SGN-26006 Signal Processing Laboratory

Image Recognition

SUBMISSION REPORT

Group: Ok_luon (An Ton, Quan Do)

Designing and training the image recognition model

Because the task's data is not large, I can upload all the files and folders needed to Google Colab and train the Machine Learning model.

1.1. Preprocessing

I used Pillow library to get the image and convert them into RGB color grading.

I also linearized the image so that it would be in the range of 0 and 1.

For the loop, I used the Tqdm library to see the processing bar running.

```
print("\nThe shape of data is: {}".format(X.shape))
print("The shape of label is: {}".format(y.shape))

100%|██████████| 4000/4000 [00:04<00:00, 869.16it/s]
100%|██████████| 4000/4000 [00:00<00:00, 1360461.89it/s]
The shape of data is: (4000, 64, 64, 3)
The shape of label is: (4000,)
```

As can be seen in the figure, the number of images is 4000.

Split train and test images

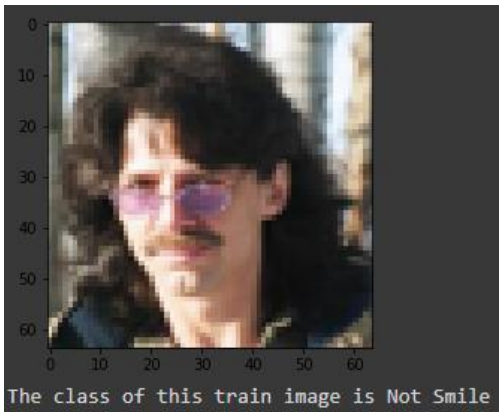
I used sklearn.model_selection.train_test_split to split the training data into training and validation data. Moreover, I used `stratify` method to distribute the data class into train and validation nicely.

```
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2, stratify=y)
print("Train size: {}\t Validation size: {}".format(X_train.shape, X_test.shape))

Train size: (3200, 64, 64, 3)    Validation size: (800, 64, 64, 3)
```

Because this task has only two classes to predict, I didn't turn them into one-hot-encoding and use binary cross-entropy for the loss calculation function later.

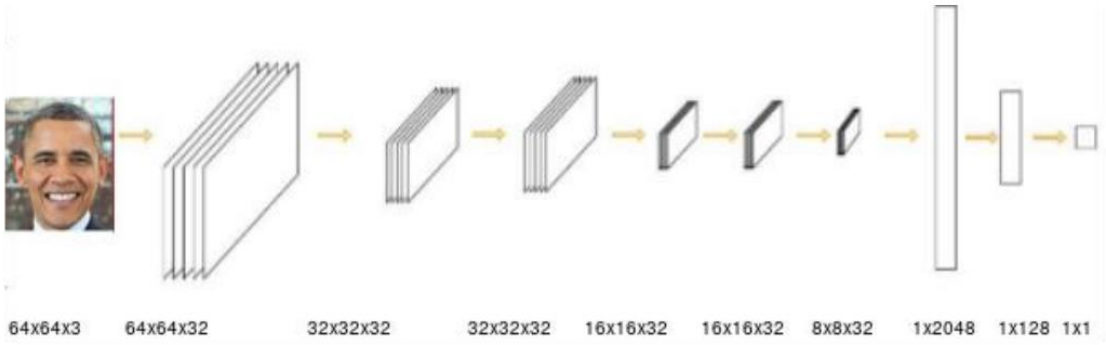
I tested a training data with its label to see if I have assigned everything correctly



1.2. Model selection and training

I used the Convolutional Neural Network (CNN) for the image recognition part of this task because it uses convolutional layers to easily extract the features and edges.

The base model includes



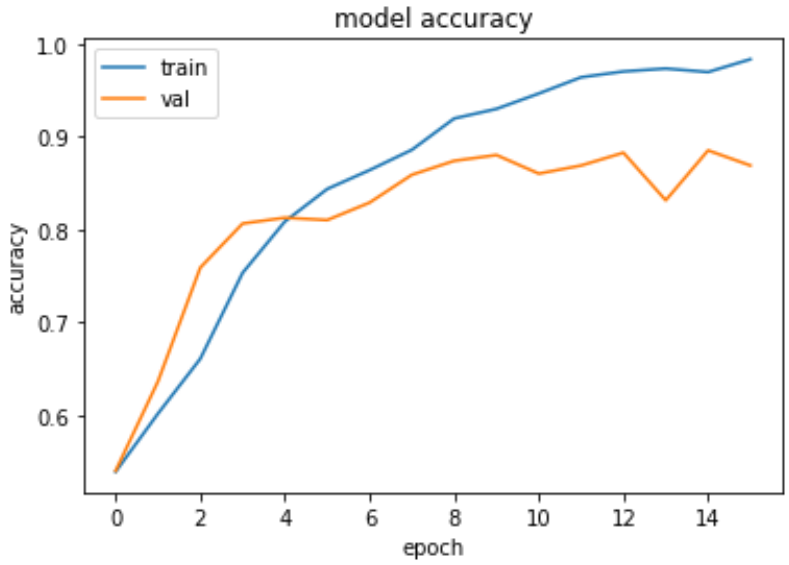
However, I tried to change a little bit from this by adding:

- BatchNormalization layer: The whole image batch is going to normalize. Therefore, it will increase the generalization and make the model predict the features better.
- Dropout: I added a dropout layer of 0.5, which, by experience, make my model perform better.
- Add more convolution layers than the base model in the above figure.
- Add more dense layers. Moreover, I added `kernel_initializer` so that the model would have the parameters to train faster.

Train the model:

- I added callbacks to make the model more efficient.
- Firstly, the ModelCheckpoint, so that I can save the weight and the model to re-train purposes or my computer got shut down, I still have the weight file to load and train again.
- Secondly, ReduceLROnPlateau is the callback that helps me reduce the optimizer's learning rate. If my model stucks in local minima because of a large learning rate, I can reduce it with patience.
- Thirdly, EarlyStopping allows my model to stop if the validation accuracy is converged.

This is the result of the training:



As can be seen, I trained the machine learning model in 50 epochs. However, because I have the EarlyStopping, and in the 8th epoch, it already converged, and I don't have to wait and can stop the learning process there.

```
Epoch 6/50
200/200 [==============================] - 4s 18ms/step - loss: 0.5321 - accuracy: 0.8395 - val_loss: 0.5735 - val_accuracy: 0.8100
Epoch 7/50
200/200 [==============================] - 4s 18ms/step - loss: 0.4741 - accuracy: 0.8581 - val_loss: 0.4757 - val_accuracy: 0.8288
Epoch 8/50
200/200 [==============================] - 3s 17ms/step - loss: 0.4212 - accuracy: 0.8792 - val_loss: 0.4351 - val_accuracy: 0.8587

Epoch 00008: ReduceLROnPlateau reducing learning rate to 0.0007000000332482159.
Epoch 9/50
200/200 [==============================] - 3s 17ms/step - loss: 0.3138 - accuracy: 0.9218 - val_loss: 0.3881 - val_accuracy: 0.8737
Epoch 10/50
200/200 [==============================] - 3s 17ms/step - loss: 0.2670 - accuracy: 0.9347 - val_loss: 0.3651 - val_accuracy: 0.8800
Epoch 11/50
200/200 [==============================] - 3s 17ms/step - loss: 0.2165 - accuracy: 0.9522 - val_loss: 0.4056 - val_accuracy: 0.8600
Epoch 12/50
200/200 [==============================] - 3s 17ms/step - loss: 0.1666 - accuracy: 0.9689 - val_loss: 0.4604 - val_accuracy: 0.8687
Epoch 13/50
200/200 [==============================] - 3s 17ms/step - loss: 0.1464 - accuracy: 0.9747 - val_loss: 0.4122 - val_accuracy: 0.8825
Epoch 14/50
```

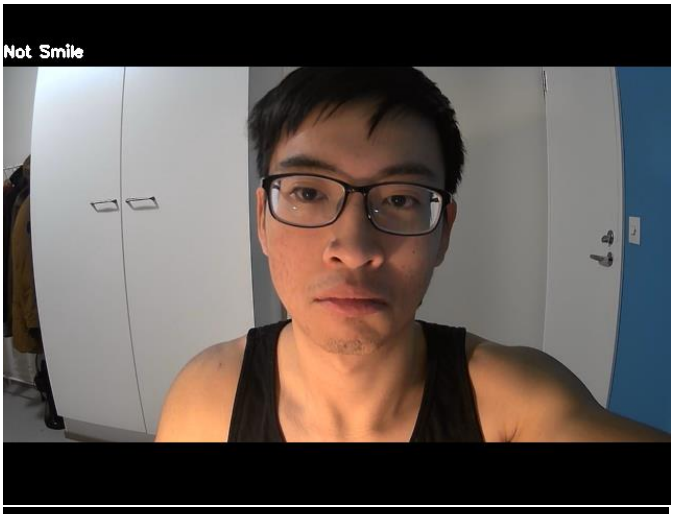
Test the machine learning model with a webcam video

I have the weight file from the training process above to plug it into the webcam test code. I do not use a pre-trained machine learning network because it would be too overkill for this task. I already achieved an 89% validation accuracy with a simple tweak, and I want to keep it short so that the inference test code should be fast enough.
Moreover, this model ensures that the test video would be real-time and high enough.

The code snippet includes the OpenCV library with the `VideoCapture` method to get the video capture, and get the image video by showing the label "Smile" or "Not Smile" in the top left of the video.

```python
cap = cv2.VideoCapture(0)
if not cap.isOpened():
    raise RuntimeError("The camera/video cannot be loaded. Check again!")
else:
    while cap.isOpened():
        ret, frame = cap.read()
        if not ret:
            break
        else:
            # Frame resize
            resized = cv2.resize(frame, (64, 64))
            # Frame normalize
            resized = resized.astype('float32')/255.0
            resized.resize((1, 64, 64, 3))
            # Model predict
            pred = model.predict(resized)
```

The video speed is fast because the network is lightweight. Moreover, the accuracy is high, so that it would ensure to predict the right facial status.



Future enhancements

Because I am using a webcam to capture the video playing and sit too far from the camera view, it would not detect correctly. The problem is that all the training data are close-up faces with a smile or not smile.

Therefore, if I want to increase the prediction faster, I would implement a region of interest that crop an area in the webcam and zoom it in.
Moreover, I would implement a simple code to test the FPS of the video and make sure the FPS is stable during the prediction process.