In [1]:
```
import numpy as np
import matplotlib.pyplot as plt
import tensorflow as tf
from tensorflow.python.keras.layers import Dense, GlobalAveragePooling2D
from tensorflow.python.keras.models import Model
from tensorflow.python.keras import layers, Sequential,losses, metrics

image_height = 48
image_width = 48
emotions_count = 8
emotion_labels = ['neutral', 'happiness', 'surprise', 'sadness',
                  'anger', 'disgust', 'fear', 'contempt']

samples = 35393 # 2~35394
training_samples = 28317  # 2~28318 (Training)
validation_samples = 3541 # 28319~31859 (PublicTest)
test_samples = 3535       # 31860~35394 (PrivateTest)
expw_samples = 35000

image_path = "./dataset/images.npy"
emotion_path = "./dataset/emotions_single.npy"
image_path_expw = "./sample35k/images.npy"
emotion_path_expw = "./sample35k/emotions.npy"
```

In [2]:
```
images = np.load(image_path)
emotions = np.load(emotion_path)
images_expw = np.load(image_path_expw)
emotions_expw = np.load(emotion_path_expw)

print(images.shape)
print(emotions.shape)
print(images_expw.shape)
print(emotions_expw.shape)
```

```
(35393, 48, 48, 1)
(35393, 8)
(35000, 48, 48, 3)
(35000, 8)
```

In [3]:
```
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
```

In [4]:

```
images = tf.convert_to_tensor(images)
images = tf.image.grayscale_to_rgb(images)
emotions = tf.convert_to_tensor(emotions)
```

In [7]:

```
images_expw = tf.convert_to_tensor(images_expw)
emotions_expw = tf.convert_to_tensor(emotions_expw)
```

In [6]:

```
images = layers.Rescaling(1./127.5, offset= -1)(images)
#images_expw = layers.Rescaling(1./127.5, offset= -1)(images_expw)
```

In [8]:

```
training_size = training_samples + validation_samples
test_size = test_samples
```

In [9]:

```
training_images = tf.concat([images_expw, images[:training_size]], 0)
test_images = images[training_size:]
training_emotions = tf.concat([emotions_expw, emotions[:training_size]], 0)
test_emotions = emotions[training_size:]
```

In [14]:

```
samples += expw_samples
print("total sample:", samples)
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)
```

```
total sample: 70393
training_images shape: (66858, 48, 48, 3)
training_emotions shape: (66858, 8)
test_images shape: (3535, 48, 48, 3)
test_emotions shape: (3535, 8)
```

In [17]:

```
from tensorflow.python.keras.layers import Dense, GlobalAveragePooling2D
from tensorflow.python.keras.models import Model
from tensorflow.python.keras import layers, Sequential,losses, metrics
from tensorflow.python.keras.applications import vgg16, resnet_v2
from tensorflow.python.keras import optimizers
from tensorflow.python.keras.optimizer_v2 import adam

base_model = vgg16.VGG16(include_top=False,
                         weights="imagenet",
                         input_shape=(48,48,3))
base_model.trainable=True
model = Sequential([
    base_model,
    layers.GlobalAveragePooling2D(),
    layers.Dense(4096, activation='relu'),
    layers.Dense(4096, activation='relu'),
    layers.Dense(emotions_count, activation='softmax'),
])

model.compile(optimizer=adam.Adam(learning_rate=1e-4),
              loss=losses.CategoricalCrossentropy(),
              metrics = [model_acc])

model.fit(x=training_images,
          y=training_emotions,
          batch_size=32,
          epochs=40,
          validation_data=(test_images, test_emotions))
```

```
C:\Users\Darkl\anaconda3\lib\site-packages\tensorflow\python\data\ops\dataset_ops.py:3703: UserWarning: Even though the `tf.confi
g.experimental_run_functions_eagerly` option is set, this option does not apply to tf.data functions. To force eager execution of
tf.data functions, please use `tf.data.experimental.enable.debug_mode()`.
  warnings.warn(
Epoch 1/40
2090/2090 [==============================] - 143s 66ms/step - loss: 1.1116 - model_acc: 0.6159 - val_loss: 0.8146 - val_model_acc:
0.7405
Epoch 2/40
```

```
2090/2090 [==============================] - 136s 65ms/step - loss: 0.8512 - model_acc: 0.7229 - val_loss: 0.6675 - val_model_acc:
0.7923
Epoch 3/40
2090/2090 [==============================] - 135s 65ms/step - loss: 0.7303 - model_acc: 0.7637 - val_loss: 0.6467 - val_model_acc:
0.7951
Epoch 4/40
2090/2090 [==============================] - 137s 65ms/step - loss: 0.6295 - model_acc: 0.7962 - val_loss: 0.5802 - val_model_acc:
0.8245
Epoch 5/40
2090/2090 [==============================] - 138s 66ms/step - loss: 0.5180 - model_acc: 0.8360 - val_loss: 0.6780 - val_model_acc:
0.8115
Epoch 6/40
2090/2090 [==============================] - 139s 66ms/step - loss: 0.4106 - model_acc: 0.8720 - val_loss: 0.6578 - val_model_acc:
0.8137
Epoch 7/40
2090/2090 [==============================] - 136s 65ms/step - loss: 0.3126 - model_acc: 0.9054 - val_loss: 0.6963 - val_model_acc:
0.8177
Epoch 8/40
2090/2090 [==============================] - 135s 65ms/step - loss: 0.2432 - model_acc: 0.9297 - val_loss: 0.7172 - val_model_acc:
0.8183
Epoch 9/40
2090/2090 [==============================] - 136s 65ms/step - loss: 0.1912 - model_acc: 0.9475 - val_loss: 0.7391 - val_model_acc:
0.8146
Epoch 10/40
2090/2090 [==============================] - 136s 65ms/step - loss: 0.1616 - model_acc: 0.9575 - val_loss: 0.8381 - val_model_acc:
0.8179
Epoch 11/40
2090/2090 [==============================] - 137s 65ms/step - loss: 0.1408 - model_acc: 0.9643 - val_loss: 0.8663 - val_model_acc:
0.8284
Epoch 12/40
2090/2090 [==============================] - 148s 71ms/step - loss: 0.1255 - model_acc: 0.9706 - val_loss: 0.8545 - val_model_acc:
0.8213
Epoch 13/40
2090/2090 [==============================] - 155s 74ms/step - loss: 0.1135 - model_acc: 0.9736 - val_loss: 0.8524 - val_model_acc:
0.8225
Epoch 14/40
2090/2090 [==============================] - 141s 67ms/step - loss: 0.1080 - model_acc: 0.9753 - val_loss: 0.8315 - val_model_acc:
0.8242
Epoch 15/40
2090/2090 [==============================] - 143s 69ms/step - loss: 0.1000 - model_acc: 0.9769 - val_loss: 1.0250 - val_model_acc:
0.8287
Epoch 16/40
2090/2090 [==============================] - 140s 67ms/step - loss: 0.0939 - model_acc: 0.9796 - val_loss: 0.7924 - val_model_acc:
0.8301
```

```
Epoch 17/40
2090/2090 [==============================] - 141s 67ms/step - loss: 0.0897 - model_acc: 0.9806 - val_loss: 0.8564 - val_model_acc:
0.8244
Epoch 18/40
2090/2090 [==============================] - 141s 67ms/step - loss: 0.0877 - model_acc: 0.9819 - val_loss: 0.8333 - val_model_acc:
0.8168
Epoch 19/40
2090/2090 [==============================] - 140s 67ms/step - loss: 0.0832 - model_acc: 0.9832 - val_loss: 0.9705 - val_model_acc:
0.8197
Epoch 20/40
2090/2090 [==============================] - 146s 70ms/step - loss: 0.0799 - model_acc: 0.9835 - val_loss: 0.9241 - val_model_acc:
0.8245
Epoch 21/40
2090/2090 [==============================] - 150s 72ms/step - loss: 0.0757 - model_acc: 0.9845 - val_loss: 0.9058 - val_model_acc:
0.8279
Epoch 22/40
 229/2090 [==>...........................] - ETA: 2:11 - loss: 0.0626 - model_acc: 0.9892
---------------------------------------------------------------------------
KeyboardInterrupt                         Traceback (most recent call last)
~\AppData\Local\Temp/ipykernel_17320/1577467506.py in <module>
     22               metrics = [model_acc])
     23
---> 24 model.fit(x=training_images,
     25           y=training_emotions,
     26           batch_size=32,

~\anaconda3\lib\site-packages\tensorflow\python\keras\engine\training.py in fit(self, x, y, batch_size, epochs, verbose, callback
s, validation_split, validation_data, shuffle, class_weight, sample_weight, initial_epoch, steps_per_epoch, validation_steps, vali
dation_batch_size, validation_freq, max_queue_size, workers, use_multiprocessing)
   1181                 _r=1):
   1182               callbacks.on_train_batch_begin(step)
-> 1183               tmp_logs = self.train_function(iterator)
   1184               if data_handler.should_sync:
   1185                 context.async_wait()

~\anaconda3\lib\site-packages\tensorflow\python\keras\engine\training.py in train_function(iterator)
    853       def train_function(iterator):
    854         """Runs a training execution with one step."""
--> 855         return step_function(self, iterator)
    856
    857     else:

~\anaconda3\lib\site-packages\tensorflow\python\keras\engine\training.py in step_function(model, iterator)
    843
```

```
    844       data = next(iterator)
--> 845       outputs = model.distribute_strategy.run(run_step, args=(data,))
    846       outputs = reduce_per_replica(
    847           outputs, self.distribute_strategy, reduction='first')

~\anaconda3\lib\site-packages\tensorflow\python\distribute\distribute_lib.py in run(***failed resolving arguments***)
   1283       fn = autograph.tf_convert(
   1284           fn, autograph_ctx.control_status_ctx(), convert_by_default=False)
-> 1285       return self._extended.call_for_each_replica(fn, args=args, kwargs=kwargs)
   1286
   1287   def reduce(self, reduce_op, value, axis):

~\anaconda3\lib\site-packages\tensorflow\python\distribute\distribute_lib.py in call_for_each_replica(self, fn, args, kwargs)
   2831       kwargs = {}
   2832     with self._container_strategy().scope():
-> 2833       return self._call_for_each_replica(fn, args, kwargs)
   2834
   2835   def _call_for_each_replica(self, fn, args, kwargs):

~\anaconda3\lib\site-packages\tensorflow\python\distribute\distribute_lib.py in _call_for_each_replica(self, fn, args, kwargs)
   3606   def _call_for_each_replica(self, fn, args, kwargs):
   3607     with ReplicaContext(self._container_strategy(), replica_id_in_sync_group=0):
-> 3608       return fn(*args, **kwargs)
   3609
   3610   def _reduce_to(self, reduce_op, value, destinations, options):

~\anaconda3\lib\site-packages\tensorflow\python\autograph\impl\api.py in wrapper(*args, **kwargs)
    595   def wrapper(*args, **kwargs):
    596     with ag_ctx.ControlStatusCtx(status=ag_ctx.Status.UNSPECIFIED):
--> 597       return func(*args, **kwargs)
    598
    599   if inspect.isfunction(func) or inspect.ismethod(func):

~\anaconda3\lib\site-packages\tensorflow\python\keras\engine\training.py in run_step(data)
    836
    837       def run_step(data):
--> 838         outputs = model.train_step(data)
    839         # Ensure counter is updated only if `train_step` succeeds.
    840         with ops.control_dependencies(_minimum_control_deps(outputs)):

~\anaconda3\lib\site-packages\tensorflow\python\keras\engine\training.py in train_step(self, data)
    798     # Run backwards pass.
    799     self.optimizer.minimize(loss, self.trainable_variables, tape=tape)
--> 800     self.compiled_metrics.update_state(y, y_pred, sample_weight)
```

```
    801     # Collect metrics to return
    802     return_metrics = {}

~\anaconda3\lib\site-packages\tensorflow\python\keras\engine\compile_utils.py in update_state(self, y_true, y_pred, sample_weight)
    458         if metric_obj is None:
    459           continue
--> 460         metric_obj.update_state(y_t, y_p, sample_weight=mask)
    461
    462       for weighted_metric_obj in weighted_metric_objs:

~\anaconda3\lib\site-packages\tensorflow\python\keras\utils\metrics_utils.py in decorated(metric_obj, *args, **kwargs)
     84
     85     with tf_utils.graph_context_for_symbolic_tensors(*args, **kwargs):
---> 86       update_op = update_state_fn(*args, **kwargs)
     87     if update_op is not None:  # update_op will be None in eager execution.
     88       metric_obj.add_update(update_op)

~\anaconda3\lib\site-packages\tensorflow\python\keras\metrics.py in update_state_fn(*args, **kwargs)
    175         control_status = ag_ctx.control_status_ctx()
    176         ag_update_state = autograph.tf_convert(obj_update_state, control_status)
--> 177         return ag_update_state(*args, **kwargs)
    178     else:
    179       if isinstance(obj.update_state, def_function.Function):

~\anaconda3\lib\site-packages\tensorflow\python\autograph\impl\api.py in wrapper(*args, **kwargs)
    690       try:
    691         with conversion_ctx:
--> 692           return converted_call(f, args, kwargs, options=options)
    693       except Exception as e:  # pylint:disable=broad-except
    694         if hasattr(e, 'ag_error_metadata'):

~\anaconda3\lib\site-packages\tensorflow\python\autograph\impl\api.py in converted_call(f, args, kwargs, caller_fn_scope, options)
    334   if conversion.is_in_allowlist_cache(f, options):
    335     logging.log(2, 'Allowlisted %s: from cache', f)
--> 336     return _call_unconverted(f, args, kwargs, options, False)
    337
    338   if ag_ctx.control_status_ctx().status == ag_ctx.Status.DISABLED:

~\anaconda3\lib\site-packages\tensorflow\python\autograph\impl\api.py in _call_unconverted(f, args, kwargs, options, update_cache)
    461
    462   if kwargs is not None:
--> 463     return f(*args, **kwargs)
    464   return f(*args)
    465
```

```
~\anaconda3\lib\site-packages\tensorflow\python\keras\metrics.py in update_state(self, y_true, y_pred, sample_weight)
    662
    663     ag_fn = autograph.tf_convert(self._fn, ag_ctx.control_status_ctx())
--> 664     matches = ag_fn(y_true, y_pred, **self._fn_kwargs)
    665     return super(MeanMetricWrapper, self).update_state(
    666         matches, sample_weight=sample_weight)

~\anaconda3\lib\site-packages\tensorflow\python\autograph\impl\api.py in wrapper(*args, **kwargs)
    690       try:
    691         with conversion_ctx:
--> 692           return converted_call(f, args, kwargs, options=options)
    693       except Exception as e:  # pylint:disable=broad-except
    694         if hasattr(e, 'ag_error_metadata'):

~\anaconda3\lib\site-packages\tensorflow\python\autograph\impl\api.py in converted_call(f, args, kwargs, caller_fn_scope, options)
    442     try:
    443       if kwargs is not None:
--> 444         result = converted_f(*effective_args, **kwargs)
    445       else:
    446         result = converted_f(*effective_args)

~\AppData\Local\Temp\tmpu_hw26xc.py in tf__model_acc(y_true, y_pred)
     45                 true = ag__.Undefined('true')
     46                 i = ag__.Undefined('i')
---> 47                 ag__.for_stmt(ag__.converted_call(ag__.ld(range), (ag__.ld(size),), None, fscope), None, loop_body, get_st
ate_1, set_state_1, ('acc',), {'iterate_names': 'i'})
     48                 try:
     49                     do_return = True

~\anaconda3\lib\site-packages\tensorflow\python\autograph\operators\control_flow.py in for_stmt(iter_, extra_test, body, get_stat
e, set_state, symbol_names, opts)
    441
    442   else:
--> 443     _py_for_stmt(iter_, extra_test, body, None, None)
    444
    445

~\anaconda3\lib\site-packages\tensorflow\python\autograph\operators\control_flow.py in _py_for_stmt(***failed resolving arguments*
**)
    470   else:
    471     for target in iter_:
--> 472       body(target)
    473
```

```
    474

~\anaconda3\lib\site-packages\tensorflow\python\autograph\operators\control_flow.py in protected_body(protected_iter)
    456     original_body = body
    457     def protected_body(protected_iter):
--> 458       original_body(protected_iter)
    459       after_iteration()
    460       before_iteration()

~\AppData\Local\Temp\tmpu_hw26xc.py in loop_body(itr)
     23                     true = ag__.ld(y_true)[ag__.ld(i)]
     24                     pred = ag__.ld(y_pred)[ag__.ld(i)]
---> 25                     index_max = ag__.converted_call(ag__.converted_call(ag__.ld(tf).argmax, (ag__.ld(pred),), None, fscope
).numpy, (), None, fscope)
     26
     27                     def get_state():

~\anaconda3\lib\site-packages\tensorflow\python\autograph\impl\api.py in converted_call(f, args, kwargs, caller_fn_scope, options)
    334   if conversion.is_in_allowlist_cache(f, options):
    335     logging.log(2, 'Allowlisted %s: from cache', f)
--> 336     return _call_unconverted(f, args, kwargs, options, False)
    337
    338   if ag_ctx.control_status_ctx().status == ag_ctx.Status.DISABLED:

~\anaconda3\lib\site-packages\tensorflow\python\autograph\impl\api.py in _call_unconverted(f, args, kwargs, options, update_cache)
    462   if kwargs is not None:
    463     return f(*args, **kwargs)
--> 464   return f(*args)
    465
    466

~\anaconda3\lib\site-packages\tensorflow\python\framework\ops.py in numpy(self)
   1092     """
   1093     # TODO(slebedev): Consider avoiding a copy for non-CPU or remote tensors.
-> 1094     maybe_arr = self._numpy()  # pylint: disable=protected-access
   1095     return maybe_arr.copy() if isinstance(maybe_arr, np.ndarray) else maybe_arr
   1096

~\anaconda3\lib\site-packages\tensorflow\python\framework\ops.py in _numpy(self)
   1058   def _numpy(self):
   1059     try:
-> 1060       return self._numpy_internal()
   1061     except core._NotOkStatusException as e:  # pylint: disable=protected-access
   1062       six.raise_from(core._status_to_exception(e.code, e.message), None)  # pylint: disable=protected-access
```

```
KeyboardInterrupt:
```

In [ ]:

```
base_model = vgg16.VGG16(include_top=False,
                         weights="imagenet",
                         input_shape=(48,48,3))
base_model.trainable=True
model = Sequential([
    base_model,
    layers.GlobalAveragePooling2D(),
    layers.Dense(4096, activation='relu'),
    layers.Dense(4096, activation='relu'),
    layers.Dense(emotions_count, activation='softmax'),
])

model.compile(optimizer='sgd',
              loss='mse',
              metrics = [model_acc])

model.fit(x=training_images,
          y=training_emotions,
          batch_size=32,
          epochs=20,
          validation_data=(test_images, test_emotions))
```