Emoji prediction

Liu Yuzhou(21100602d)

Responsable of the experiment or other information

1. Abstract

Emoji category detection is essential nowadays, as people rely on social media a lot. By viewing what kind of emoji is included in the message that the user has sent to each other, we can easily detect whether the attitude of the message is positive or negative, or even predict what kind of topic the conversation is related to. The task I will work on is to predict the emotion or the category of the emoji based on the model I have designed.

Keywords-Emoji classification, machine learning, computer vision

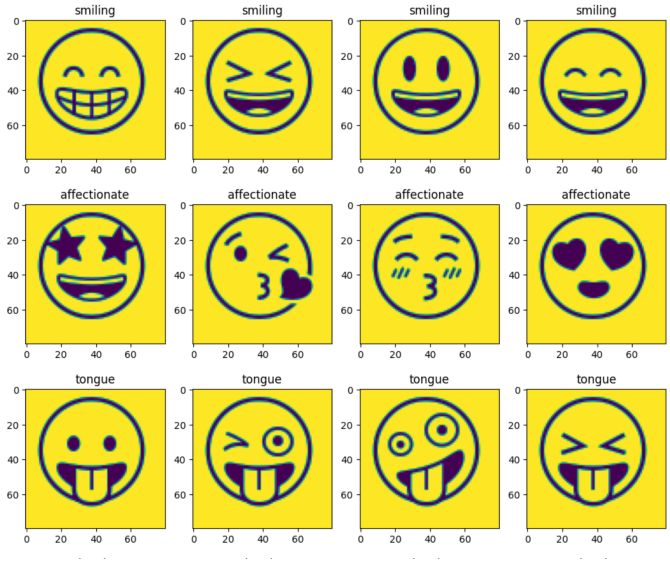

Figure 1. *Emoji kitchen emojis classes*

In this report, we will introduce:

(1)what general method I selected to train and test

(2)what and how features are extracted from the captured emojis

(3)how the model is trained, validated and tested, and how it performs in a real world scenario; here is to see how it performs on human images

(4)what problems I have encountered and how I solve them.

2. Dataset

The first challenge I encountered was the difficulty of collecting the data set. At first, I could not find an appropriate dataset online, so I tried to collect the data manually, but then the problem was that the amount of data was too limited, and the result was unsatisfactory. Finally, I got this dataset called "**Emojis Unicode Codepoints**" on Kaggle; there are two attributes describing the groups, which are **group** and **sub_group**. Because the group only have 10 unique labels, so I adapted to use the subgroup attribute. I picked 24 different labels(840 samples) from the dataset, and for each label, it has 35 different samples. Later on, to improve the test result, I also collected some .png images in order to preserve the colour information(**It also boosted previous validation performance a lot, but in sections 4&5, we will focus on the improvement on test set**).

	index	emoji	name	group	sub_group	codepoints
0	13	🥰	smiling face with hearts	Smileys & Emotion	face-positive	1F970
1	14	😍	smiling face with heart-eyes	Smileys & Emotion	face-positive	1F60D
2	15	🤩	star-struck	Smileys & Emotion	face-positive	1F929
3	16	😘	face blowing a kiss	Smileys & Emotion	face-positive	1F618
4	17	😗	kissing face	Smileys & Emotion	face-positive	1F617
...	...	...	...	...	...	...

Figure 2. *Emojis Unicode Codepoints dataset*

3. Method

3.1. Overall

To begin, I assess multiple conventional models using the dataset and determine the model that achieves the highest performance. This selected model is then utilized for the subsequent classification task associated with the **Bag of Visual Words (BoVW)** method. Afterwards, I constructed a simple **CNN** model named EMONN. Then I did a comparison between it and the traditional machine learning model. Finally, I experiment with merging the strengths of both approaches by employing the BoVW method on the output produced by the **Convolution layer**. Finally, the performance model was tested on an unseen data set generated by the emoji kitchen and a human face data set.

3.2. General Set Up

Before building my model, I simply test some of the basic machine learning models to see their performance; this will help us have an overall sense. Also have a measure to determine which model to use in the following process.

Model/Classifier	Accuracy
SVM-Linear	81.95%
Random Forest Classifier(Later just call it RFC)	79.76%
KNN	70.24%

Table 1. *Training performance on Emojis Unicode Codepoints dataset*

Because the results of SVM and RFC are close to each other, I use the K-fold method to further validate the results. Finally, I found SVM has the best performance, and the mean accuracy is around 82%. And it only generates good results when the kernel is set to be "linear". The poor performance of KNN may be related to its sensitivity to parameter setting and initialization.

3.3. BoVW(SIFT+SVM)

3.3.1. Feature extraction.

SIFT is a strong method for detecting significant features in

features, but it is most commonly used in colourful and grey images. However, in our case, as we only have binary images which only have black and white pixels, it is hard for SIFT to capture enough key points. The convolution operation is strong in edge detection and can enhance features from different directions. By combining these two methods, I believe we can explore the power of feature extraction for images.

3.3.2. Training.

Here, we can perform semi-supervised learning by first getting a word histogram for each image(Unsupervised learning). And then put the extracted histograms and labels into an SVM classifier(Supervised learning). Because the image is in binary form, which only contains white and black pixels, it should generate good performance on edge detection.

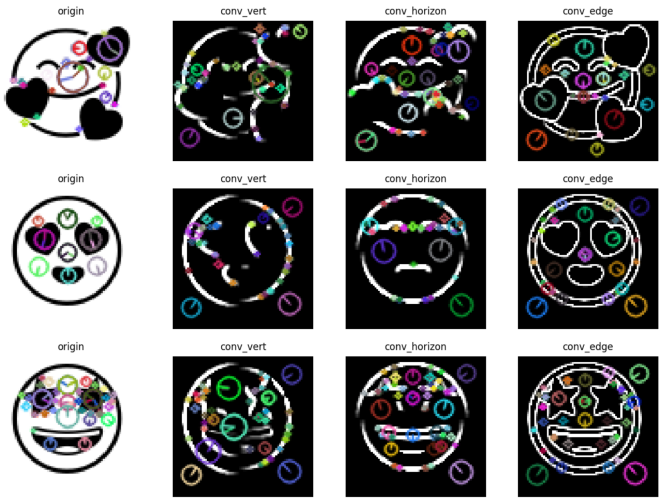

Figure 3. *SIFT on different images enhanced by different filters*

The result in **Figure 3** shows that as we enhance the image from different directions(By using different filters), the SIFT algorithm can detect the feature from a different perspective. Then, we are going to obtain histograms for different images. But there is a consideration before training: will putting all the descriptors together import redundant information, which could worsen the training result?

To answer this question, I used three ways to train the model:

1. Classification with descriptors of the original image.
2. Choose descriptors of the image which has the maximum number of **key points(kpt)** among four different images.
3. Combine all the descriptors of the four different images.

Finally, we fit the histogram of each image into the **SVM** classifier and validate the result.

The following **table 2** shows the validation results.

Model/Classifier	Accuracy
SVM with original image	78.17%
SVM with (max kpt)	82.93%
SVM with combination	85.36%

Table 2. *BoVW performance on Emojis Unicode Codepoints dataset*

The combination set seems to provide the most meaningful information as it obtains the highest accuracy.

Figure 4 shows the 10-fold validation result on the combination set.

```
SVM(combination) in fold 1's accuracy : 80.95238095238095%
SVM(combination) in fold 2's accuracy : 89.28571428571429%
SVM(combination) in fold 3's accuracy : 76.19047619047619%
SVM(combination) in fold 4's accuracy : 84.52380952380952%
SVM(combination) in fold 5's accuracy : 82.14285714285714%
SVM(combination) in fold 6's accuracy : 91.66666666666666%
SVM(combination) in fold 7's accuracy : 91.66666666666666%
SVM(combination) in fold 8's accuracy : 88.09523809523809%
SVM(combination) in fold 9's accuracy : 82.14285714285714%
SVM(combination) in fold 10's accuracy : 86.90476190476191%
The overall mean score is 85.35714285714285%
```

Figure 4. *Performance of 10-Fold Validation*

We can see that though the model accuracy sometimes got above 90, but the robustness is still not enough, as the worst case is only around 76%.

3.3.3. Difficulties & trials on improvement.

1. (**lack of descriptor**) At the first stage, I tried to train based on the different convolution results of the image individually, but some of them turned dark(or seemed totally damaged through convolution, and no kpt observed). So, it doesn't obtain any descriptor. To solve this problem, I use the combination set.
2. (**Visual word number setting**) The model should be sensitive to the number of visual words; finding an appropriate number is super hard. As for the set of SVM with the original image, the feature that can be detected is limited. So the number of words might be set to less than 150, but finally, in the combination set, the amount of descriptors is supper large, so I set the number of words to be 1000. Of course, I believe there should be a better number that balances the redundancy and variability of the information.
3. (**Choice of filter**) We can still try to optimize the choice of filters(see in section 3.5).

3.3.4. Reflection.

The performance of the model does not meet my expectations. The result was slightly improved compared with the one in the general set-up, but it did not boost. I will try to see if I can optimize the filter

3.4. A comparison between CNN and traditional way

3.4.1. Feature extraction.

I am considering using the **Local Binary Pattern(LBP)** method to extract the feature first. LBP can efficiently capture the difference between the central pixel and its surrounding pixels. CNN can go through the area of the whole image, then capture and summarize the local features in different areas of the image. I believe, by combining these two methods, we can eventually reach a decent output result.

3.4.2. Training.

Though **circle LBP** has overcome some disadvantages of the original LBP method, our dataset is too large and takes too long to process all the samples, so I stick with the traditional way. And it just takes a few seconds, maybe because calculating all the points around the circle is quite time-consuming. However, the original LBP method only gets a set of copies for

the surrounding pixels and summarizes them, which is quite efficient.

I first obtained the LBP feature for each image during the training process. Then it comes to the design of CNN, I just simply name it as EMONN. The overall structure of EMONN is to first use 2 convolution layers to feed-forward the LBP images; after each CNN layer, I inserted a max pooling layer to reduce the image size. It can remove the redundant information. Last, I flatten the image and forward it by 2 Linear layers to obtain a vector shaped (1:24) for each image, and the index of the maximum value in the vector indicates the class it belongs to.

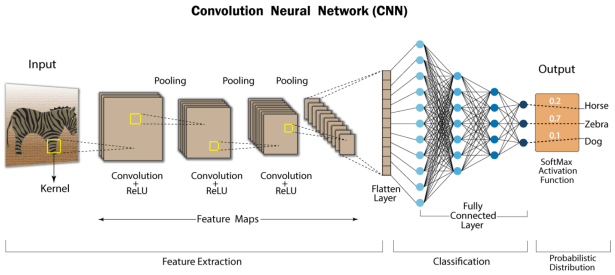

Figure 5. *Structure Of My CNN*

3.4.3. Difficulties & trials on improvement.

1.(**Problem-related to the dataset**) In the first stage, I faced the problem that the network outputs the same prediction for each sample in the validation set. I tried many ways to modify my model, but the problem still exists. However, after I looked at the dataset I collected, I found that 617/2600 emojis are labelled as person-roles in my dataset, which is the output of my model. So I rearranged the dataset, sorted all the emojis, and for each class, I picked 35 images. Finally, the problem was solved.

2. (**Loss fixed at a point**) Then, I encountered the problem that the loss was fixed at around three, and after a few epochs, the loss didn't change. I think that might have been caused by the Adam optimizer starting at a bad position, and it stuck there, so I changed it to SGD. Then the problem was solved. This problem is uncommon in traditional machine learning, as the optimization method is diverse; for example, the decision tree is built based on the ID3 algorithm, Naive Bayes relies on probability calculation... while deep learning typically relies on a gradient.

3.(**Problem of overfitting**) Finally, the result indicates that the model may face some overfitting problems; I have tried many methods to deal with the problem. While the training has around 100% accuracy, the performance on the validation set is not that good, which is around 80%. After I removed a linear layer(from three to 2), the result on the validation set increased to 83%. Although the problem was not totally solved, the result improved a little bit.

But the LBP method is not that helpful here; I think the main reason is that the image only has white and black colour, so no significant features can be indicated here.

Model/Classifier	Accuracy
CNN with LBP	82.73809523809523%
CNN without LBP	82.73809523809523%

Table 3. *Validation result*

3.4.4. Reflection.

Although the result is not as high as I expected, probably because the dataset is too small. The CNN method indeed enhanced my understanding of traditional machine learning methods. **First**, I have learned the importance of data sampling; for example, if the positive sample is the majority, but just includes a few negative samples, the model will classify all the things as positive to minimize the loss. But that is not our purpose of training. At that point, we may need to recollect or resample the data set. **Second**, compared with the traditional method, deep learning performs well when the dataset is large because the model complexity is difficult to adjust, and it can easily get overfitted **(**The second problem also exists in traditional machine learning methods if the model complexity doesn't suitable for the size of the data set**)**.

3.5. Midway between 2 methods(To work on intermediate image create by CNN)

Deep learning doesn't work well on a small data set, and the **BoVW** method may be enhanced by utilizing an optimized filter. Then, can we apply the advantage of the CNN filter to enhance the performance?

3.5.1. Feature extraction.

The answer is definitely yes. The way I am gonna work it out is to get the intermediate output from the **Convolution layer**. Then, I tried 2 ways to extract the features; the first way is to continue using BoVW method. The second is to **build histograms** for different images accordingly.

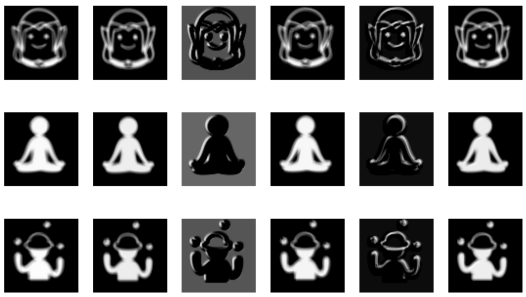

Figure 6. *Output of the first CNN layer*

Then I will explain both extraction methods one by one:

1. The BoVW method has already been illustrated in section 3.3, and the work here is similar.
2. The second method I have chosen is to use the histogram.

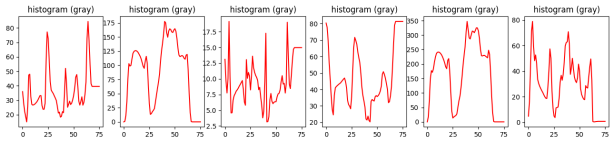

Figure 7. *Histogram of pixels for 6 different filtered images*

Here, how I am building the histogram differs from the one illustrated in the tutorial. Instead of having a statistic about each pixel value, I just **summed up the value of each row** of the image. I believe it can provide useful information for training.

3.5.2. *training.*

We can see that the CNN model has enhanced the graph from 6 different perspectives through different filters, as shown in **Figure 6**. And here, I choose a new strategy to make the prediction: **voting**. Because there are 6 independent filters obtained from the CNN model. So, for the image generated from each filter, I build a sub-group, and for each sub-group, I train it independently. At last, when doing prediction, I will get the predicted result from each group of images. Now we vote according to the collected result from each sub-group, and then the prediction is finished!

Model/Classifier	Accuracy
BoVW(SVM-Linear)	85.17%
Histogram+SVM-Linear	88.69%
Histogram+KNN Classifier	78.57%
Histogram+RFC	89.88%

Table 4. *Validation performance for different voting models(All With CNN)*

We observe from **Table 4** that the Histogram+RFC performs best among different classifiers, which is close to 90%. From **Figure 8**, we can also see that the voting strategy is pretty useful, as the accuracy for each subgroup is just around 80%, but after we apply the voting for all the groups, the accuracy is boosted to around 90%.

```
Random Forest: Accuracy for sub-group 1 SVM: 79.76%
Random Forest: Accuracy for sub-group 2 SVM: 80.36%
Random Forest: Accuracy for sub-group 3 SVM: 79.76%
Random Forest: Accuracy for sub-group 4 SVM: 81.55%
Random Forest: Accuracy for sub-group 5 SVM: 79.17%
Random Forest: Accuracy for sub-group 6 SVM: 79.17%
```

Figure 8. *Training result for each subgroup*

3.5.3. *Difficulties & trials on improvement.*

1. (**Result for BoVW does not improve**) As it shows from the **table 4**, the result for BoVW method is still around 85%, I think the potential reason could be: that sift is not suitable here, because the change of colour is still not that significant in the optimal filtered image, and the similar pattern and shape is not that easy to be find. So, I came up with the histogram method.
2. (**Overfitting for training**) Unfortunately, we still suffer from overfitting here, as the performance on the training set is always 100%, which means it is overfitted. Then, I tried to find a solution for that. The best way might be to collect more data for training, but I thought that was not effective for me. Finally, I came up with the idea of voting, and it boosted the result. However, it does not really solve the problem.

3.5.4. *Reflection.*

There could still be some limitations for this model, as the overfitting of the training set is not really solved. Maybe one day we could find a suitable model which has appropriate complexity for that, but I think the best way is to enlarge the dataset. If the problem could be solved, we can train the voting classifier based on the training set first and then apply it to the test set; I believe that could further improve our performance.

4. Test

4.1. Illustration on test result

Then, we test our model by using the unseen emoji generalized by the emoji kitchen. The following **Figure 9** is the best result I have obtained, which is 37.5%, as I only used 8 images for testing. Surprisingly, the result is not obtained by the voting model, which performs best in the training. It was obtained from the RFC.

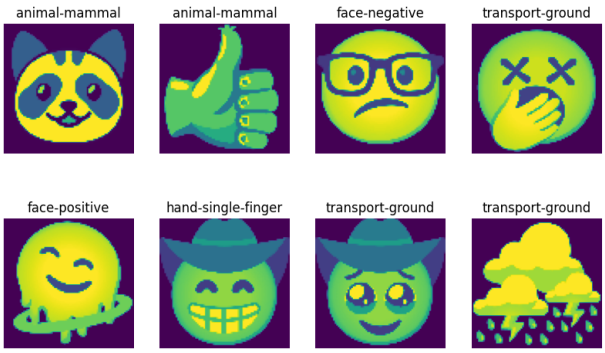

Figure 9. *Testing result for each subgroup*

Following **Table 5** is the result obtained from different models.

Model/Classifier	Accuracy
CNN	12.5%
Histogram(Voting)+RFC+CNN filter	12.5%
Histogram+SVM	25%
BoVW(SIFT)+RFC	37.5%

Table 5. *Testing result*

As the result is quite unsatisfied, I decided to collect some colourful emojis to enhance the model. Definitely, there is a gap between our test emojis and the training emojis. Because the emoji for training are in binary form, and the testing emojis are colourful images. **Figure 10** shows the emojis I have collected. Each folder contains 35 colourful images of emojis in the emoji kitchen(Totally 175).

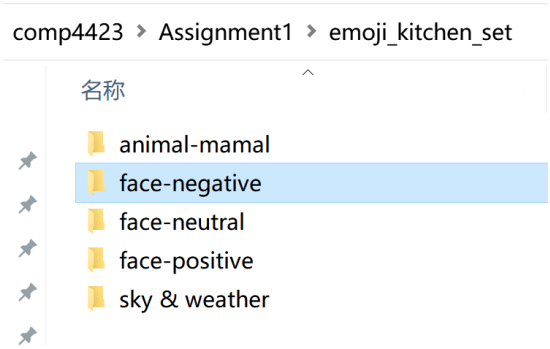

Figure 10. *Self collected emojis*

I trained the model based on the new data combined with the original data. The performance improved a lot, although it is still not that perfect, but the label seems more reasonable.

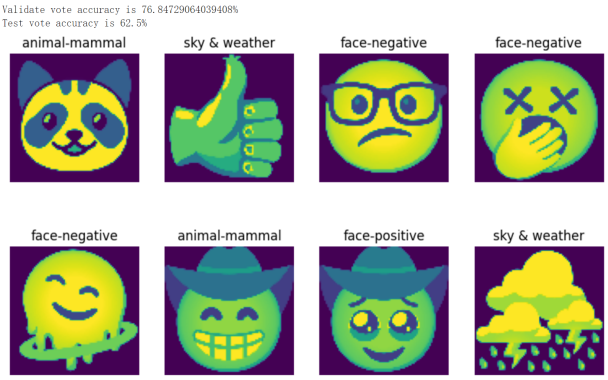

Figure 11. *Improved testing result for BoVW(With CNN filter)*

And by looking at **Figure 11** and **Figure 12**, we can see that voting really helped a lot in improving the result. As all the sub-groups didn't obtain the best accuracy as 62.5 % for testing.

```
Start k-means: 125 words, 39729 key points
Val Accuracy: 69.95%
Test Accuracy: 37.50%
Start k-means: 125 words, 50333 key points
Val Accuracy: 74.38%
Test Accuracy: 25.00%
Start k-means: 125 words, 42008 key points
Val Accuracy: 71.43%
Test Accuracy: 50.00%
Start k-means: 125 words, 36945 key points
Val Accuracy: 71.43%
Test Accuracy: 37.50%
Start k-means: 125 words, 42225 key points
Val Accuracy: 75.37%
Test Accuracy: 25.00%
```

Figure 12. *Improved testing result for BoVW(With CNN filter)*

Following **Table 6** Summarized the improved testing result.

Model/Classifier	Accuracy
CNN	50%
Histogram+RFC+CNN filter	50%
BoVW(Voting)+RFC+CNN filter	62.5%

Table 6. *Improved result*

4.2. Reflection

1. (**Test performance doesn't meet expectation**) The real-world emojis are not as simple as what we faced during training. They are combined with colourful pixels, while our training samples only consist of black and white pixels(So I think the unsatisfied result is reasonable). I did collect colourful emojis for some of the sub-groups, and we could see that the performance also improved a lot. So I believe we could boost the accuracy to nearly one hundred by collecting more colourful data. However, due to the limited time, I could not collect colourful samples for each group(which is around 850 images).
2. (**Initially poor performance for voting classifier**) From observation of testing results, I think the voting only works well when all the sub-model gives a satisfactory performance. Otherwise, it will make the situation worse(Even the correct one can be misled); we can see when the performance of all the sub-groups improves, it can even give the best result.
3. (**Point on best result**) We can see that generally, the detection result is reasonable for each testing emoji, except the hand-single-finger one. Probably because I didn't collect colourful images for that group. And I think some patterns may be misleading, to help the model learn more localised features, I believe the performance can still be improved.
4. (**Result of CNN model**) Here, the result of CNN is quite reluctant compared to other traditional machine learning models, as it does not always stop at the same point. I think it is mainly caused by the large parameter space and limited training data; if we could let the model learn more general principles, I believe its performance would be more stable and keep improving.

5. Predict Human emotion

5.1. Data collection

The following **Figure 10** shows part of the human image data set found from Kaggle, which is ed "Natural Human Face Images for Emotion Recognition".

Figure 13. *Part of human data set(Anger)*

I divided the human emotion images into three sub-groups(positive, negative and neutral) for convenience, and I picked four pictures from each set for testing. To resolve the colour information missing problem in **section 4**, I picked the three sets, which are face-positive, face-negative, and face-neutral, from my self-collected data set(105 images).

5.2. Training

First, I only use the face-related emojis for training(More specifically, the self-collected 105 emoji png data set). Here, exclude the feature extraction method I previously mentioned. I also apply a strategy called mask to help the model learn better.

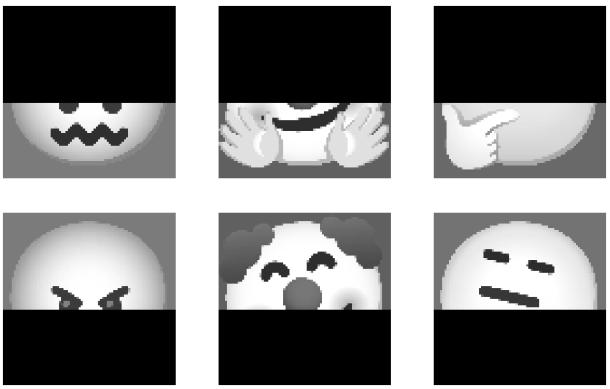

Figure 14. *Upper masked and lower masked img*

So, the mask is to create more samples by hiding the upper part and the lower part of the image, respectively. Then, creating a training sample for both images, by doing so, we tripled our training data.
The testing result based on the enlarged data set is shown below in the **Table 7**.

Model/Classifier	Accuracy
SVM(baseline)	33.33%
CNN	41.67%
Histogram(Voting)+RFC+CNN filter	33.33%
Histogram+SVM	50%
BoVW(SIFT)+RFC+CNN filter	58.33%

Table 7. *Testing result*

Figure 15 shows my best testing result (The correct labels for each row are positive, negative, and neutral, respectively). As we can see, the neutral face is really hard to detect by the model; all of them are wrong (From my perspective, it also be hard for me as a human being). The second image on the second row might be misclassified caused by the curve of its mouth; it is similar to the laugh angle.

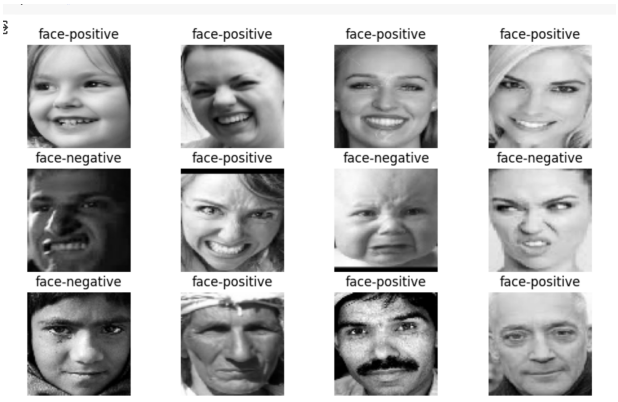

Figure 15. *Best prediction result on human emotion(58.33%)*

5.3. Reflection

1. (**key point**)The mask strategy is successful as it boosts performance. I came up with the solution because the emotion should be predicted based on the shape of the eyes and mouth of the emoji(Similar to a human). And there could be some misleading factors, such as the mask or the sunglasses worn by the character. Force the model to get some regional information(e.g. when it meets this kind of mouth, classify it as positive...).
2. (**Potential improvement**)To improve the result of this model, we could optimise the mask for each image; as we can see from **Figure 14**, some of the images are not perfectly masked(Some of the upper masks didn't hide the eyes, the lower mask didn't hide the mouth).

Also, I think the performance met my expectation as the model had never seen human images before.

6. General Limitation

1. As I cannot find any food-related object in the Kaggle data set we are using, we cannot predict any food group. To enhance the model, we may need to have more classes. Also, for some of the classes, such as face-smiling and face-tongue... because the data is too limited, I decided to have a general class for them, which is face-positive, face-neutral, and face-negative, to have more accurate prediction results; we may consider collecting more data for each sub-group.
2. For unseen datasets and human emotion prediction, the accuracy is not that good; we could add more training data to optimise the data set and put more effort into making optimised masks in order to overcome that issue.

7. SUMMARY

In this essay, we compared methods using deep learning and traditional machine learning, and combined them together in order to explore the power of feature extraction. Finally, we tested the model on some unseen images generated by the emoji kitchen and some human facial expression images, respectively. Although the result is not perfect for each method, the good news is that it keeps improving. Something noticeable in this project is the masking technique and the voting based on the different filters of the CNN layer, including histogram feature extraction mentioned in section 3.5; these techniques improved the performance of our model a lot.