Emojinator Recognition and Prediction

Xiangbei Chen, Song Luo, Ming Lyu, Chen Yin

Nov 20, 2019

Table of Content

Abstract

Deep learning is a method of machine learning that uses multiple layers to automate the process of feature extraction from inputs. It has been applied in various fields, including image recognition and text analysis. For our project, we proposed a hand gesture emojinator recognitor, which takes a real-time hand image as the input and gives the predicted emoji as the output. In the training process, we applied the Convolutional neural network and VGG16. After adjusting the hyper-parameters, we get the training accuracy 1.00 and test accuracy 0.99.

1. Introduction

Deep learning is currently a hot topic in Machine Learning and is also an essential tool in the field of Artificial Intelligence. Deep Learning is good at doing image recognition, object detection, natural language translation, trend prediction. Emoji is a visual symbol widely used in wireless communication. It includes faces, hand gestures, animals, human figures, and signs. Instead of typographic, emoji are actual pictures. As there are increasing uses of emojis, the needs for an instant generator of emojis have become urgent. With the promising results of image recognition tasks by some deep learning models, we propose a real-time emoji generator, which enables the user to generate an emoji given a corresponding human hand gesture. In this work, we applied multiple deep learning models to perform feature extraction and pre-processed our data with data augmentation strategies and morphology operations.

2. Dataset

Our dataset is from the project on GitHub[1], which contains the training images and target emojis. In this project, we want to train our model to make it able to recognize the eleven emojis, and the sample of target (output) images and training images are shown in Figure 1 and Figure 2. For each of the emoji, there are 1,200 corresponding pixel style training images, which guarantee the balance of our training set. After the training process, we use a webcam to capture real-time hand gestures and process it to the pixel style images and make real-time predictions.

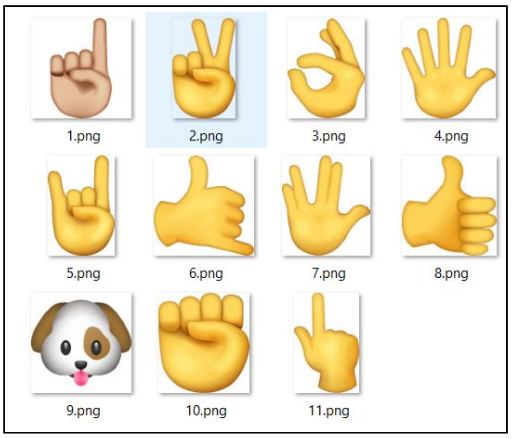

Figure 1. Sample of Target Image

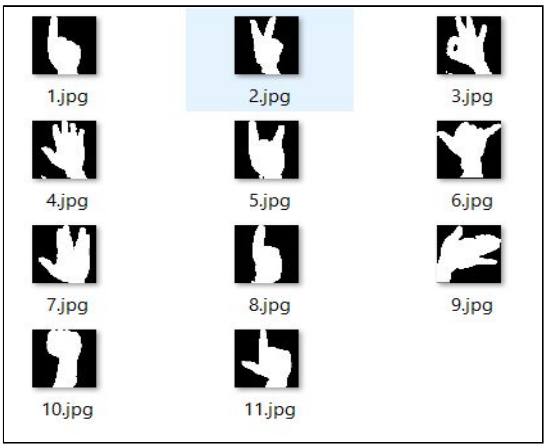

Figure 2. Sample of Training Image

2.1 Pre-processing

We found that the model we trained could not perfectly identify the hand gesture during our intermediate test. The initial idea we had was that the dataset only contains the left-hand gestures, which led to failures when recognizing right-hand gestures. We used ImageDataGenerator horizontal flip to get the other copy of the dataset, except it is all right-hand gestures this time. The result showed significant improvement, but there remained some problems. We realized that it is not realistic that we can always put our hands in the perfect position in the detected area. Therefore, we used horizontal and vertical shift and set the rotation_range to 10 and recursively combined all the results from ImageDataGenerator as a single training set to the model. We later found that the model performs well on the hands that look identical to the ones in the training dataset but performs poorly when the hands differ too much with the training dataset, i.e., longer ring fingers.

3. Models

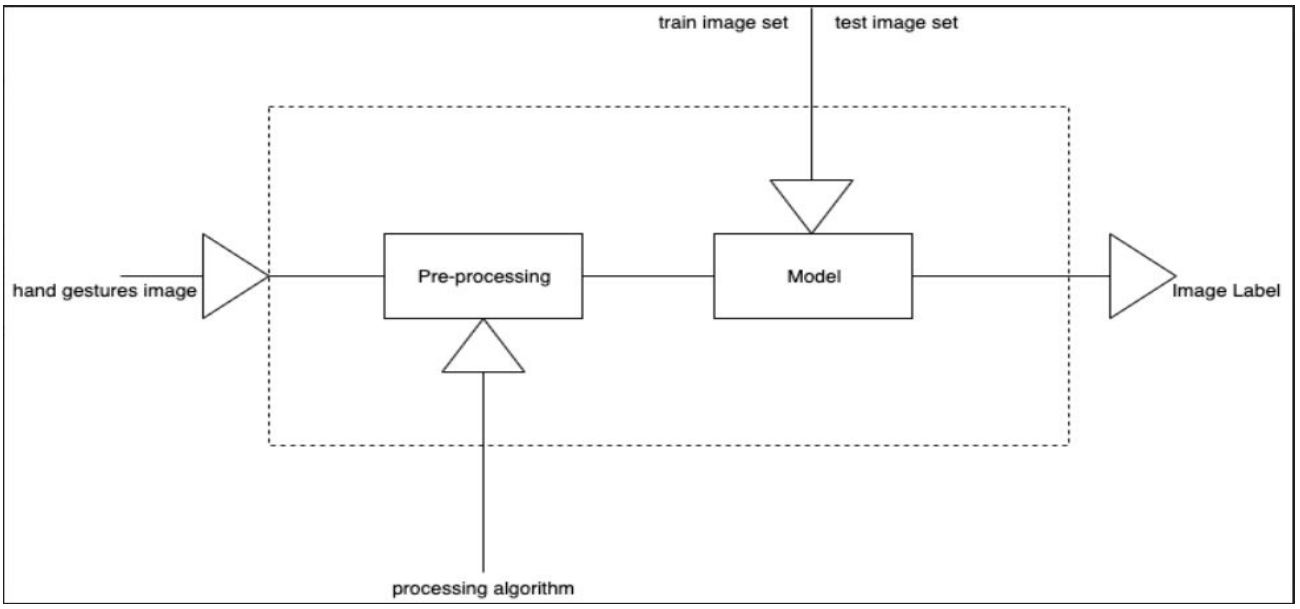

Figure 3: Workflow of the Model

3.1 Morphology Operation

Once we had the input image from the plug-in web camera, the first thing we needed to do was to unify the size of input images. Hence, we shrunk from 400*400 to 50*50 to match the resolution of the dataset. Furthermore, to match the input format of our model, which was trained by monochrome images, we filtered images by skin color range to convert to a black-white mask.

Moreover, we could not promise that if the background was clear enough to capture hand gestures without noises or little noises. We first want to ensure that the shape of the hand was correct, so we choose to erode it first.

After we gained general shapes of hand gestures, we eroded the input image by using large square 2 and dilated the image to get the relatively correct hand gesture using a large square 7.

To improve the accuracy of our system, we then blurred the images to arrive at our final input features using GaussianBlur.

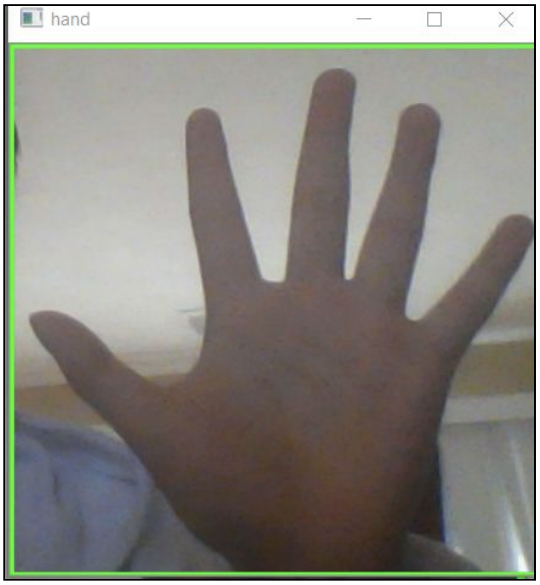

Figure 4: Input image directly from web-cam

Figure 5: Input image after morphology operation

3.2 Convolutional Neural Network

Our model uses the model of the convolutional neural network. It consists of convolutional layers, ReLu layers, and max-pooling layers. For our model, we have two sets of convolutional layers, ReLu layers, and max-pooling layers. After that, we put a fully-connected layer, a ReLu layer, and a softmax layer to predict the label. The model structure is in Figure 6. We have tried different sets of layers, and it turned out that the model in Figure 6 works the best.

```
Model: "sequential"
_________________________________________________________________
Layer (type)                 Output Shape              Param #
=================================================================
conv2d (Conv2D)              (None, 50, 50, 20)        520
_________________________________________________________________
max_pooling2d (MaxPooling2D) (None, 25, 25, 20)        0
_________________________________________________________________
conv2d_1 (Conv2D)            (None, 25, 25, 50)        25050
_________________________________________________________________
max_pooling2d_1 (MaxPooling2 (None, 12, 12, 50)        0
_________________________________________________________________
flatten (Flatten)            (None, 7200)              0
_________________________________________________________________
dense (Dense)                (None, 500)               3600500
_________________________________________________________________
dense_1 (Dense)              (None, 12)                6012
=================================================================
Total params: 3,632,082
Trainable params: 3,632,082
Non-trainable params: 0
_________________________________________________________________
```

Figure 6: CNN Model Structure

4. Results

We applied our model to the input hand gestures dataset. We had a very promising result: Both training accuracy and validation accuracy are both close to 1 (0.996, etc.), and both training loss converges to 0 after the training process is finished. The following are the learning curves of our training model.

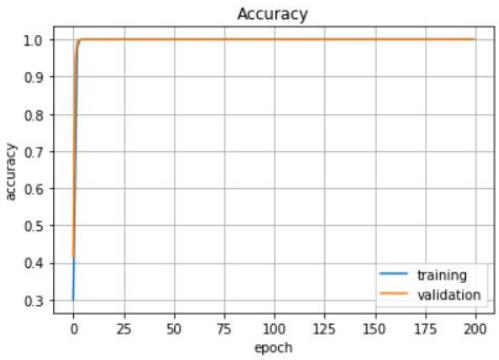

Figure 7: Accuracy vs Epoch

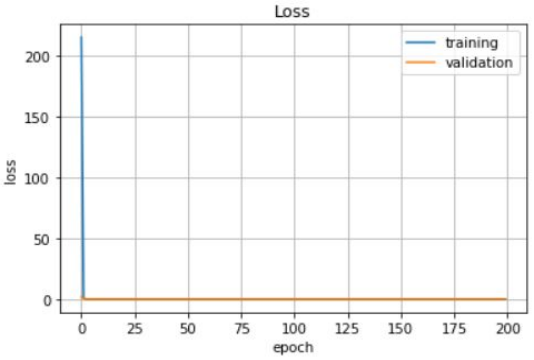

Figure 8: Loss vs Epoch

From the two graphs, we can see that the model converges very fast. During the first several epochs, the accuracy increases rapidly and reaches 1 around 20 epochs. The loss decreases to 0 around the same time. However, when we are testing our model in real-time instead of using static hand gesture images, the performance of the model is not stable. One reason is that the skin color range does not capture all parts of the hand due to the influence of light. The second reason is that we have to put the hand at the exact position in the frame as the position of the hand in the training images. Sometimes, it misclassifies hand gestures. However, if we adjust the position of our hand gestures, most of the time, our model will predict the right labels.

5. Conclusion

For this project, we get pretty good test scores, that the accuracy rate is close to 1.0. Deep learning is pretty good at image recognition, and the result reflects that. Although the accuracy is already 1, there are a couple of things that we can do to improve our model. First, we can test to find a better set of light sources and resolution of input images. For example, applying semantic segmentation pre-trained models with a label of lights will significantly reduce the unnecessary noise in our dataset. Second, we can use multiple snapshots from consecutive frames as input. From multiple label output, we can take the majority of the output labels as the final output label. Last but not least, we will want to include more datasets containing different sizes of hands to ensure our model can adapt to various hand shapes.

References:

[1] akshaybahadur21. “akshaybahadur21/Emojinator.” *GitHub*, 3 Apr. 2019, https://github.com/akshaybahadur21/Emojinator.