GR 5243 Project 5 Group 5

Implementation of CNN in Emotion Recognition

Introduction

In [0]:
```
!pip install --upgrade tensorflow
```

In [0]:
```
!pip install -q tensorflow tensorflow-datasets matplotlib
import numpy as np
import tensorflow as tf
import pandas as pd
import matplotlib.pyplot as plt
import tensorflow_datasets as tfds
import tensorflow_hub as hub
import os
import cv2
import random
from google.colab import files
from sklearn.model_selection import train_test_split
from keras.utils import to_categorical
print("TF version: ",tf.__version__)
print("Keras version:",tf.keras.__version__)
#import necessary packages
```

In [0]:
```
uploaded = files.upload()
```

In [0]:
```
data = pd.read_csv('fer2013.csv')
```

Our dataset is from Kaggle 'fer2013'. There are 34034 unique values with 7 different emotions: "Angry", "Disgust", "Fear", "Happy", "Sad", "Surprise" and "Neutral".

In [0]:
```
data.values
```

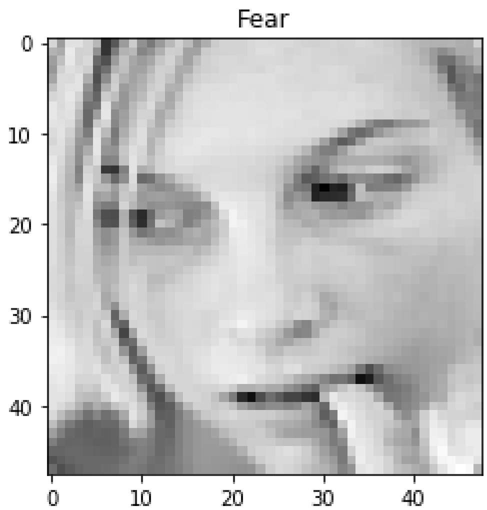
In [0]:
```
def decodeY(y):
    if y==0:
        return 'Angry'
    elif y==1:
        return 'Disgust'
    elif y==2:
        return 'Fear'
    elif y==3:
        return 'Happy'
    elif y==4:
        return 'Sad'
    elif y==5:
        return 'Surprise'
    else:
        return 'Neutral'
#Decode 0-6 to 7 different emotions
```

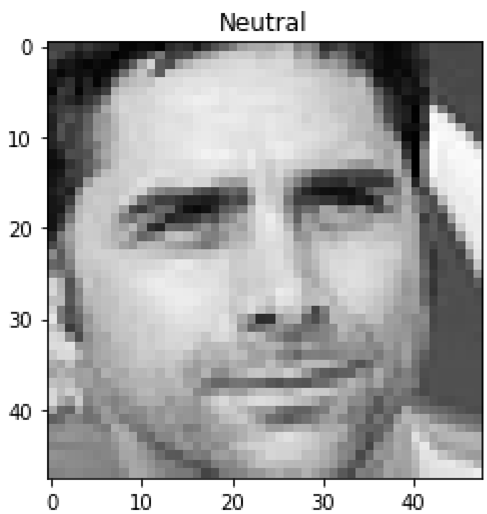
In [0]:
```
def createData(data, test_size):
    data = data.values
    y = data[:, 0]
    pixels = data[:, 1]

    data_discarded = 0
    X = []
    Y = []
    count = 0
    for ix in range(pixels.shape[0]):
            if count%1000 == 0:
                print("[INFO] {} images loaded".format(count))
            temp = np.zeros((48*48))
            p = pixels[ix].split(' ')
            if len(pixels[ix].split(' '))>=2304:
              for iy in range(temp.shape[0]):
                   temp[iy] = int(p[iy])
              X.append(temp)
              Y.append(y[ix])
              count+=1
    X_train, X_test, y_train, y_test = train_test_split(X, Y, test_size=test_s
ize, shuffle=True)
    y_train = to_categorical(y_train)
    y_test = to_categorical(y_test)
    print("[INFO] Done")

    return np.array(X_train), np.array(X_test), y_train, y_test
#Generate images based on image pixels
```

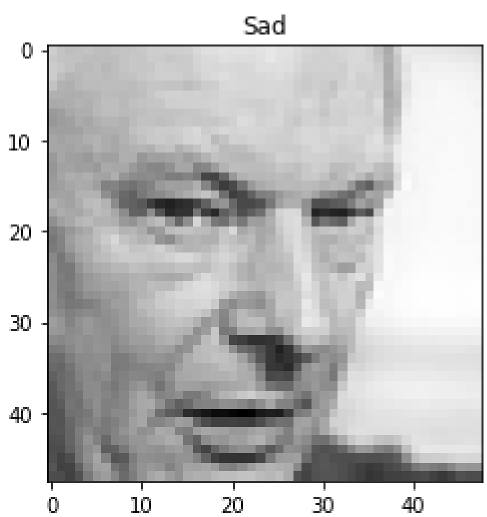
In [0]:
```
X_train, X_test, y_train, y_test = createData(data, 0.2)
#Separate training and testing sets
```

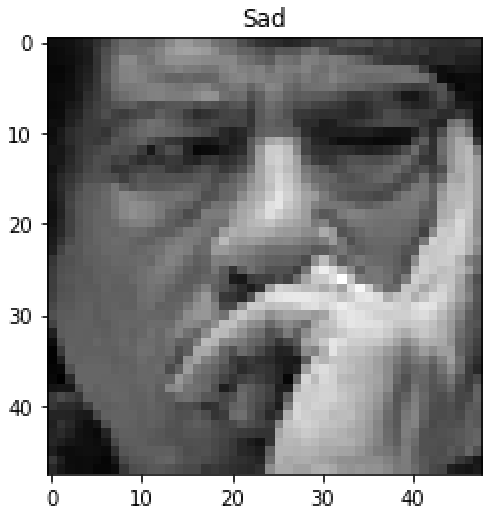
We split the dataset with 80% training and 20% testing. After decoding the pixel indices, we are able to see each image with different emotion.

In [0]:
```
def showImage(X, y):
    for ix in range(4):
        plt.figure(ix)
        plt.title(decodeY(np.argmax(y[ix])))
        plt.imshow(X[ix].reshape((48, 48)), interpolation='none', cmap='gray')
    plt.show()
showImage(X_train, y_train)
#Plot the images
```


Fear
0
10
20
30
40
0
10
20
30
40


Neutral
0
10
20
30
40
0
10
20
30
40


Sad
0
10
20
30
40
0
10
20
30
40



Method

In [0]:
```
def createData_three_channels(data, test_size):
    data = data.values
    y = data[:, 0]
    pixels = data[:, 1]
    data_discarded = 0
    X = []
    Y = []
    count = 0
    for ix in range(pixels.shape[0]):
            if count%1000 == 0:
                print("[INFO] {} images loaded".format(count))
            temp = np.zeros((48*48))
            p = pixels[ix].split(' ')
            if len(pixels[ix].split(' '))>=2304:
              for iy in range(temp.shape[0]):
                  temp[iy] = int(p[iy])
              X.append(temp)
              Y.append(y[ix])
              count+=1
    X_train, X_test, y_train, y_test = train_test_split(X, Y, test_size=test_s
ize, shuffle=True)
    y_train = to_categorical(y_train)
    y_test = to_categorical(y_test)

    X_train_3 = np.stack((X_train,)*3, axis=-1)
    X_test_3 = np.stack((X_test,)*3, axis=-1)

    print("[INFO] Done")
    return np.array(X_train_3), np.array(X_test_3), y_train, y_test
#Modify the function from single channel into three channels
```

In [0]:
```
X_train_3, X_test_3, y_train, y_test = createData_three_channels(data, 0.2)
#Load the images as well as split the training and testing sets
```

```
[INFO] 0 images loaded
[INFO] 1000 images loaded
[INFO] 2000 images loaded
[INFO] 3000 images loaded
[INFO] 4000 images loaded
[INFO] 5000 images loaded
[INFO] 6000 images loaded
[INFO] 7000 images loaded
[INFO] 8000 images loaded
[INFO] 9000 images loaded
[INFO] 10000 images loaded
[INFO] 11000 images loaded
[INFO] 12000 images loaded
[INFO] 13000 images loaded
[INFO] 14000 images loaded
[INFO] 15000 images loaded
[INFO] 16000 images loaded
[INFO] 17000 images loaded
[INFO] 18000 images loaded
[INFO] 19000 images loaded
[INFO] 20000 images loaded
[INFO] 21000 images loaded
[INFO] 22000 images loaded
[INFO] 23000 images loaded
[INFO] 24000 images loaded
[INFO] 25000 images loaded
[INFO] 26000 images loaded
[INFO] 27000 images loaded
[INFO] 28000 images loaded
[INFO] 29000 images loaded
[INFO] 30000 images loaded
[INFO] 31000 images loaded
[INFO] 32000 images loaded
[INFO] 33000 images loaded
[INFO] 34000 images loaded
[INFO] 35000 images loaded
[INFO] Done
```

In [0]:
```
X_train_3 = X_train_3.reshape((X_train_3.shape[0], 48, 48, 3))
X_test_3 = X_test_3.reshape((X_test_3.shape[0], 48, 48, 3))
#Reshape the images into three channels
```

MobileNet Transfer Learning

In [0]:
```
MobileNet = tf.keras.applications.MobileNetV2(input_shape = (48, 48, 3), inclu
de_top = False, weights = 'imagenet')
#Load MobileNet
```

In [0]:
```
model_fitted_mobilenet = tf.keras.Sequential([
     MobileNet,
     tf.keras.layers.Conv2D(128, kernel_size = (3, 3), activation = 'relu', pa
dding = 'same', name = 'conv_ex1'),
     tf.keras.layers.Flatten(),
     tf.keras.layers.Dense(1024, activation = 'relu', name = 'fully_connected_
1'),
     tf.keras.layers.Dense(1024, activation = 'relu', name = 'fully_connected_
2'),
     tf.keras.layers.Dense(512, activation = 'relu', name = 'fully_connected_
3'),
     tf.keras.layers.Dense(7, activation = 'softmax', name = 'fully_connected_
4')
                                      ])
model_fitted_mobilenet.summary()
#Construct model based on MobileNet transfer learning
```

```
Model: "sequential_2"
_________________________________________________________________
Layer (type)                 Output Shape              Param #   
=================================================================
mobilenetv2_1.00_224 (Model) (None, 2, 2, 1280)        2257984   
_________________________________________________________________
conv_ex1 (Conv2D)            (None, 2, 2, 128)         1474688   
_________________________________________________________________
flatten_2 (Flatten)          (None, 512)               0         
_________________________________________________________________
fully_connected_1 (Dense)    (None, 1024)              525312    
_________________________________________________________________
fully_connected_2 (Dense)    (None, 1024)              1049600   
_________________________________________________________________
fully_connected_3 (Dense)    (None, 512)               524800    
_________________________________________________________________
fully_connected_4 (Dense)    (None, 7)                 3591      
=================================================================
Total params: 5,835,975
Trainable params: 5,801,863
Non-trainable params: 34,112
_________________________________________________________________
```

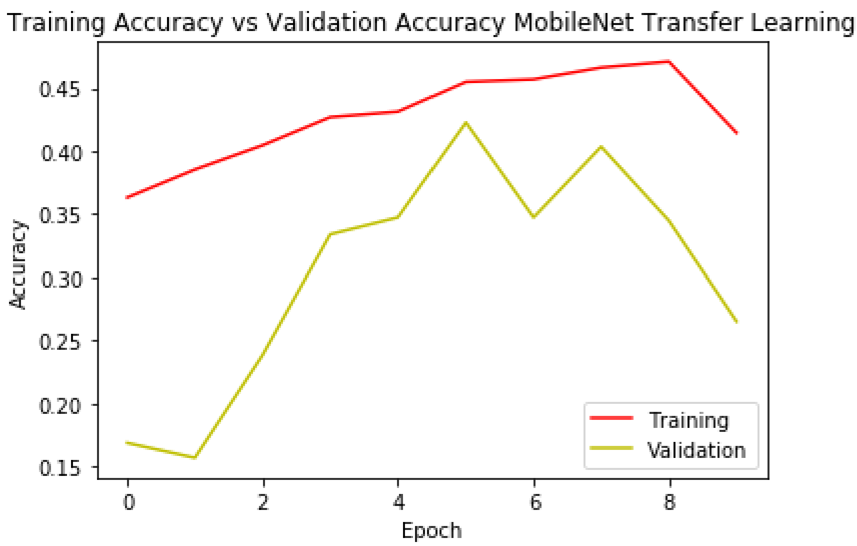
In [0]:
```
model_fitted_mobilenet.compile(optimizer = 'adam',
                       loss = 'categorical_crossentropy',
                       metrics = ['accuracy'])
#Specify loss and optimizer
```

In [0]:
```
model_fitted_1 = model_fitted_mobilenet.fit(
   X_train_3,
   y_train,
   epochs = 10,
   batch_size = 32,
   validation_data = (X_test_3, y_test))
#Train the model with 10 epochs and batch size 32
```

```
Train on 28709 samples, validate on 7178 samples
Epoch 1/10
28709/28709 [==============================] - 913s 32ms/sample - loss: 1.603
7 - accuracy: 0.3632 - val_loss: 1.8413 - val_accuracy: 0.1686
Epoch 2/10
28709/28709 [==============================] - 904s 32ms/sample - loss: 1.538
1 - accuracy: 0.3853 - val_loss: 2.1594 - val_accuracy: 0.1567
Epoch 3/10
28709/28709 [==============================] - 906s 32ms/sample - loss: 1.499
6 - accuracy: 0.4046 - val_loss: 1.9526 - val_accuracy: 0.2382
Epoch 4/10
28709/28709 [==============================] - 916s 32ms/sample - loss: 1.450
9 - accuracy: 0.4269 - val_loss: 2.7453 - val_accuracy: 0.3339
Epoch 5/10
28709/28709 [==============================] - 917s 32ms/sample - loss: 1.428
2 - accuracy: 0.4313 - val_loss: 2.2654 - val_accuracy: 0.3473
Epoch 6/10
28709/28709 [==============================] - 896s 31ms/sample - loss: 1.385
7 - accuracy: 0.4549 - val_loss: 1.7253 - val_accuracy: 0.4227
Epoch 7/10
28709/28709 [==============================] - 879s 31ms/sample - loss: 1.372
8 - accuracy: 0.4570 - val_loss: 1.9435 - val_accuracy: 0.3473
Epoch 8/10
28709/28709 [==============================] - 879s 31ms/sample - loss: 1.368
4 - accuracy: 0.4663 - val_loss: 2.0441 - val_accuracy: 0.4035
Epoch 9/10
28709/28709 [==============================] - 893s 31ms/sample - loss: 1.350
8 - accuracy: 0.4711 - val_loss: 2.0093 - val_accuracy: 0.3448
Epoch 10/10
28709/28709 [==============================] - 876s 31ms/sample - loss: 1.495
2 - accuracy: 0.4145 - val_loss: 3.4515 - val_accuracy: 0.2650
```

In [0]:
```
plt.plot(model_fitted_1.history['accuracy'], "-r", label = "Training")
plt.plot(model_fitted_1.history['val_accuracy'], "-y", label = "Validation")
plt.legend(loc = 'lower right')
plt.title('Training Accuracy vs Validation Accuracy MobileNet Transfer Learnin
g', loc = 'center')
plt.xlabel('Epoch')
plt.ylabel('Accuracy')
#Plot training accuracy vs validation accuracy
```

Out[0]: Text(0, 0.5, 'Accuracy')



ResNet Transfer Learning

In [0]:
```
ResNet = tf.keras.applications.ResNet50(input_shape = (48, 48, 3), include_top
= False, weights = 'imagenet')
#Load ResNet50
```

In [0]:
```
model_fitted_2 = tf.keras.Sequential([
     ResNet,
     tf.keras.layers.GlobalAveragePooling2D(),
     tf.keras.layers.Dense(1024, activation = 'relu'),
     tf.keras.layers.Dense(7, activation = 'softmax')
                                      ])
model_fitted_2.summary()
#Construct model based on ResNet transfer learning
```

```
Model: "sequential"
_________________________________________________________________
Layer (type)                 Output Shape              Param #   
=================================================================
resnet50 (Model)             (None, 2, 2, 2048)        23587712  
_________________________________________________________________
global_average_pooling2d (Gl (None, 2048)              0         
_________________________________________________________________
dense (Dense)                (None, 1024)              2098176   
_________________________________________________________________
dense_1 (Dense)              (None, 7)                 7175      
=================================================================
Total params: 25,693,063
Trainable params: 25,639,943
Non-trainable params: 53,120
_________________________________________________________________
```

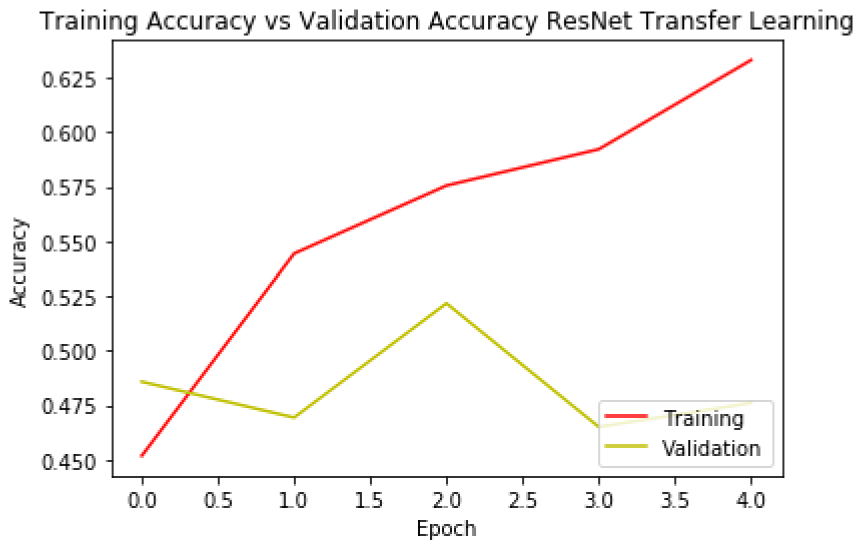
In [0]:
```
model_fitted_2.compile(optimizer = 'adam',
                       loss = 'categorical_crossentropy',
                       metrics = ['accuracy'])
#Specify loss and optimizer
```

In [0]:
```
model_fitted_resnet = model_fitted_2.fit(
   X_train_3,
   y_train,
   epochs = 5,
   batch_size = 32,
   validation_data = (X_test_3, y_test))
#Train model with 5 epochs and batch size 32
```

```
Train on 28709 samples, validate on 7178 samples
Epoch 1/5
28709/28709 [==============================] - 4632s 161ms/sample - loss: 1.4
482 - accuracy: 0.4518 - val_loss: 1.3367 - val_accuracy: 0.4858
Epoch 2/5
28709/28709 [==============================] - 4591s 160ms/sample - loss: 1.2
187 - accuracy: 0.5444 - val_loss: 1.4709 - val_accuracy: 0.4694
Epoch 3/5
28709/28709 [==============================] - 4710s 164ms/sample - loss: 1.1
310 - accuracy: 0.5755 - val_loss: 1.2533 - val_accuracy: 0.5217
Epoch 4/5
28709/28709 [==============================] - 4619s 161ms/sample - loss: 1.0
962 - accuracy: 0.5922 - val_loss: 1.5208 - val_accuracy: 0.4650
Epoch 5/5
28709/28709 [==============================] - 4628s 161ms/sample - loss: 0.9
863 - accuracy: 0.6331 - val_loss: 1.3661 - val_accuracy: 0.4760
```

In [0]:
```
plt.plot(model_fitted_resnet.history['accuracy'], "-r", label = "Training")
plt.plot(model_fitted_resnet.history['val_accuracy'], "-y", label = "Validatio
n")
plt.legend(loc = 'lower right')
plt.title('Training Accuracy vs Validation Accuracy ResNet Transfer Learning',
loc = 'center')
plt.xlabel('Epoch')
plt.ylabel('Accuracy')
#Plot training accuracy vs validation accuracy
```

Out[0]: Text(0, 0.5, 'Accuracy')



VGG19 Transfer Learning

In [0]:
```
from tensorflow.keras.applications import VGG19
vgg19 = VGG19()
#Load VGG19
```

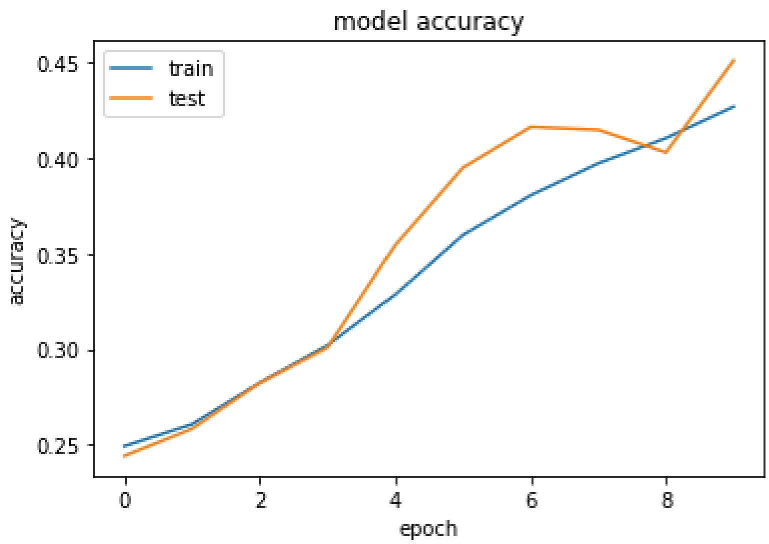
In [0]:
```
image_size = 48
image_channel = 1
input_shape = (image_size, image_size, image_channel)

vgg_transfer = tf.keras.Sequential([
     tf.keras.layers.Conv2D(32, kernel_size = (3, 3), padding = 'same',
     activation = 'relu', input_shape = input_shape, name = 'conv_1'),
     tf.keras.layers.Conv2D(32, kernel_size = (3, 3), padding = 'same',
     activation = 'relu', input_shape = input_shape, name = 'conv_12'),
     tf.keras.layers.MaxPooling2D(pool_size = (2, 2), name = 'pooling_1'),
     tf.keras.layers.Dropout(0.5),
     tf.keras.layers.Conv2D(64, kernel_size = (3, 3), padding = 'same',
     activation = 'relu', name = 'conv_2'),
     tf.keras.layers.Conv2D(64, kernel_size = (3, 3), padding = 'valid',
     activation = 'relu', name = 'conv_3'),
     tf.keras.layers.MaxPooling2D(pool_size = (2, 2), name = 'pooling_2'),
     tf.keras.layers.Dropout(0.5),
     tf.keras.layers.Conv2D(96, kernel_size = (3, 3), padding = 'same',
     activation = 'relu', name = 'conv_4'),
     tf.keras.layers.Conv2D(96, kernel_size = (3, 3), padding = 'valid',
     activation = 'relu', name = 'conv_5'),
     tf.keras.layers.MaxPooling2D(pool_size = (2, 2), name = 'pooling_3'),
     tf.keras.layers.Dropout(0.5),
     tf.keras.layers.Conv2D(128, kernel_size = (3, 3), padding = 'same',
     activation = 'relu', name = 'conv_6'),
     tf.keras.layers.Conv2D(128, kernel_size = (3, 3), padding = 'same',
     activation = 'relu', name = 'conv_7'),
     tf.keras.layers.MaxPooling2D(pool_size = (2, 2), name = 'pooling_4'),

     tf.keras.layers.Dropout(0.5),
     tf.keras.layers.Flatten(),
     tf.keras.layers.Dense(128, activation = 'relu', name = 'fully_connected_
1'),
     tf.keras.layers.Dense(7, activation = 'softmax', name = 'fully_connected_
2')
                                      ])
vgg_transfer.summary()
#Construct model based on VGG19 transfer learning
```

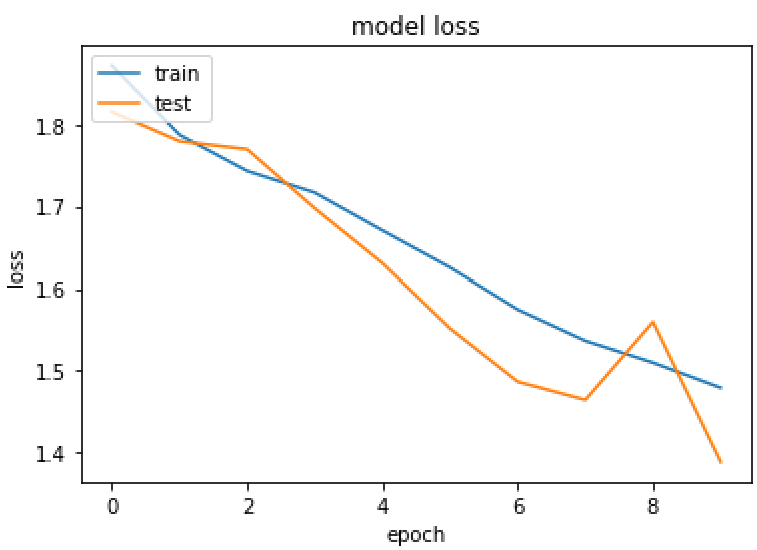
```
Model: "sequential_1"
_________________________________________________________________
Layer (type)                 Output Shape              Param #   
=================================================================
conv_1 (Conv2D)              (None, 48, 48, 32)        320       
_________________________________________________________________
conv_12 (Conv2D)             (None, 48, 48, 32)        9248      
_________________________________________________________________
pooling_1 (MaxPooling2D)     (None, 24, 24, 32)        0         
_________________________________________________________________
dropout (Dropout)            (None, 24, 24, 32)        0         
_________________________________________________________________
conv_2 (Conv2D)              (None, 24, 24, 64)        18496     
_________________________________________________________________
conv_3 (Conv2D)              (None, 22, 22, 64)        36928     
_________________________________________________________________
pooling_2 (MaxPooling2D)     (None, 11, 11, 64)        0         
_________________________________________________________________
dropout_1 (Dropout)          (None, 11, 11, 64)        0         
_________________________________________________________________
conv_4 (Conv2D)              (None, 11, 11, 96)        55392     
_________________________________________________________________
conv_5 (Conv2D)              (None, 9, 9, 96)          83040     
_________________________________________________________________
pooling_3 (MaxPooling2D)     (None, 4, 4, 96)          0         
_________________________________________________________________
dropout_2 (Dropout)          (None, 4, 4, 96)          0         
_________________________________________________________________
conv_6 (Conv2D)              (None, 4, 4, 128)         110720    
_________________________________________________________________
conv_7 (Conv2D)              (None, 4, 4, 128)         147584    
_________________________________________________________________
pooling_4 (MaxPooling2D)     (None, 2, 2, 128)         0         
_________________________________________________________________
dropout_3 (Dropout)          (None, 2, 2, 128)         0         
_________________________________________________________________
flatten (Flatten)            (None, 512)               0         
_________________________________________________________________
fully_connected_1 (Dense)    (None, 128)               65664     
_________________________________________________________________
fully_connected_2 (Dense)    (None, 7)                 903       
=================================================================
Total params: 528,295
Trainable params: 528,295
Non-trainable params: 0
_________________________________________________________________
```

In [0]:
```
batch_size = 32
epochs = 10

vgg_transfer.compile(loss = "categorical_crossentropy", optimizer = 'adam', me
trics = ['accuracy'])
history = vgg_transfer.fit(X_train, y_train,
                    epochs=epochs,
                    batch_size=batch_size,
                    validation_data=(X_test, y_test))
#Specify loss and optimizer and train the model with 10 epochs and batch size
 32
```

```
Train on 28709 samples, validate on 7178 samples
Epoch 1/10
28709/28709 [==============================] - 455s 16ms/sample - loss: 1.872
1 - accuracy: 0.2492 - val_loss: 1.8154 - val_accuracy: 0.2441
Epoch 2/10
28709/28709 [==============================] - 458s 16ms/sample - loss: 1.787
4 - accuracy: 0.2606 - val_loss: 1.7796 - val_accuracy: 0.2583
Epoch 3/10
28709/28709 [==============================] - 455s 16ms/sample - loss: 1.743
3 - accuracy: 0.2824 - val_loss: 1.7699 - val_accuracy: 0.2823
Epoch 4/10
28709/28709 [==============================] - 454s 16ms/sample - loss: 1.716
9 - accuracy: 0.3019 - val_loss: 1.6977 - val_accuracy: 0.3008
Epoch 5/10
28709/28709 [==============================] - 450s 16ms/sample - loss: 1.670
7 - accuracy: 0.3286 - val_loss: 1.6310 - val_accuracy: 0.3546
Epoch 6/10
28709/28709 [==============================] - 451s 16ms/sample - loss: 1.625
7 - accuracy: 0.3598 - val_loss: 1.5516 - val_accuracy: 0.3951
Epoch 7/10
28709/28709 [==============================] - 451s 16ms/sample - loss: 1.574
7 - accuracy: 0.3806 - val_loss: 1.4867 - val_accuracy: 0.4164
Epoch 8/10
28709/28709 [==============================] - 452s 16ms/sample - loss: 1.536
6 - accuracy: 0.3974 - val_loss: 1.4646 - val_accuracy: 0.4149
Epoch 9/10
28709/28709 [==============================] - 451s 16ms/sample - loss: 1.510
0 - accuracy: 0.4106 - val_loss: 1.5597 - val_accuracy: 0.4030
Epoch 10/10
28709/28709 [==============================] - 454s 16ms/sample - loss: 1.479
4 - accuracy: 0.4270 - val_loss: 1.3889 - val_accuracy: 0.4512
```

In [0]:
```
plt.plot(history.history['accuracy'])
plt.plot(history.history['val_accuracy'])
plt.title('model accuracy')
plt.ylabel('accuracy')
plt.xlabel('epoch')
plt.legend(['train', 'test'], loc='upper left')
plt.show()
# summarize history for loss
plt.plot(history.history['loss'])
plt.plot(history.history['val_loss'])
plt.title('model loss')
plt.ylabel('loss')
plt.xlabel('epoch')
plt.legend(['train', 'test'], loc='upper left')
plt.show()
```





Based on the validation accuracies, validation losses and training times of VGG19, MobileNet as well as ResNet Transfer Learning, VGG19 outperforms the other two. As a result, we decide to improve our model based on the VGG19 architecture with additional modifications.

Improved Model

In [0]:
```
image_size = 48
image_channel = 1
input_shape = (image_size, image_size, image_channel)
improved_model = tf.keras.Sequential([
     tf.keras.layers.Conv2D(32, kernel_size = (3, 3), padding = 'same',
     activation = 'relu', input_shape = input_shape, name = 'conv_1'),
     tf.keras.layers.MaxPooling2D(pool_size = (2, 2), name = 'pooling_1'),

     tf.keras.layers.Conv2D(64, kernel_size = (3, 3), padding = 'same',
     activation = 'relu', name = 'conv_2'),
     tf.keras.layers.Conv2D(64, kernel_size = (3, 3), padding = 'same',
     activation = 'relu', name = 'conv_3'),
     tf.keras.layers.MaxPooling2D(pool_size = (2, 2), name = 'pooling_2'),

     tf.keras.layers.Conv2D(64, kernel_size = (3, 3), padding = 'same',
     activation = 'relu', name = 'conv_4'),
     tf.keras.layers.Conv2D(64, kernel_size = (3, 3), padding = 'same',
     activation = 'relu', name = 'conv_5'),
     tf.keras.layers.MaxPooling2D(pool_size = (2, 2), name = 'pooling_3'),

     tf.keras.layers.Conv2D(128, kernel_size = (3, 3), padding = 'same',
     activation = 'relu', name = 'conv_6'),
     tf.keras.layers.Conv2D(128, kernel_size = (3, 3), padding = 'same',
     activation = 'relu', name = 'conv_7'),
     tf.keras.layers.Conv2D(128, kernel_size = (3, 3), padding = 'same',
     activation = 'relu', name = 'conv_8'),
     tf.keras.layers.MaxPooling2D(pool_size = (2, 2), name = 'pooling_4'),

     tf.keras.layers.Dropout(0.5),
     tf.keras.layers.Flatten(),
     tf.keras.layers.Dense(128, activation = 'relu', name = 'fully_connected_
1'),
     tf.keras.layers.Dense(7, activation = 'softmax', name = 'fully_connected_
2')
                                      ])
improved_model.summary()
#Construct Convolutional Neural Networks Model
```

```
Model: "sequential_10"
_________________________________________________________________
Layer (type)                 Output Shape              Param #   
=================================================================
conv_1 (Conv2D)              (None, 48, 48, 32)        320       
_________________________________________________________________
pooling_1 (MaxPooling2D)     (None, 24, 24, 32)        0         
_________________________________________________________________
conv_2 (Conv2D)              (None, 24, 24, 64)        18496     
_________________________________________________________________
conv_3 (Conv2D)              (None, 24, 24, 64)        36928     
_________________________________________________________________
pooling_2 (MaxPooling2D)     (None, 12, 12, 64)        0         
_________________________________________________________________
conv_4 (Conv2D)              (None, 12, 12, 64)        36928     
_________________________________________________________________
conv_5 (Conv2D)              (None, 12, 12, 64)        36928     
_________________________________________________________________
pooling_3 (MaxPooling2D)     (None, 6, 6, 64)          0         
_________________________________________________________________
conv_6 (Conv2D)              (None, 6, 6, 128)         73856     
_________________________________________________________________
conv_7 (Conv2D)              (None, 6, 6, 128)         147584    
_________________________________________________________________
conv_8 (Conv2D)              (None, 6, 6, 128)         147584    
_________________________________________________________________
pooling_4 (MaxPooling2D)     (None, 3, 3, 128)         0         
_________________________________________________________________
dropout_6 (Dropout)          (None, 3, 3, 128)         0         
_________________________________________________________________
flatten_10 (Flatten)         (None, 1152)              0         
_________________________________________________________________
fully_connected_1 (Dense)    (None, 128)               147584    
_________________________________________________________________
fully_connected_2 (Dense)    (None, 7)                 903       
=================================================================
Total params: 647,111
Trainable params: 647,111
Non-trainable params: 0
_________________________________________________________________
```

In [0]:

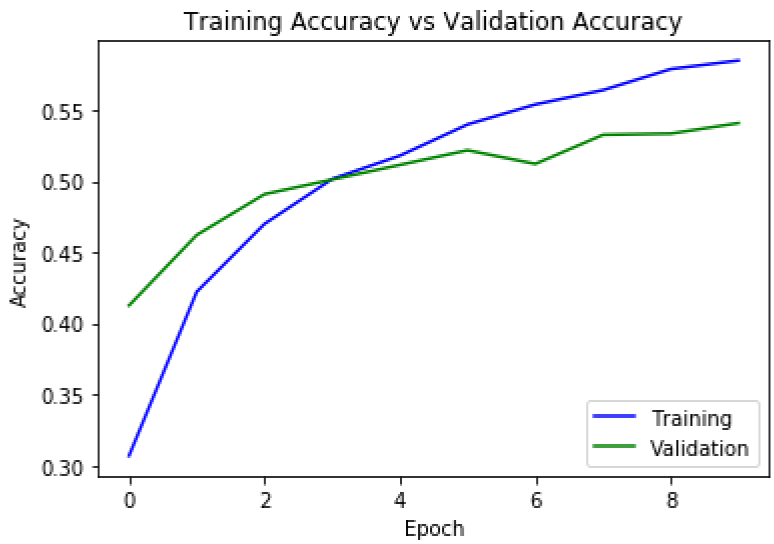
```python
improved_model.compile(optimizer = 'adam',
                       loss = 'categorical_crossentropy',
                       metrics = ['accuracy'])
#Specify loss and optimizer
```

In [0]:
```
improved_model_fitted = improved_model.fit(
   X_train,
   y_train,
   epochs = 10,
   batch_size = 32,
   validation_data = (X_test, y_test))
#Train the improved model with batch size 32 and 10 epochs
```

```
Train on 28709 samples, validate on 7178 samples
Epoch 1/10
28709/28709 [==============================] - 366s 13ms/sample - loss: 1.720
8 - accuracy: 0.3068 - val_loss: 1.5130 - val_accuracy: 0.4125
Epoch 2/10
28709/28709 [==============================] - 364s 13ms/sample - loss: 1.480
1 - accuracy: 0.4222 - val_loss: 1.4000 - val_accuracy: 0.4621
Epoch 3/10
28709/28709 [==============================] - 364s 13ms/sample - loss: 1.367
8 - accuracy: 0.4699 - val_loss: 1.3151 - val_accuracy: 0.4908
Epoch 4/10
28709/28709 [==============================] - 364s 13ms/sample - loss: 1.290
0 - accuracy: 0.5015 - val_loss: 1.2876 - val_accuracy: 0.5011
Epoch 5/10
28709/28709 [==============================] - 364s 13ms/sample - loss: 1.249
6 - accuracy: 0.5179 - val_loss: 1.2972 - val_accuracy: 0.5114
Epoch 6/10
28709/28709 [==============================] - 369s 13ms/sample - loss: 1.202
4 - accuracy: 0.5399 - val_loss: 1.2452 - val_accuracy: 0.5217
Epoch 7/10
28709/28709 [==============================] - 368s 13ms/sample - loss: 1.166
2 - accuracy: 0.5539 - val_loss: 1.2768 - val_accuracy: 0.5121
Epoch 8/10
28709/28709 [==============================] - 368s 13ms/sample - loss: 1.136
1 - accuracy: 0.5640 - val_loss: 1.2320 - val_accuracy: 0.5326
Epoch 9/10
28709/28709 [==============================] - 368s 13ms/sample - loss: 1.109
9 - accuracy: 0.5788 - val_loss: 1.2172 - val_accuracy: 0.5334
Epoch 10/10
28709/28709 [==============================] - 369s 13ms/sample - loss: 1.089
8 - accuracy: 0.5849 - val_loss: 1.2041 - val_accuracy: 0.5408
```

In [0]:
```
plt.plot(model_fitted.history['accuracy'], "-b", label = "Training")
plt.plot(model_fitted.history['val_accuracy'], "-g", label = "Validation")
plt.legend(loc = 'lower right')
plt.title('Training Accuracy vs Validation Accuracy', loc = 'center')
plt.xlabel('Epoch')
plt.ylabel('Accuracy')
#Plot training accuracy vs validation accuracy
```

Out[0]:
```
Text(0, 0.5, 'Accuracy')
```



After 10 epochs, the training accuracy is 0.5849 and validation accuracy 0.5408 which overfitting is not a concern.

Results

Compared to our first model, the loss decreases from 1.7208 to 1.0898 as well as validation loss decreases from 1.5130 to 1.2041. Both training and validation accuracy increase a significant amount. After 10 epochs, the training accuracy is 0.5849 and validation accuracy 0.5408. More importantly, the training time drops from around 450s to 360s for each epoch.

Conclusion

Throughout our model development, there are four major potential problems. Firstly, images from fer2013 have low resolutions (48 * 48 pixels) which increase the difficulty of emotion detection. Secondly, it has only one channel (greyscale) which a huge challenge for transfer learning by using pre-trained model (usually with three color channels). Thirdly, increasing number of epoch may lead to overfitting concerns since model gets the result simply based on memory instead of learning. Besides, training accuracy can always increase but validation accuracy may be stable and even decrease throughout the learning process. Lastly, some of the facial images are partially covered which dramatically increase the emotion detection difficulty. For our future works, we would like to try different loss functions as well as use parallel connected layers to make our model more comprehensive and accurate.