

```
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)
```

## Weights 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 Normalization)	(None, 12, 12, 256)	1024
conv2d_6 (Conv2D)	(None, 12, 12, 256)	590080
batch_normalization_8 (Batch Normalization)	(None, 12, 12, 256)	1024
max_pooling2d_2 (MaxPooling2D)	(None, 6, 6, 256)	0
batch_normalization_9 (Batch Normalization)	(None, 6, 6, 256)	1024
conv2d_7 (Conv2D)	(None, 6, 6, 512)	1180160
batch_normalization_10 (Batch Normalization)	(None, 6, 6, 512)	2048
conv2d_8 (Conv2D)	(None, 6, 6, 512)	2359808
batch_normalization_11 (Batch Normalization)	(None, 6, 6, 512)	2048
conv2d_9 (Conv2D)	(None, 6, 6, 512)	2359808
batch_normalization_12 (Batch Normalization)	(None, 6, 6, 512)	2048
max_pooling2d_3 (MaxPooling2D)	(None, 3, 3, 512)	0
batch_normalization_13 (Batch Normalization)	(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

```
adm = tf.keras.optimizers.Adamax(
    learning_rate=0.000001
)
loss_fn = tf.keras.losses.CategoricalCrossentropy(from_logits=True)
```

## Compiling

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

```
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 0x7f4efbe96750>

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_label i
```

## Running ...

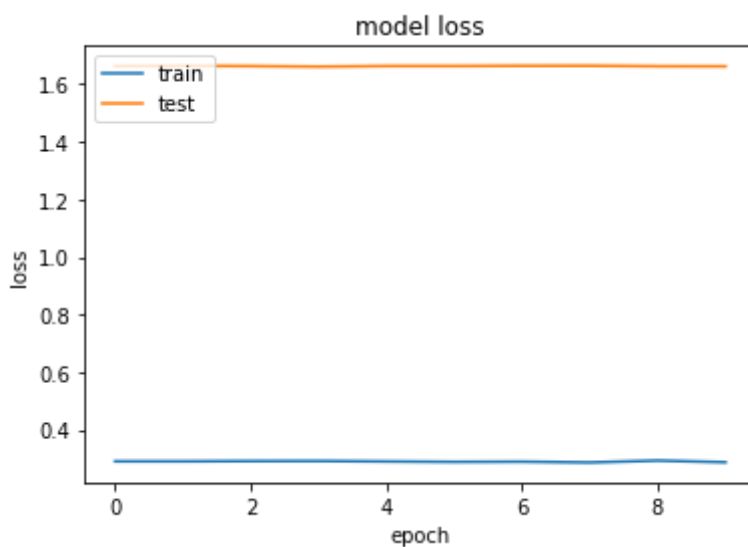
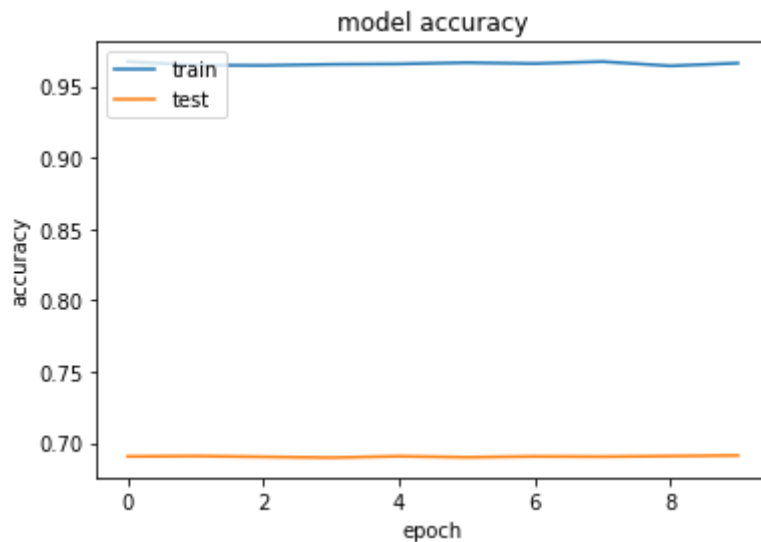
last 30 epoch

```
#history = emo_model.fit(train_set_conv,batch_size=batch_size,callbacks=[cp_callback] ,shu
history = emo_model.fit(train_set_conv,epochs=10,steps_per_epoch=x_train.shape[0]/batch_si
```

```
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()
```

```
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: 0.25552335381507874
Train accuracy: 97.87523150444031
Test loss: 1.6596894264221191
Test accuracy: 69.00250911712646
```

## Real world Test

```
def emotion_analysis(emotions):
    objects = ('angry', 'disgust', 'fear', 'happy', 'sad', 'surprise', 'neutral')
    v_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()
```

+ Code

+ Text

```
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)

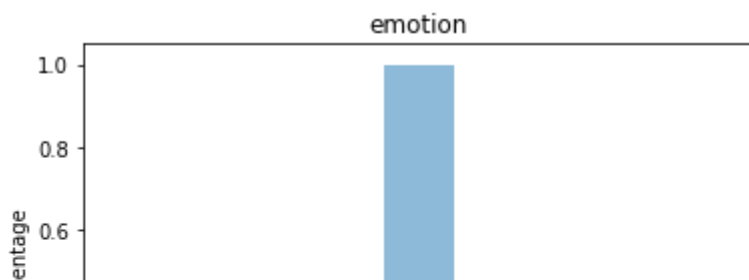
x /= 255

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()
```

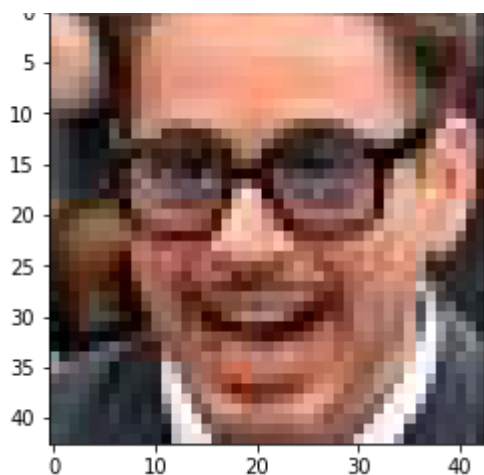
```
/usr/local/lib/python3.7/dist-packages/keras_preprocessing/image/utils.py:107: UserWarning:
  warnings.warn('grayscale is deprecated. Please use ')
WARNING:tensorflow:6 out of the last 10 calls to <function Model.make_predict_function
```



## 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
```



✓ 9s completed at 11:59 PM

● ✕