3. Facial Keypoint Detection, Complete Pipeline

May 5, 2024

0.1 Face and Facial Keypoint detection

After you've trained a neural network to detect facial keypoints, you can then apply this network to *any* image that includes faces. The neural network expects a Tensor of a certain size as input and, so, to detect any face, you'll first have to do some pre-processing.

1. Detect all the faces in an image using a face detector (we'll be using a Haar Cascade detector in this notebook).
2. Pre-process those face images so that they are grayscale, and transformed to a Tensor of the input size that your net expects. This step will be similar to the `data_transform` you created and applied in Notebook 2, whose job was tp rescale, normalize, and turn any image into a Tensor to be accepted as input to your CNN.
3. Use your trained model to detect facial keypoints on the image.

In the next python cell we load in required libraries for this section of the project.

[1]:
```
import numpy as np
import matplotlib.pyplot as plt
import matplotlib.image as mpimg
%matplotlib inline
```

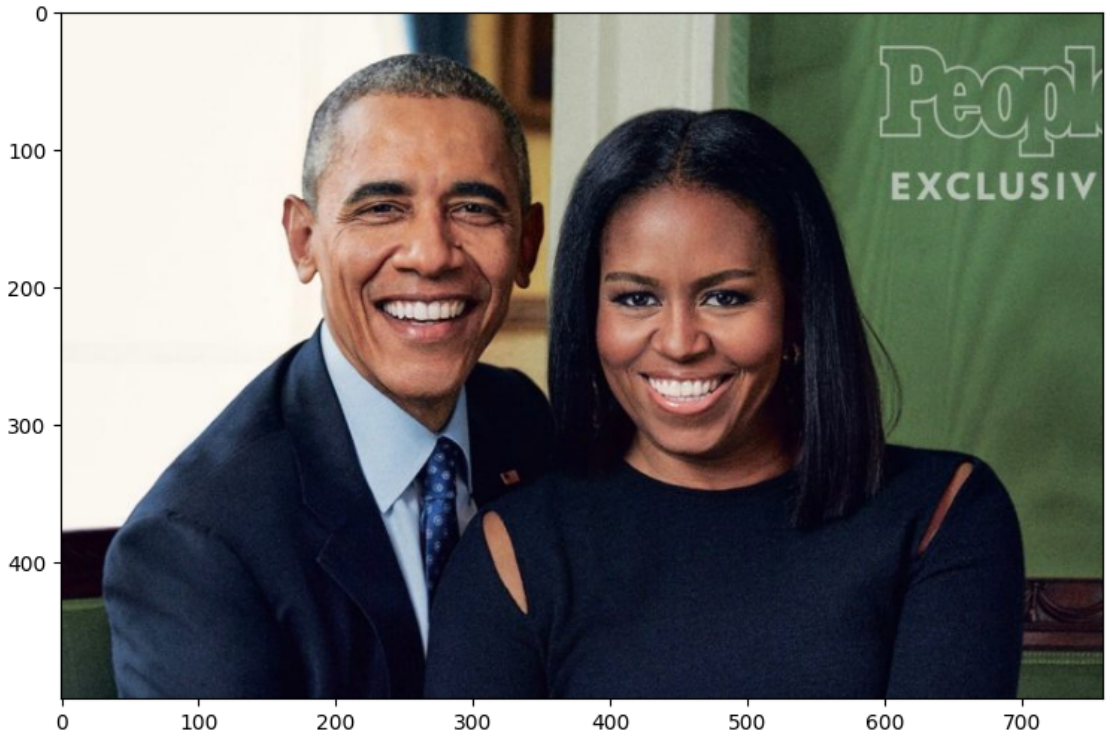
Select an image Select an image to perform facial keypoint detection on; you can select any image of faces in the `images/` directory.

[2]:
```
import cv2
# load in color image for face detection
image = cv2.imread('images/obamas.jpg')

# switch red and blue color channels
# --> by default OpenCV assumes BLUE comes first, not RED as in many images
image = cv2.cvtColor(image, cv2.COLOR_BGR2RGB)

# plot the image
fig = plt.figure(figsize=(9,9))
plt.imshow(image)
```

```
/usr/local/lib/python3.10/dist-packages/numpy/core/getlimits.py:518:
UserWarning: The value of the smallest subnormal for <class 'numpy.float64'>
```

```
type is zero.
  setattr(self, word, getattr(machar, word).flat[0])
/usr/local/lib/python3.10/dist-packages/numpy/core/getlimits.py:89: UserWarning:
The value of the smallest subnormal for <class 'numpy.float64'> type is zero.
  return self._float_to_str(self.smallest_subnormal)
```

[2]: `<matplotlib.image.AxesImage at 0x7f2d310dc4f0>`



0.2 Detect all faces in an image

Next, you'll use one of OpenCV's pre-trained Haar Cascade classifiers, all of which can be found in the `detector_architectures/` directory, to find any faces in your selected image.

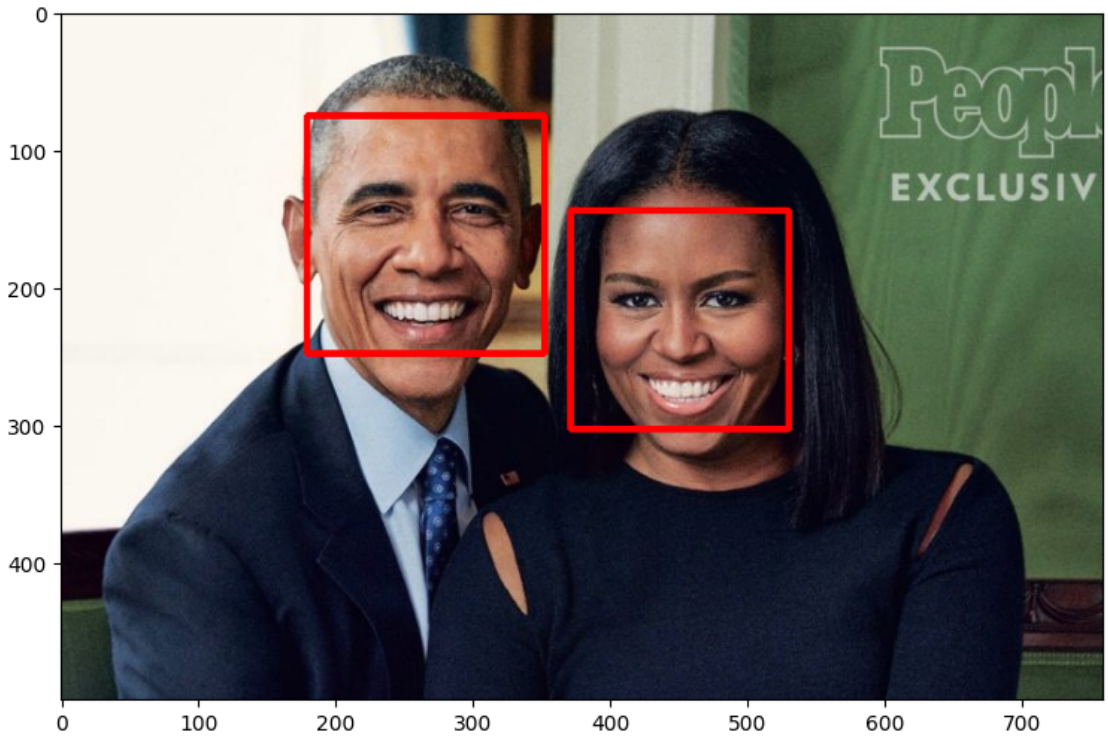
In the code below, we loop over each face in the original image and draw a red square on each face (in a copy of the original image, so as not to modify the original). You can even add eye detections as an *optional* exercise in using Haar detectors.

An example of face detection on a variety of images is shown below.

[3]:
```
# load in a haar cascade classifier for detecting frontal faces
face_cascade = cv2.CascadeClassifier('detector_architectures/
  ↪haarcascade_frontalface_default.xml')

# run the detector
```

```
# the output here is an array of detections; the corners of each detection box
# if necessary, modify these parameters until you successfully identify every␣
 ↪face in a given image
faces = face_cascade.detectMultiScale(image, 1.2, 2)

# make a copy of the original image to plot detections on
image_with_detections = image.copy()

# loop over the detected faces, mark the image where each face is found
for (x,y,w,h) in faces:
    # draw a rectangle around each detected face
    # you may also need to change the width of the rectangle drawn depending on␣
 ↪image resolution
    cv2.rectangle(image_with_detections,(x,y),(x+w,y+h),(255,0,0),3)

fig = plt.figure(figsize=(9,9))

plt.imshow(image_with_detections)
```

[3]: <matplotlib.image.AxesImage at 0x7f2d24f289d0>



0.3 Loading in a trained model

Once you have an image to work with (and, again, you can select any image of faces in the `images/` directory), the next step is to pre-process that image and feed it into your CNN facial keypoint detector.

First, load your best model by its filename.

[4]:
```
import torch
from models import Net

net = Net()

## TODO: load the best saved model parameters (by your path name)
## You'll need to un-comment the line below and add the correct name for *your* saved model
# net.load_state_dict(torch.load('saved_models/keypoints_model_1.pt'))

model_dir = 'saved_models/'
# model_name = 'keypoints_model_8.pt'
model_name = 'keypoints_model_001.pt'

net.load_state_dict(torch.load(model_dir+model_name))

## print out your net and prepare it for testing (uncomment the line below)
# net.eval()
net.eval()
```

[4]:
```
Net(
  (conv1): Sequential(
    (0): Conv2d(1, 32, kernel_size=(3, 3), stride=(1, 1))
    (1): ELU(alpha=1.0)
    (2): Dropout(p=0.1, inplace=False)
    (3): MaxPool2d(kernel_size=2, stride=2, padding=0, dilation=1,
ceil_mode=False)
  )
  (conv2): Sequential(
    (0): Conv2d(32, 64, kernel_size=(3, 3), stride=(1, 1))
    (1): ELU(alpha=1.0)
    (2): Dropout(p=0.2, inplace=False)
    (3): MaxPool2d(kernel_size=2, stride=2, padding=0, dilation=1,
ceil_mode=False)
  )
  (conv3): Sequential(
    (0): Conv2d(64, 128, kernel_size=(3, 3), stride=(1, 1))
    (1): ELU(alpha=1.0)
    (2): Dropout(p=0.3, inplace=False)
    (3): MaxPool2d(kernel_size=2, stride=2, padding=0, dilation=1,
```

```
ceil_mode=False)
  )
  (conv4): Sequential(
    (0): Conv2d(128, 256, kernel_size=(3, 3), stride=(1, 1))
    (1): ELU(alpha=1.0)
    (2): Dropout(p=0.4, inplace=False)
    (3): MaxPool2d(kernel_size=2, stride=2, padding=0, dilation=1,
ceil_mode=False)
  )
  (fc1): Sequential(
    (0): Flatten()
    (1): Linear(in_features=36864, out_features=1000, bias=True)
    (2): ELU(alpha=1.0)
    (3): Dropout(p=0.5, inplace=False)
  )
  (fc2): Sequential(
    (0): Linear(in_features=1000, out_features=1000, bias=True)
    (1): Dropout(p=0.6, inplace=False)
  )
  (fc3): Sequential(
    (0): Linear(in_features=1000, out_features=136, bias=True)
    (1): CustomizedSigmoid()
  )
)
```

0.4 Keypoint detection

Now, we'll loop over each detected face in an image (again!) only this time, you'll transform those faces in Tensors that your CNN can accept as input images.

0.4.1 TODO: Transform each detected face into an input Tensor

You'll need to perform the following steps for each detected face: 1. Convert the face from RGB to grayscale 2. Normalize the grayscale image so that its color range falls in [0,1] instead of [0,255] 3. Rescale the detected face to be the expected square size for your CNN (224x224, suggested) 4. Reshape the numpy image into a torch image.

You may find it useful to consult to transformation code in `data_load.py` to help you perform these processing steps.

0.4.2 TODO: Detect and display the predicted keypoints

After each face has been appropriately converted into an input Tensor for your network to see as input, you'll wrap that Tensor in a Variable() and can apply your `net` to each face. The ouput should be the predicted the facial keypoints. These keypoints will need to be "un-normalized" for display, and you may find it helpful to write a helper function like `show_keypoints`. You should end up with an image like the following with facial keypoints that closely match the facial features on each individual face:

[7]:
```
from torchvision import transforms

##############################################################################################
##### Instructions on how to use GPU on Pytorch, using CUDA, was obtained from␣
 ↪the link below :
##### https://www.run.ai/guides/gpu-deep-learning/pytorch-gpu
##############################################################################################

# dev = "cpu"

if torch.cuda.is_available():
    dev = "cuda:0"
else:
    dev = "cpu"
print("torch.cuda.is_available() = "+str(torch.cuda.is_available()))
device = torch.device(dev)

net = net.to(device)

##############################################################################################
##############################################################################################
##############################################################################################

def net_sample_output(image):

    # Ensure image is a tensor
    if not isinstance(image, torch.Tensor):
        image = torch.tensor(image)  # Convert to tensor if necessary

    # Add batch dimension if missing
    if len(image.shape) == 3:
        image = image.unsqueeze(0)  # Add batch dimension

    # Convert to FloatTensor and move to device
    image = image.type(torch.FloatTensor).to(device)

    # Forward pass to get net output
    output_pts = net(image)

    # Reshape to 68 x 2 keypoints
    output_pts = output_pts.view(-1, 68, 2)

    # Remove batch dimension
    image = image.squeeze(0)

    return image, output_pts
```

```
def show_all_keypoints(image, predicted_key_pts, gt_pts=None):
    """Show image with predicted keypoints"""
    # image is grayscale
    plt.imshow(image, cmap='gray')
    plt.scatter(predicted_key_pts[:, 0], predicted_key_pts[:, 1], s=20,␣
 ↪marker='.', c='m')
    # plot ground truth points as green pts
    if gt_pts is not None:
        plt.scatter(gt_pts[:, 0], gt_pts[:, 1], s=20, marker='.', c='g')

def visualize_output(test_images, test_outputs, gt_pts=None, batch_size=1):

    for i in range(batch_size):
        plt.figure(figsize=(20,10))
        ax = plt.subplot(1, batch_size, i+1)

        # un-transform the image data

        if (dev == "cuda:0"):
            # Move image to CPU before conversion to numpy
            image = test_images[i].data.cpu()  # Move to CPU first
        else:
            image = test_images[i].data   # get the image from it's Variable␣
 ↪wrapper
        image = image.numpy()   # convert to numpy array from a Tensor
        image = np.transpose(image, (1, 2, 0))   # transpose to go from torch␣
 ↪to numpy image

        # un-transform the predicted key_pts data
        if (dev == "cuda:0"):
            predicted_key_pts = test_outputs[i].data.cpu()
        else:
            predicted_key_pts = test_outputs[i].data
        predicted_key_pts = predicted_key_pts.numpy()
        # undo normalization of keypoints
        predicted_key_pts = predicted_key_pts*50.0+100

        # plot ground truth points for comparison, if they exist
        ground_truth_pts = None
        if gt_pts is not None:
            ground_truth_pts = gt_pts[i]
            ground_truth_pts = ground_truth_pts*50.0+100

        # call show_all_keypoints
        show_all_keypoints(np.squeeze(image), predicted_key_pts,␣
 ↪ground_truth_pts)
```

```
        plt.axis('off')

    plt.show()

# # call it
# visualize_output(test_images, test_outputs, gt_pts)

class Rescale(object):
    """Rescale the image in a sample to a given size.

    Args:
        output_size (tuple or int): Desired output size. If tuple, output is
            matched to output_size. If int, smaller of image edges is matched
            to output_size keeping aspect ratio the same.
    """

    def __init__(self, output_size):
        assert isinstance(output_size, (int, tuple))
        self.output_size = output_size

    def __call__(self, image):

        h, w = image.shape[:2]
        if isinstance(self.output_size, int):
            if h > w:
                new_h, new_w = self.output_size * h / w, self.output_size
            else:
                new_h, new_w = self.output_size, self.output_size * w / h
        else:
            new_h, new_w = self.output_size

        new_h, new_w = int(new_h), int(new_w)

        img = cv2.resize(image, (new_w, new_h))


        return img


class CenterCrop(object):
    """Crop randomly the image in a sample.

    Args:
        output_size (tuple or int): Desired output size. If int, square crop
            is made.
    """
```

```
    def __init__(self, output_size):
        assert isinstance(output_size, (int, tuple))
        if isinstance(output_size, int):
            self.output_size = (output_size, output_size)
        else:
            assert len(output_size) == 2
            self.output_size = output_size

    def __call__(self, image):


        h, w = image.shape[:2]
        new_h, new_w = self.output_size

        y_center, x_center = int(h/2) , int(w/2)

        top = int(max(y_center - new_h/2, 0))
        left = int(max(x_center - new_w/2, 0))

        image = image[top: top + new_h,
                      left: left + new_w]



        return image

image_copy = np.copy(image)

# loop over the detected faces from your haar cascade
for (x,y,w,h) in faces:

    padding = max(h, w)*0.25
    h=int(h+ 2*padding)
    y = max(int(y -padding), 0)

    # Select the region of interest that is the face in the image
    roi = image_copy[y:y+h, x:x+w]

    print("roi.shape = "+str(roi.shape))

    ## TODO: Convert the face region from RGB to grayscale
    gray_roi = cv2.cvtColor(roi, cv2.COLOR_BGR2GRAY)
#     print("gray_roi = "+ str(gray_roi))
    print("gray_roi.shape = "+ str(gray_roi.shape))

    ## TODO: Normalize the grayscale image so that its color range falls in␣
 ↪[0,1] instead of [0,255]
```

```
    gray_roi = gray_roi / 255.0

    ## TODO: Rescale the detected face to be the expected square size for your␣
 ↪CNN (224x224, suggested)
    target_size=224

#     smaller_dim = min(gray_roi.shape[0], gray_roi.shape[1])
#     factor = target_size/smaller_dim

#     temp_h = round(factor*gray_roi.shape[0])
#     temp_w = round(factor*gray_roi.shape[1])

#     composed = transforms.Compose([Rescale(target_size),
#                                    RandomCrop(target_size)])

    ###############################################################
#     composed = transforms.Compose([Rescale(target_size),
#                                    CenterCrop(target_size)])

    composed = transforms.Compose([Rescale(target_size)])
    ###############################################################

    gray_roi_transformed = composed(gray_roi)

    gray_roi_transformed = cv2.resize(gray_roi_transformed, (target_size,␣
 ↪target_size))

#     print("gray_roi_transformed.shape = "+str(gray_roi_transformed.shape))
#     plt.imshow(gray_roi_transformed, cmap='gray')
#     plt.show()

#     input("pause")
#     continue


#     plt.show()
#     input("Press Enter to continue...")



#     resized_roi = cv2.resize(gray_roi, (temp_h, temp_w))



    ## TODO: Reshape the numpy image shape (H x W x C) into a torch image shape␣
 ↪(C x H x W)
```

```
    # if image has no grayscale color channel, add one
    if(len(gray_roi_transformed.shape) == 2):
        # add that third color dim
        gray_roi_transformed = gray_roi_transformed.
 ↪reshape(gray_roi_transformed.shape[0],
                                                          ␣
 ↪gray_roi_transformed.shape[1],
                                                           1)

#     print("gray_roi_transformed.shape = "+str(gray_roi_transformed.shape))
#     input("pause")

#     print("gray_roi_transformed.shape = "+str(gray_roi_transformed.shape))
#     plt.imshow(gray_roi_transformed[:,:,0], cmap='gray')
#     plt.show()
#     input("pause")
#     continue

    # swap color axis because
    # numpy image: H x W x C
    # torch image: C X H X W
    gray_roi_transformed = gray_roi_transformed.transpose((2, 0, 1))

#     print("gray_roi_transformed.shape = "+str(gray_roi_transformed.shape))
#     plt.imshow(gray_roi_transformed[0,:,:], cmap='gray')
#     plt.show()
#     input("pause")
#     print("resized_roi[:,120] = "+str(resized_roi[:,120]))
#     print("min(resized_roi) = "+str(min(resized_roi)))
#     print("max(resized_roi) = "+str(max(resized_roi)))

#     continue

    image_tensor , output_pts = net_sample_output(gray_roi_transformed)

    print("output_pts = "+str(output_pts))
```

```
torch.cuda.is_available() = True
roi.shape = (259, 173, 3)
gray_roi.shape = (259, 173)
output_pts = tensor([[[ 14.4483, 106.9501],
         [ 15.3710, 126.7018],
         [ 19.0937, 144.9249],
         [ 20.4304, 162.9601],
```

```
[ 27.9561, 181.1247],
[ 39.4285, 196.8143],
[ 54.3797, 207.0533],
[ 72.6782, 216.3813],
[102.1561, 221.2265],
[124.5757, 215.7425],
[138.3255, 208.9348],
[146.4734, 199.6535],
[157.0957, 186.7906],
[165.7630, 167.6166],
[168.5548, 152.6493],
[171.1913, 134.1652],
[172.6989, 114.5730],
[ 39.7705,  94.5324],
[ 52.5978,  91.2107],
[ 64.7820,  90.6246],
[ 76.6539,  91.9209],
[ 88.7447,  95.2478],
[128.2447,  97.6643],
[136.7655,  96.4851],
[148.3386,  95.3944],
[158.6706,  98.0517],
[166.2509, 100.4871],
[107.5255, 115.2086],
[108.7853, 128.1936],
[109.3278, 141.6586],
[110.8739, 150.7757],
[ 91.6025, 154.5873],
[ 99.9465, 157.8535],
[107.8256, 159.4784],
[115.5404, 159.2188],
[119.3542, 156.4612],
[ 53.9858, 110.5126],
[ 64.3800, 108.0668],
[ 74.9079, 108.7650],
[ 82.6168, 111.8569],
[ 74.8641, 114.3950],
[ 64.3070, 115.0928],
[126.2428, 114.1529],
[136.1799, 112.0853],
[144.9835, 111.7153],
[150.8152, 114.2215],
[144.5420, 117.7513],
[135.3085, 117.4378],
[ 68.9476, 173.7313],
[ 84.0070, 171.3678],
[ 99.8598, 169.2657],
[109.0332, 172.4717],
```

```
          [113.0250, 169.5653],
          [125.3919, 172.7865],
          [132.1424, 174.9070],
          [121.8541, 182.2947],
          [115.0645, 186.0109],
          [105.8191, 186.4933],
          [ 94.4966, 185.1952],
          [ 84.6919, 181.3766],
          [ 72.6532, 173.4285],
          [ 93.6673, 175.0707],
          [105.4057, 175.8245],
          [115.3982, 176.0549],
          [130.7220, 174.3210],
          [114.1133, 177.1151],
          [104.8638, 178.1111],
          [ 95.8659, 178.1247]]], device='cuda:0', grad_fn=<ViewBackward0>)
roi.shape = (238, 159, 3)
gray_roi.shape = (238, 159)
output_pts = tensor([[[ 36.7389, 102.8438],
          [ 37.6161, 122.8760],
          [ 41.5837, 140.3754],
          [ 43.4341, 157.7435],
          [ 50.8421, 175.9789],
          [ 62.6020, 192.1023],
          [ 76.7681, 203.5159],
          [ 92.9060, 212.5380],
          [120.8589, 217.5696],
          [142.6469, 210.8953],
          [157.6479, 202.1674],
          [168.8904, 191.8616],
          [179.8290, 177.0228],
          [187.5741, 157.6785],
          [190.8305, 142.1263],
          [192.4655, 123.5201],
          [193.0319, 103.9781],
          [ 57.4247,  92.5687],
          [ 67.4501,  89.3846],
          [ 78.3970,  89.0061],
          [ 89.9027,  91.0196],
          [100.7615,  94.4540],
          [137.3654,  95.2345],
          [145.8595,  93.2587],
          [156.7343,  91.6106],
          [167.6874,  92.6328],
          [176.8756,  94.4377],
          [118.6243, 113.7775],
          [119.6025, 126.9886],
          [118.8384, 141.4258],
```

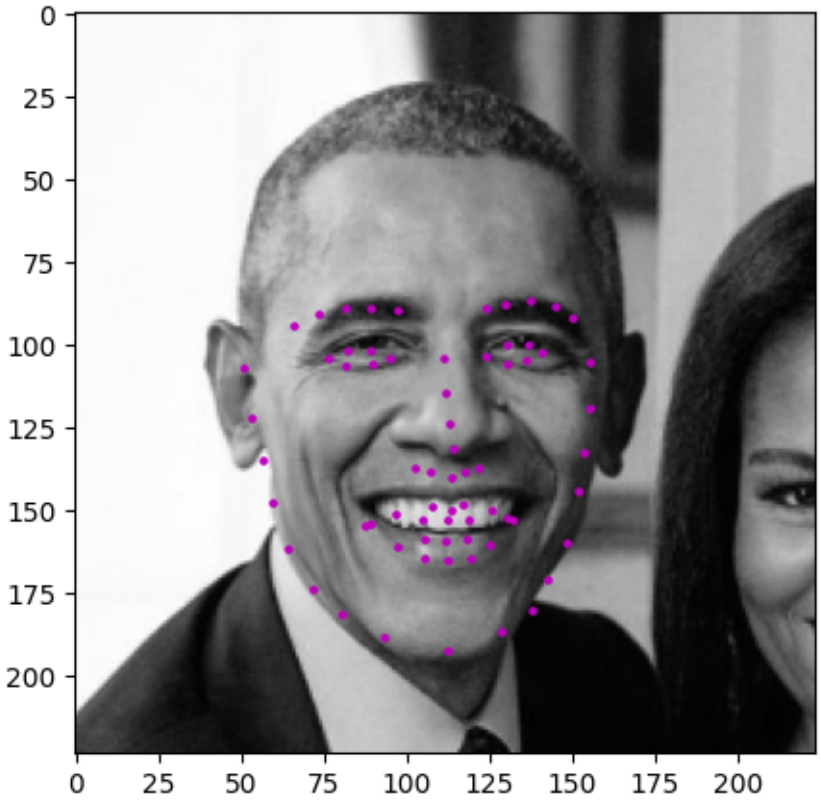
```
           [120.9616, 149.8242],
           [106.3516, 153.0459],
           [113.1318, 156.1665],
           [119.8676, 157.5798],
           [127.7238, 156.8866],
           [132.5688, 154.1602],
           [ 70.8066, 108.5998],
           [ 79.8494, 106.7854],
           [ 89.4388, 106.8668],
           [ 97.7454, 109.4190],
           [ 89.9937, 112.9426],
           [ 79.6356, 113.0029],
           [137.8718, 110.5502],
           [146.9797, 108.4590],
           [155.3184, 107.9026],
           [162.7569, 109.4606],
           [155.4439, 114.1171],
           [146.2135, 113.4787],
           [ 86.9057, 171.1361],
           [ 99.3431, 169.4917],
           [113.5435, 167.7705],
           [121.3686, 170.3966],
           [125.3082, 167.7410],
           [138.1745, 169.7048],
           [147.0196, 170.9546],
           [136.2540, 179.1194],
           [129.3008, 183.1821],
           [120.7943, 184.2940],
           [109.8132, 183.2965],
           [100.9874, 179.1947],
           [ 90.2954, 171.0496],
           [108.5288, 172.5518],
           [119.5949, 173.6084],
           [128.7141, 173.3101],
           [145.7669, 170.3953],
           [128.2504, 174.2729],
           [119.7890, 175.5885],
           [111.0079, 175.6653]]], device='cuda:0', grad_fn=<ViewBackward0>)
```

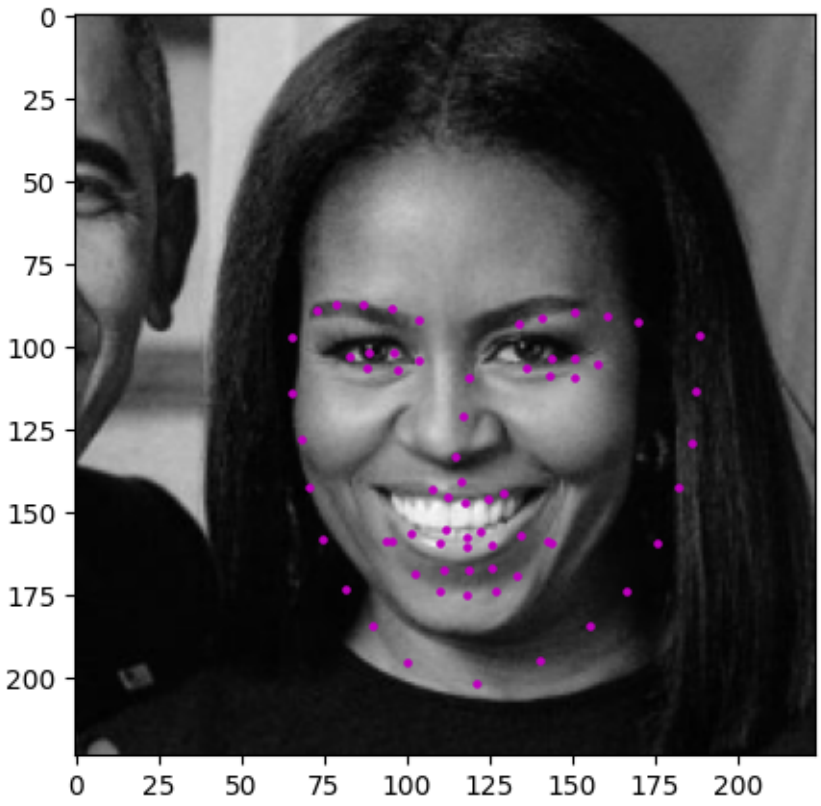
[45]:
```python
import matplotlib.image as mpimg

# image = cv2.imread('images/obamas.jpg')
image = mpimg.imread('images/obamas.jpg')

def show_all_keypoints(image, predicted_key_pts, gt_pts=None):
    """Show image with predicted keypoints"""
    # image is grayscale
```

```
    plt.imshow(image, cmap='gray')
    plt.scatter(predicted_key_pts[:, 0], predicted_key_pts[:, 1], s=20,␣
 ↪marker='.', c='m')
    # plot ground truth points as green pts
    if gt_pts is not None:
        plt.scatter(gt_pts[:, 0], gt_pts[:, 1], s=20, marker='.', c='g')

image_copy = np.copy(image)

# loop over the detected faces from your haar cascade
for (x,y,w,h) in faces:

    padding = max(h, w)*0.425

    if ((x - padding) >= 0) and ((y - padding) >=0):
        w = int(w + 2*padding)
        x = max(int(x -padding), 0)
        h = int(h + 2*padding)
        y = max(int(y -padding), 0)



    # Select the region of interest that is the face in the image
    roi = image_copy[y:y+h, x:x+w]

    ## TODO: Convert the face region from RGB to grayscale
    roi = cv2.cvtColor(roi, cv2.COLOR_RGB2GRAY)



    ## TODO: Normalize the grayscale image so that its color range falls in␣
 ↪[0,1] instead of [0,255]
    roi = roi/255.0


    ## TODO: Rescale the detected face to be the expected square size for your␣
 ↪CNN (224x224, suggested)
    img = cv2.resize(roi, (224, 224))
    img_copy = np.copy(img)

    ## TODO: Reshape the numpy image shape (H x W x C) into a torch image shape␣
 ↪(C x H x W)
    img_copy = img_copy.reshape(img_copy.shape[0], img_copy.shape[1], 1)
    img_copy = img_copy.transpose((2, 0, 1))

#     print("np.max(img_copy) = "+str(np.max(img_copy)))
#     print("np.min(img_copy) = "+str(np.min(img_copy)))
```

```
    ## TODO: Make facial keypoint predictions using your loaded, trained␣
 ↪network
    img_copy = torch.from_numpy(img_copy)
    if dev == "cpu":
        fin = img_copy.type(torch.FloatTensor)
    else:
        fin = img_copy.type(torch.FloatTensor).cuda()

    fin = fin.unsqueeze(0)

    ###############################################################

#     print("fin.shape = "+str(fin.shape))
#     print("fin = "+str(fin))


    ###############################################################

    predicted_key_pts = net(fin)
    pred = predicted_key_pts.view(predicted_key_pts.size()[0], 68, -1)
    if dev == "cpu":
        pred_n = np.copy(pred.data.numpy())
    else:
        pred_n = np.copy(pred.data.cpu().numpy())
    pred_n = pred_n.squeeze(0)


    print("pred_n.shape = "+str(pred_n.shape))
#     print("pred_n = "+str(pred_n))


    ## TODO: Display each detected face and the corresponding keypoints

    img_copy = img_copy.squeeze(0)  # Remove batch dimension
    fig = plt.figure(figsize=(5, 5))
    show_all_keypoints(img_copy, pred_n)

#     print("pred_n = "+str(pred_n))
```

```
pred_n.shape = (68, 2)
pred_n.shape = (68, 2)
```


0
25
50
75
100
125
150
175
200
0
25
50
75
100
125
150
175
200


0
25
50
75
100
125
150
175
200
0
25
50
75
100
125
150
175
200

[ ]:

[ ]: