1. Model architecture

Layer (type)	Output Shape	Param #
input_1 (InputLayer)	(None, 96, 96, 3)	0
block1_conv1 (Conv2D)	(None, 96, 96, 64)	1792
block1_conv2 (Conv2D)	(None, 96, 96, 64)	36928
block1_pool (MaxPooling2D)	(None, 48, 48, 64)	0
block2_conv1 (Conv2D)	(None, 48, 48, 128)	73856
block2_conv2 (Conv2D)	(None, 48, 48, 128)	147584
block2_pool (MaxPooling2D)	(None, 24, 24, 128)	0
block3_conv1 (Conv2D)	(None, 24, 24, 256)	295168
block3_conv2 (Conv2D)	(None, 24, 24, 256)	590080
block3_conv3 (Conv2D)	(None, 24, 24, 256)	590080
block3_pool (MaxPooling2D)	(None, 12, 12, 256)	0
batch_normalization_1 (Batch	(None, 12, 12, 256)	1024
conv2d_1 (Conv2D)	(None, 10, 10, 512)	1180160
batch_normalization_2 (Batch	(None, 10, 10, 512)	2048
activation_1 (Activation)	(None, 10, 10, 512)	0
conv2d_2 (Conv2D)	(None, 10, 10, 256)	131328
dropout_1 (Dropout)	(None, 10, 10, 256)	0
conv2d_3 (Conv2D)	(None, 10, 10, 128)	32896
dropout_2 (Dropout)	(None, 10, 10, 128)	0
global_average_pooling2d_1 (	(None, 128)	0
dense_1 (Dense)	(None, 7)	903

My model is a combination of truncated Vgg16 and NiN. I take 3 blocks of Vgg16, connected with a NiN block. The NiN is Network-In-Network, it has two significant improvements in neural network, one is MLP-conv(multilayer perceptron), and the other one is GAP(global average pooling). In my project, the MLP-conv consists of a 3x3 conv layer and two 1x1 conv layers. To avoid gradient vanish and overfitting, batch normalization is added between 3x3 conv layer and "Relu" activation, and Dropout follows 1x1 conv layer. The GAP layer takes the average of each feature map, the resulting vector is fed directly into the softmax layer.

2. My tries
    a. learning rate
       I set the learning rate as 0.1 when training a new model, with decay equals to 1e-6. When fine-tuning the Vgg model, the learning rate was set as 1e-4, to avoid ruining the pre-trained Vgg model.
    b. ImageDataGenerator
       This function has lots of configures, and I tried several values for each config element. When I set a value, I save the transformed images to check if they look awkward, I adjust the configuration values until all the generated images seem close to true training pictures.
    c. Input image size
       The origin input of Vgg16 is 224x224 size, but our training datas are 48x48. We should not upscale the origin image to 224x224 size because of the poor quality, and the 48x48 size is neither desirable, because the feature map will be small after the 3 blocks of Vgg16. Finally, I chose 96x96 size, it's a balance of mid-layer size and image quality.
    d. Batch Normalization and Dropout
       Before I add the batch normalization and Dropout in my model, I find it's easy to get overfitting, no matter how much data augmented by ImageDataGenerator. With batch normalization between conv layer and "Relu" activation, the overfitting reduced. After adding Dropout, I can train more epochs before overfitting.

3. Training approaches
    a. Transfer learning

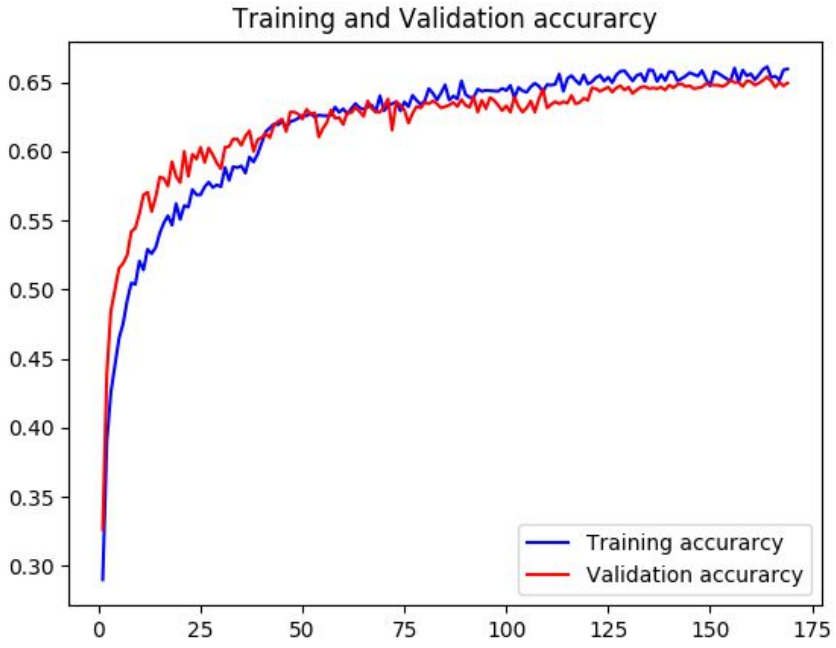
       A pre-trained Vgg model is good at extracting image features, in this project, I replaced 2 blocks and fully connected layers of Vgg16 with a NiN block. Firstly, freeze all the layers of Vgg and train the NiN block. After the NiN is well trained, unfrozen the Vgg, fine tune them with the NiN block at a low learning rate.
    b. Data augement

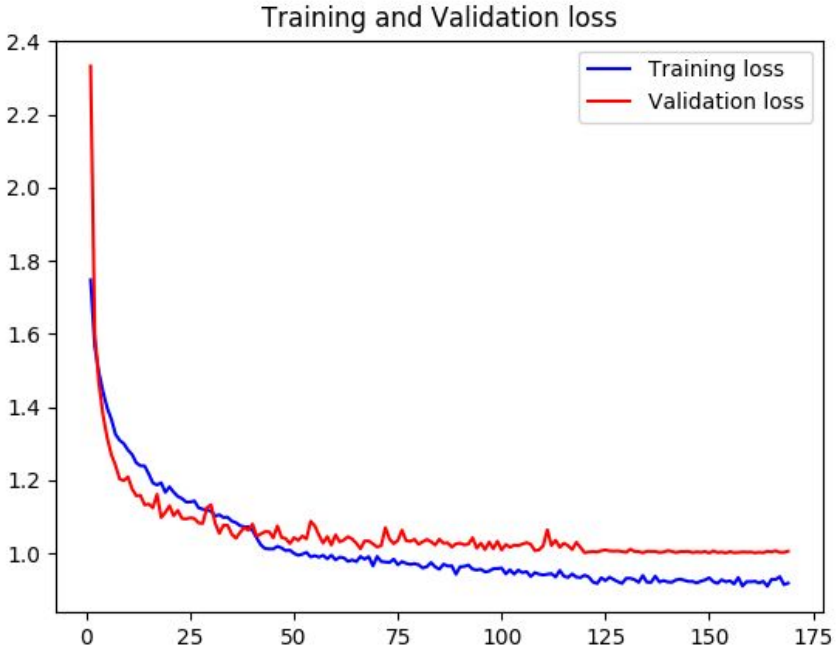
       In the training data set, faces in different sizes and positions, different tilting angles, and some facing left but some others are in the right direction. These unimportant features allow us to modify them to enrich our training data set. The Keras ImageDataGenerator is a function that can transform images randomly, within the user defined restrictions. In this project, I defined the restrictions as ImageDataGenerator(rotation_range=40, zoom_range=0.4, width_shift_range=0.2, height_shift_range=0.2, shear_range = 0.1, horizontal_flip=True, fill_mode='nearest', dtype='uint8'). Those generated images can reduce overfitting, and this data augement method can save memories.
    c. Image Contrast Enhancement

       In our training data, some images are too dark or too bright, it’s hard to see faces. To avoid the effects of light, histogram equalization was used to balance pixel distribution and adjust the picture contrast.

4. Accuracy and Loss

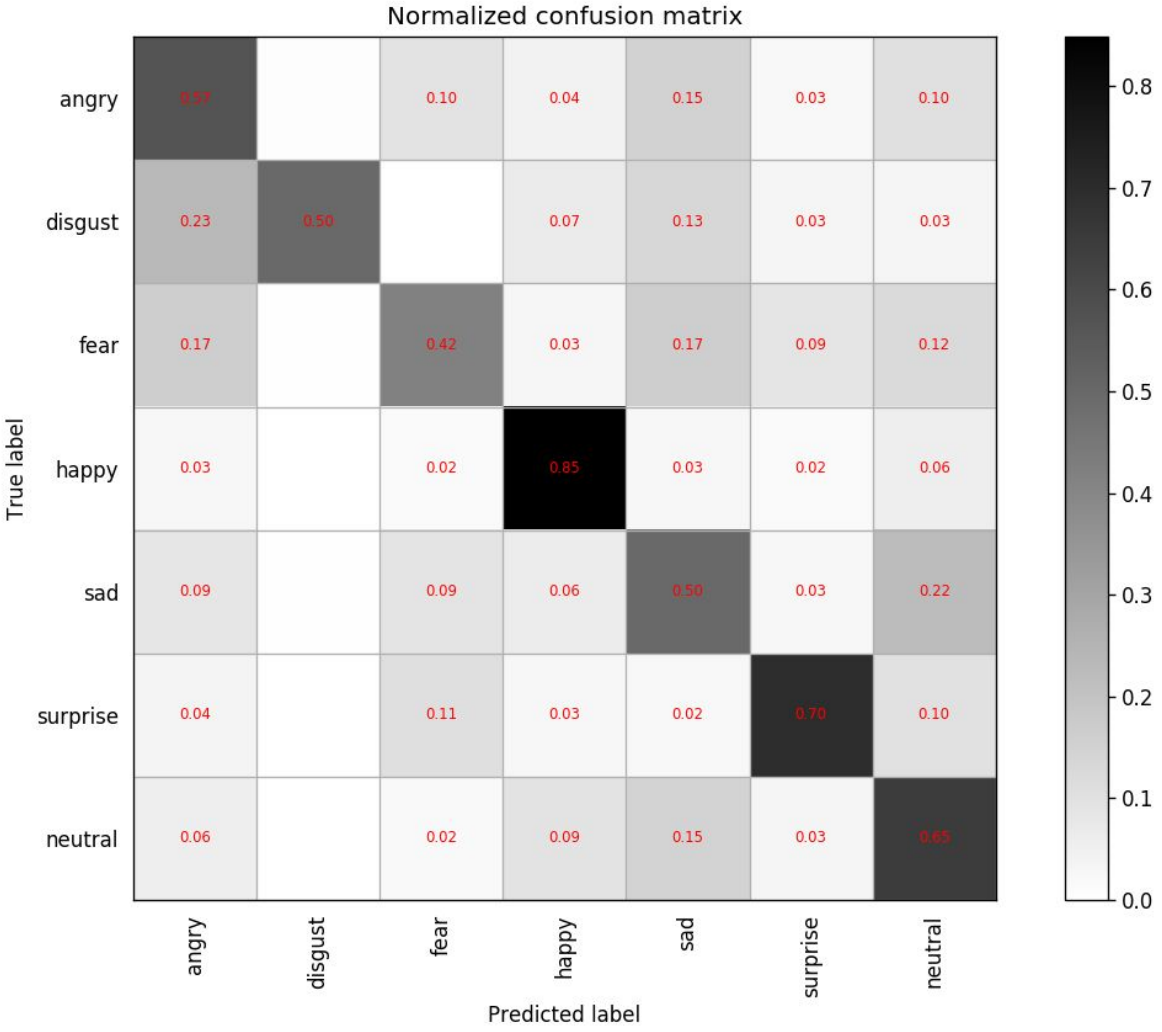
   There is a significant increase near the 50th epoch, it is caused by a break-and-continue training. After this break, starts fine-tuning the Vgg16, which introduced progress.





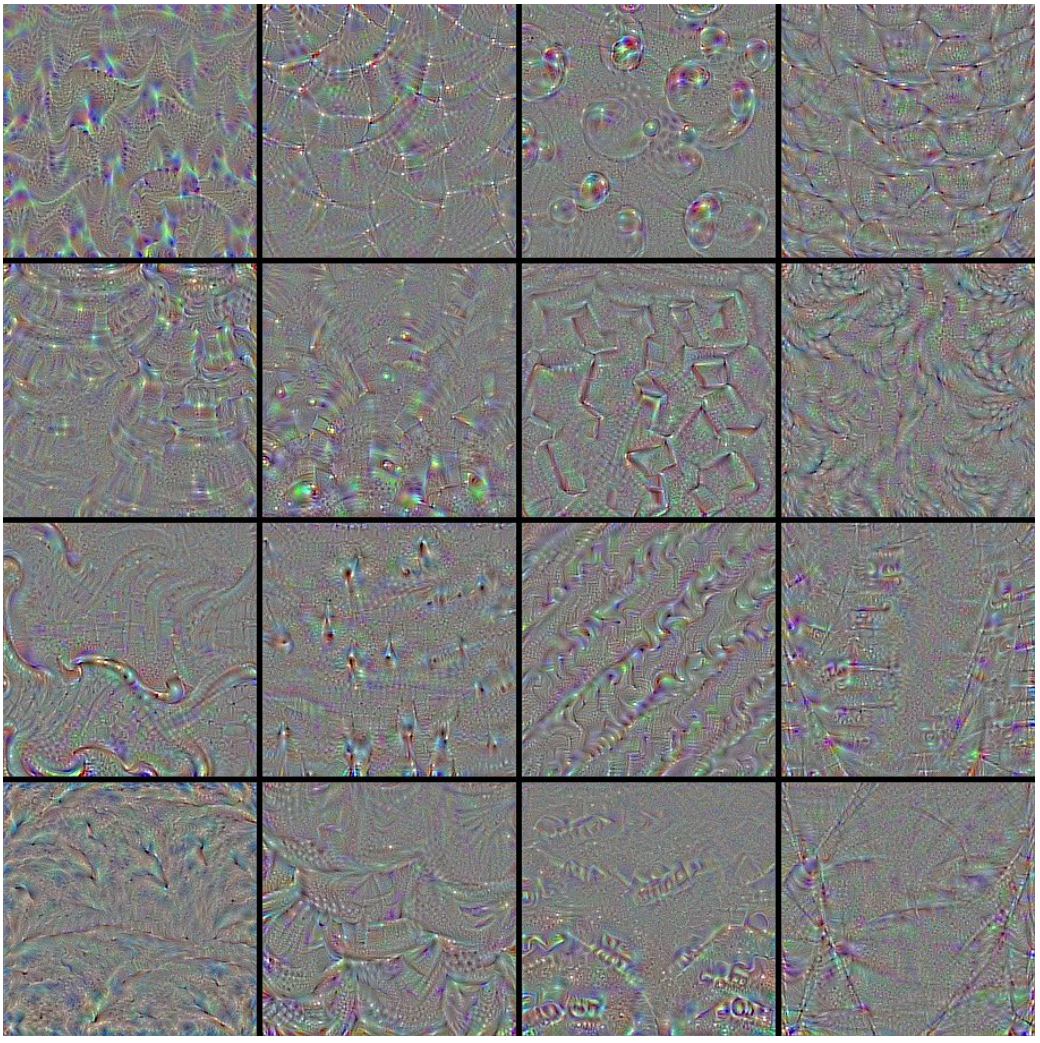
5. Confusion Matrix

   In the training data set, the “disgust” data are rare, therefore, for other kinds of sentiment, it’s not easy to be misjudged as disgust.



6. Filter visualization

   Filter visualization was used in this project to explain the model. Randomly choose some filters in the target convolution layer, then get their gradient loss and do gradient ascent to add gradient loss on random noise, we can find what features those filters are extracting.



If we add gradient loss of each filter on the origin image, we can find out what those filters do on our images. The following image demonstrated that the last conv layer can extract eyes and facial contour features.

7. Typical sentiment samples

   The following 4 images standard 4 emotions, they are:

   Happy, Fear, Sad and Surprise, we find the important areas on the face are highlighted by neural networks.