Instructions

This document is a template, and you are not required to follow it exactly. However, the kinds of questions we ask here are the kinds of questions we want you to focus on. While you might have answered similar questions to these in your project presentations, we want you to go into a lot more detail in this write-up; you can refer to the Lab homeworks for ideas on how to present your data or results.

You don't have to answer every question in this template, but you should answer roughly this many questions. Your answers to such questions should be paragraph-length, not just a bullet point. You likely still have questions of your own -- that's okay! We want you to convey what you've learned, how you've learned it, and demonstrate that the content from the course has influenced how you've thought about this project.

Project Name

Project mentor: Yuli

Borui Feng bfeng9@jh.edu (mailto:bfeng9@jh.edu), Minghui Song msong23@jh.edu (mailto:msong23@jh.edu), Naiyu Zhang nzhang54@jh.edu (mailto:nzhang54@jh.edu)

Git repo link: https://github.com/MollyZh/Machine-Learning-2022-Fall-Final.git (https://github.com/MollyZh/Machine-Learning-2022-Fall-Final.git)

Outline and Deliverables

Uncompleted Deliverables

Coonsidering the time limitation and model complexity, we do not introduce feature engineering and augmentation in our model.

1. "Expect to complete #1": Compare different feature selection and feature engineering layers
2. "Would like to accomplish #3": Test if adding augmentation could improve performance

Completed Deliverables

1. "Must complete #1": We used gradient descend method with Adam optimizer to minimize our crossentropy loss functions in our CNN and VGG16 models.
2. "Must complete #2": We applied convolutional neural networks (CNN) for facial expression recognition. We used 2 convolutional and 2 max pooling layers to summarize the picture informations.
3. "Must complete #3": We added a dropout rate of 0.2 to prevent overfitting.
4. "Would like to accomplish #1": We used VGG16model which was proposed in mgeNet competition as our pre-trained model. By adding flatten dense ,relu dense, and softmax dense, we realized facial expression recognition in another way.
5. "Would like to accomplish #2": Increase training depth, apply multiple convolutions together and compare their performances. We have built our own model with two convolutional layers (which has already achieved the goal of "multiple convolutions together"). But we add a VGG16 based model for comparison which has even more convolutional layers.
6. "Expect to complete #2": Compare performance with CNN and gradient descend method.
7. "Expect to complete #3": Improve classification accuracy by adjusting parameters. Although our model are not sensitive to change of hyperparameters, we did try to change some of the hyperparameters to get a better model when coding.

Additional Deliverables

1. Added class_weights during training to try to solve the issue of data imbalance (Although the result shows that it doesn't work very well)
2. Visualized data before building model to get a better sense of the dataset
3. Provide various metrics in the end to evaluate the model and give a comprehensive visualization of the result

Preliminaries

What problem were you trying to solve or understand?

Our goal is to perform facial emotion recognition with image pixels vector with grayscale value. It's a multi-class classfication problem. We plan to assign each facial picture to one of seven emotions: anger, happiness, fear, surprise, disgust, sadness and neutral.

Facial expression recognition technology is one of the most widely used technologies for emotion recognition. The technology is based on the Facial Expression Coding System (FACS) proposed by the famous American psychologist Paul Aikman and emotional psychology master Izzard.

There are a wide range of real-world application scenarios. For example, in banking, it can be used to detect suspicious behavior and warn of potential terrorist threats. In marketing, it can be used to identify customer age, gender and mood, to instruct more suitable marketing methods accordingly. In Advertising and movie industry, it can be used to predict how well a promo or commercial will pay off.

This problem is similar to the Handwritten number recognition problem we discussed in class. We will use filters to conclude the mode of pictures, then use max pooling for filter responses to gain robustness to the exact spatial location of features, finally apply softmax function to realize multi-class classfication.

The expression recognition problem is unique in that the size of reliable expression data is small and the expressions are not deterministic. In addition to neutrality, humans have 6 basic expressions and 15 compound expressions. Expression annotations in the face expression recognition problem may be mixed with subjective emotions of the annotator, and thus there is a lot of noise.

The ethical aspects of the facial expression recognition problem are somewhat controversial.Many photos from people's networks are used for training without their permission. It's also said that steming from an imbalance training database, many face recognition algorithms recognize white males to a greater degree. Another ethical issue that has been emerging recently regarding expression recognition is that some algorithms can modify the expressions of people in photos based on expression recognition, and these modifications to photos may also have ethical and legal implications.

Dataset

We use the dataset from kaggle "Challenges in Representation Learning: Facial Expression Recognition Challenge" (https://www.kaggle.com/competitions/challenges-in-representation-learning-facial-expression-recognition-challenge/data?select=icml_face_data.csv) as our data source. Those data are collected by Pierre-Luc Carrier and Aaron Courville, as part of an ongoing research project.(Cite from kaggle dataset description)

The dataset includes 35,887 samples, with input 48x48 pixel grayscale images of faces and output one of 7 labels: 0=Angry, 1=Disgust, 2=Fear, 3=Happy, 4=Sad, 5=Surprise, 6=Neutral. All the faces are pre-processed so that they are roughly centered and occupy the same amount of space in each image. Our task is to assign each face to six basic emotions or neutral.

We choose this dataset for two reasons: 1. This dataset is open source and does not have any copyright issues. 2. This dataset has been well pre-processed, we don't need to spend extra effort on data cleaning.

We splited this dataset as 8:1:1, which means train size=28709, validation size=3589, and test size=3589.

In [1]:
```
# Load your data and print 2-3 examples
#import data
import pandas as pd
import_data = pd.read_csv('icml_face_data.csv')
```

In [2]:
```
import_data.head()
```

Out[2]:

	emotion	Usage	pixels
0	0	Training	70 80 82 72 58 58 60 63 54 58 60 48 89 115 121...
1	0	Training	151 150 147 155 148 133 111 140 170 174 182 15...
2	2	Training	231 212 156 164 174 138 161 173 182 200 106 38...
3	4	Training	24 32 36 30 32 23 19 20 30 41 21 22 32 34 21 1...
4	6	Training	4 0 0 0 0 0 0 0 0 0 0 0 0 3 15 23 28 48 50 58 84...

Pre-processing

We don't do much on data preprocessing since our data is quite complete and there is no missing data. So we only need to reshape our data into the correct input dimension which is (sample_size,48,48,1) for CNN-like model and (sample_size,48*48) for baseline model. In addition, we also divide the original pixel values by 255 and then minus 0.5 to make a simple normalization.

Our dataset is not class-balanced. Accoring to the histograms of 7 labels we plotted in the main jupyter notebook, we have more samples of "happy" and very few samples of "disgust". We tried to solve this problem by adding sample weights to our models. The logic for sample weights is that classes with fewer samples should have higher sample weights.

For categorical output labels, we use softmax functions to get the probabilities of every category. Our prediction is the category with the highest probability.

In [3]:
```
import copy
#process raw data into training, validation and testing
training = copy.deepcopy(import_data[import_data[' Usage'] == 'Training'])
training.drop(columns = [' Usage'],inplace = True)
training.reset_index(drop = True,inplace = True)
validation = copy.deepcopy(import_data[import_data[' Usage'] == 'PrivateTest'])
validation.drop(columns = [' Usage'],inplace = True)
validation.reset_index(drop = True,inplace = True)
testing = copy.deepcopy(import_data[import_data[' Usage'] == 'PublicTest'])
testing.drop(columns = [' Usage'],inplace = True)
testing.reset_index(drop = True,inplace = True)
```

In [4]:
```
import numpy as np
# import functions in our project
%run functions_used.ipynb

#reshape data for CNN and VGG and do scalarization
xtrain,ytrain = reshape_data_CNN(training)
xval,yval = reshape_data_CNN(validation)
xtest,ytest = reshape_data_CNN(testing)
```

In [5]:
```
# input data before pre-process
import_data[' pixels'][0]
```

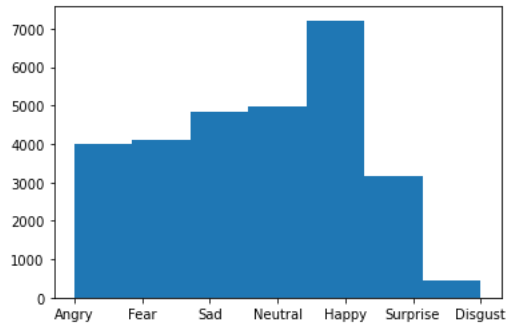
Out[5]:
```
'70 80 82 72 58 58 60 63 54 58 60 48 89 115 121 119 115 110 98 91 84 84 90 99 110 126 143 153 158 171 169 172 169 165
129 110 113 107 95 79 66 62 56 57 61 52 43 41 65 61 58 57 56 69 75 70 65 56 54 105 146 154 151 151 155 155 150 147 14
7 148 152 158 164 172 177 182 186 189 188 190 188 180 167 116 95 103 97 77 72 62 55 58 54 56 52 44 50 43 54 64 63 71
68 64 52 66 119 156 161 164 163 164 167 168 170 174 175 176 178 179 183 187 190 195 197 198 197 198 195 191 190 145 8
6 100 90 65 57 60 54 51 41 49 56 47 38 44 63 55 46 52 54 55 83 138 157 158 165 168 172 171 173 176 179 179 180 182 18
5 187 189 189 192 197 200 199 196 198 200 198 197 177 91 87 96 58 58 59 51 42 37 41 47 45 37 35 36 30 41 47 59 94 141
159 161 161 164 170 171 172 176 178 179 182 183 183 187 189 192 192 194 195 200 200 199 199 200 201 197 193 111 71 10
8 69 55 61 51 42 43 56 54 44 24 29 31 45 61 72 100 136 150 159 163 162 163 170 172 171 174 177 177 180 187 186 187 18
9 192 192 194 195 196 197 199 200 201 200 197 201 137 58 98 92 57 62 53 47 41 40 51 43 24 35 52 63 75 104 129 143 149
158 162 164 166 171 173 172 174 178 178 179 187 188 188 191 193 194 195 198 199 199 197 198 197 197 197 201 164 52 78
87 69 58 56 50 54 39 44 42 26 31 49 65 91 119 134 145 147 152 159 163 167 171 170 169 174 178 178 179 187 187 185 187
190 188 187 191 197 201 199 199 200 197 196 197 182 58 62 77 61 60 55 49 59 52 54 44 22 30 47 68 102 123 136 144 148
150 153 157 167 172 173 170 171 177 179 178 186 190 186 189 196 193 191 194 190 190 192 197 201 203 199 194 189 69 48
74 56 60 57 50 59 59 51 41 20 34 47 79 111 132 139 143 145 147 150 151 160 169 172 171 167 171 177 177 174 180 182 18
1 192 196 189 192 198 195 194 196 198 201 202 195 189 70 39 69 61 61 61 53 59 59 45 40 26 40 61 93 124 135 138 142 14
4 146 151 152 158 165 168 168 165 161 164 173 172 167 172 167 180 198 198 193 199 195 194 198 200 198 197 195 190 65
35 68 59 59 62 57 60 59 50 44 32 54 90 115 132 137 138 140 144 146 146 156 165 168 174 176 176 175 168 168 169 171 17
5 171 172 192 194 184 198 205 201 194 195 193 195 192 186 57 38 72 65 57 62 58 57 60 54 49 47 79 116 130 138 141 141
139 141 143 145 157 164 164 166 173 174 176 179 179 176 181 189 188 173 180 175 160 182 189 198 192 189 190 190 188 1
```

In [6]:
```
# # input data after pre-process
xtrain[0]
```

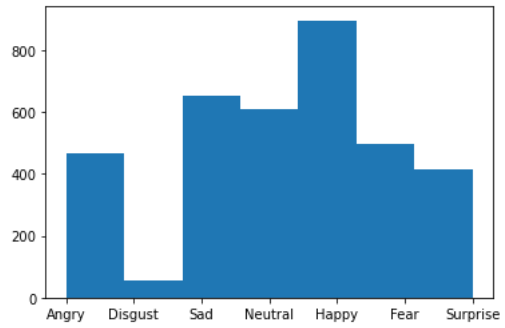
Out[6]:
```
array([[[-0.7254902 ],
        [-0.68627451],
        [-0.67843137],
        ...,
        [-0.79607843],
        [-0.83137255],
        [-0.83921569]],

       [[-0.74509804],
        [-0.76078431],
        [-0.77254902],
        ...,
        [-0.78039216],
        [-0.79607843],
        [-0.82745098]],

       [[-0.80392157],
        [-0.83137255],
        [-0.78823529],
        ...,
        [-0.80784314],
        [-0.78039216],
        [-0.81568627]],

       ...,

       [[-0.64313725],
        [-0.74509804],
        [-0.83529412],
        ...,
        [-0.71764706],
        [-0.78039216],
        [-0.83137255]],

       [[-0.69803922],
        [-0.67843137],
        [-0.69019608],
        ...,
        [-0.58823529],
        [-0.7254902 ],
        [-0.81960784]],

       [[-0.69803922],
        [-0.71764706],
        [-0.67058824],
        ...,
        [-0.58431373],
        [-0.57254902],
        [-0.67843137]]])
```

In [7]:
```
# Visualize the distribution of data before and after pre-processing.
import matplotlib.pyplot as plt
label_list = ['Angry', 'Disgust', 'Fear', 'Happy', 'Sad', 'Surprise', 'Neutral']
describe_data(training,validation,testing,xtrain,label_list)
```

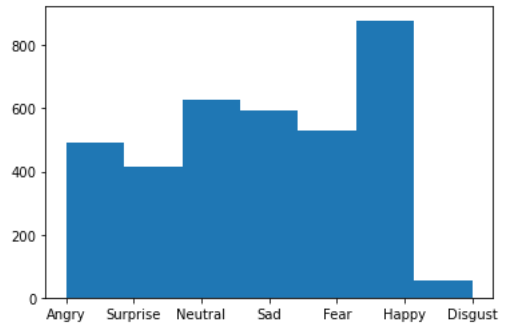
```
train size: 28709
validation size: 3589
test size: 3589
training set class distribution:
```



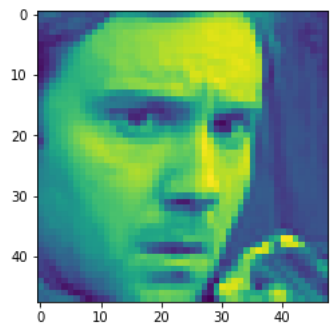

```
testing set class distribution:
```



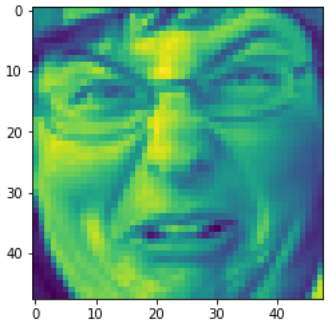

```
validation set class distribution:
```



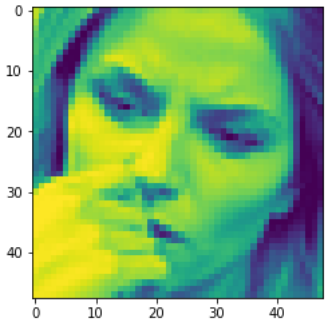

```
Angry
```



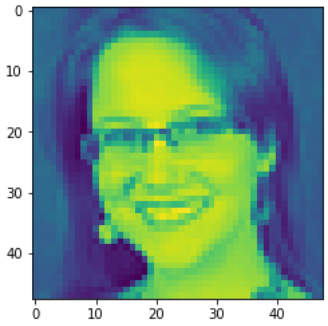

```
Disgust
```



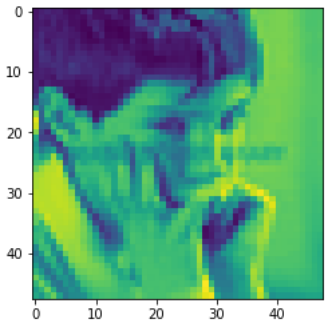
```
Fear
```



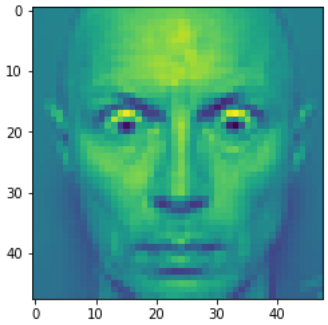
```
Happy
```



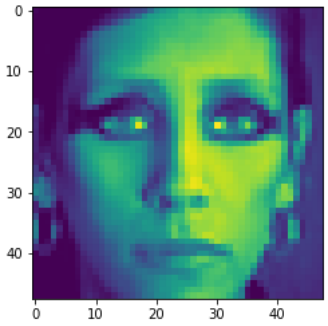
```
Sad
```



```
Surprise
```



```
Neutral
```



Models and Evaluation

Experimental Setup

How did you evaluate your methods? Why is that a reasonable evaluation metric for the task?

We evalute our final model result mainly based on test accuracy. Since there are 7 classes in our final prediction and there is no particular class which is more important than others and we consider positive and negative cases equally (for each class), we then decide accuracy would be a good metrics for us to evalute the predictability of our model. However, in addition to that, in order to benefit from what accuracy can tell us but not limited to it, while using test accuracy as our main metric, we also include many other metrics to help us get a more general sense of our model and how our model performs in each perspective. The full list of the metrics we output will be listed below. Also, we add early stopping to our model to prevent over-fitting where we consider validation accuracy to be the metric we want to monitor during the train process. A full list of the metrics we output: test accuracy, test loss, precision (of each class), recall (of each class), f1-score (of each class), train and validation accuracy against numbers of epochs trained, train and validation loss agains numbers of epochs trained, confusion matrix (whose information is sort of contained in metrics mentioned above, but we still output the confusion matrix to have a more clear sense of the predictions our model made and gain a better visulization)

What did you use for your loss function to train your models? Did you try multiple loss functions? Why or why not?

We use cross-entropy (or categorical cross-entropy as called in keras) to be our loss function to train our model. We didn't use other loss functions since cross-entropy, being the most popular multi-class loss function, is logically and empirically well suitted to our problem and we have conducted researches online and found that most of related works use cross-entropy as their loss function. As a result, we consider cross-entropy is a valid and appropriate loss function for our model.

How did you split your data into train and test sets? Why?

The train test split process for our project is trivial. Since our data is from 'Challenges in Representation Learning: Facial Expression Recognition Challenge Learn facial expressions from an image' on kaggle which has already specified and pre-splited the train, validation and test sets we should use for our model for us.

In [8]:
```
# Code for loss functions, evaluation metrics or link to Git repo
#loss functions are shown in codes in the methods section below
#display metrics for CNN and VGG model
def describe_CNN(model,history,xtest,ytest,label_list,is_vgg16):
    metrics_grapher_CNN(history)
    model_summary_CNN(model)
    if is_vgg16:
        xtest = np.repeat(xtest, repeats=3, axis=3)
    ypred_CNN = model.predict(xtest)
    CNNpred,true = convert_labels_CNN(ypred_CNN,ytest.values,label_list)
    print(classification_report(true, CNNpred))
    draw_confusion(true,CNNpred,label_list)
    return None
#display metrics for other models
def describe_general(model,xtest,ytest,label_list):
    ypred = model.predict(xtest)
    pred,true = convert_labels_general(ypred,ytest.to_numpy(),label_list)
    print(classification_report(true, pred))
    draw_confusion(true,pred,label_list)
    return accuracy_score(pred,true)
```

Baselines

What baselines did you compare against? Why are these reasonable?

We have consided three baseline models which are logistic regression, random forest (with grid search to tune hyper-parameters) and naive bayes. We consider these models because these models are simple and popular models which have been widely used. Comparing with these simple models, we can have a more clear sense of what advantages our model have.

Did you look at related work to contextualize how others methods or baselines have performed on this dataset/task? If so, how did those methods do?

We have seen related works on kaggle and found that most of the related works using CNN-like models have an accuracy around 54%, which is similar as the result of our model.

Methods

What methods did you choose? Why did you choose them?

We have chosen two models. One is self-bulid standard CNN model and one is a model based on a pre-trained model called VGG16 (transfer learning). The self-build model is our main model. We have constructed the model in a way that it's not too complex but not too simple. The detailed construction of the model is shown in the code below. And we decide to include this VGG16 model to see how transfer learning can do for us and make a comparision between our self-build model and the model we build based on the pre-trained VGG16 model.

How did you train these methods, and how did you evaluate them? Why?

model.fit() method in keras is used for the training process. The hyperparameters are shown in the code below and the ealuation process is given in the experimental setup section above.

Which methods were easy/difficult to implement and train? Why?

The two models are quite similar in terms of the difficulty of implementation. The vgg16 based model is relatively easier to train since we only need to train the three dense layers in the end instead of the whole model, which is a benefit of transfer learning.

For each method, what hyperparameters did you evaluate? How sensitive was your model's performance to different hyperparameter settings?

Since neural network models are time consuming when using Gridsearch-like method to tune hyperparameters, so we didn't tune every hyperparameters in the model. Particularly, we have tried different batch size, different number of epochs and different hidden units. It turns out our model is not quite sensitive to the change of hyperparameters. In other words, we got quite similar results with different hyperparameter settings.

In [9]:
```
# Code for training models, or link to your Git repository
#definte VGG model
def VGG16_model(xtrain,ytrain,xval,yval,class_weights,filter_num,filter_size,pooling_size,dropout_rate):
    xtrain_colored = np.repeat(xtrain, repeats=3, axis=3)
    xval_colored = np.repeat(xval, repeats=3, axis=3)
    base_model = VGG16(weights="imagenet", include_top=False, input_shape= (48, 48, 3))
    base_model.trainable = False
    model = Sequential()
    model.add(base_model)
    model.add(Flatten())
    model.add(Dense(128, activation='relu'))
    model.add(Dense(128, activation='relu'))
    model.add(Dense(7, activation='softmax'))
    model.compile( 'adam',loss='categorical_crossentropy',metrics=['accuracy'])
    callback = tf.keras.callbacks.EarlyStopping(monitor='val_loss', patience=3,verbose = 1)
    history = model.fit(xtrain_colored, ytrain,validation_data=(xval_colored, yval),class_weight = class_weights,
                        epochs=100, batch_size=100, callbacks=[callback])
    return model,history
#define CNN model
#conv - pooling - conv- pooling- flatten - dense - dense
def CNN_model(xtrain,ytrain,xval,yval,class_weights,filter_num,filter_size,pooling_size,dropout_rate):
    model = Sequential()
    model.add(Conv2D(filters = filter_num, kernel_size =  filter_size, activation='relu',padding="same" ,
                     input_shape=(48, 48, 1)))
    model.add(MaxPooling2D(pool_size=pooling_size))
    model.add(Dropout(dropout_rate))
    model.add(Conv2D(filters = filter_num, kernel_size = filter_size,padding="same" , activation='relu'))
    model.add(MaxPooling2D(pool_size=pooling_size))
    model.add(Flatten())
    model.add(Dense(128, activation='relu'))
    model.add(Dense(7, activation='softmax'))
    model.compile( 'adam',loss='categorical_crossentropy',metrics=['accuracy'])
    callback = tf.keras.callbacks.EarlyStopping(monitor='val_loss', patience=3,verbose = 1)
    history = model.fit(xtrain, ytrain,validation_data=(xval, yval),class_weight = class_weights,epochs=100,
                        batch_size=100, callbacks=[callback])
    return model,history
#define hyperparameters for CNN and VGG
class_weights  = get_class_weights(training)
filter_num = 64
filter_size = 3
pooling_size = 2
dropout_rate = 0.2
#train CNN model
model,history = CNN_model(xtrain= xtrain,ytrain= ytrain,xval = xval,yval = yval,class_weights = class_weights,
                      filter_num =filter_num,filter_size = filter_size
                      ,pooling_size = pooling_size ,dropout_rate = dropout_rate)
#train VGG model
VGGmodel,VGGhistory = VGG16_model(xtrain= xtrain,ytrain= ytrain,xval = xval,yval = yval,class_weights = class_weights,
                      filter_num =filter_num,filter_size = filter_size
                      ,pooling_size = pooling_size ,dropout_rate = dropout_rate)
```

```
Epoch 1/100
288/288 [==============================] - 44s 150ms/step - loss: 0.2774 - accuracy: 0.2833 - val_loss: 1.6058 - val_
accuracy: 0.4018
Epoch 2/100
288/288 [==============================] - 43s 148ms/step - loss: 0.2382 - accuracy: 0.4079 - val_loss: 1.5613 - val_
accuracy: 0.4224
Epoch 3/100
288/288 [==============================] - 41s 143ms/step - loss: 0.2159 - accuracy: 0.4675 - val_loss: 1.3776 - val_
accuracy: 0.4817
Epoch 4/100
288/288 [==============================] - 41s 143ms/step - loss: 0.2018 - accuracy: 0.5019 - val_loss: 1.4106 - val_
accuracy: 0.4851
Epoch 5/100
288/288 [==============================] - 42s 146ms/step - loss: 0.1907 - accuracy: 0.5357 - val_loss: 1.2966 - val_
accuracy: 0.5171
Epoch 6/100
288/288 [==============================] - 42s 146ms/step - loss: 0.1799 - accuracy: 0.5627 - val_loss: 1.2960 - val_
accuracy: 0.5210
Epoch 7/100
288/288 [==============================] - 41s 141ms/step - loss: 0.1700 - accuracy: 0.5862 - val_loss: 1.2769 - val_
accuracy: 0.5344
Epoch 8/100
288/288 [==============================] - 42s 146ms/step - loss: 0.1596 - accuracy: 0.6106 - val_loss: 1.2790 - val_
accuracy: 0.5316
Epoch 9/100
288/288 [==============================] - 42s 145ms/step - loss: 0.1487 - accuracy: 0.6384 - val_loss: 1.3192 - val_
accuracy: 0.5322
Epoch 10/100
288/288 [==============================] - 42s 146ms/step - loss: 0.1377 - accuracy: 0.6642 - val_loss: 1.3090 - val_
accuracy: 0.5350
Epoch 10: early stopping
Epoch 1/100
288/288 [==============================] - 77s 265ms/step - loss: 0.2645 - accuracy: 0.3283 - val_loss: 1.5994 - val_
accuracy: 0.3809
Epoch 2/100
288/288 [==============================] - 76s 265ms/step - loss: 0.2466 - accuracy: 0.3850 - val_loss: 1.5707 - val_
accuracy: 0.4009
Epoch 3/100
288/288 [==============================] - 77s 266ms/step - loss: 0.2398 - accuracy: 0.4077 - val_loss: 1.5873 - val_
accuracy: 0.3970
Epoch 4/100
288/288 [==============================] - 78s 273ms/step - loss: 0.2340 - accuracy: 0.4267 - val_loss: 1.5382 - val_
accuracy: 0.4093
Epoch 5/100
288/288 [==============================] - 78s 271ms/step - loss: 0.2284 - accuracy: 0.4414 - val_loss: 1.5637 - val_
accuracy: 0.4132
Epoch 6/100
288/288 [==============================] - 78s 272ms/step - loss: 0.2238 - accuracy: 0.4554 - val_loss: 1.5417 - val_
accuracy: 0.4232
Epoch 7/100
288/288 [==============================] - 77s 268ms/step - loss: 0.2179 - accuracy: 0.4723 - val_loss: 1.5504 - val_
accuracy: 0.4213
Epoch 7: early stopping
```

Results

Show tables comparing your methods to the baselines.

We didn't write code for generating tables comparing methods to baseline but below we show the results of our two CNN-like models and the baseline models. The comparison between accuracy in each category is shown in the presentation. Here we list the overall accuracy of each model. It can be seen that CNN model with early stop performs the best.

Model	Accuracy
CNN Model	0.53
VGG Model	0.42
Logistic Regression Model	0.38
Random Forest Model	0.41
Naive Bayes Model	0.23

What about these results surprised you? Why?

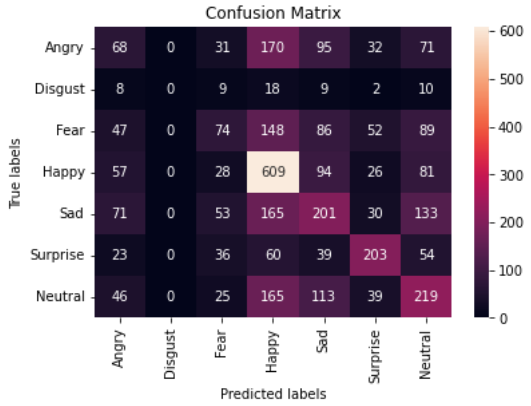
Since in the data visualization section, we have found that there is an imbalance of our dataset. So although we have added a class_weights during training to try to solve this issue, it's not surprising that our final result shows that our models don't predict the 'disgust' class (the one with less samples) very well.

Did your models over- or under-fit? How can you tell? What did you do to address these issues?

Our models are not over- or under- fit. We prevent this from happening by adding a dropout layer and an early stopping function.

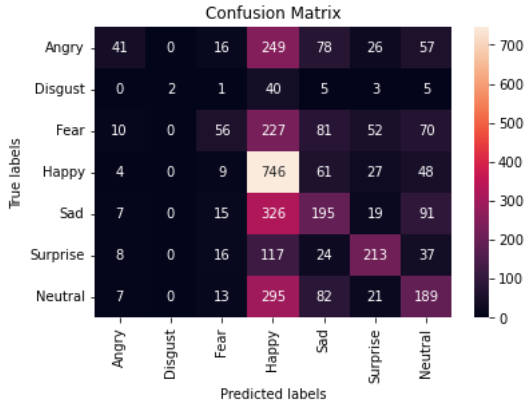
In [14]:
```
#show results of baseline models
#reshape data for other models
acc_list = []
model_list = []
xtrain_general,ytrain_general = reshape_data_general(training)
xval_general,yval_general = reshape_data_general(validation)
xtest_general,ytest_general = reshape_data_general(testing)
#train logistic regression
print('Logistic Regression')
clf = LogisticRegression().fit(xtrain_general,ytrain_general)
acc_list.append(describe_general(clf,xtest_general,ytest_general,label_list))
model_list.append('Logistic Regression')
```

```
Logistic Regression
              precision    recall  f1-score   support

       Angry       0.21      0.15      0.17       467
     Disgust       0.00      0.00      0.00        56
        Fear       0.29      0.15      0.20       496
       Happy       0.46      0.68      0.55       895
     Neutral       0.33      0.36      0.35       607
         Sad       0.32      0.31      0.31       653
    Surprise       0.53      0.49      0.51       415

    accuracy                           0.38      3589
   macro avg       0.31      0.30      0.30      3589
weighted avg       0.36      0.38      0.36      3589
```



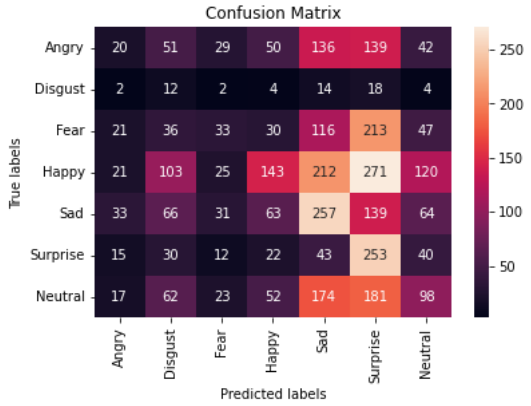
In [15]:
```
#show results of baseline models
#random forest
print('Random Forest')
clf = RandomForestClassifier()
params = {'max_depth':[5,10],
          'n_estimators': [10,100]}
grid = GridSearchCV(clf, param_grid=params, cv=4,verbose = 3).fit(xtrain_general, ytrain_general)
clf = grid.best_estimator_
print(clf)
acc_list.append(describe_general(clf,xtest_general,ytest_general,label_list))
model_list.append('Random Forest')
```

```
Random Forest
Fitting 4 folds for each of 4 candidates, totalling 16 fits
[CV 1/4] END ......max_depth=5, n_estimators=10;, score=0.317 total time=   1.9s
[CV 2/4] END ......max_depth=5, n_estimators=10;, score=0.309 total time=   1.9s
[CV 3/4] END ......max_depth=5, n_estimators=10;, score=0.319 total time=   1.9s
[CV 4/4] END ......max_depth=5, n_estimators=10;, score=0.319 total time=   1.9s
[CV 1/4] END .....max_depth=5, n_estimators=100;, score=0.325 total time=  17.6s
[CV 2/4] END .....max_depth=5, n_estimators=100;, score=0.317 total time=  17.0s
[CV 3/4] END .....max_depth=5, n_estimators=100;, score=0.327 total time=  16.9s
[CV 4/4] END .....max_depth=5, n_estimators=100;, score=0.323 total time=  16.9s
[CV 1/4] END .....max_depth=10, n_estimators=10;, score=0.362 total time=   3.7s
[CV 2/4] END .....max_depth=10, n_estimators=10;, score=0.364 total time=   3.7s
[CV 3/4] END .....max_depth=10, n_estimators=10;, score=0.365 total time=   3.6s
[CV 4/4] END .....max_depth=10, n_estimators=10;, score=0.365 total time=   3.7s
[CV 1/4] END ....max_depth=10, n_estimators=100;, score=0.397 total time=  35.4s
[CV 2/4] END ....max_depth=10, n_estimators=100;, score=0.403 total time=  35.4s
[CV 3/4] END ....max_depth=10, n_estimators=100;, score=0.408 total time=  35.3s
[CV 4/4] END ....max_depth=10, n_estimators=100;, score=0.402 total time=  35.4s
RandomForestClassifier(max_depth=10)
              precision    recall  f1-score   support

       Angry       0.53      0.09      0.15       467
     Disgust       1.00      0.04      0.07        56
        Fear       0.44      0.11      0.18       496
       Happy       0.37      0.83      0.52       895
     Neutral       0.38      0.31      0.34       607
         Sad       0.37      0.30      0.33       653
    Surprise       0.59      0.51      0.55       415

    accuracy                           0.40      3589
   macro avg       0.53      0.31      0.31      3589
weighted avg       0.44      0.40      0.36      3589
```

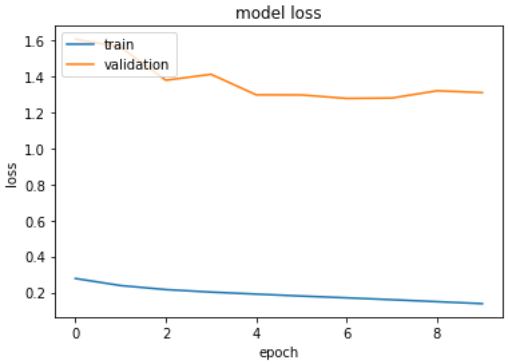


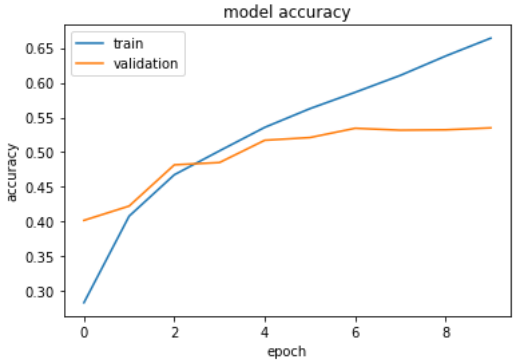
In [16]:
```
#show results of baseline models
#NaiveBayes
print('Naive Bayes')
clf = GaussianNB().fit(xtrain_general,ytrain_general)
acc_list.append(describe_general(clf,xtest_general,ytest_general,label_list))
model_list.append('Naive Bayes')
#print accuracy for models other than CNN and VGG
for i in range(len(model_list)):
    print(model_list[i]+' accuracy: '+str(acc_list[i]))
```

```
Naive Bayes
              precision    recall  f1-score   support

       Angry       0.16      0.04      0.07       467
     Disgust       0.03      0.21      0.06        56
        Fear       0.21      0.07      0.10       496
       Happy       0.39      0.16      0.23       895
     Neutral       0.24      0.16      0.19       607
         Sad       0.27      0.39      0.32       653
    Surprise       0.21      0.61      0.31       415

    accuracy                           0.23      3589
   macro avg       0.22      0.24      0.18      3589
weighted avg       0.26      0.23      0.21      3589

Logistic Regression accuracy: 0.3828364446921148
Random Forest accuracy: 0.4017832265254946
Naive Bayes accuracy: 0.22736138200055725
```

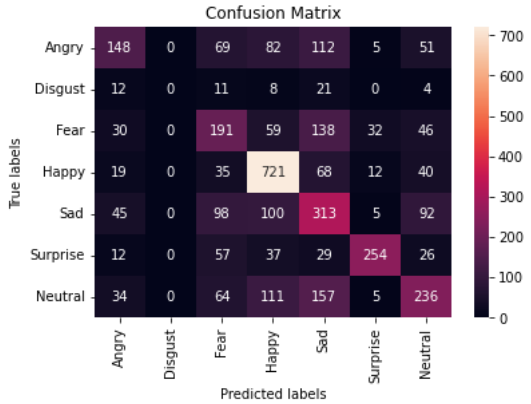


In [11]:
```
# Show plots or visualizations of your evaluation metric(s) on the train and test sets.
#   What do these plots show about over- or under-fitting?
#   You may borrow from how we visualized results in the Lab homeworks.
#   Are there aspects of your results that are difficult to visualize? Why?
#plot for training accuracy and loss and evaluation metrics of test set are all shown here
#describe CNN result
describe_CNN(model,history,xtest,ytest,label_list,False)
```

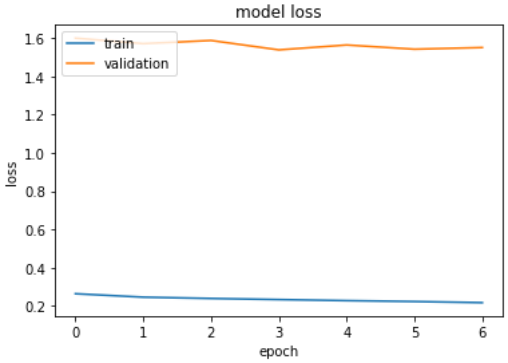


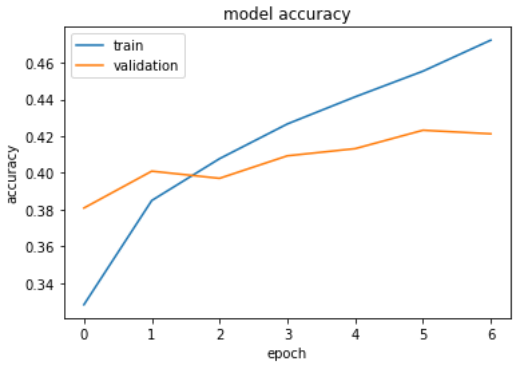



```
Model: "sequential"
_________________________________________________________________
 Layer (type)                Output Shape              Param #
=================================================================
 conv2d (Conv2D)             (None, 48, 48, 64)        640

 max_pooling2d (MaxPooling2D  (None, 24, 24, 64)       0
 )

 dropout (Dropout)           (None, 24, 24, 64)        0

 conv2d_1 (Conv2D)           (None, 24, 24, 64)        36928

 max_pooling2d_1 (MaxPooling  (None, 12, 12, 64)       0
 2D)

 flatten (Flatten)           (None, 9216)              0

 dense (Dense)               (None, 128)               1179776

 dense_1 (Dense)             (None, 7)                 903

=================================================================
Total params: 1,218,247
Trainable params: 1,218,247
Non-trainable params: 0
_________________________________________________________________
113/113 [==============================] - 1s 10ms/step
              precision    recall  f1-score   support

       Angry       0.49      0.32      0.39       467
     Disgust       0.00      0.00      0.00        56
        Fear       0.36      0.39      0.37       496
       Happy       0.64      0.81      0.72       895
     Neutral       0.48      0.39      0.43       607
         Sad       0.37      0.48      0.42       653
    Surprise       0.81      0.61      0.70       415

    accuracy                           0.52      3589
   macro avg       0.45      0.43      0.43      3589
weighted avg       0.52      0.52      0.51      3589
```

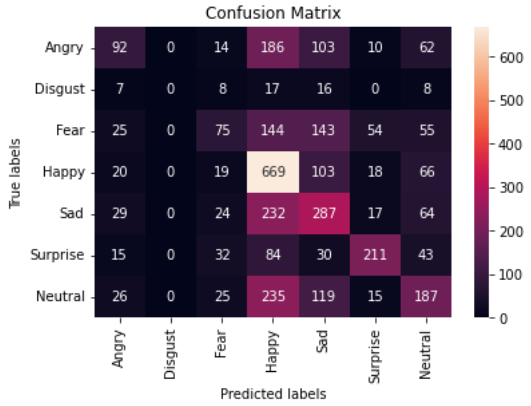

Confusion Matrix
Angry 148 0 69 82 112 5 51
Disgust 12 0 11 8 21 0 4
Fear 30 0 191 59 138 32 46
Happy 19 0 35 721 68 12 40
Sad 45 0 98 100 313 5 92
Surprise 12 0 57 37 29 254 26
Neutral 34 0 64 111 157 5 236
True labels
Angry
Disgust
Fear
Happy
Sad
Surprise
Neutral
Predicted labels
700
600
500
400
300
200
100
0

In [12]:
```
#descibe VGG result
describe_CNN(VGGmodel,VGGhistory,xtest,ytest,label_list,True)
```






```
Model: "sequential_1"
_________________________________________________________________
 Layer (type)                Output Shape              Param #
=================================================================
 vgg16 (Functional)          (None, 1, 1, 512)         14714688

 flatten_1 (Flatten)         (None, 512)               0

 dense_2 (Dense)             (None, 128)               65664

 dense_3 (Dense)             (None, 128)               16512

 dense_4 (Dense)             (None, 7)                 903

=================================================================
Total params: 14,797,767
Trainable params: 83,079
Non-trainable params: 14,714,688
_________________________________________________________________
113/113 [==============================] - 10s 91ms/step
              precision    recall  f1-score   support

       Angry       0.43      0.20      0.27       467
     Disgust       0.00      0.00      0.00        56
        Fear       0.38      0.15      0.22       496
       Happy       0.43      0.75      0.54       895
     Neutral       0.39      0.31      0.34       607
         Sad       0.36      0.44      0.39       653
    Surprise       0.65      0.51      0.57       415

    accuracy                           0.42      3589
   macro avg       0.38      0.34      0.33      3589
weighted avg       0.42      0.42      0.40      3589
```



Discussion

What you've learned

Note: you don't have to answer all of these, and you can answer other questions if you'd like. We just want you to demonstrate what you've learned from the project.

What concepts from lecture/breakout were most relevant to your project? How so?

The model we use to classify facial expression is based on CNN from the lecture. In terms of loss function, considering that it is a multi-classification problem, we choose cross-entropy as the loss function. After the fully connected layer, we use a softmax layer to normalize the probability to 1, which is easier for data processing. And we compare with a classic VGG model. The core of VGG is five sets of convolution operations, each set of convolutions is followed by maximum pooling as well as a fully connected layer and a classification layer.

What aspects of your project did you find most surprising?

Although we have already considered rebalancing data to improve the performance, the accuracy of disgust is still unstasified. There might be two reasons. The first one is that data after rebalancing is still enough to train. The second one is that the defination of happy, surprise, etc. is ambiguous. Labelled data might be not accurate. Next, we could introduce attention module for specific expressions, pay attention to detailed information, and provide further support for improving classification.

What was the most helpful feedback you received during your presentation? Why?

We are inspired by feedback related to the accuracy. One group said that we could combine realistic condition or context to define a facial expression. For example, we prefer to define angry in a quarrel.

If you had two more weeks to work on this project, what would you do next? Why?

As said before, we might conduct experiments with introducing attention module to improve the performance. Also, we would accomplish uncompleted deliverables. 1. Compare different feature selection and feature engineering layers. 2. Test if adding augmentation could improve performance.

In [ ]: