EECS 605 Emotion Recognition Project

Dongyao Zhang

Abstract

In this project, I build a React web app that given a human face image as input, can tell its emotion and give its predicted probability of emotions. The model is trained using VGG19 on FER2013 dataset and deployed on AWS. The website is published with Github and Heroku.

1 Problem description

The recognition of emotions is of great importance in developing intelligent systems, and emotion analysis also benefits research and finding in fields such as psychology and sociology. Given a human face image as input, the emotion recognition system is expected to tell the emotion on the face with its predicted probability of emotions.

2 Method

2.1 Neural networks

The architecture used is VGG19 [1], which consists of small convolution filters to increase the networks depth, leading to better performance on classification tasks. Its architecture is shown in Table. 1.

Layer	Hyperparameters
Conv	in channels=3, out channels=64, kernel size=3, padding=1, BatchNorm, ReLU
Conv	in channels=64, out channels=64, kernel size=3, padding=1, BatchNorm, ReLU
Maxpool	kernel size=2, stride=2
Conv	in channels=64, out channels=128, kernel size=3, padding=1, BatchNorm, ReLU
Conv	in channels=128, out channels=128, kernel size=3, padding=1, BatchNorm, ReLU
Maxpool	kernel size=2, stride=2
Conv	in channels=64, out channels=256, kernel size=3, padding=1, BatchNorm, ReLU
Conv	in channels=256, out channels=256, kernel size=3, padding=1, BatchNorm, ReLU
Conv	in channels=256, out channels=256, kernel size=3, padding=1, BatchNorm, ReLU
Convl	in channels=256, out channels=256, kernel size=3, padding=1, BatchNorm, ReLU
Maxpool	kernel size=2, stride=2
Conv	in channels=256, out channels=512, kernel size=3, padding=1, BatchNorm, ReLU
Conv	in channels=512, out channels=512, kernel size=3, padding=1, BatchNorm, ReLU
Conv	in channels=512, out channels=512, kernel size=3, padding=1, BatchNorm, ReLU
Conv	in channels=512, out channels=512, kernel size=3, padding=1, BatchNorm, ReLU
Maxpool	kernel size=2, stride=2
Conv	in channels=512, out channels=512, kernel size=3, padding=1, BatchNorm, ReLU
Conv	in channels=512, out channels=512, kernel size=3, padding=1, BatchNorm, ReLU
Conv	in channels=512, out channels=512, kernel size=3, padding=1, BatchNorm, ReLU
Conv	in channels=512, out channels=512, kernel size=3, padding=1, BatchNorm, ReLU
Maxpool	kernel size=2, stride=2
Avgpool	kernel size=1, stride=1
Dropout	$p = 0.5$
Linear + (Softmax)	$512 \rightarrow 7$

Table 1: VGG19 architecure

2.2 Data processing

The dataset used for training is FER2013, which consists of 48×48 gray scale images with 7 emotion labels. It has 28709 examples as training set, 3589 examples as public test set, and 3589 examples as private test set.

For training, I preprocess the grayscale image by concatenating itself three times along the channel and converting the 0-255 pixel value range to 0-1. For each image, I crop ten $44 \times 44 \times 3$ around the center of it to augment the data and increase robustness.

For testing, I convert a RGB image to a grayscale image with pixel value range 0-1, resize it to 44×44 and concatenating itself three times along the channel to get a $44 \times 44 \times 3$ processed image.

2.3 Training process

The nerual networks is trained with 0.01 as learning rate, 128 as batch size, Cross Entropy as loss and SGD as optimizer for 250 epochs. During training, the learning rate decays with a rate of 0.9 after 80 epochs.

2.4 Evaluation

The method is mainly evaluated by accuracy, which is computed by

$$\text{accuracy} = \frac{\text{True Positive} + \text{False Negative}}{\text{True Positive} + \text{True Negative} + \text{False Positive} + \text{False Negative}}.$$

The accuracy of VGG19 is about 72.39% for all emotions.

The confusion matrix in Fig. 1 shows other metrics for the emotions. The true positive rate for Angry, Digust, Fear, Happy, Sad, Surprise and Neutral are 0.52, 0.73, 0.59, 0.88, 0.62, 0.80 and 0.75 respectively. Happy and Fear have the highest and lowest rate. Besides, the algorithm is more likely to classify Disgust as Angry, Fear as Sad or Sad as Neutral with rate of 0.18, 0.17 and 0.17 respectively. This is understandable since the facial expressions for those emotions may be similar.

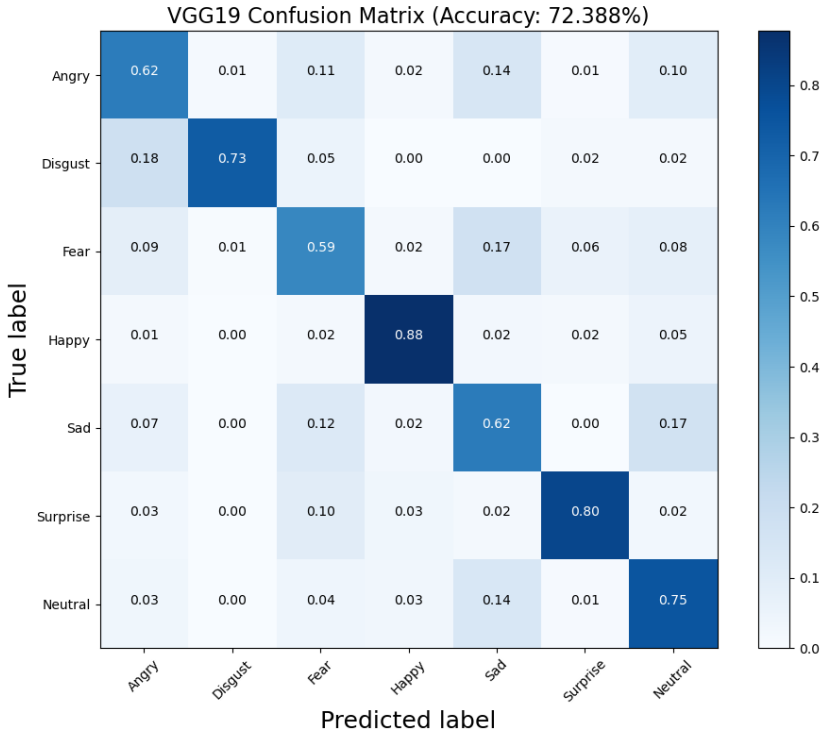
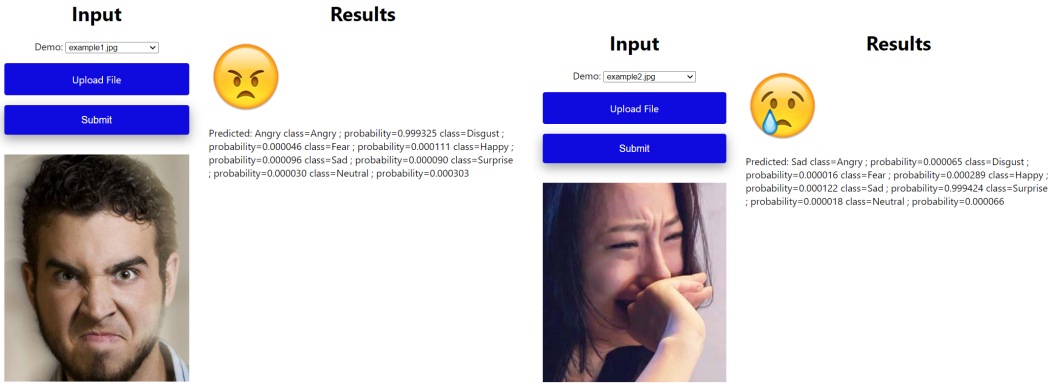


Figure 1: Confusion matrix

2.5 Results

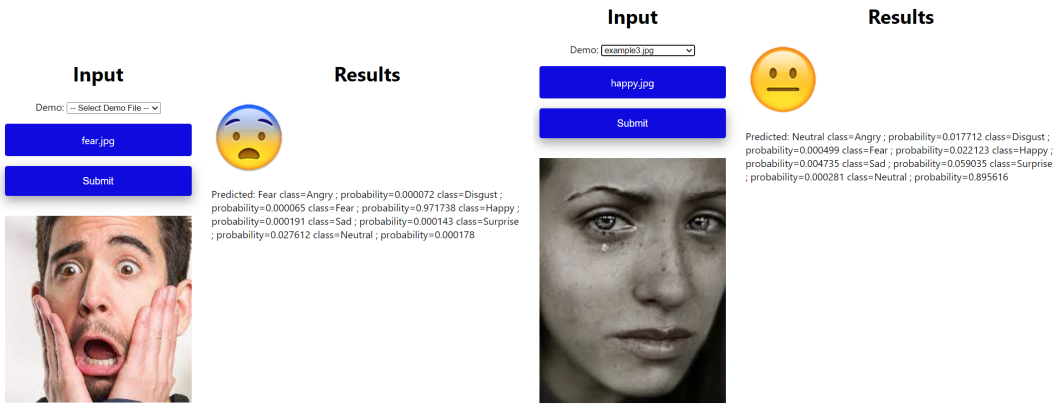
A. Good examples

In most of the cases, the algorithm works well with accurate prediction. Fig. 2 shows several examples works well on the website. The website uses an emoji face to show the predicted emotion, and gives the predicted probabilities for it.



(a) Example 1

(b) Example 2



(c) Example 3

(d) Example 4

Figure 2: Good examples

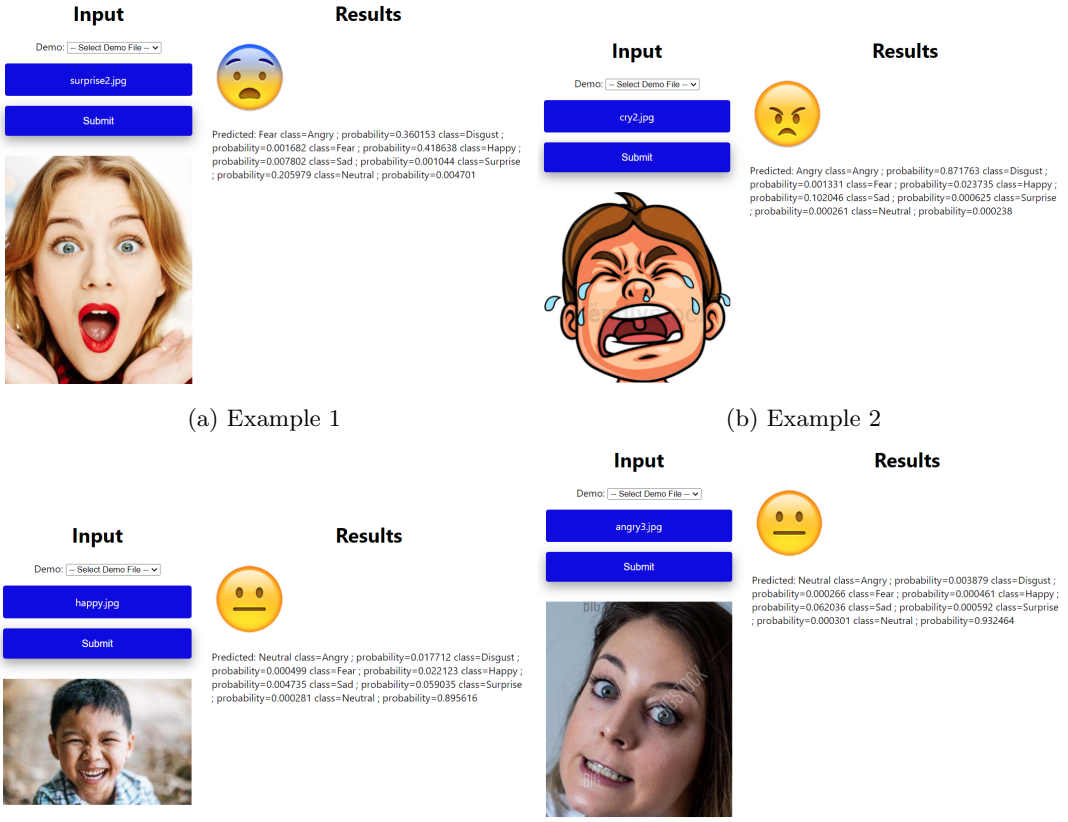
B. Bad examples

Some bad cases can be seen in Fig. 3.

There are several reasons for the bad results. Firstly, the dataset used for training is a human face dataset, in which each image contains a front view face taking up a large portion. The learned networks may not be able to predict the emotion for other types of faces such as a cartoon character face. When the face on the image is not large enough, it also cannot give a precise prediction. Secondly, emotions are intrinsically complicated. Different facial expressions can convey similar emotions, while similar expressions may involves different emotions. The boundaries between some emotions are not clear, and different people have different interpretation of emotions. Therefore, the algorithm might misunderstand some emotions.

3 Deployment

I export the networks architecture and the learned parameters into an onnx model. For the deployment, I put the model, code for inference and transmitting the data, required packages and the emotion images into a .zip file, and upload it to the Lambda Function. I use a POST API to connect the React app front end with the Cloud backend so that the users can submit an image as input and get the result produced from the AWS cloud.



(a) Example 1

(b) Example 2

(c) Example 3

(d) Example 4

Figure 3: Bad examples

I also add a dropdown menu providing some example images for demo on the website. This is completed by putting the example images into an AWS Bucket and using a POST/GET API to connect the frontend and backend together.

The deployment and connection between frontend and backend can be seen in Fig. 4.

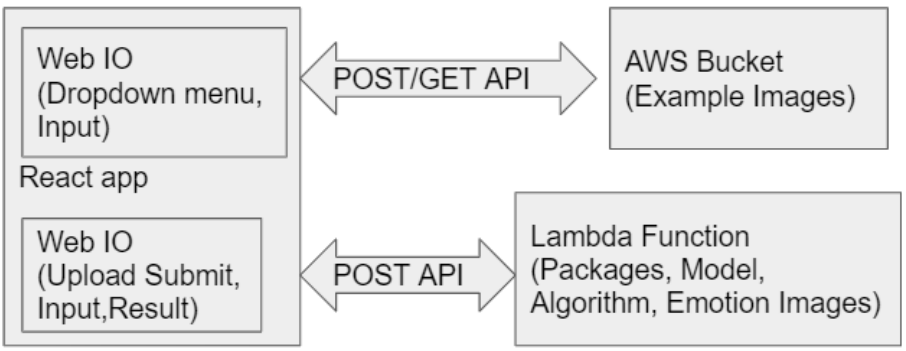


Figure 4: Deployment structure

4 Challenges

The challenge in the training process is the limited computational resource and slow learning after several epochs. I solved running out of memory problem by using a smaller batch size and reduced

the learning rate as the learning process slowed down to encourage better learning.

The challenge in the deployment is that loading the original model requires large packages such as torch, torchvision. The loading process is slow, which could exceed the API timeout in a cold start, which happens when the Lambda function is invoked for the first time, and those packages take much storage, which cannot be unzipped on cloud. I tried other tools including ECS, Cloud9 for the packages and algorithm. Finally, I was able to restore the networks and the parameters as an onnx model. Then I did not need the large packages for the deploy and could perform a faster inference on cloud.

5 Remaining challenges

The remaining challenges are to have a better data visualization section and an active learning component. The data visualization may require deploying packages like matplotlib or visualization tools for web development. To add an active learning component or other more interesting components, better ideas for data loading, algorithm implementation and deployment are required.

References

[1] Karen Simonyan and Andrew Zisserman. Very deep convolutional networks for large-scale image recognition. *arXiv preprint arXiv:1409.1556*, 2014.