In [1]:

```
import csv, json, random
import numpy as np
import matplotlib.pyplot as plt
import tensorflow as tf
```

In [4]:

```
samples = 35393 # 2~35394

image_height = 48
image_width = 48
emotions_count = 8

images = []
emotions = []
emotion_labels = []
with open('./dataset.csv') as file:
    reader = csv.reader(file)
    for line in reader:
        image_pixels = line[-1].split()
        if len(image_pixels) == 1:
            emotion_labels = line[2:2+emotions_count]
            assert(emotion_labels == ['neutral', 'happiness', 'surprise', 'sadness', 'anger
            continue

        image = []
        for i in range(image_height):
            row = []
            for j in range(image_width):
                row.append(image_pixels[image_width*i+j])
            row = list(map(int, row))
            image.append(row)
        images.append(image)

        emotion = []
        for i in range(2, 2+emotions_count):
            emotion.append(line[i])
        emotion = list(map(float, emotion))
        emotions.append(emotion)

images = np.array(images).reshape(samples, image_height, image_width, 1)
emotions = np.array(emotions)

print("images shape:", images.shape)
print("emotions shape:", emotions.shape)
print("emotion_labels:", emotion_labels)


# # check the last 9 images
# plt.figure(figsize=(10, 10))
# for i in range(35384,35393):
#     ax = plt.subplot(3, 3, i-35383)
#     plt.imshow(images[i].astype("uint8"))
#     plt.title(emotion_labels[np.argmax(emotions[i])])
#     plt.axis("off")
```

```
images shape: (35393, 48, 48, 1)
emotions shape: (35393, 8)
emotion_labels: ['neutral', 'happiness', 'surprise', 'sadness', 'anger', 'di
sgust', 'fear', 'contempt']
```

In [5]:

```
emotions
```

Out[5]:

```
array([[0.4     , 0.      , 0.      , ..., 0.2     , 0.      , 0.      ],
       [0.75    , 0.      , 0.125   , ..., 0.      , 0.      , 0.      ],
       [0.555556, 0.      , 0.      , ..., 0.      , 0.      , 0.      ],
       ...,
       [0.      , 0.      , 0.      , ..., 0.1     , 0.      , 0.2     ],
       [0.      , 1.      , 0.      , ..., 0.      , 0.      , 0.      ],
       [0.222222, 0.      , 0.      , ..., 0.111111, 0.      , 0.      ]])
```

In [6]:

```
for i in range(emotions.shape[0]):
    max_indices = []
    maximum = np.amax(emotions[i])
    for j in range(emotions.shape[1]):
        if emotions[i][j] == maximum:
            max_indices.append(j)
    no_of_maxs = len(max_indices)
    for j in range(emotions.shape[1]):
        if j in max_indices:
            emotions[i][j] = 1/no_of_maxs
        else:
            emotions[i][j] = 0
emotions
```

Out[6]:

```
array([[1., 0., 0., ..., 0., 0., 0.],
       [1., 0., 0., ..., 0., 0., 0.],
       [1., 0., 0., ..., 0., 0., 0.],
       ...,
       [0., 0., 0., ..., 0., 0., 0.],
       [0., 1., 0., ..., 0., 0., 0.],
       [0., 0., 0., ..., 0., 0., 0.]])
```

In [7]:

```
for i in range(10,30):
    print(emotions[i])
```

```
[0. 0. 0. 0. 1. 0. 0. 0.]
[1. 0. 0. 0. 0. 0. 0. 0.]
[1. 0. 0. 0. 0. 0. 0. 0.]
[1. 0. 0. 0. 0. 0. 0. 0.]
[0. 1. 0. 0. 0. 0. 0. 0.]
[0. 0. 1. 0. 0. 0. 0. 0.]
[0. 1. 0. 0. 0. 0. 0. 0.]
[0. 0. 0. 1. 0. 0. 0. 0.]
[1. 0. 0. 0. 0. 0. 0. 0.]
[1. 0. 0. 0. 0. 0. 0. 0.]
[0. 0. 0. 0. 0. 0. 1. 0.]
[0. 0. 0. 0. 1. 0. 0. 0.]
[0. 1. 0. 0. 0. 0. 0. 0.]
[0. 1. 0. 0. 0. 0. 0. 0.]
[0.5 0.  0.5 0.  0.  0.  0.  0. ]
[0. 0. 0. 0. 1. 0. 0. 0.]
[0. 1. 0. 0. 0. 0. 0. 0.]
[0. 0. 1. 0. 0. 0. 0. 0.]
[0. 1. 0. 0. 0. 0. 0. 0.]
[1. 0. 0. 0. 0. 0. 0. 0.]
```

In [8]:

```
1/2
```

Out[8]:

```
0.5
```

In [9]:

```
images = tf.convert_to_tensor(images)
emotions = tf.convert_to_tensor(emotions)
images = tf.image.grayscale_to_rgb(images)
print("images shape:", images.shape)
print("emotions shape:", emotions.shape)
```

```
images shape: (35393, 48, 48, 3)
emotions shape: (35393, 8)
```

In [10]:

```
# images = tf.image.resize(images, [224,224])
# print("images shape:", images.shape)
```

In [11]:

```
from tensorflow.python.keras.applications.vgg16 import preprocess_input
from tensorflow.python.keras import layers

# choose one method:
images = layers.Rescaling(1./127.5, offset= -1)(images)
#images = preprocess_input(images)
```

In [12]:

```
training_samples = 28317  # 2~28318 (Training)
validation_samples = 3541 # 28319~31859 (PublicTest)
test_samples = 3535       # 31860~35394 (PrivateTest)

training_size = training_samples + validation_samples
test_size = test_samples

training_images = images[:training_size]
test_images = images[training_size:]
training_emotions = emotions[:training_size]
test_emotions = emotions[training_size:]

print("training_images shape:", training_images.shape)
print("training_emotions shape:", training_emotions.shape)
print("test_images shape:", test_images.shape)
print("test_emotions shape:", test_emotions.shape)
```

```
training_images shape: (31858, 48, 48, 3)
training_emotions shape: (31858, 8)
test_images shape: (3535, 48, 48, 3)
test_emotions shape: (3535, 8)
```

In [13]:

```
# # check the last 9 images
# plt.figure(figsize=(10, 10))
# for i in range(6):
#     ax = plt.subplot(3, 3, i+1)
#     plt.imshow(special_images[i].astype("uint8"))
#     plt.axis("off")
```

In [14]:

```
tf.config.run_functions_eagerly(True)
def model_acc(y_true, y_pred):
    size = y_true.shape[0]
    acc = 0
    for i in range(size):
        true = y_true[i]
        pred = y_pred[i]
        index_max = tf.argmax(pred).numpy()
        if true[index_max].numpy()==tf.reduce_max(true).numpy():
            acc += 1
    return acc/size
def loss_func(y_true, y_pred):
    size = y_true.shape[0]
    loss = 0
    for i in range(size):
        true = y_true[i]
        pred = y_pred[i]
        index_max = tf.argmax(pred).numpy()
        if true[index_max].numpy()!=tf.reduce_max(true).numpy():
            loss += 1
    return loss/size
```

In [15]:

```
from tensorflow.python.keras.applications.vgg16 import VGG16
from tensorflow.python.keras.layers import Dense, GlobalAveragePooling2D
from tensorflow.python.keras.models import Model
from tensorflow.python.keras import layers, Sequential

vgg_model = VGG16(include_top=False, weights="imagenet", input_shape=(48,48,3))
vgg_model.trainable=False
model = Sequential([
    vgg_model,
    layers.GlobalAveragePooling2D(),
    layers.Dense(1024, activation='relu'),
    layers.Dense(emotions_count, activation='softmax'),
])
#model.summary()
```

In [16]:

```
from tensorflow.python.keras import losses, metrics

#m = top_k_categorical_accuracy(y_true, y_pred, k=3)
#model.compile(optimizer='sgd', loss=losses.CategoricalCrossentropy(), metrics = [metrics.C
model.compile(optimizer='adam', loss=losses.CategoricalCrossentropy(), metrics = [model_acc
```

In [17]:

```
model.fit(x=training_images,
        y=training_emotions,
        batch_size=32,
        epochs=10,
        validation_data=(test_images, test_emotions))
```

```
C:\Users\Darkl\AppData\Roaming\Python\Python38\site-packages\tensorflow\pyth
on\data\ops\dataset_ops.py:4211: UserWarning: Even though the `tf.config.exp
erimental_run_functions_eagerly` option is set, this option does not apply t
o tf.data functions. To force eager execution of tf.data functions, please u
se `tf.data.experimental.enable_debug_mode()`.
  warnings.warn(

Epoch 1/10
996/996 [==============================] - 865s 869ms/step - loss: 1.3427 -
model_acc: 0.5156 - val_loss: 1.2979 - val_model_acc: 0.5364
Epoch 2/10
996/996 [==============================] - 860s 863ms/step - loss: 1.2241 -
model_acc: 0.5599 - val_loss: 1.2503 - val_model_acc: 0.5464
Epoch 3/10
996/996 [==============================] - 860s 864ms/step - loss: 1.1559 -
model_acc: 0.5877 - val_loss: 1.2163 - val_model_acc: 0.5628
Epoch 4/10
996/996 [==============================] - 862s 866ms/step - loss: 1.0921 -
model_acc: 0.6180 - val_loss: 1.2049 - val_model_acc: 0.5757
Epoch 5/10
996/996 [==============================] - 873s 876ms/step - loss: 1.0272 -
model_acc: 0.6416 - val_loss: 1.1879 - val_model_acc: 0.5745
Epoch 6/10
996/996 [==============================] - 866s 870ms/step - loss: 0.9626 -
model_acc: 0.6666 - val_loss: 1.2157 - val_model_acc: 0.5802
Epoch 7/10
996/996 [==============================] - 865s 869ms/step - loss: 0.9017 -
model_acc: 0.6892 - val_loss: 1.2101 - val_model_acc: 0.5855
Epoch 8/10
996/996 [==============================] - 865s 869ms/step - loss: 0.8356 -
model_acc: 0.7151 - val_loss: 1.2166 - val_model_acc: 0.5942
Epoch 9/10
996/996 [==============================] - 866s 869ms/step - loss: 0.7759 -
model_acc: 0.7394 - val_loss: 1.2362 - val_model_acc: 0.5880
Epoch 10/10
996/996 [==============================] - 866s 869ms/step - loss: 0.7180 -
model_acc: 0.7620 - val_loss: 1.3056 - val_model_acc: 0.5801
```

Out[17]:

```
<tensorflow.python.keras.callbacks.History at 0x23326616190>
```

In [18]:

```
m = metrics.CategoricalAccuracy()
m.update_state([[0, 1, 1], [0, 1, 0]], [[0.1, 0.9, 0.98],[0.05, 0.95, 0]])
m.result().numpy()
```

Out[18]:

```
0.5
```

In [ ]:

```
```