_test, y_train, y_test =␣
 ↪train_test_split(hog_features,labels,test_size=0.2)
print('Training data and target sizes: \n{}, {}'.format(X_train.shape,y_train.
 ↪shape))
print('Test data and target sizes: \n{}, {}'.format(X_test.shape,y_test.shape))
X_train.shape
```

```
Training data and target sizes:
(48000, 128), (48000,)
Test data and target sizes:
(12000, 128), (12000,)
```

[131]: (48000, 128)

[134]:
```
test_accuracy = []
scaler = StandardScaler()
X_scaled = scaler.fit_transform(X_train)
classifier = KNeighborsClassifier(n_neighbors=3,algorithm='brute')
classifier.fit(X_scaled, y_train)
X_test_scaled = scaler.fit_transform(X_test)
y_pred = classifier.predict(X_test_scaled)
test_accuracy = classifier.score(scaler.transform(X_test), y_test)
print("Accuracy: "+str(accuracy_score(y_test, y_pred)))
print('\n')
print(classification_report(y_test,y_pred))
```

```
Accuracy: 0.94525


              precision    recall  f1-score   support

           0       0.95      0.98      0.97      1187
           1       0.97      0.99      0.98      1319
           2       0.96      0.93      0.95      1184
           3       0.93      0.93      0.93      1225
           4       0.95      0.91      0.93      1120
           5       0.97      0.94      0.95      1069
           6       0.97      0.99      0.98      1177
           7       0.94      0.91      0.92      1319
           8       0.94      0.91      0.92      1201
           9       0.87      0.95      0.91      1199

    accuracy                           0.95     12000
   macro avg       0.95      0.94      0.95     12000
weighted avg       0.95      0.95      0.95     12000
```

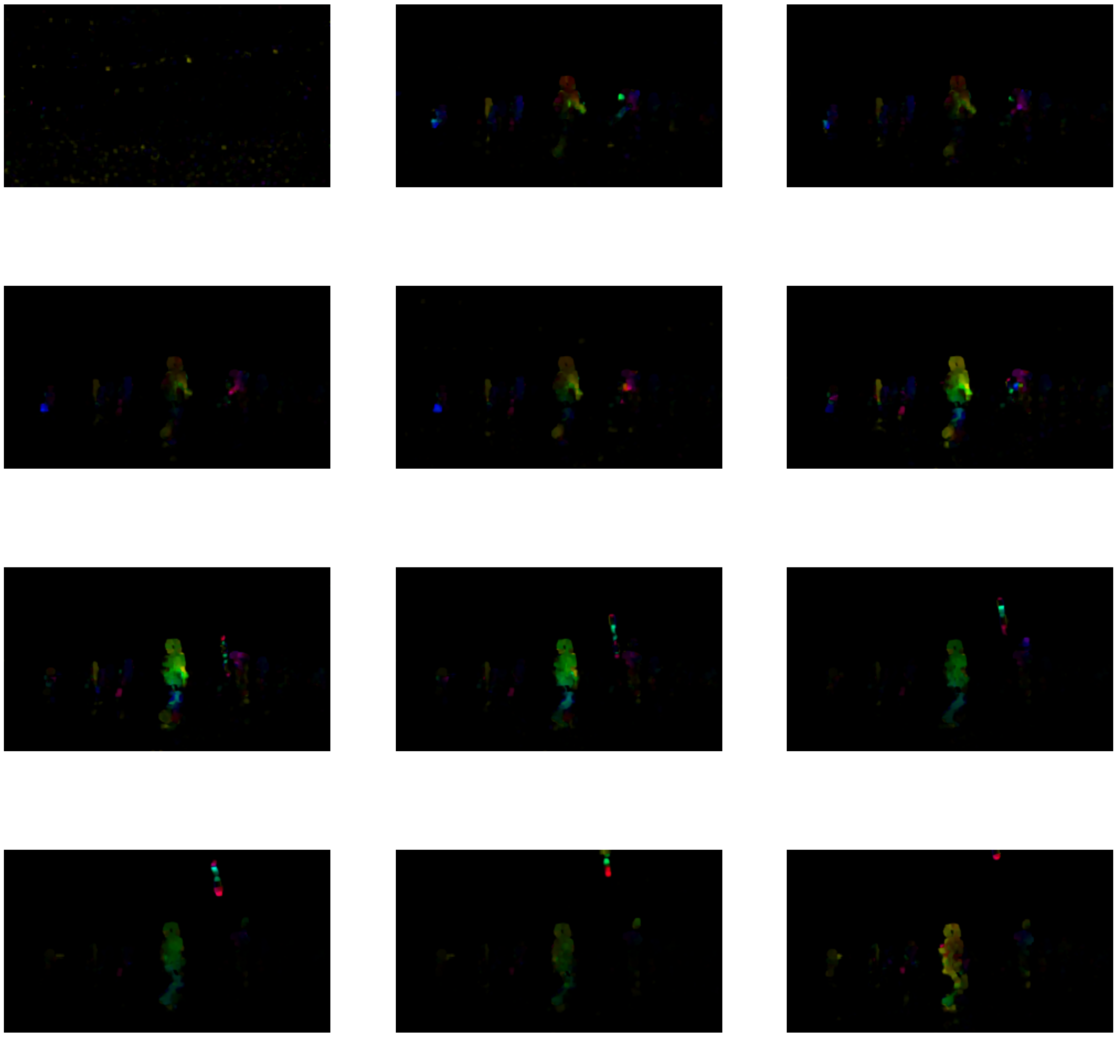
1.11 Part II

1.12 Q4 Optical Flow

1.12.1 Dense Optical Flow

[3]:
```
cap = cv2.VideoCapture("badminton_Trim.mp4")

ret, frame1 = cap.read()
prvs = cv2.cvtColor(frame1,cv2.COLOR_BGR2GRAY)
hsv = np.zeros_like(frame1)
hsv[...,1] = 255
i =1
plt.figure(figsize=(18, 18))
```

```python
while(1):
    ret, frame2 = cap.read()
    next = cv2.cvtColor(frame2,cv2.COLOR_BGR2GRAY)

    flow = cv2.calcOpticalFlowFarneback(prvs,next, None, 0.5, 3, 15, 3, 5, 1.2, 
 ↪0)

    mag, ang = cv2.cartToPolar(flow[...,0], flow[...,1])
    hsv[...,0] = ang*180/np.pi/2
    hsv[...,2] = cv2.normalize(mag,None,0,255,cv2.NORM_MINMAX)
    rgb = cv2.cvtColor(hsv,cv2.COLOR_HSV2BGR)
    if i<13:

        ax = plt.subplot(4,3,i)
        plt.imshow(rgb)
        plt.axis("off")

    i += 1
    cv2.imshow('frame2',rgb)
    k = cv2.waitKey(30) & 0xff
    if not ret:
        break
    elif k == 27:
        break
    elif k == ord('s'):
        cv2.imwrite('opticalfb.png',frame2)
        cv2.imwrite('opticalhsv.png',rgb)
    prvs = next

cap.release()
cv2.destroyAllWindows()
```

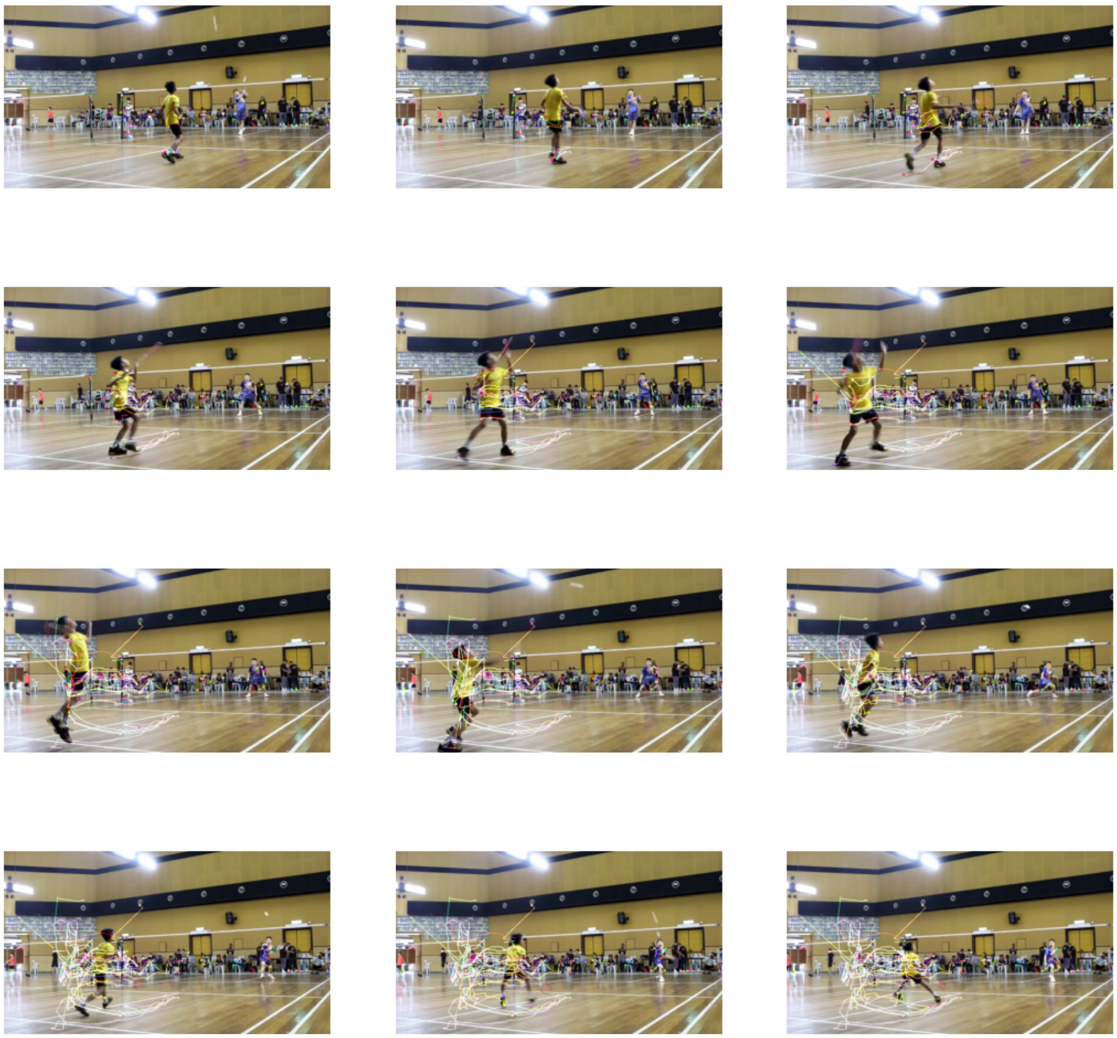


1.12.2 LK feature tracking

[4]:
```
cap = cv2.VideoCapture('badminton_Trim.mp4')

# params for ShiTomasi corner detection
feature_params = dict( maxCorners = 200,
                       qualityLevel = 0.4,
                       minDistance = 10,
                       blockSize = 7 )

# Parameters for lucas kanade optical flow
lk_params = dict( winSize  = (15,15),
                  maxLevel = 2,
                  criteria = (cv2.TERM_CRITERIA_EPS | cv2.TERM_CRITERIA_COUNT, 10, 0.03))
```

```python
# Create some random colors
color = np.random.randint(0,255,(100,3))

# Take first frame and find corners in it
ret, old_frame = cap.read()
old_gray = cv2.cvtColor(old_frame, cv2.COLOR_BGR2GRAY)
p0 = cv2.goodFeaturesToTrack(old_gray, mask = None, **feature_params)

# Create a mask image for drawing purposes
mask = np.zeros_like(old_frame)
plt.figure(figsize=(18, 18))
j = 1
idx = 1
while(1):
    ret,frame = cap.read()
    frame_gray = cv2.cvtColor(frame, cv2.COLOR_BGR2GRAY)

    # calculate optical flow
    p1, st, err = cv2.calcOpticalFlowPyrLK(old_gray, frame_gray, p0, None,␣
↪**lk_params)

    # Select good points
    good_new = p1[st==1]
    good_old = p0[st==1]

    # draw the tracks
    for i,(new,old) in enumerate(zip(good_new,good_old)):
        a,b = new.ravel()
        c,d = old.ravel()
        a,b,c,d = int(a),int(b),int(c),int(d)
        mask = cv2.line(mask, (int(a),int(b)),(int(c),int(d)), color[i].
↪tolist(), 2)
        frame = cv2.circle(frame,(a,b),5,color[i].tolist(),-1)
    img = cv2.add(frame,mask)

    if j in np.linspace(10,100,num=12,dtype=int):
        plt.subplot(4,3,idx)
        plt.imshow(img[:,:,::-1])
        plt.axis("off")
        idx+=1

    j += 1
    cv2.imshow('frame',img)
    k = cv2.waitKey(30) & 0xff
    if not ret:
        break
```

```
    elif k == 27:
        break

    # Now update the previous frame and previous points
    old_gray = frame_gray.copy()
    p0 = good_new.reshape(-1,1,2)

cap.release()
cv2.destroyAllWindows()
```



[ ]:
```

```

part2

April 17, 2022

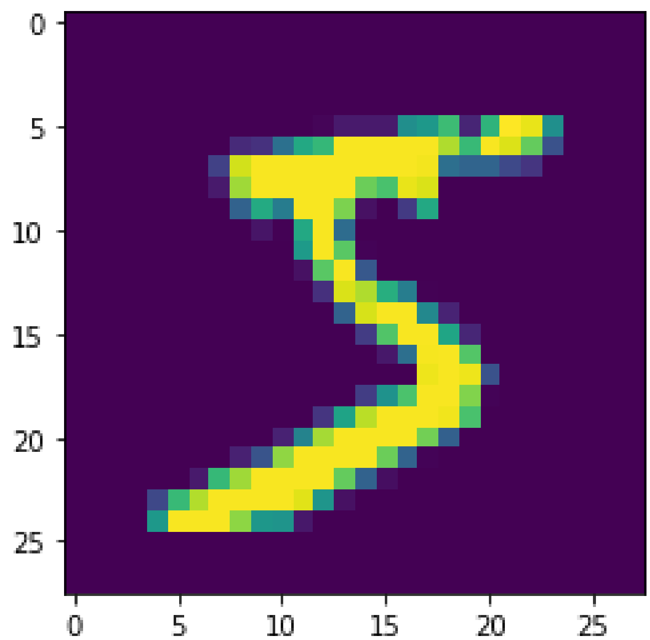
0.1 Q8 MNIST with Transfer Learning

[1]:
```
#Deep Learning
import tensorflow as tf
from tensorflow import keras
from tensorflow.keras import layers as L
from tensorflow.keras import layers
from tensorflow.keras.models import Sequential, Model
from tensorflow.keras.preprocessing.image import ImageDataGenerator

#Data Visualizations
import matplotlib.pyplot as plt
# !pip install seaborn
import os
import cv2
import numpy as np
import seaborn as sns
import struct
from tqdm import tqdm
from random import shuffle
import pandas as pd
```

[2]:
```
with open('train-images-idx3-ubyte/train-images.idx3-ubyte', 'rb') as f:
    magic, size = struct.unpack('>II', f.read(8))
    nrows, ncols = struct.unpack('>II', f.read(8))
    data = np.fromfile(f, dtype=np.dtype(np.uint8)).newbyteorder(">")
    data = data.reshape((size,nrows,ncols))
with open('train-labels-idx1-ubyte/train-labels.idx1-ubyte', 'rb') as i:
    magic, size = struct.unpack('>II', i.read(8))
    data_1 = np.fromfile(i, dtype=np.dtype(np.uint8)).newbyteorder(">")

data_train, labels_train = data, data_1
```

[3]:
```
plt.imshow(data_train[0])
labels_train[0]
```

[3]: 5



[4]:
```
from sklearn.model_selection import train_test_split
X_train, X_test, y_train, y_test =␣
↪train_test_split(data_train,labels_train,random_state=5,stratify=labels_train,test_size=0.
↪3)
```

[2]:
```
from tensorflow.keras.preprocessing.image import load_img, img_to_array,␣
↪array_to_img

def change_size(image):
    img = array_to_img(image, scale=False) #returns PIL Image
    img = img.resize((75, 75)) #resize image
    img = img.convert(mode='RGB') #makes 3 channels
    arr = img_to_array(img) #convert back to array
    return arr.astype(np.float64)
```

[6]:
```
train_arr = np.array(X_train).reshape(-1, 28, 28, 1)
train_arr.shape
```

[6]: (42000, 28, 28, 1)

[7]:
```
train_arr_75 = [change_size(img) for img in tqdm(train_arr)]
del train_arr
train_arr_75 = np.array(train_arr_75)
train_arr_75.shape
```

```
100%|                                                          |
```

```
42000/42000 [00:10<00:00, 3825.42it/s]
```

[7]:
```
(42000, 75, 75, 3)
```

[ ]:

[8]:
```
test_arr = np.array(X_test).reshape(-1, 28, 28, 1)
test_arr.shape
```

[8]:
```
(18000, 28, 28, 1)
```

[9]:
```
test_arr_75 = [change_size(img) for img in tqdm(test_arr)]
del test_arr
test_arr_75 = np.array(test_arr_75)
test_arr_75.shape
```

```
100%|                                        |
18000/18000 [00:04<00:00, 3909.40it/s]
```

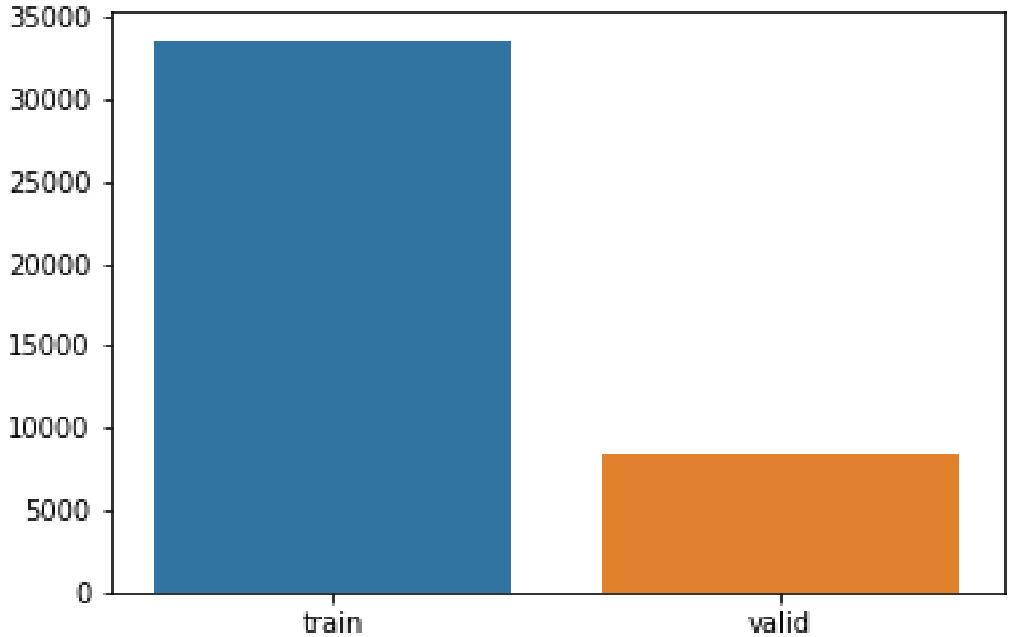
[9]:
```
(18000, 75, 75, 3)
```

[ ]:

[10]:
```
y_train = tf.keras.utils.to_categorical(y_train)
y_testc = tf.keras.utils.to_categorical(y_test)
image_gen = ImageDataGenerator(rescale=1./255, #easier for network to interpret␣
 ↪numbers in range [0,1]
                              zoom_range=0.1,
                              width_shift_range=0.2,
                              height_shift_range=0.2,
                              validation_split=0.2) # 80/20 train/val split

train_generator = image_gen.flow(train_arr_75,
                                 y_train,
                                batch_size=32,
                                shuffle=True,
                                subset='training',
                                seed=42)
valid_generator = image_gen.flow(train_arr_75,
                                 y_train,
                                batch_size=16,
                                shuffle=True,
                                subset='validation')
test_generator = image_gen.flow(test_arr_75,
                                 y_testc,
                                batch_size=32,
                                seed=42,shuffle=False)
del train_arr_75 #saves RAM
```

[11]:
```
sns.barplot(x = ['train', 'valid'], y= [train_generator.n, valid_generator.n])
```

[11]: <AxesSubplot:>



[12]:
```
base_model = tf.keras.applications.resnet50.ResNet50(input_shape = (75, 75, 3),
                                include_top = False,
                                weights = 'imagenet')
# base_model.trainable = False
model = Sequential()

model.add(base_model)

model.add(L.Flatten())
model.add(L.Dense(256, activation='relu'))
model.add(L.Dense(128, activation='relu'))
model.add(L.Dense(10, activation='softmax'))

model.compile(optimizer=keras.optimizers.Adam(learning_rate=0.0001),␣
 ↪loss='categorical_crossentropy', metrics=['accuracy'])
#Do not use default learning rate since it is too high!
```

[13]:
```
for layer in model.layers[0].layers:
    if layer.name == 'conv5_block1_0_conv':
        break
    layer.trainable=False
```

[14]:
```
model.summary()
```

```
Model: "sequential"

_________________________________________________________________
Layer (type)                 Output Shape              Param #
=================================================================
resnet50 (Functional)        (None, 3, 3, 2048)        23587712

_________________________________________________________________
flatten (Flatten)            (None, 18432)             0

_________________________________________________________________
dense (Dense)                (None, 256)               4718848

_________________________________________________________________
dense_1 (Dense)              (None, 128)               32896

_________________________________________________________________
dense_2 (Dense)              (None, 10)                1290
=================================================================
Total params: 28,340,746
Trainable params: 16,842,378
Non-trainable params: 11,498,368

_________________________________________________________________
```

[15]:
```
epochs = 5
history = model.fit(train_generator, validation_data=valid_generator,␣
↪epochs=epochs,
          steps_per_epoch=train_generator.n//train_generator.batch_size,
         validation_steps=valid_generator.n//valid_generator.batch_size)
```

```
Epoch 1/5
1050/1050 [==============================] - 139s 120ms/step - loss: 0.1840 -
accuracy: 0.9461 - val_loss: 0.2477 - val_accuracy: 0.9173
Epoch 2/5
1050/1050 [==============================] - 126s 119ms/step - loss: 0.0699 -
accuracy: 0.9811 - val_loss: 0.0658 - val_accuracy: 0.9830
Epoch 3/5
1050/1050 [==============================] - 126s 120ms/step - loss: 0.0560 -
accuracy: 0.9845 - val_loss: 0.0804 - val_accuracy: 0.9800
Epoch 4/5
1050/1050 [==============================] - 126s 120ms/step - loss: 0.0536 -
accuracy: 0.9854 - val_loss: 0.0428 - val_accuracy: 0.9892
Epoch 5/5
1050/1050 [==============================] - 126s 120ms/step - loss: 0.0438 -
accuracy: 0.9878 - val_loss: 0.0885 - val_accuracy: 0.9775
```

[16]:
```
pred = model.predict(test_generator)
predictions = np.argmax(pred,axis = 1)
```

[17]:
```
tf.math.confusion_matrix(y_test,predictions)
```

[17]: <tf.Tensor: shape=(10, 10), dtype=int32, numpy=

```
array([[1763,    0,    2,    1,    0,    0,    5,    2,    2,    2],
       [   0, 2020,    0,    0,    0,    0,    0,    3,    0,    0],
       [   0,   10, 1726,   17,    0,    0,    0,   33,    1,    0],
       [   0,    0,    0, 1831,    0,    2,    0,    6,    0,    0],
       [   0,   26,    0,    0, 1655,    0,    4,   52,    0,   16],
       [   0,    0,    0,   14,    0, 1601,    5,    3,    1,    2],
       [   5,    2,    3,    0,    0,    3, 1762,    0,    0,    0],
       [   0,    9,    1,    1,    0,    0,    0, 1869,    0,    0],
       [   0,    5,    1,   37,    4,    7,    8,    6, 1674,   13],
       [   1,    7,    0,   21,    4,    2,    0,   47,    2, 1701]])>
```

[18]:

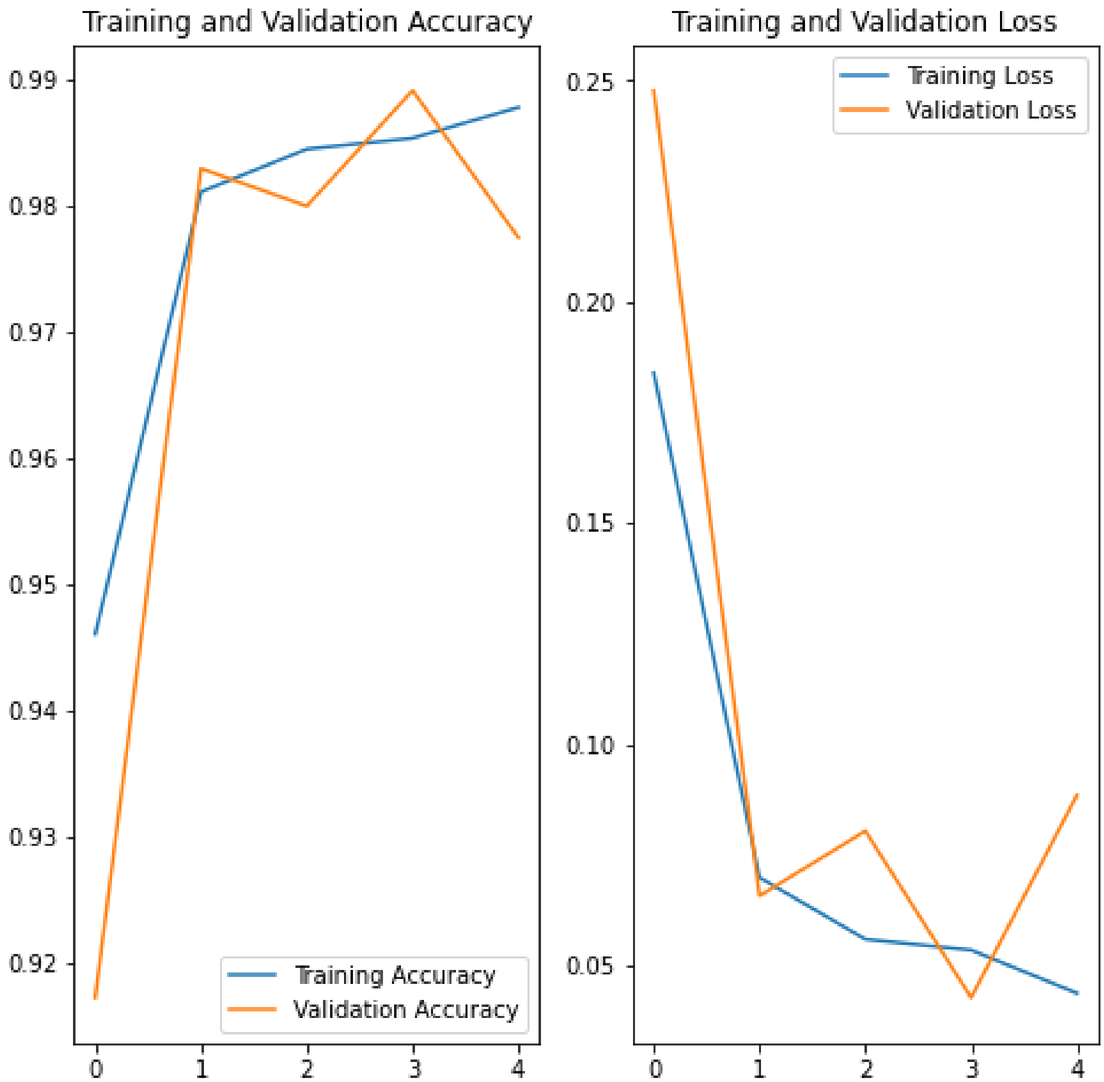
```
from sklearn.metrics import classification_report
print(classification_report(y_test,predictions))
```

```
              precision    recall  f1-score   support

           0       1.00      0.99      0.99      1777
           1       0.97      1.00      0.98      2023
           2       1.00      0.97      0.98      1787
           3       0.95      1.00      0.97      1839
           4       1.00      0.94      0.97      1753
           5       0.99      0.98      0.99      1626
           6       0.99      0.99      0.99      1775
           7       0.92      0.99      0.96      1880
           8       1.00      0.95      0.97      1755
           9       0.98      0.95      0.97      1785

    accuracy                           0.98     18000
   macro avg       0.98      0.98      0.98     18000
weighted avg       0.98      0.98      0.98     18000
```

[19]:

```
acc = history.history["accuracy"]
val_acc = history.history["val_accuracy"]

loss = history.history["loss"]
val_loss = history.history["val_loss"]
epochs = epochs
epochs_range = range(epochs)

plt.figure(figsize=(8, 8))
plt.subplot(1, 2, 1)
plt.plot(epochs_range, acc, label="Training Accuracy")
plt.plot(epochs_range, val_acc, label="Validation Accuracy")
plt.legend(loc="lower right")
plt.title("Training and Validation Accuracy")
```

```
plt.subplot(1, 2, 2)
plt.plot(epochs_range, loss, label="Training Loss")
plt.plot(epochs_range, val_loss, label="Validation Loss")
plt.legend(loc="upper right")
plt.title("Training and Validation Loss")
plt.show()
```



0.2 Building a CNN from Scratch

[20]:
```
# num_classes = 10

model2 = Sequential(
    [
```

```
        layers.InputLayer((75,75,3)),

        layers.Conv2D(16, 3, padding="same", activation="relu"),
        layers.MaxPooling2D(),
        layers.Conv2D(32, 3, padding="same", activation="relu"),
        layers.MaxPooling2D(),
        layers.Dropout(0.2),
        layers.Conv2D(64, 3, padding="same", activation="relu"),
        layers.MaxPooling2D(),


        layers.Flatten(),
        layers.Dense(512, activation="relu"),
        layers.Dense(128, activation="relu"),
        layers.Dense(128, activation="relu"),
        layers.Dense(10,activation='softmax'),
    ]
)


model2.compile(optimizer=keras.optimizers.Adam(learning_rate=0.0001),␣
 ↪loss='categorical_crossentropy', metrics=['accuracy'])
```

[21]:
```
model2.summary()
```

```
Model: "sequential_1"
_________________________________________________________________
Layer (type)                 Output Shape              Param #
=================================================================
conv2d (Conv2D)              (None, 75, 75, 16)        448
_________________________________________________________________
max_pooling2d (MaxPooling2D) (None, 37, 37, 16)        0
_________________________________________________________________
conv2d_1 (Conv2D)            (None, 37, 37, 32)        4640
_________________________________________________________________
max_pooling2d_1 (MaxPooling2 (None, 18, 18, 32)        0
_________________________________________________________________
dropout (Dropout)            (None, 18, 18, 32)        0
_________________________________________________________________
conv2d_2 (Conv2D)            (None, 18, 18, 64)        18496
_________________________________________________________________
max_pooling2d_2 (MaxPooling2 (None, 9, 9, 64)          0
_________________________________________________________________
flatten_1 (Flatten)          (None, 5184)              0
_________________________________________________________________
dense_3 (Dense)              (None, 512)               2654720
_________________________________________________________________
dense_4 (Dense)              (None, 128)               65664
```

```
_________________________________________________________________
dense_5 (Dense)              (None, 128)               16512
_________________________________________________________________
dense_6 (Dense)              (None, 10)                1290
=================================================================
Total params: 2,761,770
Trainable params: 2,761,770
Non-trainable params: 0
_________________________________________________________________
```

[22]:

```
epochs2 = 10
history = model2.fit(train_generator, validation_data=valid_generator,␣
 ↪epochs=epochs2,
          steps_per_epoch=train_generator.n//train_generator.batch_size,
         validation_steps=valid_generator.n//valid_generator.batch_size)
```

```
Epoch 1/10
1050/1050 [==============================] - 51s 47ms/step - loss: 0.7384 -
accuracy: 0.7547 - val_loss: 0.3290 - val_accuracy: 0.8976
Epoch 2/10
1050/1050 [==============================] - 49s 47ms/step - loss: 0.2546 -
accuracy: 0.9204 - val_loss: 0.2190 - val_accuracy: 0.9352
Epoch 3/10
1050/1050 [==============================] - 50s 48ms/step - loss: 0.1895 -
accuracy: 0.9411 - val_loss: 0.1824 - val_accuracy: 0.9445
Epoch 4/10
1050/1050 [==============================] - 50s 47ms/step - loss: 0.1559 -
accuracy: 0.9513 - val_loss: 0.1453 - val_accuracy: 0.9543
Epoch 5/10
1050/1050 [==============================] - 50s 47ms/step - loss: 0.1277 -
accuracy: 0.9601 - val_loss: 0.1311 - val_accuracy: 0.9585
Epoch 6/10
1050/1050 [==============================] - 49s 47ms/step - loss: 0.1211 -
accuracy: 0.9623 - val_loss: 0.1157 - val_accuracy: 0.9651
Epoch 7/10
1050/1050 [==============================] - 50s 47ms/step - loss: 0.1050 -
accuracy: 0.9669 - val_loss: 0.1045 - val_accuracy: 0.9670
Epoch 8/10
1050/1050 [==============================] - 49s 47ms/step - loss: 0.0936 -
accuracy: 0.9701 - val_loss: 0.0829 - val_accuracy: 0.9743
Epoch 9/10
1050/1050 [==============================] - 49s 47ms/step - loss: 0.0882 -
accuracy: 0.9718 - val_loss: 0.0812 - val_accuracy: 0.9761
Epoch 10/10
1050/1050 [==============================] - 49s 47ms/step - loss: 0.0798 -
accuracy: 0.9749 - val_loss: 0.0860 - val_accuracy: 0.9750
```

[23]:
```python
pred = model2.predict(test_generator)
predictions = np.argmax(pred,axis = 1)
```

[24]:
```python
tf.math.confusion_matrix(y_test,predictions)
```

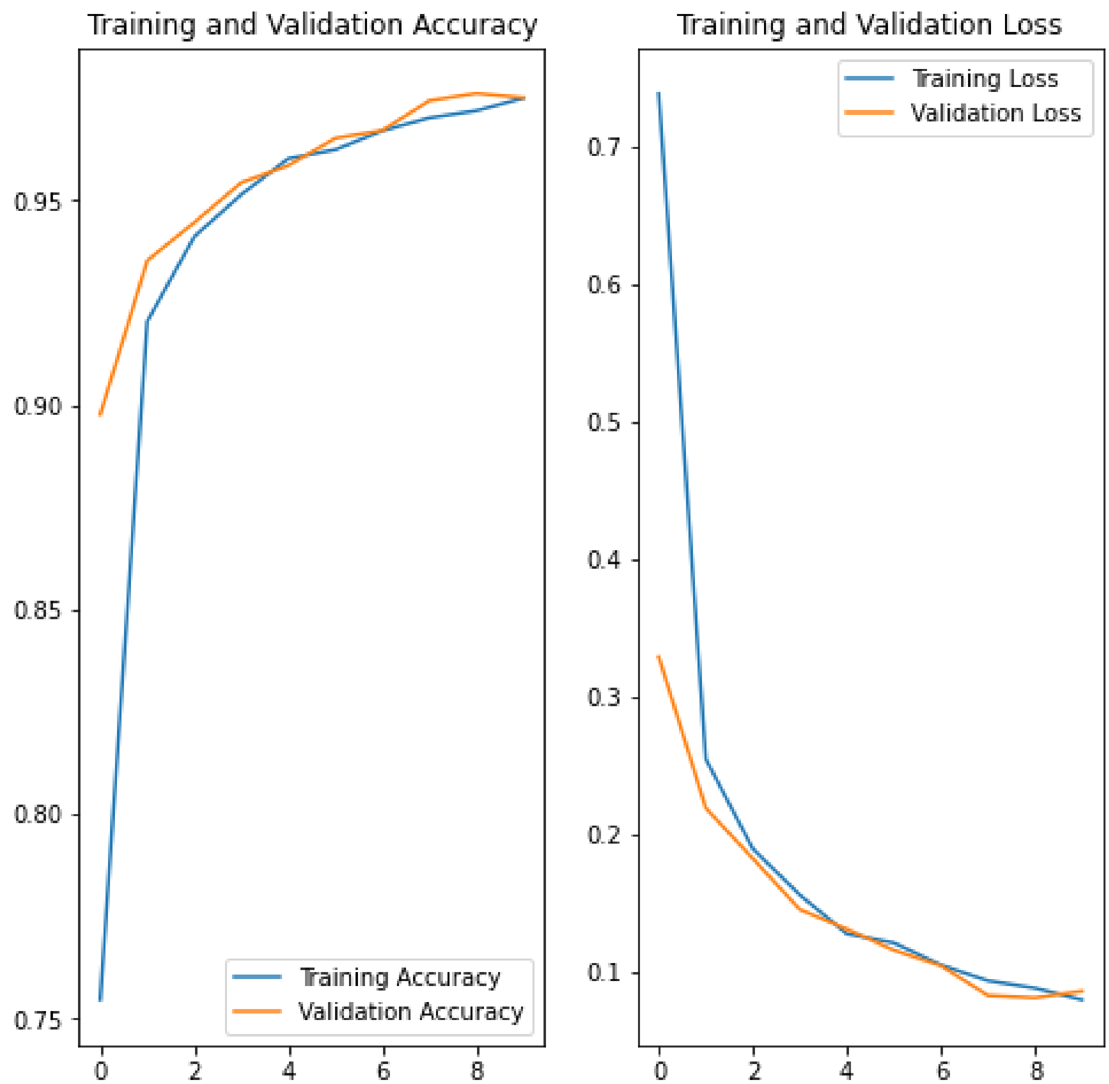
[24]:
```
<tf.Tensor: shape=(10, 10), dtype=int32, numpy=
array([[1741,    1,    2,    0,    1,    4,   15,    3,    6,    4],
       [   1, 2003,    0,    2,    9,    1,    1,    5,    1,    0],
       [   5,   15, 1698,   15,    9,    3,    1,   23,   15,    3],
       [   1,    2,   13, 1792,    0,   16,    0,    4,    7,    4],
       [   1,    2,    0,    0, 1727,    1,    4,    2,    1,   15],
       [   1,    0,    1,    4,    2, 1596,   11,    0,    7,    4],
       [   2,    3,    1,    0,    4,   11, 1752,    0,    2,    0],
       [   0,    4,    5,    6,    9,    3,    0, 1842,    1,   10],
       [   8,    3,    3,    5,    7,   15,   16,    2, 1672,   24],
       [   4,    3,    2,    6,   25,   13,    0,   13,    3, 1716]])>
```

[25]:
```python
from sklearn.metrics import classification_report
print(classification_report(y_test,predictions))
```

```
              precision    recall  f1-score   support

           0       0.99      0.98      0.98      1777
           1       0.98      0.99      0.99      2023
           2       0.98      0.95      0.97      1787
           3       0.98      0.97      0.98      1839
           4       0.96      0.99      0.97      1753
           5       0.96      0.98      0.97      1626
           6       0.97      0.99      0.98      1775
           7       0.97      0.98      0.98      1880
           8       0.97      0.95      0.96      1755
           9       0.96      0.96      0.96      1785

    accuracy                           0.97     18000
   macro avg       0.97      0.97      0.97     18000
weighted avg       0.97      0.97      0.97     18000
```

[26]:
```python
acc = history.history["accuracy"]
val_acc = history.history["val_accuracy"]

loss = history.history["loss"]
val_loss = history.history["val_loss"]
epochs = epochs2
epochs_range = range(epochs)

plt.figure(figsize=(8, 8))
plt.subplot(1, 2, 1)
```

```
plt.plot(epochs_range, acc, label="Training Accuracy")
plt.plot(epochs_range, val_acc, label="Validation Accuracy")
plt.legend(loc="lower right")
plt.title("Training and Validation Accuracy")

plt.subplot(1, 2, 2)
plt.plot(epochs_range, loss, label="Training Loss")
plt.plot(epochs_range, val_loss, label="Validation Loss")
plt.legend(loc="upper right")
plt.title("Training and Validation Loss")
plt.show()
```



From the data we can see that the CNN based Classifiers perform much better compared to the SVM classifier. This is the result of using complex interlinked neuralnets which in theory can approximate any function perfectly(Universal approximation Theorem).

[27]:
```
model2.save('model2.h5')
```

[28]:
```
model2.save_weights('model2_weights.h5')
```

0.3 Q3 Training SVM on CNN Features

[2]:
```
# path ="archive (1)/real_and_f#renaming real and fake directories
real = "archive/real_and_fake_face_detection/real_and_fake_face/training_real"
fake = "archive/real_and_fake_face_detection/real_and_fake_face/training_fake"
#we're creating a list of real and fake images
real_path = os.listdir(real)
fake_path = os.listdir(fake)
```

[3]:
```
img_size = int(75)
def create_data():
    training_data = []
    y=[]
    for img in tqdm(real_path):
        path = os.path.join(real, img)
#         label = [1]
        try:
            image = cv2.resize( cv2.imread(path), (img_size,img_size) )
            training_data.append(np.array(image))
            y.append(1)
        except:
            continue

    for img in tqdm(fake_path):
        path = os.path.join(fake, img)
#         label = [0]
        try:
            image = cv2.resize(cv2.imread(path), (img_size,img_size))
            training_data.append(np.array(image))
            y.append(0)
        except: continue

    return(training_data,y)




data,labels = create_data()
```

```
100%|
 | 1081/1081 [00:08<00:00, 124.58it/s]
100%|
```

```
| 961/961 [00:07<00:00, 127.58it/s]
```

[4]:
```
data = np.array(data)
labels = np.array(labels)
```

[5]:
```
len(data)
```

[5]: 2041

[6]:
```
from sklearn.model_selection import train_test_split
X_train, X_test, y_train, y_test =␣
↪train_test_split(data,labels,random_state=6,test_size=0.3,stratify=labels)
```

[7]:
```
y_trainn = tf.keras.utils.to_categorical(y_train)
y_testc = tf.keras.utils.to_categorical(y_test)
image_gen = ImageDataGenerator(rescale=1./255, #easier for network to interpret␣
↪numbers in range [0,1]
                              zoom_range=0.1,
                              width_shift_range=0.2,
                              height_shift_range=0.2,
                              validation_split=0.2) # 80/20 train/val split
```

[8]:
```
X_train.shape,y_trainn.shape
```

[8]: ((1428, 75, 75, 3), (1428, 2))

[9]:
```
# num_classes = 10

model3 = Sequential(
    [
        layers.InputLayer((75,75,3)),

        layers.Conv2D(16, 3, padding="same", activation="relu"),
        layers.MaxPooling2D(),
        layers.Conv2D(32, 3, padding="same", activation="relu"),
        layers.MaxPooling2D(),
        layers.Conv2D(64, 3, padding="same", activation="relu"),
        layers.MaxPooling2D(),
        layers.Dropout(0.2),
        layers.Conv2D(128, 3, padding="same", activation="relu"),
        layers.MaxPooling2D(),


        layers.Flatten(),
        layers.Dense(128, activation="relu"),
        layers.Dropout(0.2),
        layers.Dense(256, activation="relu"),
        layers.Dense(512, activation="relu"),
```

```
        layers.Dense(1024, activation="relu"),
        layers.Dense(2,activation='softmax'),
    ]
)


model3.compile(optimizer=keras.optimizers.Adam(learning_rate=0.0001), loss=tf.
 ↪keras.losses.binary_crossentropy, metrics=['accuracy'])
```

[10]:
```
model3.summary()
```

```
Model: "sequential"
_________________________________________________________________
Layer (type)                 Output Shape              Param #
=================================================================
conv2d (Conv2D)              (None, 75, 75, 16)        448
_________________________________________________________________
max_pooling2d (MaxPooling2D) (None, 37, 37, 16)        0
_________________________________________________________________
conv2d_1 (Conv2D)            (None, 37, 37, 32)        4640
_________________________________________________________________
max_pooling2d_1 (MaxPooling2 (None, 18, 18, 32)        0
_________________________________________________________________
conv2d_2 (Conv2D)            (None, 18, 18, 64)        18496
_________________________________________________________________
max_pooling2d_2 (MaxPooling2 (None, 9, 9, 64)          0
_________________________________________________________________
dropout (Dropout)            (None, 9, 9, 64)          0
_________________________________________________________________
conv2d_3 (Conv2D)            (None, 9, 9, 128)         73856
_________________________________________________________________
max_pooling2d_3 (MaxPooling2 (None, 4, 4, 128)         0
_________________________________________________________________
flatten (Flatten)            (None, 2048)              0
_________________________________________________________________
dense (Dense)                (None, 128)               262272
_________________________________________________________________
dropout_1 (Dropout)          (None, 128)               0
_________________________________________________________________
dense_1 (Dense)              (None, 256)               33024
_________________________________________________________________
dense_2 (Dense)              (None, 512)               131584
_________________________________________________________________
dense_3 (Dense)              (None, 1024)              525312
_________________________________________________________________
dense_4 (Dense)              (None, 2)                 2050
=================================================================
Total params: 1,051,682
```

```
Trainable params: 1,051,682
Non-trainable params: 0
_________________________________________________________________
```

[11]:
```
history = model3.fit(X_train,y_trainn,epochs = 100,verbose =
↪1,validation_split=0.1)
```

```
Epoch 1/100
41/41 [==============================] - 4s 23ms/step - loss: 0.9631 - accuracy:
0.5136 - val_loss: 0.7491 - val_accuracy: 0.4685
Epoch 2/100
41/41 [==============================] - 0s 9ms/step - loss: 0.7341 - accuracy:
0.5315 - val_loss: 0.7215 - val_accuracy: 0.4476
Epoch 3/100
41/41 [==============================] - 0s 9ms/step - loss: 0.7110 - accuracy:
0.5268 - val_loss: 0.6981 - val_accuracy: 0.5245
Epoch 4/100
41/41 [==============================] - 0s 9ms/step - loss: 0.7104 - accuracy:
0.5268 - val_loss: 0.7078 - val_accuracy: 0.5245
Epoch 5/100
41/41 [==============================] - 0s 9ms/step - loss: 0.7047 - accuracy:
0.5198 - val_loss: 0.7227 - val_accuracy: 0.4545
Epoch 6/100
41/41 [==============================] - 0s 9ms/step - loss: 0.7006 - accuracy:
0.5113 - val_loss: 0.6914 - val_accuracy: 0.5315
Epoch 7/100
41/41 [==============================] - 0s 9ms/step - loss: 0.6889 - accuracy:
0.5549 - val_loss: 0.7050 - val_accuracy: 0.4545
Epoch 8/100
41/41 [==============================] - 0s 9ms/step - loss: 0.6910 - accuracy:
0.5237 - val_loss: 0.7044 - val_accuracy: 0.5245
Epoch 9/100
41/41 [==============================] - 0s 9ms/step - loss: 0.6920 - accuracy:
0.5121 - val_loss: 0.6984 - val_accuracy: 0.4895
Epoch 10/100
41/41 [==============================] - 0s 9ms/step - loss: 0.6886 - accuracy:
0.5525 - val_loss: 0.7019 - val_accuracy: 0.5105
Epoch 11/100
41/41 [==============================] - 0s 9ms/step - loss: 0.6866 - accuracy:
0.5541 - val_loss: 0.6986 - val_accuracy: 0.4825
Epoch 12/100
41/41 [==============================] - 0s 9ms/step - loss: 0.6778 - accuracy:
0.5813 - val_loss: 0.6892 - val_accuracy: 0.5455
Epoch 13/100
41/41 [==============================] - 0s 9ms/step - loss: 0.6866 - accuracy:
0.5564 - val_loss: 0.6891 - val_accuracy: 0.5385
Epoch 14/100
41/41 [==============================] - 0s 9ms/step - loss: 0.6760 - accuracy:
```

```
0.5798 - val_loss: 0.6803 - val_accuracy: 0.5524
Epoch 15/100
41/41 [==============================] - 0s 9ms/step - loss: 0.6662 - accuracy:
0.5899 - val_loss: 0.6899 - val_accuracy: 0.5594
Epoch 16/100
41/41 [==============================] - 0s 9ms/step - loss: 0.6822 - accuracy:
0.5767 - val_loss: 0.6908 - val_accuracy: 0.5594
Epoch 17/100
41/41 [==============================] - 0s 9ms/step - loss: 0.6915 - accuracy:
0.5424 - val_loss: 0.6898 - val_accuracy: 0.5105
Epoch 18/100
41/41 [==============================] - 0s 9ms/step - loss: 0.6737 - accuracy:
0.5673 - val_loss: 0.6831 - val_accuracy: 0.5245
Epoch 19/100
41/41 [==============================] - 0s 9ms/step - loss: 0.6611 - accuracy:
0.6031 - val_loss: 0.6715 - val_accuracy: 0.5804
Epoch 20/100
41/41 [==============================] - 0s 9ms/step - loss: 0.6628 - accuracy:
0.5813 - val_loss: 0.6931 - val_accuracy: 0.5594
Epoch 21/100
41/41 [==============================] - 0s 9ms/step - loss: 0.6618 - accuracy:
0.6031 - val_loss: 0.6789 - val_accuracy: 0.5524
Epoch 22/100
41/41 [==============================] - 0s 9ms/step - loss: 0.6333 - accuracy:
0.6304 - val_loss: 0.6804 - val_accuracy: 0.6084
Epoch 23/100
41/41 [==============================] - 0s 9ms/step - loss: 0.6396 - accuracy:
0.6272 - val_loss: 0.6799 - val_accuracy: 0.5734
Epoch 24/100
41/41 [==============================] - 0s 9ms/step - loss: 0.6395 - accuracy:
0.6226 - val_loss: 0.6655 - val_accuracy: 0.5874
Epoch 25/100
41/41 [==============================] - 0s 9ms/step - loss: 0.6336 - accuracy:
0.6397 - val_loss: 0.6721 - val_accuracy: 0.6154
Epoch 26/100
41/41 [==============================] - 0s 9ms/step - loss: 0.6104 - accuracy:
0.6700 - val_loss: 0.6671 - val_accuracy: 0.6154
Epoch 27/100
41/41 [==============================] - 0s 9ms/step - loss: 0.6303 - accuracy:
0.6327 - val_loss: 0.6548 - val_accuracy: 0.6434
Epoch 28/100
41/41 [==============================] - 0s 9ms/step - loss: 0.6133 - accuracy:
0.6545 - val_loss: 0.6523 - val_accuracy: 0.6364
Epoch 29/100
41/41 [==============================] - 0s 9ms/step - loss: 0.5781 - accuracy:
0.6934 - val_loss: 0.6756 - val_accuracy: 0.6294
Epoch 30/100
41/41 [==============================] - 0s 9ms/step - loss: 0.5794 - accuracy:
```

```
0.6669 - val_loss: 0.6633 - val_accuracy: 0.6154
Epoch 31/100
41/41 [==============================] - 0s 9ms/step - loss: 0.5544 - accuracy:
0.7019 - val_loss: 0.6804 - val_accuracy: 0.6643
Epoch 32/100
41/41 [==============================] - 0s 9ms/step - loss: 0.5623 - accuracy:
0.7027 - val_loss: 0.6661 - val_accuracy: 0.6154
Epoch 33/100
41/41 [==============================] - 0s 9ms/step - loss: 0.5244 - accuracy:
0.7284 - val_loss: 0.6938 - val_accuracy: 0.6294
Epoch 34/100
41/41 [==============================] - 0s 9ms/step - loss: 0.5338 - accuracy:
0.7136 - val_loss: 0.7022 - val_accuracy: 0.6014
Epoch 35/100
41/41 [==============================] - 0s 9ms/step - loss: 0.5099 - accuracy:
0.7416 - val_loss: 0.7158 - val_accuracy: 0.6224
Epoch 36/100
41/41 [==============================] - 0s 9ms/step - loss: 0.5233 - accuracy:
0.7268 - val_loss: 0.7013 - val_accuracy: 0.6084
Epoch 37/100
41/41 [==============================] - 0s 9ms/step - loss: 0.4933 - accuracy:
0.7556 - val_loss: 0.7095 - val_accuracy: 0.6014
Epoch 38/100
41/41 [==============================] - 0s 9ms/step - loss: 0.4514 - accuracy:
0.7813 - val_loss: 0.7674 - val_accuracy: 0.6224
Epoch 39/100
41/41 [==============================] - 0s 9ms/step - loss: 0.4417 - accuracy:
0.7798 - val_loss: 0.8628 - val_accuracy: 0.5385
Epoch 40/100
41/41 [==============================] - 0s 9ms/step - loss: 0.4412 - accuracy:
0.7844 - val_loss: 0.7304 - val_accuracy: 0.6573
Epoch 41/100
41/41 [==============================] - 0s 9ms/step - loss: 0.4169 - accuracy:
0.8093 - val_loss: 0.7422 - val_accuracy: 0.6154
Epoch 42/100
41/41 [==============================] - 0s 9ms/step - loss: 0.3978 - accuracy:
0.8039 - val_loss: 0.7913 - val_accuracy: 0.5594
Epoch 43/100
41/41 [==============================] - 0s 10ms/step - loss: 0.3781 - accuracy:
0.8202 - val_loss: 0.8408 - val_accuracy: 0.5804
Epoch 44/100
41/41 [==============================] - 0s 9ms/step - loss: 0.3872 - accuracy:
0.8031 - val_loss: 0.9138 - val_accuracy: 0.5874
Epoch 45/100
41/41 [==============================] - 0s 10ms/step - loss: 0.3507 - accuracy:
0.8358 - val_loss: 0.8799 - val_accuracy: 0.5734
Epoch 46/100
41/41 [==============================] - 0s 9ms/step - loss: 0.3511 - accuracy:
```

```
0.8405 - val_loss: 0.8267 - val_accuracy: 0.6224
Epoch 47/100
41/41 [==============================] - 0s 9ms/step - loss: 0.3208 - accuracy:
0.8482 - val_loss: 0.9313 - val_accuracy: 0.6294
Epoch 48/100
41/41 [==============================] - 0s 9ms/step - loss: 0.3169 - accuracy:
0.8490 - val_loss: 0.8948 - val_accuracy: 0.6154
Epoch 49/100
41/41 [==============================] - 0s 9ms/step - loss: 0.3082 - accuracy:
0.8584 - val_loss: 0.9506 - val_accuracy: 0.6014
Epoch 50/100
41/41 [==============================] - 0s 9ms/step - loss: 0.2880 - accuracy:
0.8794 - val_loss: 0.9768 - val_accuracy: 0.5944
Epoch 51/100
41/41 [==============================] - 0s 10ms/step - loss: 0.2516 - accuracy:
0.8926 - val_loss: 1.0769 - val_accuracy: 0.6154
Epoch 52/100
41/41 [==============================] - 0s 9ms/step - loss: 0.2510 - accuracy:
0.8942 - val_loss: 0.9792 - val_accuracy: 0.6573
Epoch 53/100
41/41 [==============================] - 0s 9ms/step - loss: 0.2429 - accuracy:
0.8918 - val_loss: 0.9493 - val_accuracy: 0.5874
Epoch 54/100
41/41 [==============================] - 0s 9ms/step - loss: 0.2370 - accuracy:
0.8973 - val_loss: 0.9615 - val_accuracy: 0.6154
Epoch 55/100
41/41 [==============================] - 0s 9ms/step - loss: 0.2579 - accuracy:
0.8856 - val_loss: 1.0012 - val_accuracy: 0.6364
Epoch 56/100
41/41 [==============================] - 0s 9ms/step - loss: 0.2034 - accuracy:
0.9222 - val_loss: 1.0855 - val_accuracy: 0.5944
Epoch 57/100
41/41 [==============================] - 0s 9ms/step - loss: 0.2133 - accuracy:
0.9136 - val_loss: 1.0633 - val_accuracy: 0.6364
Epoch 58/100
41/41 [==============================] - 0s 9ms/step - loss: 0.2101 - accuracy:
0.9043 - val_loss: 1.0856 - val_accuracy: 0.6084
Epoch 59/100
41/41 [==============================] - 0s 9ms/step - loss: 0.1831 - accuracy:
0.9214 - val_loss: 1.2164 - val_accuracy: 0.6014
Epoch 60/100
41/41 [==============================] - 0s 9ms/step - loss: 0.1859 - accuracy:
0.9230 - val_loss: 1.1486 - val_accuracy: 0.6014
Epoch 61/100
41/41 [==============================] - 0s 9ms/step - loss: 0.1775 - accuracy:
0.9284 - val_loss: 1.0530 - val_accuracy: 0.6084
Epoch 62/100
41/41 [==============================] - 0s 9ms/step - loss: 0.1767 - accuracy:
```

```
0.9284 - val_loss: 1.2933 - val_accuracy: 0.5804
Epoch 63/100
41/41 [==============================] - 0s 9ms/step - loss: 0.1504 - accuracy:
0.9409 - val_loss: 1.2170 - val_accuracy: 0.5944
Epoch 64/100
41/41 [==============================] - 0s 9ms/step - loss: 0.1394 - accuracy:
0.9455 - val_loss: 1.2989 - val_accuracy: 0.5874
Epoch 65/100
41/41 [==============================] - 0s 9ms/step - loss: 0.1621 - accuracy:
0.9401 - val_loss: 1.2088 - val_accuracy: 0.5804
Epoch 66/100
41/41 [==============================] - 0s 9ms/step - loss: 0.1779 - accuracy:
0.9323 - val_loss: 1.1638 - val_accuracy: 0.6434
Epoch 67/100
41/41 [==============================] - 0s 9ms/step - loss: 0.1277 - accuracy:
0.9556 - val_loss: 1.2015 - val_accuracy: 0.6224
Epoch 68/100
41/41 [==============================] - 0s 9ms/step - loss: 0.1372 - accuracy:
0.9471 - val_loss: 1.2325 - val_accuracy: 0.5874
Epoch 69/100
41/41 [==============================] - 0s 9ms/step - loss: 0.1163 - accuracy:
0.9580 - val_loss: 1.3661 - val_accuracy: 0.6434
Epoch 70/100
41/41 [==============================] - 0s 9ms/step - loss: 0.1102 - accuracy:
0.9549 - val_loss: 1.4536 - val_accuracy: 0.6294
Epoch 71/100
41/41 [==============================] - 0s 9ms/step - loss: 0.1531 - accuracy:
0.9424 - val_loss: 1.1828 - val_accuracy: 0.5944
Epoch 72/100
41/41 [==============================] - 0s 9ms/step - loss: 0.1528 - accuracy:
0.9409 - val_loss: 1.3034 - val_accuracy: 0.6434
Epoch 73/100
41/41 [==============================] - 0s 9ms/step - loss: 0.1002 - accuracy:
0.9626 - val_loss: 1.3573 - val_accuracy: 0.5874
Epoch 74/100
41/41 [==============================] - 0s 10ms/step - loss: 0.1002 - accuracy:
0.9619 - val_loss: 1.3357 - val_accuracy: 0.6224
Epoch 75/100
41/41 [==============================] - 0s 9ms/step - loss: 0.0954 - accuracy:
0.9634 - val_loss: 1.5496 - val_accuracy: 0.5874
Epoch 76/100
41/41 [==============================] - 0s 9ms/step - loss: 0.0952 - accuracy:
0.9634 - val_loss: 1.6474 - val_accuracy: 0.6084
Epoch 77/100
41/41 [==============================] - 0s 9ms/step - loss: 0.0881 - accuracy:
0.9665 - val_loss: 1.6203 - val_accuracy: 0.5734
Epoch 78/100
41/41 [==============================] - 0s 10ms/step - loss: 0.1161 - accuracy:
```

```
0.9541 - val_loss: 1.3598 - val_accuracy: 0.5874
Epoch 79/100
41/41 [==============================] - 0s 9ms/step - loss: 0.0828 - accuracy:
0.9704 - val_loss: 1.6233 - val_accuracy: 0.6154
Epoch 80/100
41/41 [==============================] - 0s 9ms/step - loss: 0.0954 - accuracy:
0.9650 - val_loss: 1.4430 - val_accuracy: 0.6224
Epoch 81/100
41/41 [==============================] - 0s 9ms/step - loss: 0.1118 - accuracy:
0.9588 - val_loss: 1.4387 - val_accuracy: 0.6154
Epoch 82/100
41/41 [==============================] - 0s 9ms/step - loss: 0.0751 - accuracy:
0.9728 - val_loss: 1.8059 - val_accuracy: 0.5944
Epoch 83/100
41/41 [==============================] - 0s 9ms/step - loss: 0.0928 - accuracy:
0.9634 - val_loss: 1.4681 - val_accuracy: 0.6154
Epoch 84/100
41/41 [==============================] - 0s 9ms/step - loss: 0.0946 - accuracy:
0.9619 - val_loss: 1.5775 - val_accuracy: 0.5874
Epoch 85/100
41/41 [==============================] - 0s 9ms/step - loss: 0.0839 - accuracy:
0.9681 - val_loss: 1.5140 - val_accuracy: 0.6014
Epoch 86/100
41/41 [==============================] - 0s 9ms/step - loss: 0.0573 - accuracy:
0.9790 - val_loss: 1.6748 - val_accuracy: 0.5944
Epoch 87/100
41/41 [==============================] - 0s 9ms/step - loss: 0.0660 - accuracy:
0.9751 - val_loss: 1.7580 - val_accuracy: 0.5944
Epoch 88/100
41/41 [==============================] - 0s 9ms/step - loss: 0.0700 - accuracy:
0.9767 - val_loss: 1.7946 - val_accuracy: 0.5874
Epoch 89/100
41/41 [==============================] - 0s 9ms/step - loss: 0.0891 - accuracy:
0.9681 - val_loss: 1.6785 - val_accuracy: 0.5874
Epoch 90/100
41/41 [==============================] - 0s 9ms/step - loss: 0.0498 - accuracy:
0.9782 - val_loss: 1.9690 - val_accuracy: 0.5734
Epoch 91/100
41/41 [==============================] - 0s 9ms/step - loss: 0.0695 - accuracy:
0.9759 - val_loss: 1.6861 - val_accuracy: 0.6154
Epoch 92/100
41/41 [==============================] - 0s 9ms/step - loss: 0.0532 - accuracy:
0.9813 - val_loss: 1.8555 - val_accuracy: 0.6084
Epoch 93/100
41/41 [==============================] - 0s 9ms/step - loss: 0.0757 - accuracy:
0.9728 - val_loss: 1.6787 - val_accuracy: 0.5944
Epoch 94/100
41/41 [==============================] - 0s 9ms/step - loss: 0.1283 - accuracy:
```

```
0.9510 - val_loss: 1.3652 - val_accuracy: 0.6084
Epoch 95/100
41/41 [==============================] - 0s 9ms/step - loss: 0.0923 - accuracy:
0.9650 - val_loss: 1.5103 - val_accuracy: 0.6014
Epoch 96/100
41/41 [==============================] - 0s 9ms/step - loss: 0.0573 - accuracy:
0.9767 - val_loss: 1.6472 - val_accuracy: 0.5944
Epoch 97/100
41/41 [==============================] - 0s 10ms/step - loss: 0.0538 - accuracy:
0.9813 - val_loss: 1.6313 - val_accuracy: 0.6084
Epoch 98/100
41/41 [==============================] - 0s 9ms/step - loss: 0.0311 - accuracy:
0.9907 - val_loss: 1.8636 - val_accuracy: 0.6154
Epoch 99/100
41/41 [==============================] - 0s 9ms/step - loss: 0.0598 - accuracy:
0.9767 - val_loss: 1.5896 - val_accuracy: 0.6154
Epoch 100/100
41/41 [==============================] - 0s 9ms/step - loss: 0.0598 - accuracy:
0.9751 - val_loss: 1.7902 - val_accuracy: 0.5804
```

[12]:

```
# Taking the CNN outputs and feeding to SVM
model_feat = Model(inputs=model3.input,outputs=model3.get_layer('dense_3').
 ↪output)

feat_train = model_feat.predict(X_train)
print(feat_train.shape)

feat_test = model_feat.predict(X_test)
print(feat_test.shape)
```

```
(1428, 1024)
(613, 1024)
```

[ ]:

```
```

[13]:

```
from sklearn.svm import SVC

svm = SVC(kernel='rbf')

svm.fit(feat_train,np.argmax(y_trainn,axis=1))

print('Training on SVM complete')
```

```
Training on SVM complete
```

[14]:

```
svm.score(feat_train,np.argmax(y_trainn,axis=1))
```

[14]: 0.9642857142857143

[15]:

```
predictions = svm.predict(feat_test)
from sklearn.metrics import classification_report
print(classification_report(y_test,predictions))
```

```
              precision    recall  f1-score   support

           0       0.56      0.59      0.57       288
           1       0.62      0.60      0.61       325

    accuracy                           0.59       613
   macro avg       0.59      0.59      0.59       613
weighted avg       0.59      0.59      0.59       613
```

[16]:

```
model3.save('model3.h5')
model3.save_weights('model3_weights.h5')
```

[ ]:

```
```