Human Face Identification System

Abstract:

The project is to develop a deep learning system that can identify a person by the front face image. It can be used for check-in/ check-out system and security surveillance system. By collecting front face images and training with a deep learning network, it shows good results in a lab-like face detection testing.

Introduction:

There are needs to identify persons by front faces for check-in/ check-out system or security surveillance system. It can guard security 7/24 and potentially be able to notify when a target person shows up or an unauthorized person comes in.

Haar Cascade algorithm has been developed to detect human face (1,2) in an image for years. By Haar Cascade, it can be used to tell whether there is someone in an image or no one in an image. It can pre-process images to get front face rectangle for next stage processing.

With front face image available, a deep learning system can be used to identify a person by model. In this project, a deep learning system with LeNet network was built to train for person S1,

person S2, and others. Therefore, there are four categories to identify, S1, S2, unknown person, or no-one.

To increase accuracy for prediction, data augmentation has been used to increase image samples. Keras ImageDataGenerator class (5) can be used for data augmentation. Each category was augmented to around 1000 images for training and validation for equal weights.

With 28x28 gray level face images, LeNet network was trained quickly in DIGITS.

Background / Formulation: In the beginning, four categories of data were collected for S1, S2, others, and no-one. S1, S2, and others are all human faces. Images of S1 and S2 are from a single person respectively, while Others are from lots of different people. However, images of no-one is a challenge task. Lots of different images were collected among tree, mountain, fence, window, cat, dog, horse, butterfly, flower, etc,

Due to the diversity of images in the no-one category, it can’t be trained to get a good model. Training in deep learning network failed to converge.

Chapter 22 of Adrian Rosebrock’s Deep Learning for Computer Vision with Python (4) describes a face smile detection case.

Haar Cascade was used to detect human faces in the first phase. Then faces images were passed to deep learning to detect a smile on the face.

The no-one images were removed from training and Haar Cascade was used instead. In DIGITS, LeNet Network was used to train face detection on 28x28 grayscale images with 20 epochs and Adam optimization for three categories of images, person S1, person S2, and others. Adam (6) stands Adaptive Moment Estimation, which was presented by Kingma and Ba in 2015 to improve over stochastic gradient descend (SGD). Adam contains advantages of Adaptive Gradient Algorithm (AdaGrad) and Root Mean Square Propagation (RMSProp). As LeNet is simpler than AlexNet or GoogleNet, it finished quickly. Fig. 1 shows screenshot of the training in DIGITS.

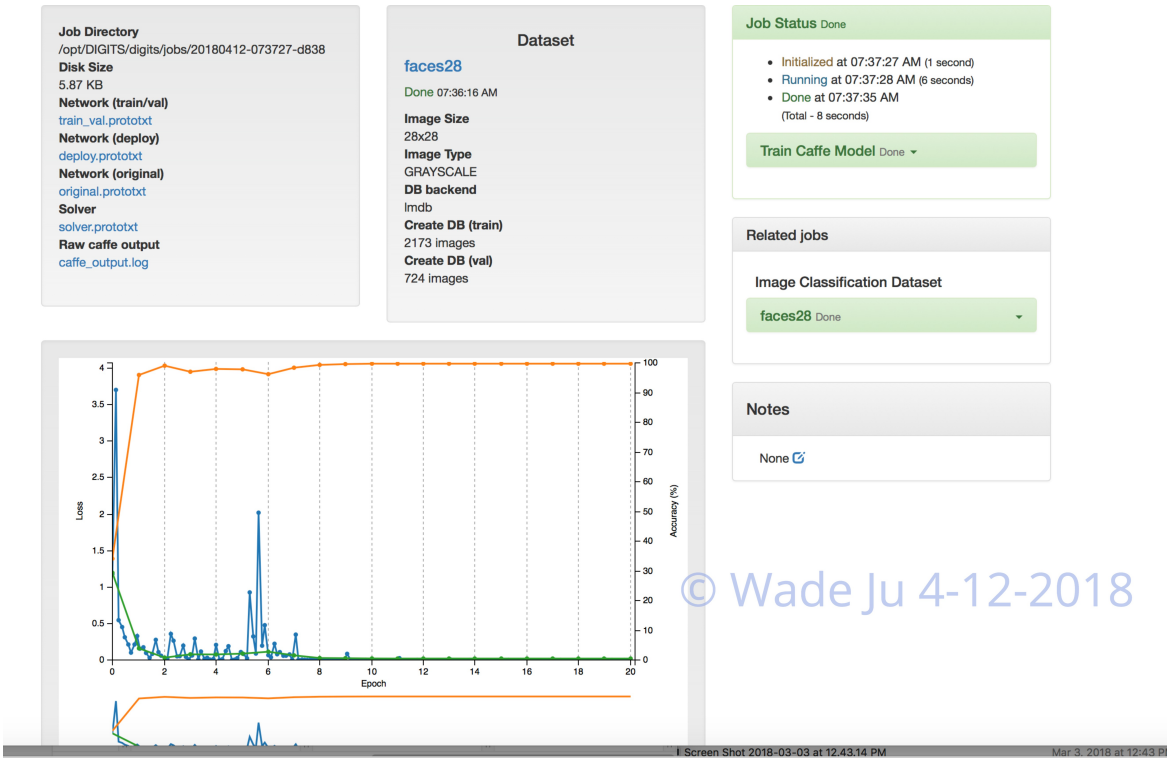


Fig. 1 Training Face Images on DIGITS

Data Acquisition: To simplify the task to identify faces, training images were cut to faces for training and validation. For person S1 and person S2, there were about 20 images collected respectively with manual labels. For others, about 80 different faces were collected manually as well. Table 1 shows some images for S1 and S2.

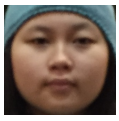
S1							
S2							

Table 1 Face Images for Person S1 and Person S2

In DetectFace,ipynb, function findFace(gray_img) uses detector.detectMultiScale to detect faces in an image. If there are faces found, an array of rect objects will be returned. Below shows a simplified source code of findFace function.

```
def findFace(gray_img):
    rects = detector.detectMultiScale(gray_img, scaleFactor=1.1,
            minNeighbors=5, minSize=(30, 30),
            flags=cv2.CASCADE_SCALE_IMAGE)
    if len(rects) <= 0:
        print(" no face found!")
    else:
        print(" found {} faces".format(n))

    # loop over face bounding boxes
    for (fX, fY, fW, fH) in rects:
        # extract the ROI of the face from the grayscale image,
        # resize it to a fixed 28x28 pixels, and then prepare the
        # ROI for classification via the CNN
        roi = gray_img[fY:fY + fH, fX:fX + fW]
        roi_img = cv2.resize(roi, (24, 24))
        # normalize each pixel
        roi = roi.astype("float") / 255.0
        roi = img_to_array(roi)
        roi = np.expand_dims(roi, axis=0)
```

By using gen_imgs.ipynb, each category has around 1000 images. It's good to keep each category with similar amount of training dataset to prevent from weighting problems in training.

With ImageDataGenerator from Keras, it can create a generator to generate images with variations on rotation angles, width/ height shifts, shear, zoom, etc.

```
aug = ImageDataGenerator(rotation_range=10, width_shift_range=0.1,
height_shift_range=0.1, shear_range=0.2, zoom_range=0.2,
horizontal_flip=False, fill_mode="nearest" )
```

Flow() function of the data augmenter can specify source image and target image attributes. By looping through the image_gen object, new images will be generated in the target directory with specified attributes and variance from the source image. Table 2 shows some augmented images.

```
image_gen = aug.flow(get_image(file), save_to_dir="./images/faces/
NoOne", save_prefix="gen_", save_format="jpg")
```

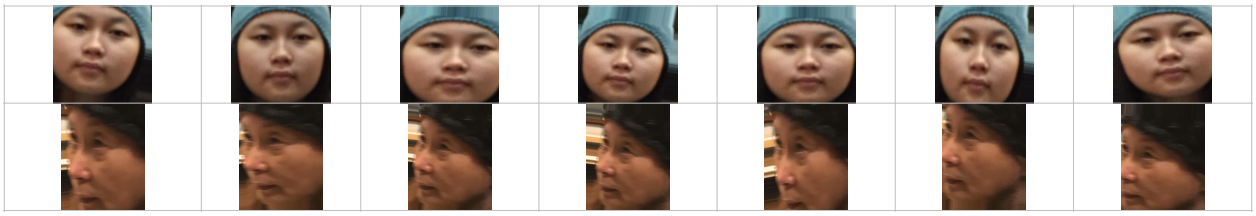


Table 2 Augmented Images

The training data were uploaded to S3, https://s3-us-west-2.amazonaws.com/wtest7272/faces.zip.

Test data were uploaded to S3 as well, https://s3-us-west-2.amazonaws.com/wtest7272/Test.zip.

Results: Table 3 shows a list of test images. Fig. 2 shows test results in DIGITS. The trained model identifies all three cases perfectly.



Table 3 Test Image

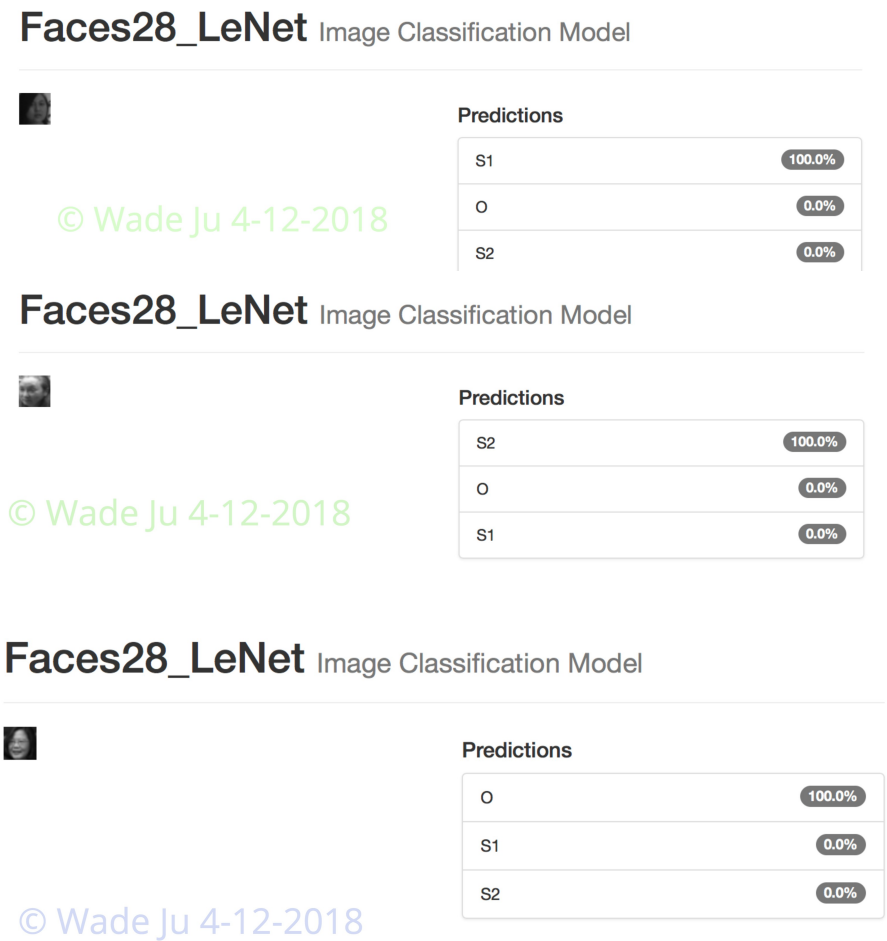


Fig. 2 Test Results on DIGITS

For Haar Cascade in DetectFace.ipynb, which has false positives on detecting faces among people and background objects.

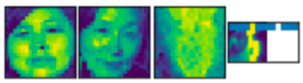
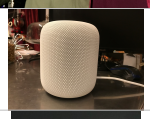
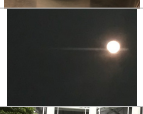
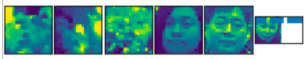
Test Image	Detected Faces	Results	Accuracy
		Two correct faces and two false positives	50%
	no face found	Correct	100%
	no face found	Correct	100%
		Three correct faces and three false positives.	50%

Table 3 for Hair Cascade Testing

Discussion: The training time and accuracy from LeNet for face detection are good with face only images provided. However, using Haar Cascade algorithm to preprocess image and get face out of image is not as good. There were 50% false positives in two cases.

For fixed surveillance camera, we can further train images with background to tell difference from with person(s) and without a person.

For dynamic environment, we would need better algorithms to handle it. One way is to segment image using fully convolution network (FCN). If there is a segment like human shape, Hair Cascade can be used to get a front face and pass it to LeNet for identification.

Conclusion / Future Work: An important task to improve is to accurately get the portion of the front face of a person. The testing performed is a lab-like testing. There are ways to improve for real-life cases.

Another future work is to put in hardware to test and improve. Thanks for the discount for Nvidia Jetson TX2! I am looking to receiving it soon.

References

1. https://docs.opencv.org/3.4.1/d7/d8b/tutorial_py_face_detection.html

2. https://en.wikipedia.org/wiki/Haar-like_feature

3. https://github.com/opencv/opencv/blob/master/data/haarcascades/haarcascade_frontalface_default.xml

4. Deep Learning For Computer Vision With Python 2017 Dr. Adrian Rosebrock

5. https://keras.io/preprocessing/image/

6. Gentle Introduction to the Adam Optimization Algorithm for Deep Learning https://machinelearningmastery.com/adam-optimization-algorithm-for-deep-learning/