```
from google.colab import drive
drive.mount('/content/drive')
```

```
    Mounted at /content/drive
```

```
import numpy as np
```

```
import tensorflow as tf
```

Data loading from csv file

```
with open("/content/drive/MyDrive/Colab Notebooks/dataset2/fer2013.csv") as f:
    content = f.readlines()
```

```
lines = np.array(content)
```

```
num_of_instances = lines.size
print("number of instances: ",num_of_instances)
print("instance length: ",len(lines[1].split(",")[1].split(" ")))
```

```
    number of instances:  35888
    instance length:  2304
```

```
num_classes = 7 #angry, disgust, fear, happy, sad, surprise, neutral
```

```
x_train, y_train, x_test, y_test = [], [], [], []
```

```
for i in range(1,num_of_instances):
  emotion, img, usage = lines[i].split(",")
  val = img.split(" ")
  pixels = np.array(val, 'float32')
  emotion = tf.keras.utils.to_categorical(emotion, num_classes)
  if 'Training' in usage:
    y_train.append(emotion)
    x_train.append(pixels)
  elif 'PublicTest' in usage:
    y_test.append(emotion)
    x_test.append(pixels)
  elif 'PrivateTest' in usage:
    y_test.append(emotion)
    x_test.append(pixels)
```

```
x_train = np.array(x_train, 'float32')
y_train = np.array(y_train, 'float32')
x_test = np.array(x_test, 'float32')
y_test = np.array(y_test, 'float32')
```

```
x_train /= 255 #normalize inputs between [0, 1]
x_test /= 255

x_train = x_train.reshape(x_train.shape[0], 48, 48, 1)
x_test = x_test.reshape(x_test.shape[0], 48, 48, 1)
```

Checking Dims

```
print(x_train.shape)
print(y_train.shape)
print(x_test.shape)
print(y_test.shape)
```

```
    (28709, 48, 48, 1)
    (28709, 7)
    (7178, 48, 48, 1)
    (7178, 7)
```

Weigths initializer

not used

```
#initializer = tf.keras.initializers.RandomNormal(mean=0., stddev=0.00001, seed=1234646445
```

vgg16 Model

```
# Build the model
emo_model = tf.keras.Sequential([
  tf.keras.layers.Conv2D(64,kernel_size =3, activation='relu', padding ='same', kernel_reg
  tf.keras.layers.BatchNormalization(),
  tf.keras.layers.Conv2D(64,kernel_size =3, activation='relu', padding ='same', kernel_reg
  tf.keras.layers.BatchNormalization(),
  tf.keras.layers.MaxPool2D(pool_size =2, strides =2, padding ='same'),
  tf.keras.layers.BatchNormalization(),
  tf.keras.layers.Conv2D(128,kernel_size =3, activation='relu', padding ='same', kernel_re
  tf.keras.layers.BatchNormalization(),
  tf.keras.layers.Conv2D(128,kernel_size =3, activation='relu', padding ='same', kernel_re
  tf.keras.layers.BatchNormalization(),
  tf.keras.layers.MaxPool2D(pool_size =2, strides =2, padding ='same'),
  tf.keras.layers.BatchNormalization(),
  tf.keras.layers.Conv2D(256,kernel_size =3, activation='relu', padding ='same', kernel_re
  tf.keras.layers.BatchNormalization(),
  tf.keras.layers.Conv2D(256,kernel_size =3, activation='relu', padding ='same', kernel_re
  tf.keras.layers.BatchNormalization(),
  tf.keras.layers.Conv2D(256,kernel_size =3, activation='relu', padding ='same', kernel_re
  tf.keras.layers.BatchNormalization(),
  tf.keras.layers.MaxPool2D(pool_size =2, strides =2, padding ='same'),
  tf.keras.layers.BatchNormalization(),
  tf.keras.layers.Conv2D(512,kernel_size =3, activation='relu', padding ='same', kernel_re
```

```
  tf.keras.layers.BatchNormalization(),
  tf.keras.layers.Conv2D(512,kernel_size =3, activation='relu', padding ='same', kernel_re
  tf.keras.layers.BatchNormalization(),
  tf.keras.layers.Conv2D(512,kernel_size =3, activation='relu', padding ='same', kernel_re
  tf.keras.layers.BatchNormalization(),
  tf.keras.layers.MaxPool2D(pool_size =2, strides =2, padding ='same'),
  tf.keras.layers.BatchNormalization(),
  tf.keras.layers.Conv2D(512,kernel_size =3, activation='relu', padding ='same', kernel_re
  tf.keras.layers.BatchNormalization(),
  tf.keras.layers.Conv2D(512,kernel_size =3, activation='relu', padding ='same', kernel_re
  tf.keras.layers.BatchNormalization(),
  tf.keras.layers.Conv2D(512,kernel_size =3, activation='relu', padding ='same', kernel_re
  tf.keras.layers.BatchNormalization(),
  tf.keras.layers.MaxPool2D(pool_size =2, strides =2, padding ='same'),
  tf.keras.layers.BatchNormalization(),
  tf.keras.layers.Flatten(),
  tf.keras.layers.Dense(units = 4096, activation ='relu', kernel_initializer='he_normal'),
  tf.keras.layers.Dropout(0.5),
  tf.keras.layers.Dense(units = 4096, activation ='relu', kernel_initializer='he_normal'),
  tf.keras.layers.Dropout(0.5),
  tf.keras.layers.Dense(units = 1000, activation ='relu', kernel_initializer='he_normal'),
  tf.keras.layers.Dense(units = 7, activation ='softmax')
  ])
```

```
emo_model.summary()
```

```
    Model: "sequential"
    _____
    Layer (type)                 Output Shape              Param #
    =================================================================
    conv2d (Conv2D)              (None, 48, 48, 64)        640
    _____
    batch_normalization (BatchNo (None, 48, 48, 64)        256
    _____
    conv2d_1 (Conv2D)            (None, 48, 48, 64)        36928
    _____
    batch_normalization_1 (Batch (None, 48, 48, 64)        256
    _____
    max_pooling2d (MaxPooling2D) (None, 24, 24, 64)        0
    _____
    batch_normalization_2 (Batch (None, 24, 24, 64)        256
    _____
    conv2d_2 (Conv2D)            (None, 24, 24, 128)       73856
    _____
    batch_normalization_3 (Batch (None, 24, 24, 128)       512
    _____
    conv2d_3 (Conv2D)            (None, 24, 24, 128)       147584
    _____
    batch_normalization_4 (Batch (None, 24, 24, 128)       512
    _____
    max_pooling2d_1 (MaxPooling2 (None, 12, 12, 128)       0
    _____
    batch_normalization_5 (Batch (None, 12, 12, 128)       512
    _____
    conv2d_4 (Conv2D)            (None, 12, 12, 256)       295168
    _____
    batch_normalization_6 (Batch (None, 12, 12, 256)       1024
```

```
_____
conv2d_5 (Conv2D)              (None, 12, 12, 256)        590080
_____
batch_normalization_7 (Batch   (None, 12, 12, 256)        1024
_____
conv2d_6 (Conv2D)              (None, 12, 12, 256)        590080
_____
batch_normalization_8 (Batch   (None, 12, 12, 256)        1024
_____
max_pooling2d_2 (MaxPooling2   (None, 6, 6, 256)          0
_____
batch_normalization_9 (Batch   (None, 6, 6, 256)          1024
_____
conv2d_7 (Conv2D)              (None, 6, 6, 512)          1180160
_____
batch_normalization_10 (Batc   (None, 6, 6, 512)          2048
_____
conv2d_8 (Conv2D)              (None, 6, 6, 512)          2359808
_____
batch_normalization_11 (Batc   (None, 6, 6, 512)          2048
_____
conv2d_9 (Conv2D)              (None, 6, 6, 512)          2359808
_____
batch_normalization_12 (Batc   (None, 6, 6, 512)          2048
_____
max_pooling2d_3 (MaxPooling2   (None, 3, 3, 512)          0
_____
batch normalization 13 (Batc   (None, 3, 3, 512)          2048
```

Input Output test

```
predictions = emo_model(x_train[1:2]).numpy()
print(predictions)

    [[0.14286013 0.14287986 0.142858   0.14287551 0.14280865 0.1429946
      0.14272323]]
```

Learning rate decay

not used

```
#lr_schedule = tf.keras.optimizers.schedules.ExponentialDecay(initial_learning_rate=0.1,de
```

Optimizer Loss Function and Metrics

```
sgd = tf.keras.optimizers.SGD(
    learning_rate=0.000001, momentum=0.99, nesterov=True
)
loss_fn = tf.keras.losses.CategoricalCrossentropy(from_logits=True)
```

Compiling

```
emo_model.compile(optimizer=sgd,
```

```
            loss=loss_fn,
            metrics=['accuracy'])
```

Checkpoints

```
emo_model.load_weights("/content/drive/MyDrive/Colab Notebooks/checkpoints/model7/cp.ckpt"

    <tensorflow.python.training.tracking.util.CheckpointLoadStatus at 0x7f50dcbea510>


checkpoint_path = "/content/drive/MyDrive/Colab Notebooks/checkpoints/model7/cp.ckpt"

cp_callback = tf.keras.callbacks.ModelCheckpoint(filepath=checkpoint_path,monitor='accurac
```

Data Augmentation

```
##### Include Little Data Augmentation
batch_size = 60 # try several values

train_DataGen = tf.keras.preprocessing.image.ImageDataGenerator(zoom_range=0.2,
                                                    width_shift_range=0.1,
                                                    height_shift_range = 0.1,
                                                    horizontal_flip=True)
train_set_conv = train_DataGen.flow(x_train, y_train, batch_size=batch_size) # train_lab i
```

Running ...

last 30 epoch

```
#history = emo_model.fit(train_set_conv,batch_size=batch_size,callbacks=[cp_callback] ,shu
history = emo_model.fit(train_set_conv,epochs=160,steps_per_epoch=x_train.shape[0]/batch_s

    Epoch 1/160
    478/478 [==============================] - 25s 48ms/step - loss: 0.3284 - accuracy

    Epoch 00001: accuracy improved from 0.95475 to 0.95531, saving model to /content/d
    Epoch 2/160
    478/478 [==============================] - 23s 48ms/step - loss: 0.3243 - accuracy

    Epoch 00002: accuracy improved from 0.95531 to 0.95548, saving model to /content/d
    Epoch 3/160
    478/478 [==============================] - 23s 48ms/step - loss: 0.3287 - accuracy

    Epoch 00003: accuracy improved from 0.95548 to 0.95562, saving model to /content/d
    Epoch 4/160
    478/478 [==============================] - 23s 48ms/step - loss: 0.3323 - accuracy

    Epoch 00004: accuracy did not improve from 0.95562
    Epoch 5/160
    478/478 [==============================] - 23s 47ms/step - loss: 0.3225 - accuracy

    Epoch 00005: accuracy improved from 0.95562 to 0.95608, saving model to /content/d
    Epoch 6/160
```

```
478/478 [==============================] - 23s 48ms/step - loss: 0.3264 - accuracy

Epoch 00006: accuracy did not improve from 0.95608
Epoch 7/160
478/478 [==============================] - 23s 47ms/step - loss: 0.3209 - accuracy

Epoch 00007: accuracy did not improve from 0.95608
Epoch 8/160
478/478 [==============================] - 23s 47ms/step - loss: 0.3238 - accuracy

Epoch 00008: accuracy did not improve from 0.95608
Epoch 9/160
478/478 [==============================] - 23s 47ms/step - loss: 0.3280 - accuracy

Epoch 00009: accuracy did not improve from 0.95608
Epoch 10/160
478/478 [==============================] - 23s 47ms/step - loss: 0.3260 - accuracy

Epoch 00010: accuracy did not improve from 0.95608
Epoch 11/160
478/478 [==============================] - 22s 47ms/step - loss: 0.3272 - accuracy

Epoch 00011: accuracy did not improve from 0.95608
Epoch 12/160
478/478 [==============================] - 23s 47ms/step - loss: 0.3204 - accuracy

Epoch 00012: accuracy did not improve from 0.95608
Epoch 13/160
478/478 [==============================] - 23s 47ms/step - loss: 0.3283 - accuracy

Epoch 00013: accuracy did not improve from 0.95608
Epoch 14/160
478/478 [==============================] - 23s 47ms/step - loss: 0.3269 - accuracy

Epoch 00014: accuracy did not improve from 0.95608
Epoch 15/160
478/478 [==============================] - 23s 47ms/step - loss: 0.3229 - accuracy
```

```python
import matplotlib.pyplot as plt
# list all data in history
print(history.history.keys())
# summarize history for accuracy
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()
```
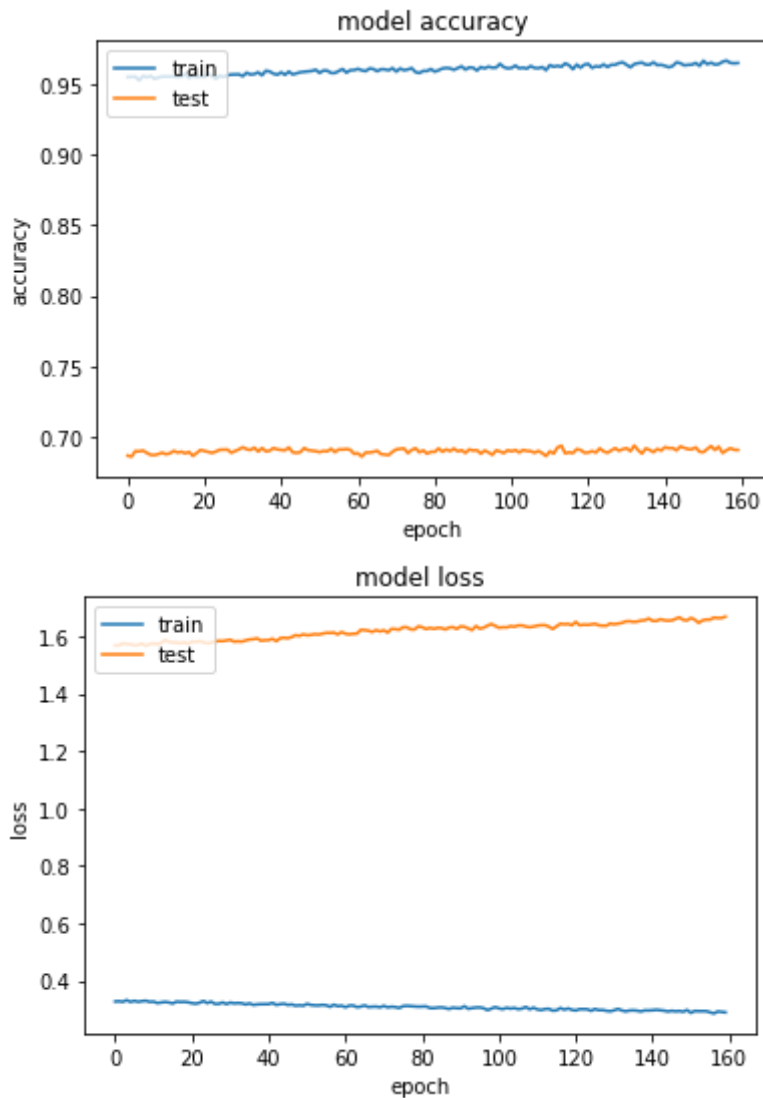
```
dict_keys(['loss', 'accuracy', 'val_loss', 'val_accuracy'])
```


Train Test results

```
train_score = emo_model.evaluate(x_train, y_train, verbose=0)
print('Train loss:', train_score[0])
print('Train accuracy:', 100*train_score[1])

test_score = emo_model.evaluate(x_test, y_test, verbose=0)
print('Test loss:', test_score[0])
print('Test accuracy:', 100*test_score[1])
```

```
Train loss: 0.25686976313591003
Train accuracy: 97.80556559562683
Test loss: 1.6681681871414185
Test accuracy: 69.03036832809448
```

Real world Test

```
def emotion_analysis(emotions):
    objects = ('angry', 'disgust', 'fear', 'happy', 'sad', 'surprise', 'neutral')
    y_pos = np.arange(len(objects))
```

```
    plt.bar(y_pos, emotions, align='center', alpha=0.5)
    plt.xticks(y_pos, objects)
    plt.ylabel('percentage')
    plt.title('emotion')

    plt.show()


from keras.preprocessing import image
from keras.preprocessing.image import ImageDataGenerator

file = '/content/drive/MyDrive/Colab Notebooks/test/test2.png'
true_image = image.load_img(file)
img = image.load_img(file, grayscale=True, target_size=(48, 48))

x = image.img_to_array(img)
x = np.expand_dims(x, axis = 0)

x /= 255

custom = emo_model.predict(x)
emotion_analysis(custom[0])
print(custom[0])

x = np.array(x, 'float32')
x = x.reshape([48, 48]);

plt.gray()
plt.imshow(true_image)
plt.show()
```
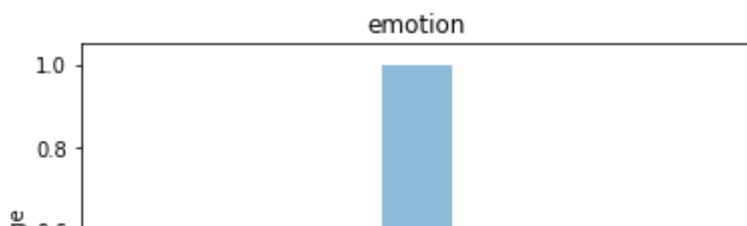
```
/usr/local/lib/python3.7/dist-packages/keras_preprocessing/image/utils.py:107: UserWa
  warnings.warn('grayscale is deprecated. Please use '
WARNING:tensorflow:5 out of the last 9 calls to <function Model.make_predict_functior
```
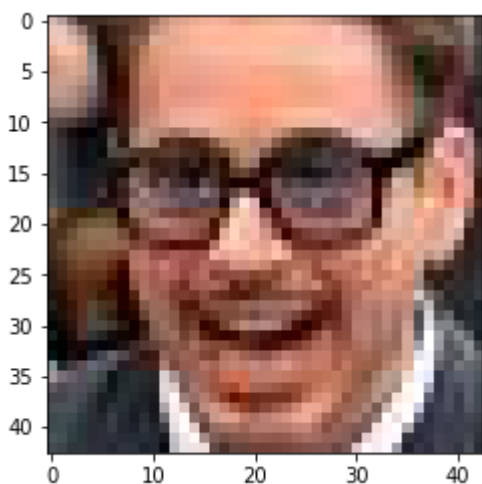

Saving model

```
print(emo_model.save('/content/drive/MyDrive/Colab Notebooks/checkpoints/SuccessModel4'))
```

```
INFO:tensorflow:Assets written to: /content/drive/MyDrive/Colab Notebooks/checkpoints
None
```

```
3.0244765e-04 1.3936314e-04]
```


✓ 0s   completed at 11:15 PM   ● ✕