Symbolic Understanding of Anime Using Deep Learning

Jingyang Li
Department of Computer Science
Golisano College of Computing and Information Sciences
Rochester Institute of Technology
Rochester, NY 14623
jl8930@rit.edu

Abstract—With the exponential growth of images available on the Internet, virtual characters can be seen almost everywhere. Nowadays, since real world images are easy for machines to study, what about virtual images? This paper demonstrates a novel method of explaining virtual image meanings from its visual symbols.

Index Terms—Image Description; Anime; Deep Learning

I. INTRODUCTION

Image understanding and captioning are well developed as advances in language modeling and image recognition. The benefits of image understanding are not only labeling enormous amount of online images at low cost, but also merge Natural Language Processing and Computer Vision together which are two fields of great importance in Artificial Intelligence.

This area may be developed for a long time by utilizing deep learning [1], [2], [3], [4]; nevertheless, it is not an all-inclusive area yet. All the image captioning methods or models available are based on real-world scenes, which is relatively trivial and familiar because the objects follow particular patterns and rules. While our focus is on the virtual scenes in images, namely anime.

Anime has its own visual vocabulary that can be intuitive and simple to understand for human. Lines, colors and deformations are used to convey emotions in anime. Anime watchers usually take these pictorial words for granted. For example, anger can be expressed by some simple straight lines which can also be interpreted as awkwardness in different scenes for anime watchers.

However, situation is vastly different for machines. A machine is not able to process the visual symbols and extract emotional information like human brain does in most cases. The reason is anime may contain different art styles from different artists, and also unfamiliar or unknown elements compared to the real-world scenes. In this case, one of the biggest challenges for us is to find and catch the most important features and ignore the least important ones.

To cope with this, we realized that particular symbols do play a very important role in animes meaning expression such as cross popping veins,flushing lines, tear drops, and so on. They are technically invariant and usable since in most cases they convey similar meanings even in different scenes. Therefore, in this paper, we approach the problem by detecting and training features as these symbols by applying convolutional neural networks.

A. Motivation

As the era of virtual reality comes, an enormous amount of virtual scenes which consists of even larger amount of virtual images will pop out. Thus, it is critical whether our current image understanding methods will also work on virtual images or not as it may be applied in daily life.

To illustrate if current image understanding methods will fail to decode the deeper meaning of anime images such as emotions, we run a comparison test on both anime images and real-world images using the Google Cloud Vision API which claims to be a powerful image understanding tool encapsulating many machine learning models. Here is one pair of results we have obtained.

![Motivation Figure: Current top level image understanding tool fails on anime image after comparison test](image.png)

It is obvious that this tool is capable of extracting main features from the angry old man image and confidently says...
that anger is his current emotion. Whereas it does not successfully recognize, detect or understand the emotion expressed from this anime character. Therefore, we believe it is of great value to come up with a method to handle virtual image understanding. Apart from this, we can also evaluate different deep learning method’s performance on this type of images which has not been deeply developed before.

II. BACKGROUND

Since most studies[5], [2] on image understanding are based on real-world images which is different from virtual images, we need to adjust the methods learned from other articles [6] while seeking our way to solve this problem. Here, we mainly focus on discussing several image understanding and deep learning related methods.

A. Image Understanding

Recent studies on image understanding show that there are particular persuasive strategies that images use to convey the meaning behind them to viewers such as symbolic references. Besides, it is proved that computer vision system do have the capability of understanding such strategies []. Thus, we believe special symbol is a vital component to help us tackle the virtual image understanding problem.

B. Convolutional Neural Networks

There has been a lot of previous work illustrating how Convolutional Neural Networks (CNNs) is the best way to classify images. We also believe that with proper training of Convolutional Neural Networks, it is capable of classifying virtual images.

C. Feature Matching

The task of feature matching has been explored. We decide to apply current feature matching methods to see which one gives the best results, and also validate whether it is possible to have a decent result on virtual images. Scale-invariant feature transform (SIFT) [7], [8] is considered to give the best results because anime symbols prefer to have scale or angle transformation rather than shape transformation. Besides, ORB [9] and FLANN [10] are also tested in our study.

III. METHOD

Overview

The ultimate goal of our method is to generate descriptions of anime images. The first phase of our work consists of obtaining high-quality labeled anime images. We do this by collecting images from Internet and have Amazon Mechanical Turk labeled them. Then we train neural networks for different labels, and use special symbol as a vital factor of face expression. At last, we generate description for each image based on the training results.

A. Data Preparation

We achieved this part by dividing it into three steps: First we collect raw data from the Internet, then get labeled data from Amazon Mechanical Turk, clean the data at last.

1) Collecting Images:

It is hard to collect a large amount of anime images without a web crawler program. Therefore, We implemented a web crawler to collect images from konachan.net [11] using Python packages such as beautifulsoup and requests. We have collected around 120,000 images as the raw dataset. However, there is no uniform size for the images we collected since the website does not classify images by sizes.

2) Labeling Images:

We randomly chose 7,500 images from the raw dataset to be labeled. They are all uploaded onto a server which provides specific link to each one of the images. Besides, as the following figure shows, a template which consists of an image and ten related questions is made for each worker on Amazon Mechanical Turks to work on. These questions include:

- Whether the image contains inappropriate content
- Whether the image is interesting or not
- How happy does this image background make you feel
- How stimulating is this image
- What kind of anime do you think this is
- How many people are in the image
- What’s the facial expression
- Does this image contain cross-popping vein
- Does this image contain flush lines

![Fig. 2. Amazon Mechanical Turk Template Part I](image)

![Fig. 3. Amazon Mechanical Turk Template Part II](image)
By bringing up with these questions, we are able to get labels to help build attributes of images. However, one potential issue we ran into is that our labeled data may not be clean and accurate enough, since we are limited to the budget and workers may not be well encouraged to treat this survey seriously. Therefore, the final description accuracy may be negatively influenced more or less.

3) Preprocessing Data:

After the labeled data are collected, we first filter out images with inappropriate content for the concern of morality. Then we filter out unusable data which mainly consists of two different kinds of data:

- blank instances or the ones with more than two vacant labels
- instances with worker’s working time lower than ten seconds (we assume no one is capable of finishing ten questions within ten seconds unless the survey is not treated seriously)

B. Feature Matching

The very important task of this project is to prove and show that with recognition of special symbols in anime, we are confident to decide the particular meaning of character’s facial expression or the type of anime. In this study, we chose two special symbols:

- cross-popping veins
- flush lines

At the beginning of our study, we assume current feature matching methods can perform good if not excellent on virtual images. However, we did not realize we were wrong until we have applied three most popular feature matching methods and they all failed.

- Brute-Force Matching with SIFT Descriptors and Ratio Test
- Brute-Force Matching with ORB Descriptors [9]
- FLANN(Fast Library for Approximate Nearest Neighbors) Based Matching [12]

To seek the cause of this failure case, different techniques are applied to make comparisons such as resize, blur, image segmentation, etc. The first solution we come up with is to use a template of cross-popping veins and match it with an image with a highly similar symbol in it. We try to resize the template image so that the scale won’t be a factor that affects the result. Besides, Gaussian blur is applied to get rid of some noise.

It is obvious that both of our methods failed. However, the left example shows some of the turning points of cross-popping vein are detected successfully, they just match with wrong target.

The next thing we have tried is matching two different images with similar cross-popping veins and image layout. As we can see from the figure below, cross-popping veins are not matched successfully, but it is noticeable that there are several pairs of edge points of the hair area on both images.

We keep asking ourselves whether it is because the two images we chose are still not similar enough. Thus, we extract two characters which have similar features from the same image. However, the result is still disappointing even if the cross-popping veins on two characters have the exact style.

At this point, it becomes more and more unclear to us whether it is possible to match symbols in virtual images or not with current feature matching methods. Therefore, the last
thing we want to try is crop a small patch which contains the cross-popping vein from an image as the template. Then try to see if they can successfully match with each other, especially the symbol area.

The results is somehow delightful as it is shown in the figure above. Not only there are a lot of feature points matching with the right target points, but also it clearly shows that most of the feature points are along the cross-popping vein edges, which means the cross-popping vein is successfully detected and matched. This helps prove that even with current feature matching methods, there is a way to do the job.

Now here comes the question, why did the several previous cases fail? We believe it is because of the noise, which is also the most common obstacle in feature matching work. Combined with all the examples and comparisons we have made so far, it is noticeable that cross-popping vein is usually located at a certain area on an anime character, which is either side of its forehead. The most obvious common feature of this area is that it is covered by hair which often are presented as triangles consist of multiple sharp straight lines by Mangaka. This is the main cause of failure of feature matching methods in our opinions, since both the hair and cross-popping vein are interwoven with each other with very similar style and features.

C. Convolutional Neural Networks Training

In this study, convolutional neural network is a vital part. As mentioned previously, we believe that it will still be an effective way to deal with virtual images because theory is to continuously training the neural network to make it remember the important features and abandon the irrelevant ones which should remain the same. Currently we have trained two classifiers: One is the “interestingness” classifier; the other one is facial expression classifier. Both classifiers are trained based on using ”Very Deep Convolutional Networks for Large-Scale Image Recognition” (VGG16) architecture [6].

1) VGG16:

Why do we use VGG16 as base structure instead of completely retraining our own neural networks? The answer is quite simple: Because it is can be used out-of-box and very popular in image recognition. First of all, Keras provides VGG16 by default and it is easy for us to modify it a little bit based on it. Besides, VGG16 is a well known architecture for dealing with similar problems as ours. In this study, we think VGG16 would give us an important baseline for comparisons in the future. Even though we may pre-train the model beforehand, due to its weaknesses such as time consuming and large weight, it is thought to be the best option for us at this point.

2) “Interestingness” Classifier:

This classifier is trained using labels from the question "whether this image is interesting or not". What we have done is splitting the 7,500 images into two big chunks: training and validation (we are using ten-fold validation which means the training dataset is nine times the size as the validation dataset). Each one of them has two subsets: interesting or not interesting. Then we set the parameters and run the convolutional neural network. At last we got around 66 percent overall accuracy.

3) Facial Expression Classifier:

The first task is extracting character faces from anime images. We utilize an open-source face detector called "lbpcascadeanimeface" [?] which was trained based on 20,000 anime images previously. After test, we found it is the most reliable and stable tool we can use at this moment, and the accuracy is over 85%. 2,466 faces are detected and
extracted for training the facial expression classifier. Then based on the corresponding label, we divide these images into seven subsets for both training and validation. Similarly, ten fold validation is also applied in this classifier. What’s different from the “Interestingness” classifier is we use "categorical" instead of "binary" for classifier class because we have more than two classes to be classified into. After running 100 epochs, the overall accuracy we got is around 63%, which in our opinion is satisfying because the cleanness of our source data is doubtful. Here are some test result we have retained:

![Fig. 11. CNN Test Figure: Number shows the confidence of label "Angry"
](image1)

![Fig. 12. CNN Test Figure: Number shows the confidence of label "Neutral"
](image2)

From the results it is obvious that "Neutral" has higher accuracy than "Angry" and "Happy", this is because we have the largest dataset labeled as "Neutral". Apart from this, "Angry" has the lowest accuracy and this is why we try to use symbol recognition to increase the accuracy of it, which means whenever a cross-popping vein is detected in this image, it is safe and confident for the classifier to say this character is angry.

The following figure is the confusion matrix of facial expression classifier.

These two figures also support our assumption above as we can tell that Neutral and Happy are the two labels with highest detection accuracy generated by the classifier.

![Fig. 13. CNN Test Figure: Number shows the confidence of label "Happy"
](image3)

![Fig. 14. Confusion Matrix Figure: facial expression classifier
](image4)

**IV. Future Work**

This project aims at generating simple descriptions for anime images, but more and more obstacles pop out in the process of our study which left us on the half way to our goal. Therefore, there are several important tasks left need to be done:

- "Anime type" classifier (background extraction)
- "Stimulating" classifier
- Cross-popping vein and flush lines detector
- Description generating and test

"Anime type" and "Stimulating" classifier are similar to "Interestingness" classifier which can be straightforward using convolutional neural networks.

Cross-popping vein is difficult as we have discussed above, but we believe with proper combination of techniques it can be solved. Currently there are three things we believe is the key to this solution: First of all, in common cases, cross-popping vein is always red. Thus, the non-red parts can be ignored. Secondly, the cross-popping vein is often center symmetric which provides us a pattern to focus on. At last, it always appears on the one side of character’s forehead, which even help reduce the region of detection once the character’s face is recognized successfully.

Flush lines is a feature which we haven’t got our hands on but it is thought to be an even harder task to deal with. The
reason is there are different styles of flush lines as the below example shows. In some cases, the flush lines are not parallel to each other as the following figure shows.

![Fig. 15. Intersecting flush lines](image)

Apart from the styles, its color also varies in different cases. We currently cannot think of an efficient solution to this symbol, but we will try our best to seek for it.

V. EXPERIMENTS, RESULTS AND DISCUSSION

As it is mentioned before, we don’t believe that our dataset is clean and accurate. This assumption was proved when we were going through the images and corresponding labels, a good amount of which are obviously mislabeled. The only way to solve this problem is to relabel the images using more reliable workforce such as the RIT anime club which are full of anime fans who will be more supportive and also familiar with anime. Another potential issue we have noticed is that due to the low resolution of our source images, our neural networks might not be trained as effective as it should be. This could be solved if we can get large amount of high quality images, preferably with over 1920 * 1080 pixels. Moreover, since we are using VGG16 architecture, the result might be more satisfying if deeper neural networks are applied, such as VGG19, GoogLeNet, ResNet, etc.

VI. CONCLUSION

In conclusion, we found that the cross-popping vein is highly correlated to facial expression "Anger", and we believe we made the right assumption that it is possible to let machine understand virtual image’s meaning by detecting and recognizing some key symbols in the scene. We also proves that current methods like SIFT matching and Convolutional Neural Networks are capable of dealing with virtual images only if proper preparation work and training are provided. This study is far from accomplishment, more challenges are waiting ahead and more hard work needs to be done. However, this is also the essence of each outstanding study, and we believe that we are on the right tack.

ACKNOWLEDGMENT

The author would like to thank Dr. Ifeoma Nwogu for advising and instructing along the process of this project, and Dr. Minseok Kwon for teaching us presenting and writing skills.

REFERENCES

APPENDIX

This appendix is for reporting detailed information of implementation.

A. Data Preparation Programs

1) Source Images Collection: This is implemented in "collect_data.py" file. The pseudo code for download function is as follows:

```python
def download():
    if file exists:
        print "File exists!"
        return
    try:
        get url
        for filename:
            write chunk
        return filename
    except:
        KeyboardInterrupt, Exception
```

The whole program is very straightforward, for each url parsed in, iteratively write data into local file. Then in the main function, call the download function with different urls.

2) Face Images Generation: This part is implemented in "anime_face_detection.py" file. The pseudo code for detect function is as follows:

```python
def detect():
    cascade = CascadeClassifier()
    faces = cascade.detectMultiScale()
    for face in faces:
        save face to local
```

Then loop to save faces from each image from all images, and the faces from same image will be saved with the same prefix name but different postfix index.

3) Amazon MTurk Template: The following HTML code section is the content source code of the Amazon MTurk template. One potential improvement can be making the question single-choice instead of multiple-choice which might lead to more meaningless workload.

```html
<fieldset><label>1. Does this image contain any nudity or sexual content (for adults or above age 18 only)?</label>
</fieldset>

<div class="radio"><label><input name="Q1Answer" type="radio" value="Yes" />Yes</label></div>

<div class="radio"><label><input name="Q1Answer" type="radio" value="No" />No</label></div>

<fieldset>
<fieldset><label>2. Is this image interesting or not?</label>
</fieldset>

<div class="radio"><label><input name="Q2Answer" type="radio" value="Yes" />Yes</label></div>

<div class="radio"><label><input name="Q2Answer" type="radio" value="No" />No</label></div>

<fieldset>
<fieldset><label>3. How happy does the image background make you feel?</label>
</fieldset>

<div class="radio"><label><input name="Q3Answer" type="checkbox" value="High" />High</label></div>

<div class="radio"><label><input name="Q3Answer" type="checkbox" value="Neutral" />Neutral</label></div>

<fieldset>
<fieldset><label>4. How stimulating is this image?</label>
</fieldset>

<div class="radio"><label><input name="Q4Answer" type="checkbox" value="High" />High</label></div>

<div class="radio"><label><input name="Q4Answer" type="checkbox" value="Neutral" />Neutral</label></div>

<fieldset>
<fieldset><label>5. What kind of Anime do you like?</label>
</fieldset>

<div class="radio"><label><input name="Q5Answer" type="checkbox" value="High" />High</label></div>

<div class="radio"><label><input name="Q5Answer" type="checkbox" value="Neutral" />Neutral</label></div>

</fieldset>
```
you think this is?

6. How many people are there in the source image?

7. What is the facial expression of this character?

8. Is there a cross-popping vein in the source image (similar to the one in the following example)?

9. Are there flush lines on character's face in the source image (similar to the ones in the following example)?

10. Any additional comments on this image?
4) Classify Face expressions: This program aims at distributing face images with different labels to its corresponding folder, and also abandoning those with multiple labels.

```python
# Reading CSV file
df = pd.read_csv("HIT_results.csv")
df.dropna(axis=0, thresh=10, inplace=True)

df = df[df['Answer'.Q7Answer'].str.contains('Not_clear')==False]

anger = df[df['Answer'.Q7Answer'].str.contains('Anger')==True]
disgust = df[df['Answer'.Q7Answer'].str.contains('Disgust')==True]
happy = df[df['Answer'.Q7Answer'].str.contains('Happy')==True]
sad = df[df['Answer'.Q7Answer'].str.contains('Sad')==True]
scared = df[df['Answer'.Q7Answer'].str.contains('Scared')==True]
surprise = df[df['Answer'.Q7Answer'].str.contains('Surprise')==True]
neutral = df[df['Answer'.Q7Answer']==''Neutral'']

anger_name_list = anger['Input'.image_url'].tolist()
disgust_name_list = disgust['Input'.image_url'].tolist()
happy_name_list = happy['Input'.image_url'].tolist()
sad_name_list = sad['Input'.image_url'].tolist()
scared_name_list = scared['Input'.image_url'].tolist()
surprise_name_list = surprise['Input'.image_url'].tolist()
neutral_name_list = neutral['Input'.image_url'].tolist()

anger_img_list = []
disgust_img_list = []
happy_img_list = []
sad_img_list = []
scared_img_list = []
surprise_img_list = []
neutral_img_list = []

for i in range(len(anger_name_list)):
    anger_img_list.append(anger_name_list[i].split('/')[-1])
for i in range(len(happy_name_list)):
    happy_img_list.append(happy_name_list[i].split('/')[-1])
for i in range(len(sad_name_list)):
    sad_img_list.append(sad_name_list[i].split('/')[-1])
for i in range(len(scared_name_list)):
    scared_img_list.append(scared_name_list[i].split('/')[-1])
for i in range(len(surprise_name_list)):
    surprise_img_list.append(surprise_name_list[i].split('/')[-1])
for i in range(len(neutral_name_list)):
    neutral_img_list.append(neutral_name_list[i].split('/')[-1])

face_list = glob('faces/*')

for name in face_list:
    original_name = os.path.basename(name).split('_')[0] + '_' + os.path.basename(name).split('_')[1]
    if original_name in anger_img_list:
        shutil.copy("%s\%s name, '. face_expression/train/Anger/"
        print ('file copied: %s' % name)
    if original_name in disgust_img_list:
        shutil.copy("%s\%s name, '. face_expression/train/Disgust/"
        print ('file copied: %s' % name)
    if original_name in happy_img_list:
        shutil.copy("%s\%s name, '. face_expression/train/Happy/"
        print ('file copied: %s' % name)
    if original_name in sad_img_list:
        shutil.copy("%s\%s name, '. face_expression/train/Sad/"
        print ('file copied: %s' % name)
    if original_name in scared_img_list:
        shutil.copy("%s\%s name, '. face_expression/train/Scared/"
        print ('file copied: %s' % name)
```

Rochester Institute of Technology 9 | Page
B. Feature Matching

This part contains three different files: SIFT matching.py, ORB matching.py, FLANN matching.py. As the name shows, each one of them represents a feature matching method I have tried. In the final report, I took SIFT matching as an example. Since all three programs have similar process, one pseudo code example will be shown below:

```python
# if original_name in surprise_img_list:
    shutil.copy('%s' % name, './face_expression/train/Surprise/
#     print ('file copied: %s' % name)

if original_name in neutral_img_list:
    shutil.copy('%s' % name, './face_expression/train/Neutral/
    print ('file copied: %s' % name)

# for name in interesting_img_list:
    # shutil.copy('.training/%s' % name, './training/interesting/Yes/
    # print ('file copied: %s' % name)

img1 = "1.jpg"
img2 = "2.jpg"
sift = SIFT_create
features1 = sift.detect(img1)
features2 = sift.detect(img2)
matcher = matcher_create(img1, img2)
for x, y in matcher:
    if x matches y:
        draw lines between x and y on img3
plt.show(img3)
```

C. CNN training

This section contains two CNNs: interestingness CNN and facial expression CNN.

1) Interestingness CNN:

```python
model = Sequential()
model.add(Convolution2D(32, (3, 3),
    input_shape=(96,96,3), data_format='channels_first'))
model.add(Activation('relu'))
model.add(MaxPooling2D(pool_size=(2, 2)))
model.add(Convolution2D(32, (3, 3)))
model.add(Activation('relu'))
model.add(MaxPooling2D(pool_size=(2, 2)))
model.add(Convolution2D(64, (3, 3)))
model.add(Activation('relu'))
model.add(MaxPooling2D(pool_size=(2, 2)))
model.add(Flatten())
model.add(Dense(64))
model.add(Activation('relu'))
model.add(Dropout(0.5))
model.add(Dense(1))
model.add(Activation('softmax'))
model.compile(loss='binary_crossentropy',
    optimizer='rmsprop',
    metrics=['accuracy'])

batch_size = 16
train_datagen = ImageDataGenerator(
    rescale = 1./255,
    shear_range = 0.2,
    zoom_range = 0.2,
    horizontal_flip = True)

test_datagen = ImageDataGenerator(rescale = 1./255)

train_generator = train_datagen.flow_from_directory(
    './interesting/train',
    target_size=(96,96),
    batch_size=batch_size,
    class_mode='binary',
)

validation_generator = test_datagen.flow_from_directory(
    './interesting/validation',
    target_size=(96,96),
    batch_size=batch_size,
    class_mode='binary',
)

H = model.fit_generator(
    train_generator,
    steps_per_epoch=2000//batch_size,
    epochs=50,
    validation_data=validation_generator,
    validation_steps=800//batch_size)

model.save_weights('first_try.h5')
```

2) Facial Expression CNN:

```python
img_width, img_height = 96, 96
train_data_dir = 'face_expression/train'
```
```python
validation_data_dir = 'face_expression/validation'
bv_train_samples = 800
nb_validation_samples = 100
epochs = 50
batch_size = 10

if K.image_data_format() == 'channels_first':
    input_shape = (3, img_width, img_height)
else:
    input_shape = (img_width, img_height, 3)

model = Sequential()
model.add(Conv2D(32, (3, 3), input_shape=input_shape))
model.add(Activation('relu'))
model.add(MaxPooling2D(pool_size=(2, 2)))

model.add(Conv2D(32, (3, 3)))
model.add(Activation('relu'))
model.add(MaxPooling2D(pool_size=(2, 2)))

model.add(Conv2D(64, (3, 3)))
model.add(Activation('relu'))
model.add(MaxPooling2D(pool_size=(2, 2)))

model.add(Flatten())
model.add(Dense(64))
model.add(Activation('relu'))
model.add(Dense(7))
model.add(Activation('softmax'))

model.compile(loss='categorical_crossentropy',
              optimizer='rmsprop',
              metrics=['accuracy'])

train_generator = train_datagen.
flow_from_directory(
    train_data_dir,
    target_size=(img_width, img_height),
    batch_size=batch_size,
    class_mode='categorical') #

validation_generator = test_datagen.
flow_from_directory(
    validation_data_dir,
    target_size=(img_width, img_height),
    batch_size=batch_size,
    class_mode='categorical') #

H = model.fit_generator(
    train_generator,
    steps_per_epoch=nb_train_samples // batch_size,
    epochs=epochs,
    validation_data=validation_generator,
    validation_steps=nb_validation_samples // batch_size)

print("[INFO] serializing network...")
model.save('face_model_softmax.h5')

D. Classifier Testing
Take facial expression classifier for example:

```
# load the image
for filename in file_list:
    image = cv2.imread(filename)
    orig = image.copy()

    # pre-process the image for classification
    image = cv2.resize(image, (norm_size, norm_size))
    image = image.astype("float") / 255.0
    image = img_to_array(image)
    image = np.expand_dims(image, axis=0)

    # classify the input image
    result = model.predict(image)[0]
    # print(result)
    proba = np.max(result)
    label = str(np.where(result == proba)[0])
    # print(label)
    alist.append(int(label[1]))
    label = "\{:.2f\}%".format(label, proba * 100)

    # draw the label on the image
    output = Image.resize(orig, width = 400)
    output = orig.resize(400, Image.ANTIALIAS)
    cv2.putText(orig, label, (3, 25), cv2.FONT_HERSEY_SIMPLEX,
                0.4, (0, 255, 0), 2)
    # show the output image
    cv2.imshow("Output", orig)
    cv2.waitKey(0)
    cv2.imwrite("Output.png", orig)

E. Notice

All detailed code comments are included in the submission.