```
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=0.01, momentum=0.95, 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

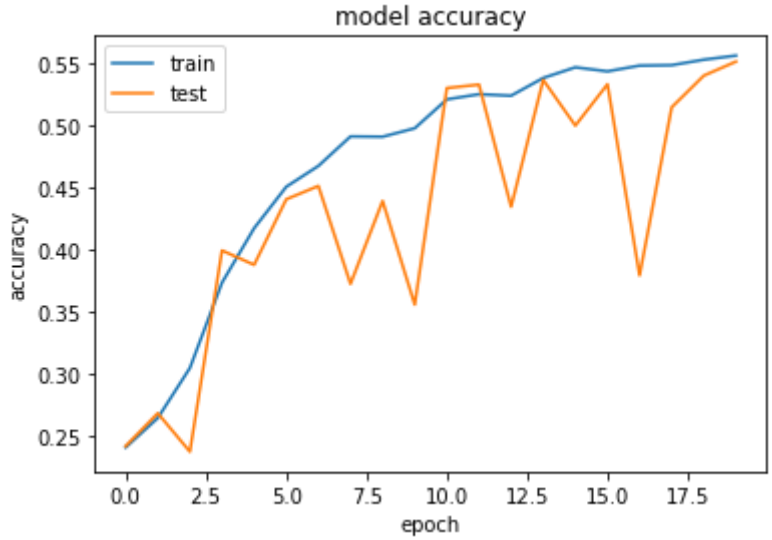
```
#history = emo_model.fit(train_set_conv,batch_size=batch_size,callbacks=[cp_callback] ,shu
history = emo_model.fit(train_set_conv,epochs=20,steps_per_epoch=x_train.shape[0]/batch_si
```

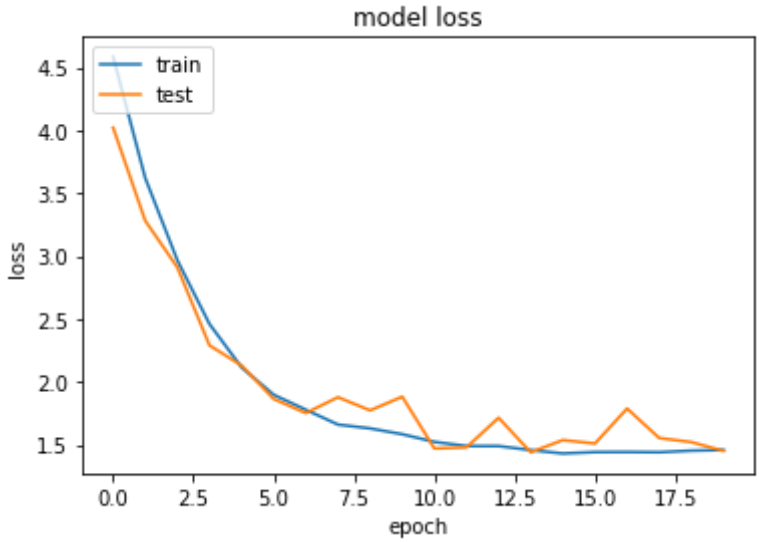
```
    Epoch 1/20
    478/478 [==============================] - 27s 51ms/step - loss: 4.9745 - accuracy

    Epoch 00001: accuracy improved from -inf to 0.24027, saving model to /content/driv
    Epoch 2/20
    478/478 [==============================] - 23s 48ms/step - loss: 3.8092 - accuracy

    Epoch 00002: accuracy improved from 0.24027 to 0.26392, saving model to /content/d
    Epoch 3/20
    478/478 [==============================] - 23s 49ms/step - loss: 3.1119 - accuracy

    Epoch 00003: accuracy improved from 0.26392 to 0.30423, saving model to /content/d
    Epoch 4/20
    478/478 [==============================] - 23s 49ms/step - loss: 2.5712 - accuracy

    Epoch 00004: accuracy improved from 0.30423 to 0.37295, saving model to /content/d
    Epoch 5/20
    478/478 [==============================] - 23s 48ms/step - loss: 2.1857 - accuracy

    Epoch 00005: accuracy improved from 0.37295 to 0.41680, saving model to /content/d
    Epoch 6/20
```

```
478/478 [==============================] - 23s 48ms/step - loss: 1.9387 - accuracy

Epoch 00006: accuracy improved from 0.41680 to 0.45017, saving model to /content/d
Epoch 7/20
478/478 [==============================] - 23s 48ms/step - loss: 1.7986 - accuracy

Epoch 00007: accuracy improved from 0.45017 to 0.46696, saving model to /content/d
Epoch 8/20
478/478 [==============================] - 23s 48ms/step - loss: 1.6705 - accuracy

Epoch 00008: accuracy improved from 0.46696 to 0.49072, saving model to /content/d
Epoch 9/20
478/478 [==============================] - 23s 48ms/step - loss: 1.6467 - accuracy

Epoch 00009: accuracy did not improve from 0.49072
Epoch 10/20
478/478 [==============================] - 23s 48ms/step - loss: 1.5866 - accuracy

Epoch 00010: accuracy improved from 0.49072 to 0.49730, saving model to /content/d
Epoch 11/20
478/478 [==============================] - 23s 48ms/step - loss: 1.5263 - accuracy

Epoch 00011: accuracy improved from 0.49730 to 0.52057, saving model to /content/d
Epoch 12/20
478/478 [==============================] - 23s 48ms/step - loss: 1.4916 - accuracy

Epoch 00012: accuracy improved from 0.52057 to 0.52471, saving model to /content/d
Epoch 13/20
478/478 [==============================] - 23s 48ms/step - loss: 1.4845 - accuracy

Epoch 00013: accuracy did not improve from 0.52471
Epoch 14/20
478/478 [==============================] - 23s 48ms/step - loss: 1.4657 - accuracy

Epoch 00014: accuracy improved from 0.52471 to 0.53781, saving model to /content/d
Epoch 15/20
478/478 [==============================] - 23s 48ms/step - loss: 1.4376 - accuracy
```

```
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])
```

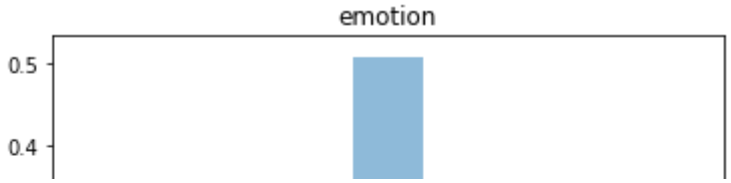
```
Train loss: 1.418951392173767
Train accuracy: 56.278520822525024
Test loss: 1.4507994651794434
Test accuracy: 55.098915100097656
```

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 '
```

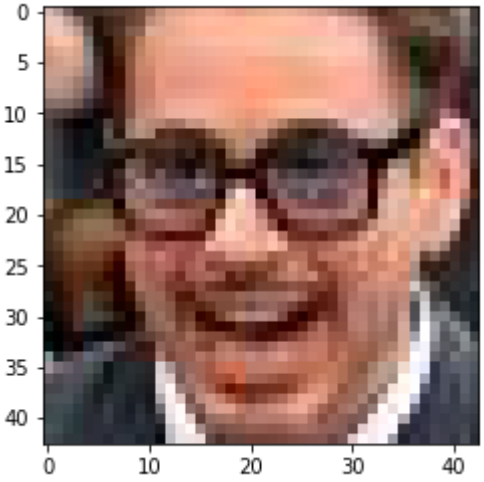


Saving model

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

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

```
[0.08202116 0.00676601 0.06842619 0.50902873 0.11190096 0.05613279
 0.16572419]
```



✓ 1s completed at 7:56 PM