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

    Drive already mounted at /content/drive; to attempt to forcibly remount, call drive.mount("/content/drive", force_remount=True).

```
import tensorflow as tf
import os
import matplotlib.pyplot as plt
import re
import shutil
import numpy as np
```

```
gpus = tf.config.experimental.list_logical_devices('GPU')
strategy = tf.distribute.MirroredStrategy(gpus)
```

```
raw_data = "/content/drive/MyDrive/archive/flowers"
```

```
flowers = ['tulip', 'orchids', 'peonies', 'hydrangeas', 'lilies', 'gardenias', 'garden_roses', 'daisies', 'hibiscus', 'bougainvillea']

for flower in flowers:
    directory = flower
    if not os.path.exists(directory):
        os.mkdir(directory)
    else:
        print(f"The directory '{directory}' already exists.")
```

```
import os

# รายชื่อของดอกไม้
flowers = ['tulip', 'orchids', 'peonies', 'hydrangeas', 'lilies', 'gardenias', 'garden_roses', 'daisies', 'hibiscus', 'bougainvillea']

for flower in flowers:
    folder_path = f"/content/drive/MyDrive/archive/flowers/{flower}"  # เปลี่ยน "ระบุที่อยู่ของโฟลเดอร์" เป็นที่อยู่ที่คุณต้องการ
    os.makedirs(folder_path, exist_ok=True)
    print(f"Flowers {flower} ที่ {folder_path}")
```

    Flowers tulip ที่ /content/drive/MyDrive/archive/flowers/tulip
    Flowers orchids ที่ /content/drive/MyDrive/archive/flowers/orchids
    Flowers peonies ที่ /content/drive/MyDrive/archive/flowers/peonies
    Flowers hydrangeas ที่ /content/drive/MyDrive/archive/flowers/hydrangeas
    Flowers lilies ที่ /content/drive/MyDrive/archive/flowers/lilies
    Flowers gardenias ที่ /content/drive/MyDrive/archive/flowers/gardenias
    Flowers garden_roses ที่ /content/drive/MyDrive/archive/flowers/garden_roses
    Flowers daisies ที่ /content/drive/MyDrive/archive/flowers/daisies
    Flowers hibiscus ที่ /content/drive/MyDrive/archive/flowers/hibiscus
    Flowers bougainvillea ที่ /content/drive/MyDrive/archive/flowers/bougainvillea

```
train_data_dir = '/content/drive/MyDrive/archive/flowers'
```

```
from tensorflow.keras.models import Sequential
from tensorflow.keras.layers import Dense

# สร้างโมเดล Sequential
model = Sequential()

# เพิ่มเลเยอร์
model.add(Dense(units=64, activation='relu', input_dim=100))
model.add(Dense(units=10, activation='softmax'))

# คอมไพล์โมเดล
model.compile(optimizer='adam', loss='categorical_crossentropy', metrics=['accuracy'])
```

```
from tensorflow.keras.preprocessing.image import ImageDataGenerator

# สร้างชุดข้อมูลการฝึก
train_datagen = ImageDataGenerator(rescale=1./255, shear_range=0.2, zoom_range=0.2, horizontal_flip=True)

train_generator = train_datagen.flow_from_directory(
    train_data_dir,
    target_size=(img_width, img_height),
    batch_size=batch_size,
    class_mode='categorical'
```

```
)
```

Found 0 images belonging to 10 classes.

```
for i in flowers:
    print(os.listdir(f"/content/Flowers/{i}")[:10])
```

```
import cv2
import imghdr
```

```
import os
import re
import shutil

# ระบุที่อยู่ของโฟลเดอร์ที่มีรูปภาพ
raw_data_folder = '/content/drive/MyDrive/archive/flowers'

# สร้างโฟลเดอร์ปลายทาง
destination_base_folder = 'Flowers'

# ลูปที่ดึงข้อมูลจากโฟลเดอร์ raw_data_folder
for folder_name in os.listdir(raw_data_folder):
    # สร้างโฟลเดอร์ปลายทางขึ้นมาใหม่โดยใช้ชื่อโฟลเดอร์จากชื่อโฟลเดอร์เดิม
    destination_folder = os.path.join(destination_base_folder, folder_name)
    os.makedirs(destination_folder, exist_ok=True)

    # ลูปที่ดึงข้อมูลจากโฟลเดอร์รูปภาพ
    for file_name in os.listdir(os.path.join(raw_data_folder, flower)):
        # ทำการคัดลอกไฟล์ภาพไปยังโฟลเดอร์ปลายทาง
        source_path = os.path.join(raw_data_folder, folder_name, file_name)
        destination_path = os.path.join(destination_folder, file_name)
        shutil.copyfile(source_path, destination_path)

print("เสร็จสิ้นการคัดลอกภาพไปยังโฟลเดอร์ปลายทาง")
```

เสร็จสิ้นการคัดลอกภาพไปยังโฟลเดอร์ปลายทาง

```
img = cv2.imread("/content/drive/MyDrive/archive/flowers/daisies_00040.jpg")
plt.imshow(img)
plt.show()
```


```
plt.imshow(cv2.cvtColor(img, cv2.COLOR_BGR2RGB))
```

```
<matplotlib.image.AxesImage at 0x78eef3c24dc0>
```


```
tf.data.Dataset??
```

```
data = tf.keras.utils.image_dataset_from_directory('/content/drive/MyDrive/archive/')
```

```
Found 733 files belonging to 1 classes.
```

```
data_iterator = data.as_numpy_iterator()
```

```
batch = data_iterator.next()
```

```
# สมมติว่า labels เป็นลิสต์ของชื่อโฟลเดอร์
labels = data.class_names

# ให้ผลลัพธ์เป็นสตริงทั้งหมด
labels_str = ', '.join(labels)
print(labels_str)
```

```
flowers
```

```
import os

# ระบุที่อยู่ของโฟลเดอร์ที่ต้องการแสดง
folder_path = '/content/drive/MyDrive/archive/flowers/'

# ใช้ os.listdir() เพื่อดึงลิสต์ของไฟล์ในโฟลเดอร์
file_list = os.listdir(folder_path)

# แสดงลิสต์ของไฟล์
for file in file_list:
    print(file)
```

```
tulip_00014.jpg
tulip_00054.jpg
tulip_00051.jpg
tulip_00033.jpg
tulip_00037.jpg
tulip_00041.jpg
tulip_00029.jpg
tulip_00066.jpg
tulip_00013.jpg
tulip_00021.jpg
tulip_00012.jpg
tulip_00015.jpg
tulip_00080.jpg
tulip_00074.jpg
tulip_00075.jpg
tulip_00077.jpg
tulip_00071.jpg
tulip_00084.jpg
tulip_00070.jpg
tulip_00083.jpg
tulip_00081.jpg
tulip_00079.jpg
tulip_00076.jpg
tulip
orchids
peonies
hydrangeas
lilies
gardenias
garden_roses
daisies
hibiscus
bougainvillea
```

```
batch[1]
```

```
array([0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0,
       0, 0, 0, 0, 0, 0, 0, 0, 0], dtype=int32)
```

```
fig, ax = plt.subplots(ncols= 4, figsize= (20, 20))
for idx, img in enumerate(batch[0][:4]):
    ax[idx].imshow(img.astype(int))
    ax[idx].title.set_text(labels[batch[1][idx]])
```


```
data = data.map(lambda x,y : (x/255, y))
```

```
batch = data.as_numpy_iterator().next()
```

```
print(batch[0].min(), batch[0].max())
```

```
0.0 1.0
```

```
print(f"Data is split into {len(data)} batches and Each batch has {len(batch[0])} images.")
```

```
Data is split into 23 batches and Each batch has 32 images.
```

```
train_size = int(len(data)* .7)
val_size = int(len(data)* .2) + 1
test_size = int(len(data)* .1)
```

```
train_size + val_size + test_size
```

```
23
```

```
train = data.take(train_size)
val = data.skip(train_size).take(val_size)
test = data.skip(train_size + val_size).take(test_size)
```

```
from tensorflow.keras.models import Sequential
from tensorflow.keras.layers import Conv2D, MaxPooling2D, Dense, Flatten
```

```
with strategy.scope():
    model = Sequential()

    model.add(Conv2D(filters= 16, kernel_size= (3,3), strides= 1, activation= 'relu', input_shape= (256, 256, 3)))
    model.add(MaxPooling2D())

    model.add(Conv2D(filters= 32, kernel_size= (3,3), strides= 1, activation= 'relu'))
    model.add(MaxPooling2D())

    model.add(Conv2D(filters= 16, kernel_size= (3,3), strides= 1, activation= 'relu'))
    model.add(MaxPooling2D())

    model.add(Flatten())

    model.add(Dense(units= 256, activation= 'relu'))
    model.add(Dense(units= 11, activation= 'softmax'))

    model.compile(optimizer= 'adam', loss= tf.losses.sparse_categorical_crossentropy, metrics = ['accuracy'] )
```

```
model.summary()
```

```
Model: "sequential_8"
_________________________________________________________________
 Layer (type)                Output Shape              Param #
=================================================================
 conv2d (Conv2D)             (None, 254, 254, 16)      448

 max_pooling2d (MaxPooling2  (None, 127, 127, 16)      0
 D)

 conv2d_1 (Conv2D)           (None, 125, 125, 32)      4640

 max_pooling2d_1 (MaxPoolin  (None, 62, 62, 32)        0
 g2D)

 conv2d_2 (Conv2D)           (None, 60, 60, 16)        4624

 max_pooling2d_2 (MaxPoolin  (None, 30, 30, 16)        0
 g2D)

 flatten (Flatten)           (None, 14400)             0

 dense_10 (Dense)            (None, 256)               3686656

 dense_11 (Dense)            (None, 11)                2827

=================================================================
Total params: 3699195 (14.11 MB)
Trainable params: 3699195 (14.11 MB)
Non-trainable params: 0 (0.00 Byte)
_________________________________________________________________
```

```
logdir = 'logs'
tensorboard_callback = tf.keras.callbacks.TensorBoard(log_dir = logdir)
```

```
%%time
with strategy.scope():
    hist = model.fit(train, epochs= 20, validation_data= val, callbacks= [tensorboard_callback])
```

```
Epoch 1/20
16/16 [==============================] - 71s 4s/step - loss: 0.1438 - accuracy: 0.9902 - val_loss: 0.0000e+00 - val_accuracy: 1.0000
Epoch 2/20
16/16 [==============================] - 66s 4s/step - loss: 0.0000e+00 - accuracy: 1.0000 - val_loss: 0.0000e+00 - val_accuracy: 1.0000
Epoch 3/20
16/16 [==============================] - 65s 3s/step - loss: 0.0000e+00 - accuracy: 1.0000 - val_loss: 0.0000e+00 - val_accuracy: 1.0000
Epoch 4/20
16/16 [==============================] - 65s 4s/step - loss: 0.0000e+00 - accuracy: 1.0000 - val_loss: 0.0000e+00 - val_accuracy: 1.0000
Epoch 5/20
16/16 [==============================] - 71s 4s/step - loss: 0.0000e+00 - accuracy: 1.0000 - val_loss: 0.0000e+00 - val_accuracy: 1.0000
Epoch 6/20
16/16 [==============================] - 85s 5s/step - loss: 0.0000e+00 - accuracy: 1.0000 - val_loss: 0.0000e+00 - val_accuracy: 1.0000
Epoch 7/20
16/16 [==============================] - 64s 4s/step - loss: 0.0000e+00 - accuracy: 1.0000 - val_loss: 0.0000e+00 - val_accuracy: 1.0000
```

```
Epoch 8/20
16/16 [==============================] - 64s 4s/step - loss: 0.0000e+00 - accuracy: 1.0000 - val_loss: 0.0000e+00 - val_accuracy: 1.0000
Epoch 9/20
16/16 [==============================] - 68s 4s/step - loss: 0.0000e+00 - accuracy: 1.0000 - val_loss: 0.0000e+00 - val_accuracy: 1.0000
Epoch 10/20
16/16 [==============================] - 66s 4s/step - loss: 0.0000e+00 - accuracy: 1.0000 - val_loss: 0.0000e+00 - val_accuracy: 1.0000
Epoch 11/20
16/16 [==============================] - 64s 4s/step - loss: 0.0000e+00 - accuracy: 1.0000 - val_loss: 0.0000e+00 - val_accuracy: 1.0000
Epoch 12/20
16/16 [==============================] - 64s 4s/step - loss: 0.0000e+00 - accuracy: 1.0000 - val_loss: 0.0000e+00 - val_accuracy: 1.0000
Epoch 13/20
16/16 [==============================] - 63s 4s/step - loss: 0.0000e+00 - accuracy: 1.0000 - val_loss: 0.0000e+00 - val_accuracy: 1.0000
Epoch 14/20
16/16 [==============================] - 84s 5s/step - loss: 0.0000e+00 - accuracy: 1.0000 - val_loss: 0.0000e+00 - val_accuracy: 1.0000
Epoch 15/20
16/16 [==============================] - 67s 4s/step - loss: 0.0000e+00 - accuracy: 1.0000 - val_loss: 0.0000e+00 - val_accuracy: 1.0000
Epoch 16/20
16/16 [==============================] - 61s 4s/step - loss: 0.0000e+00 - accuracy: 1.0000 - val_loss: 0.0000e+00 - val_accuracy: 1.0000
Epoch 17/20
16/16 [==============================] - 69s 4s/step - loss: 0.0000e+00 - accuracy: 1.0000 - val_loss: 0.0000e+00 - val_accuracy: 1.0000
Epoch 18/20
16/16 [==============================] - 66s 4s/step - loss: 0.0000e+00 - accuracy: 1.0000 - val_loss: 0.0000e+00 - val_accuracy: 1.0000
Epoch 19/20
16/16 [==============================] - 64s 4s/step - loss: 0.0000e+00 - accuracy: 1.0000 - val_loss: 0.0000e+00 - val_accuracy: 1.0000
Epoch 20/20
16/16 [==============================] - 64s 4s/step - loss: 0.0000e+00 - accuracy: 1.0000 - val_loss: 0.0000e+00 - val_accuracy: 1.0000
CPU times: user 26min 27s, sys: 3min 32s, total: 30min
Wall time: 28min 47s
```

```
pip install pyswarms
```

```
Collecting pyswarms
  Downloading pyswarms-1.3.0-py2.py3-none-any.whl (104 kB)
     ━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━ 104.1/104.1 kB 2.5 MB/s eta 0:00:00
Requirement already satisfied: scipy in /usr/local/lib/python3.10/dist-packages (from pyswarms) (1.11.4)
Requirement already satisfied: numpy in /usr/local/lib/python3.10/dist-packages (from pyswarms) (1.25.2)
Requirement already satisfied: matplotlib>=1.3.1 in /usr/local/lib/python3.10/dist-packages (from pyswarms) (3.7.1)
Requirement already satisfied: attrs in /usr/local/lib/python3.10/dist-packages (from pyswarms) (23.2.0)
Requirement already satisfied: tqdm in /usr/local/lib/python3.10/dist-packages (from pyswarms) (4.66.2)
Requirement already satisfied: future in /usr/local/lib/python3.10/dist-packages (from pyswarms) (0.18.3)
Requirement already satisfied: pyyaml in /usr/local/lib/python3.10/dist-packages (from pyswarms) (6.0.1)
Requirement already satisfied: contourpy>=1.0.1 in /usr/local/lib/python3.10/dist-packages (from matplotlib>=1.3.1->pyswarms) (1.2.0)
Requirement already satisfied: cycler>=0.10 in /usr/local/lib/python3.10/dist-packages (from matplotlib>=1.3.1->pyswarms) (0.12.1)
Requirement already satisfied: fonttools>=4.22.0 in /usr/local/lib/python3.10/dist-packages (from matplotlib>=1.3.1->pyswarms) (4.49.0)
Requirement already satisfied: kiwisolver>=1.0.1 in /usr/local/lib/python3.10/dist-packages (from matplotlib>=1.3.1->pyswarms) (1.4.5)
Requirement already satisfied: packaging>=20.0 in /usr/local/lib/python3.10/dist-packages (from matplotlib>=1.3.1->pyswarms) (23.2)
Requirement already satisfied: pillow>=6.2.0 in /usr/local/lib/python3.10/dist-packages (from matplotlib>=1.3.1->pyswarms) (9.4.0)
Requirement already satisfied: pyparsing>=2.3.1 in /usr/local/lib/python3.10/dist-packages (from matplotlib>=1.3.1->pyswarms) (3.1.2)
Requirement already satisfied: python-dateutil>=2.7 in /usr/local/lib/python3.10/dist-packages (from matplotlib>=1.3.1->pyswarms) (2.8.2)
Requirement already satisfied: six>=1.5 in /usr/local/lib/python3.10/dist-packages (from python-dateutil>=2.7->matplotlib>=1.3.1->pyswarms) (1.16.0)
Installing collected packages: pyswarms
Successfully installed pyswarms-1.3.0
```

```
import pyswarms as ps
```

```
import pyswarms as ps
from pyswarms.utils.functions import single_obj as fx
# Set-up hyperparameters
options = {'c1': 0.5, 'c2': 0.3, 'w':0.9}
# Call instance of PSO
optimizer = ps.single.GlobalBestPSO(n_particles=10, dimensions=2, options=options)
# Perform optimization
best_cost, best_pos = optimizer.optimize(fx.sphere, iters=100)
```

```
2024-03-12 13:21:10,355 - pyswarms.single.global_best - INFO - Optimize for 100 iters with {'c1': 0.5, 'c2': 0.3, 'w': 0.9}
pyswarms.single.global_best: 100%|██████████|100/100, best_cost=3.27e-7
2024-03-12 13:21:10,927 - pyswarms.single.global_best - INFO - Optimization finished | best cost: 3.267342555430617e-07, best pos: [ 0.00034693 -0.00045428]
```

```
optimizer.cost_history
# Obtain the position history
optimizer.pos_history
# Obtain the velocity history
optimizer.velocity_history
```

```
       [-0.00098943, -0.00607862],
       [-0.07347618,  0.00281994],
       [-0.00233653, -0.00548867],
       [-0.00028809, -0.01332976],
       [-0.00163603,  0.00232189],
       [ 0.0001481 ,  0.00398642],
       [-0.00081117,  0.00182075]]),
 array([[ 0.00458535,  0.00039868],
       [ 0.00372251, -0.0026566 ],
       [-0.00201755,  0.01896387],
       [-0.00160385,  0.01987966],
       [-0.05830731,  0.00291763],
       [-0.001361  , -0.00329347],
       [-0.00045562, -0.00502117],
       [ 0.01269335,  0.00393791],
       [ 0.00013396,  0.00397663],
       [-0.00083765,  0.00969661]]),
 array([[-5.89893050e-04,  2.48864147e-04],
       [ 9.47727746e-03, -1.36837113e-04],
       [ 2.26858660e-05,  1.62685588e-02],
       [-4.41562220e-04,  2.77291358e-02],
       [ 3.32701810e-03,  3.34092871e-03],
       [ 1.00387330e-03,  1.37222488e-03],
       [-5.48971370e-04,  2.14177510e-03],
       [ 2.41789786e-02,  3.91062558e-03],
       [ 9.93718390e-05,  2.79321675e-03],
       [-4.96987192e-04,  1.06821648e-02]]),
 array([[-2.63735499e-03, -1.08658353e-05],
       [ 1.73172240e-02,  2.87315033e-03],
       [ 7.41058694e-04,  1.10238989e-05],
       [ 1.86253256e-03,  1.79993246e-02],
       [ 3.68645134e-02,  1.29085990e-03],
       [ 2.43138847e-03,  4.41227300e-03],
       [ 5.00514755e-05,  6.77743478e-03],
       [ 1.81560693e-02,  1.57340730e-03],
       [-2.86373972e-06,  8.00377349e-04],
       [-1.12844990e-04,  6.95960007e-03]]),
 array([[-4.46309042e-03, -2.17987679e-04],
       [ 1.87911395e-02,  3.54859042e-03],
       [ 1.42610223e-03, -1.39053487e-02],
       [ 2.87970683e-03,  4.81808071e-03],
       [ 6.38673186e-02,  1.19044340e-04],
       [ 4.19174545e-03,  5.44109552e-03],
       [ 2.18647760e-04,  1.02894182e-02],
       [ 6.90223955e-03, -7.47180050e-04],
       [-1.02987445e-04, -9.64180766e-04],
       [ 4.50673111e-04,  4.13277078e-05]])]
```

```
optimizer.mean_pbest_history
```

```
9.361090823895291e-05,
9.163873460388231e-05,
8.832296818807713e-05,
8.071887939053134e-05,
8.071887939053134e-05,
7.852101914923592e-05,
7.852101914923592e-05,
7.71748610904789e-05,
6.587054293244517e-05,
6.566233252243985e-05,
6.566233252243985e-05,
6.566233252243985e-05,
6.566233252243985e-05,
6.470737227753217e-05,
6.470737227753217e-05,
6.470737227753217e-05,
6.470737227753217e-05,
3.888172614899696e-05]
```

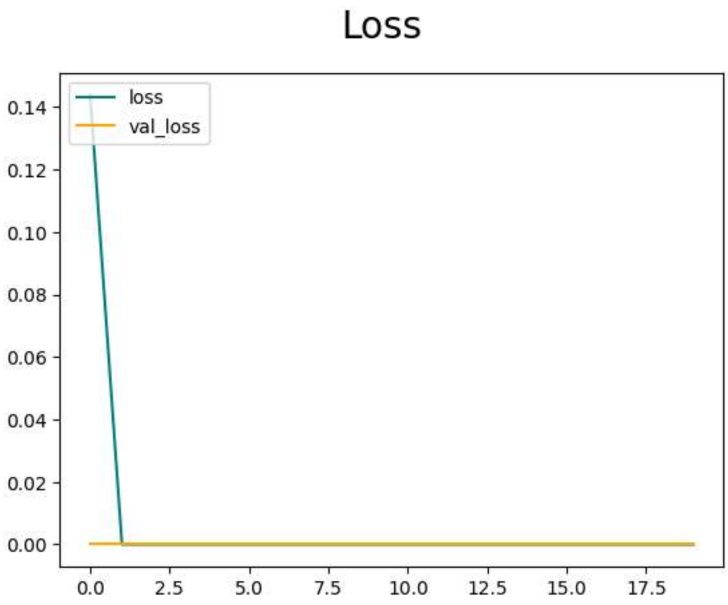
optimizer.mean_neighbor_history

```
0.00026333830329294957,
0.00026333830329294957,
0.00011490351568525205,
0.00011490351568525205,
0.00011490351568525205,
0.00011490351568525205,
0.00011490351568525205,
0.00011490351568525205,
0.00011490351568525205,
0.00011490351568525205,
0.00011490351568525205,
0.00011490351568525205,
0.00011490351568525205,
7.503546844105037e-05,
7.503546844105037e-05,
4.834955810808699e-05,
4.834955810808699e-05,
4.834955810808699e-05,
4.834955810808699e-05,
4.412325288484706e-05,
4.412325288484706e-05,
4.412325288484706e-05,
4.412325288484706e-05,
2.990780064915466e-06,
2.990780064915466e-06,
2.990780064915466e-06,
2.990780064915466e-06,
2.990780064915466e-06,
2.990780064915466e-06,
2.990780064915466e-06,
2.990780064915466e-06,
2.990780064915466e-06,
2.990780064915466e-06,
4.4305812427182724e-07,
4.4305812427182724e-07,
4.4305812427182724e-07,
4.4305812427182724e-07,
4.4305812427182724e-07,
4.4305812427182724e-07,
4.4305812427182724e-07,
4.4305812427182724e-07,
4.4305812427182724e-07,
4.4305812427182724e-07,
4.4305812427182724e-07,
4.4305812427182724e-07,
4.4305812427182724e-07,
4.4305812427182724e-07,
4.4305812427182724e-07,
4.4305812427182724e-07,
3.267342555430617e-07,
3.267342555430617e-07,
3.267342555430617e-07,
3.267342555430617e-07,
3.267342555430617e-07,
3.267342555430617e-07,
3.267342555430617e-07,
3.267342555430617e-07,
3.267342555430617e-07]
```

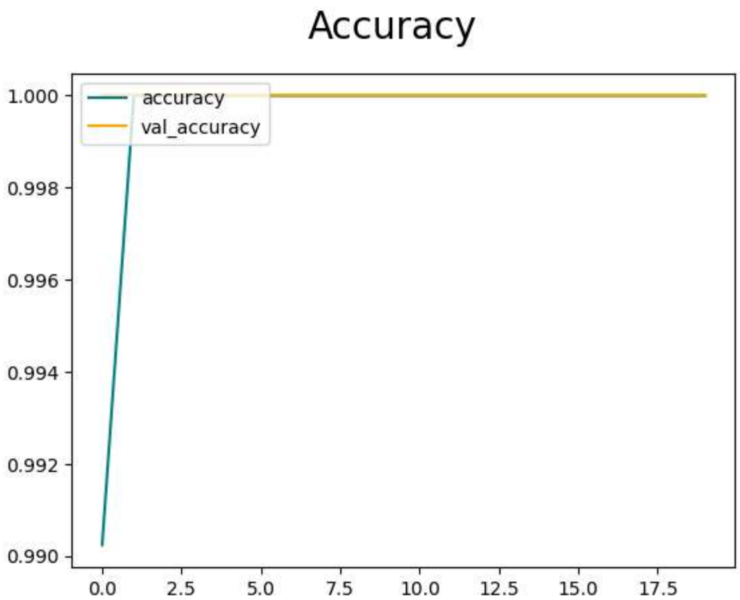
```
from pyswarms.utils.search import RandomSearch
from pyswarms.utils.functions import single_obj as fx

# Set-up choices for the parameters
options = {
    'c1': (1,5),
    'c2': (6,10),
    'w': (2,5),
    'k': (11, 15),
    'p': 1
}

# Create a RandomSearch object
# n_selection_iters is the number of iterations to run the searcher
# iters is the number of iterations to run the optimizer
g = RandomSearch(ps.single.LocalBestPSO, n_particles=40,
            dimensions=20, options=options, objective_func=fx.sphere,
            iters=10, n_selection_iters=100)

best_score, best_options = g.search()
```

```
 0.44628679 0.26609265 0.17750266 0.78799271 0.36581474 0.21714182
 0.44361822 0.4278085  0.12984199 0.21755224 0.50600618 0.15336393
 0.33118971 0.70369082]
2024-03-12 13:22:20,582 - pyswarms.single.local_best - INFO - Optimize for 10 iters with {'c1': 4.442242519501345, 'c2': 6.759319465621623, 'w': 2.26173026
pyswarms.single.local_best: 100%|██████████|10/10, best_cost=4.28
2024-03-12 13:22:20,609 - pyswarms.single.local_best - INFO - Optimization finished | best cost: 4.2798714205675, best pos: [0.57657897 0.24749877 0.5503
 0.01983157 0.06797336 0.78669029 0.36022608 0.30155529 0.37785334
 0.89538133 0.6395926  0.57960499 0.18940229 0.56771685 0.05942604
 0.11840609 0.34541483]
2024-03-12 13:22:20,623 - pyswarms.single.local_best - INFO - Optimize for 10 iters with {'c1': 4.625162286082112, 'c2': 8.961305615551971, 'w': 3.84660806
pyswarms.single.local_best: 100%|██████████|10/10, best_cost=4.3
2024-03-12 13:22:20,650 - pyswarms.single.local_best - INFO - Optimization finished | best cost: 4.301011094897623, best pos: [0.45113607 0.49857334 0.08
 0.46163045 0.60480908 0.31757369 0.71876855 0.49393918 0.54601451
 0.41189365 0.85509883 0.00794443 0.33451929 0.04980575 0.36179531
 0.31172385 0.5010991 ]
2024-03-12 13:22:20,661 - pyswarms.single.local_best - INFO - Optimize for 10 iters with {'c1': 1.430525577100605, 'c2': 8.782438174426128, 'w': 3.46451716
pyswarms.single.local_best: 100%|██████████|10/10, best_cost=4.15
2024-03-12 13:22:20,686 - pyswarms.single.local_best - INFO - Optimization finished | best cost: 4.1494303107121056, best pos: [0.02319298 0.46669347 0.3
 0.0713255  0.96402351 0.14084763 0.62547535 0.33750515 0.28971062
 0.64171337 0.53358643 0.61916328 0.46338873 0.29267837 0.33355703
 0.73974708 0.3763702 ]
2024-03-12 13:22:20,697 - pyswarms.single.local_best - INFO - Optimize for 10 iters with {'c1': 4.068609354312912, 'c2': 7.512133859733403, 'w': 2.7460024
pyswarms.single.local_best: 100%|██████████|10/10, best_cost=4
2024-03-12 13:22:20,729 - pyswarms.single.local_best - INFO - Optimization finished | best cost: 4.000509896362768, best pos: [0.38878316 0.39361592 0.03
 0.49289045 0.12283039 0.42249425 0.69125061 0.2940199  0.83822202
 0.53194493 0.38110238 0.60201205 0.06747217 0.5729236  0.53939808
 0.07893549 0.2152394 ]
2024-03-12 13:22:20,741 - pyswarms.single.local_best - INFO - Optimize for 10 iters with {'c1': 2.981743007347104, 'c2': 9.42823974419204, 'w': 3.69534706
pyswarms.single.local_best: 100%|██████████|10/10, best_cost=2.49
2024-03-12 13:22:20,769 - pyswarms.single.local_best - INFO - Optimization finished | best cost: 2.4870802472577145, best pos: [0.10261379 0.48253914 0.2
 0.3256612  0.38711281 0.28830176 0.30586556 0.06028609 0.40474672
 0.35270474 0.62177887 0.23517612 0.39157667 0.03443529 0.17713097
 0.03589566 0.60707267]
2024-03-12 13:22:20,780 - pyswarms.single.local_best - INFO - Optimize for 10 iters with {'c1': 2.7011213041968647, 'c2': 6.7812389452123725, 'w': 4.476962
pyswarms.single.local_best: 100%|██████████|10/10, best_cost=4.3
2024-03-12 13:22:20,803 - pyswarms.single.local_best - INFO - Optimization finished | best cost: 4.295444825117076, best pos: [0.09263069 0.65718721 0.22
 0.09838809 0.77778984 0.10953612 0.15475954 0.07911193 0.83261794
 0.28281868 0.44031602 0.06289534 0.58659975 0.73360196 0.034239
 0.04219925 0.00477063]
2024-03-12 13:22:20,816 - pyswarms.single.local_best - INFO - Optimize for 10 iters with {'c1': 2.694106566601279, 'c2': 8.153763541361617, 'w': 2.51584725
pyswarms.single.local_best: 100%|██████████|10/10, best_cost=4.18
2024-03-12 13:22:20,840 - pyswarms.single.local_best - INFO - Optimization finished | best cost: 4.1821881250537745, best pos: [0.1177001  0.09697208 0.0
 0.17716469 0.6825353  0.05865899 0.44985386 0.22534561 0.44151352
 0.15162278 0.77035422 0.90890903 0.23517822 0.38947859 0.19399031
 0.40285463 0.50994645]
2024-03-12 13:22:20,851 - pyswarms.single.local_best - INFO - Optimize for 10 iters with {'c1': 1.8206809741635879, 'c2': 6.707885474398157, 'w': 4.0723807
pyswarms.single.local_best: 100%|██████████|10/10, best_cost=5.02
2024-03-12 13:22:20,876 - pyswarms.single.local_best - INFO - Optimization finished | best cost: 5.018381555405124, best pos: [0.42788465 0.30665488 0.46
 0.90047572 0.19073858 0.02058713 0.61927491 0.50657728 0.58579795
 0.78363317 0.1220423  0.70120264 0.10157251 0.08128927 0.26885672
 0.35809029 0.4123971 ]
2024-03-12 13:22:20,888 - pyswarms.single.local_best - INFO - Optimize for 10 iters with {'c1': 1.5480874549597088, 'c2': 8.371945901847068, 'w': 4.7514967
pyswarms.single.local_best: 100%|██████████|10/10, best_cost=4.46
2024-03-12 13:22:20,914 - pyswarms.single.local_best - INFO - Optimization finished | best cost: 4.460064858800242, best pos: [0.70794646 0.26010413 0.64
 0.19697486 0.09345524 0.00990392 0.52334967 0.19859618 0.03032486
 0.35581695 0.41166721 0.74973154 0.90111987 0.11369253 0.9751315
 0.35385372 0.21886904]
```

```
fig = plt.figure()
plt.plot(hist.history['loss'], color='teal', label='loss')
plt.plot(hist.history['val_loss'], color='orange', label='val_loss')
fig.suptitle('Loss', fontsize=20)
plt.legend(loc='upper left')
plt.show()
```




```
fig = plt.figure()
plt.plot(hist.history['accuracy'], color='teal', label='accuracy')
plt.plot(hist.history['val_accuracy'], color='orange', label='val_accuracy')
fig.suptitle('Accuracy', fontsize=20)
plt.legend(loc='upper left')
plt.show()
```




```
from tensorflow.keras.metrics import CategoricalAccuracy
```

```
acc = CategoricalAccuracy()
```

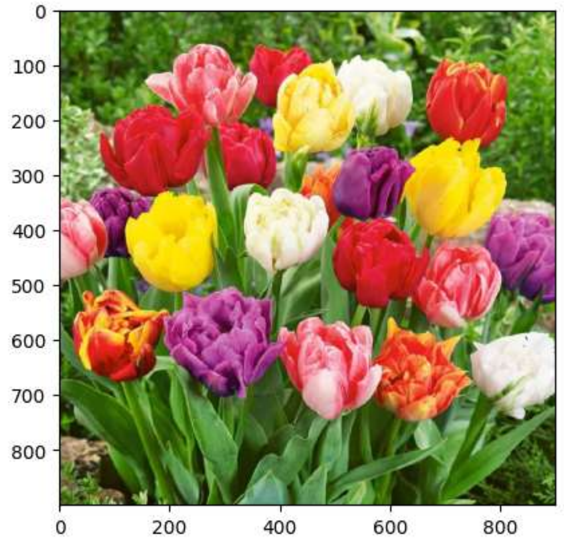
```
for batch in test.as_numpy_iterator():
    X,y = batch
    yhat = model.predict(X)
    yhat = [i.argmax() for i in yhat]
    acc.update_state(y, yhat)
```

```
1/1 [==============================] - 1s 756ms/step
1/1 [==============================] - 1s 651ms/step
```

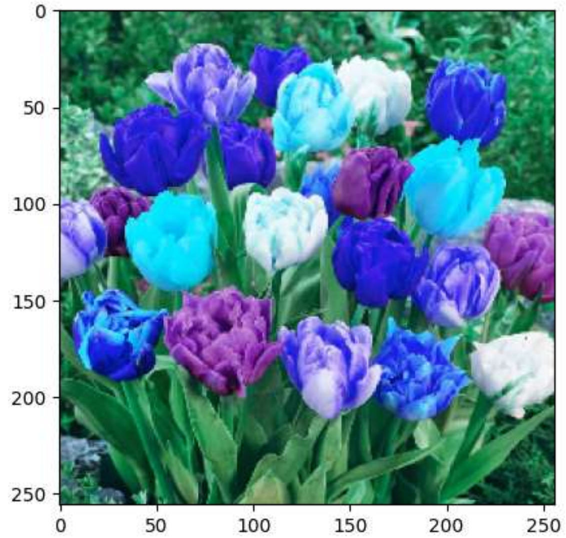
```
print(f'Accuracy: {round(acc.result().numpy() * 100, 2)} %')
```

```
Accuracy: 100.0 %
```

```
img = cv2.imread('/content/drive/MyDrive/archive/flowers/tulip_00022.jpg')
plt.imshow(cv2.cvtColor(img, cv2.COLOR_BGR2RGB))
plt.show()
```




```
resize = tf.image.resize(img, (256, 256))
plt.imshow(resize.numpy().astype(int))
plt.show()
```




```
yhat = model.predict(np.expand_dims(resize/255,0))
```

```
1/1 [==============================] - 0s 228ms/step
```

```
labels[yhat.argmax()]
```

```
'flowers'
```

```
def predict_flower(model, img_path):
    img = cv2.imread(img_path)
    resize = tf.image.resize(img, (256, 256))
    yhat = model.predict(np.expand_dims(resize/255,0))
    return f'{labels[yhat.argmax()]}'
```

```
predict_flower(model, '/content/drive/MyDrive/archive/flowers/hibiscus_00067.jpg')
```

```
1/1 [==============================] - 0s 299ms/step
'flowers'
```

```
from tensorflow.keras.models import load_model
```

```
model.save('multiclass_flower_classifier.h5')
```

```
/usr/local/lib/python3.10/dist-packages/keras/src/engine/training.py:3103: UserWarning: You are saving your model as an HDF5 file via `model.save()`. This file fo
  saving_api.save_model(
```

```
loaded_model = load_model('multiclass_flower_classifier.h5')
```

```
predict_flower(loaded_model, '/content/drive/MyDrive/archive/flowers/hydrangeas_00036.jpg')
```

```
1/1 [==============================] - 0s 117ms/step
'flowers'
```