104 Types of Flower Classification with TPUs

In [1]:
```
import math, re, os
import tensorflow as tf
import numpy as np
from matplotlib import pyplot as plt
from sklearn.metrics import f1_score, precision_score, recall_score, confusion_matrix
print("Tensorflow version " + tf.__version__)
AUTO = tf.data.experimental.AUTOTUNE
```

```
D1022 21:38:22.909670386      15 config.cc:119]                       gRPC EXPERIMENT tcp_frame_size_tuning
OFF (default:OFF)
D1022 21:38:22.909693269      15 config.cc:119]                       gRPC EXPERIMENT tcp_rcv_lowat
OFF (default:OFF)
D1022 21:38:22.909696454      15 config.cc:119]                       gRPC EXPERIMENT peer_state_based_framing
OFF (default:OFF)
D1022 21:38:22.909698924      15 config.cc:119]                       gRPC EXPERIMENT flow_control_fixes
ON  (default:ON)
D1022 21:38:22.909701209      15 config.cc:119]                       gRPC EXPERIMENT memory_pressure_controller
OFF (default:OFF)
D1022 21:38:22.909705062      15 config.cc:119]                       gRPC EXPERIMENT unconstrained_max_quota_buffer_s
ize OFF (default:OFF)
D1022 21:38:22.909707489      15 config.cc:119]                       gRPC EXPERIMENT new_hpack_huffman_decoder
ON  (default:ON)
D1022 21:38:22.909709786      15 config.cc:119]                       gRPC EXPERIMENT event_engine_client
OFF (default:OFF)
D1022 21:38:22.909712017      15 config.cc:119]                       gRPC EXPERIMENT monitoring_experiment
ON  (default:ON)
D1022 21:38:22.909714262      15 config.cc:119]                       gRPC EXPERIMENT promise_based_client_call
OFF (default:OFF)
D1022 21:38:22.909716429      15 config.cc:119]                       gRPC EXPERIMENT free_large_allocator
OFF (default:OFF)
D1022 21:38:22.909718590      15 config.cc:119]                       gRPC EXPERIMENT promise_based_server_call
OFF (default:OFF)
D1022 21:38:22.909720908      15 config.cc:119]                       gRPC EXPERIMENT transport_supplies_client_latenc
y   OFF (default:OFF)
D1022 21:38:22.909723016      15 config.cc:119]                       gRPC EXPERIMENT event_engine_listener
OFF (default:OFF)
I1022 21:38:22.909945444      15 ev_epoll1_linux.cc:122]              grpc epoll fd: 62
D1022 21:38:22.909956923      15 ev_posix.cc:144]                     Using polling engine: epoll1
D1022 21:38:22.909976772      15 dns_resolver_ares.cc:822]            Using ares dns resolver
D1022 21:38:22.922863174      15 lb_policy_registry.cc:46]            registering LB policy factory for "priority_expe
rimental"
D1022 21:38:22.922876805      15 lb_policy_registry.cc:46]            registering LB policy factory for "outlier_detec
tion_experimental"
D1022 21:38:22.922881066      15 lb_policy_registry.cc:46]            registering LB policy factory for "weighted_targ
et_experimental"
D1022 21:38:22.922883800      15 lb_policy_registry.cc:46]            registering LB policy factory for "pick_first"
D1022 21:38:22.922886599      15 lb_policy_registry.cc:46]            registering LB policy factory for "round_robin"
D1022 21:38:22.922889561      15 lb_policy_registry.cc:46]            registering LB policy factory for "weighted_roun
d_robin_experimental"
D1022 21:38:22.922897101      15 lb_policy_registry.cc:46]            registering LB policy factory for "ring_hash_exp
erimental"
D1022 21:38:22.922916472      15 lb_policy_registry.cc:46]            registering LB policy factory for "grpclb"
D1022 21:38:22.922946684      15 lb_policy_registry.cc:46]            registering LB policy factory for "rls_experimen
```

```
tal"
D1022 21:38:22.922962324      15 lb_policy_registry.cc:46]            registering LB policy factory for "xds_cluster_m
anager_experimental"
D1022 21:38:22.922965838      15 lb_policy_registry.cc:46]            registering LB policy factory for "xds_cluster_i
mpl_experimental"
D1022 21:38:22.922968832      15 lb_policy_registry.cc:46]            registering LB policy factory for "cds_experimen
tal"
D1022 21:38:22.922975834      15 lb_policy_registry.cc:46]            registering LB policy factory for "xds_cluster_r
esolver_experimental"
D1022 21:38:22.922978941      15 lb_policy_registry.cc:46]            registering LB policy factory for "xds_override_
host_experimental"
D1022 21:38:22.922981752      15 lb_policy_registry.cc:46]            registering LB policy factory for "xds_wrr_local
ity_experimental"
D1022 21:38:22.922985932      15 certificate_provider_registry.cc:35]  registering certificate provider factory for "fi
le_watcher"
I1022 21:38:22.926493876      15 socket_utils_common_posix.cc:408]    Disabling AF_INET6 sockets because ::1 is not av
ailable.
I1022 21:38:22.940522295     240 socket_utils_common_posix.cc:337]    TCP_USER_TIMEOUT is available. TCP_USER_TIMEOUT
will be used thereafter
E1022 21:38:22.947372221     240 oauth2_credentials.cc:236]           oauth_fetch: UNKNOWN:C-ares status is not ARES_S
UCCESS qtype=A name=metadata.google.internal. is_balancer=0: Domain name not found {grpc_status:2, created_time:"2023-1
0-22T21:38:22.947355967+00:00"}
Tensorflow version 2.12.0
```

TPU or GPU detection

In [2]:

```python
# NEW on TPU in TensorFlow 24: shorter cross-compatible TPU/GPU/multi-GPU/cluster-GPU detection code

try: # detect TPUs
    tpu = tf.distribute.cluster_resolver.TPUClusterResolver.connect() # TPU detection
    strategy = tf.distribute.TPUStrategy(tpu)
except ValueError: # detect GPUs
    strategy = tf.distribute.MirroredStrategy() # for GPU or multi-GPU machines
    #strategy = tf.distribute.get_strategy() # default strategy that works on CPU and single GPU
    #strategy = tf.distribute.experimental.MultiWorkerMirroredStrategy() # for clusters of multi-GPU machines

print("Number of accelerators: ", strategy.num_replicas_in_sync)
```

```
INFO:tensorflow:Deallocate tpu buffers before initializing tpu system.
INFO:tensorflow:Initializing the TPU system: local
INFO:tensorflow:Finished initializing TPU system.
INFO:tensorflow:Found TPU system:
INFO:tensorflow:*** Num TPU Cores: 8
INFO:tensorflow:*** Num TPU Workers: 1
INFO:tensorflow:*** Num TPU Cores Per Worker: 8
INFO:tensorflow:*** Available Device: _DeviceAttributes(/job:localhost/replica:0/task:0/device:CPU:0, CPU, 0, 0)
INFO:tensorflow:*** Available Device: _DeviceAttributes(/job:localhost/replica:0/task:0/device:TPU:0, TPU, 0, 0)
INFO:tensorflow:*** Available Device: _DeviceAttributes(/job:localhost/replica:0/task:0/device:TPU:1, TPU, 0, 0)
INFO:tensorflow:*** Available Device: _DeviceAttributes(/job:localhost/replica:0/task:0/device:TPU:2, TPU, 0, 0)
INFO:tensorflow:*** Available Device: _DeviceAttributes(/job:localhost/replica:0/task:0/device:TPU:3, TPU, 0, 0)
INFO:tensorflow:*** Available Device: _DeviceAttributes(/job:localhost/replica:0/task:0/device:TPU:4, TPU, 0, 0)
INFO:tensorflow:*** Available Device: _DeviceAttributes(/job:localhost/replica:0/task:0/device:TPU:5, TPU, 0, 0)
INFO:tensorflow:*** Available Device: _DeviceAttributes(/job:localhost/replica:0/task:0/device:TPU:6, TPU, 0, 0)
INFO:tensorflow:*** Available Device: _DeviceAttributes(/job:localhost/replica:0/task:0/device:TPU:7, TPU, 0, 0)
INFO:tensorflow:*** Available Device: _DeviceAttributes(/job:localhost/replica:0/task:0/device:TPU_SYSTEM:0, TPU_SYSTE
M, 0, 0)
Number of accelerators:  8
```

Configuration

In [4]:

```
IMAGE_SIZE = [512, 512] # At this size, a GPU will run out of memory. Use the TPU.
                        # For GPU training, please select 224 x 224 px image size.
EPOCHS = 12
BATCH_SIZE = 16 * strategy.num_replicas_in_sync

GCS_PATH_SELECT = { # available image sizes
    192: GCS_DS_PATH + '/tfrecords-jpeg-192x192',
    224: GCS_DS_PATH + '/tfrecords-jpeg-224x224',
    331: GCS_DS_PATH + '/tfrecords-jpeg-331x331',
    512: GCS_DS_PATH + '/tfrecords-jpeg-512x512'
}
GCS_PATH = GCS_PATH_SELECT[IMAGE_SIZE[0]]

TRAINING_FILENAMES = tf.io.gfile.glob(GCS_PATH + '/train/*.tfrec')
VALIDATION_FILENAMES = tf.io.gfile.glob(GCS_PATH + '/val/*.tfrec')
TEST_FILENAMES = tf.io.gfile.glob(GCS_PATH + '/test/*.tfrec') # predictions on this dataset should be submitted for the

CLASSES = ['pink primrose',    'hard-leaved pocket orchid', 'canterbury bells', 'sweet pea',     'wild geranium',     '
           'snapdragon',       "colt's foot",               'king protea',      'spear thistle', 'yellow iris',       '
           'fire lily',        'pincushion flower',         'fritillary',       'red ginger',    'grape hyacinth',    '
           'carnation',        'garden phlox',              'love in the mist', 'cosmos',        'alpine sea holly',  '
```

```
          'barberton daisy',  'daffodil',              'sword lily',       'poinsettia',    'bolero deep blue',  '
          'petunia',          'wild pansy',            'primula',          'sunflower',     'lilac hibiscus',    '
          'cautleya spicata', 'japanese anemone',      'black-eyed susan', 'silverbush',    'californian poppy', '
          'gazania',          'azalea',                'water lily',       'rose',          'thorn apple',       '
          'frangipani',       'clematis',              'hibiscus',         'columbine',     'desert-rose',       '
          'hippeastrum ',     'bee balm',              'pink quill',       'foxglove',      'bougainvillea',     '
          'trumpet creeper',  'blackberry lily',       'common tulip',     'wild rose']
```

Visualization utilities

In [5]:

```
# numpy and matplotlib defaults
np.set_printoptions(threshold=15, linewidth=80)

def batch_to_numpy_images_and_labels(data):
    images, labels = data
    numpy_images = images.numpy()
    numpy_labels = labels.numpy()
    if numpy_labels.dtype == object: # binary string in this case, these are image ID strings
        numpy_labels = [None for _ in enumerate(numpy_images)]
    # If no labels, only image IDs, return None for labels (this is the case for test data)
    return numpy_images, numpy_labels

def title_from_label_and_target(label, correct_label):
    if correct_label is None:
        return CLASSES[label], True
    correct = (label == correct_label)
    return "{} [{}{}{}]".format(CLASSES[label], 'OK' if correct else 'NO', u"\u2192" if not correct else '',
                                CLASSES[correct_label] if not correct else ''), correct

def display_one_flower(image, title, subplot, red=False, titlesize=16):
    plt.subplot(*subplot)
    plt.axis('off')
    plt.imshow(image)
    if len(title) > 0:
        plt.title(title, fontsize=int(titlesize) if not red else int(titlesize/1.2), color='red' if red else 'black', f
    return (subplot[0], subplot[1], subplot[2]+1)

def display_batch_of_images(databatch, predictions=None):
    """This will work with:
    display_batch_of_images(images)
    display_batch_of_images(images, predictions)
    display_batch_of_images((images, labels))
    display_batch_of_images((images, labels), predictions)
```

```
    """
    # data
    images, labels = batch_to_numpy_images_and_labels(databatch)
    if labels is None:
        labels = [None for _ in enumerate(images)]

    # auto-squaring: this will drop data that does not fit into square or square-ish rectangle
    rows = int(math.sqrt(len(images)))
    cols = len(images)//rows

    # size and spacing
    FIGSIZE = 13.0
    SPACING = 0.1
    subplot=(rows,cols,1)
    if rows < cols:
        plt.figure(figsize=(FIGSIZE,FIGSIZE/cols*rows))
    else:
        plt.figure(figsize=(FIGSIZE/rows*cols,FIGSIZE))

    # display
    for i, (image, label) in enumerate(zip(images[:rows*cols], labels[:rows*cols])):
        title = '' if label is None else CLASSES[label]
        correct = True
        if predictions is not None:
            title, correct = title_from_label_and_target(predictions[i], label)
        dynamic_titlesize = FIGSIZE*SPACING/max(rows,cols)*40+3 # magic formula tested to work from 1x1 to 10x10 images
        subplot = display_one_flower(image, title, subplot, not correct, titlesize=dynamic_titlesize)

    #layout
    plt.tight_layout()
    if label is None and predictions is None:
        plt.subplots_adjust(wspace=0, hspace=0)
    else:
        plt.subplots_adjust(wspace=SPACING, hspace=SPACING)
    plt.show()

def display_confusion_matrix(cmat, score, precision, recall):
    plt.figure(figsize=(15,15))
    ax = plt.gca()
    ax.matshow(cmat, cmap='Reds')
    ax.set_xticks(range(len(CLASSES)))
    ax.set_xticklabels(CLASSES, fontdict={'fontsize': 7})
    plt.setp(ax.get_xticklabels(), rotation=45, ha="left", rotation_mode="anchor")
    ax.set_yticks(range(len(CLASSES)))
    ax.set_yticklabels(CLASSES, fontdict={'fontsize': 7})
```

```
    plt.setp(ax.get_yticklabels(), rotation=45, ha="right", rotation_mode="anchor")
    titlestring = ""
    if score is not None:
        titlestring += 'f1 = {:.3f} '.format(score)
    if precision is not None:
        titlestring += '\nprecision = {:.3f} '.format(precision)
    if recall is not None:
        titlestring += '\nrecall = {:.3f} '.format(recall)
    if len(titlestring) > 0:
        ax.text(101, 1, titlestring, fontdict={'fontsize': 18, 'horizontalalignment':'right', 'verticalalignment':'top'
    plt.show()

def display_training_curves(training, validation, title, subplot):
    if subplot%10==1: # set up the subplots on the first call
        plt.subplots(figsize=(10,10), facecolor='#F0F0F0')
        plt.tight_layout()
    ax = plt.subplot(subplot)
    ax.set_facecolor('#F8F8F8')
    ax.plot(training)
    ax.plot(validation)
    ax.set_title('model '+ title)
    ax.set_ylabel(title)
    #ax.set_ylim(0.28,1.05)
    ax.set_xlabel('epoch')
    ax.legend(['train', 'valid.'])
```

Datasets

In [6]:

```
def decode_image(image_data):
    image = tf.image.decode_jpeg(image_data, channels=3)  # image format uint8 [0,255]
    image = tf.reshape(image, [*IMAGE_SIZE, 3]) # explicit size needed for TPU
    return image

def read_labeled_tfrecord(example):
    LABELED_TFREC_FORMAT = {
        "image": tf.io.FixedLenFeature([], tf.string), # tf.string means bytestring
        "class": tf.io.FixedLenFeature([], tf.int64),  # shape [] means single element
    }
    example = tf.io.parse_single_example(example, LABELED_TFREC_FORMAT)
    image = decode_image(example['image'])
    label = tf.cast(example['class'], tf.int32)
    return image, label # returns a dataset of (image, label) pairs
```

```python
def read_unlabeled_tfrecord(example):
    UNLABELED_TFREC_FORMAT = {
        "image": tf.io.FixedLenFeature([], tf.string), # tf.string means bytestring
        "id": tf.io.FixedLenFeature([], tf.string),  # shape [] means single element
        # class is missing, this competitions's challenge is to predict flower classes for the test dataset
    }
    example = tf.io.parse_single_example(example, UNLABELED_TFREC_FORMAT)
    image = decode_image(example['image'])
    idnum = example['id']
    return image, idnum # returns a dataset of image(s)

def load_dataset(filenames, labeled=True, ordered=False):
    # Read from TFRecords. For optimal performance, reading from multiple files at once and
    # disregarding data order. Order does not matter since we will be shuffling the data anyway.

    ignore_order = tf.data.Options()
    if not ordered:
        ignore_order.experimental_deterministic = False # disable order, increase speed

    dataset = tf.data.TFRecordDataset(filenames, num_parallel_reads=AUTO) # automatically interleaves reads from multip
    dataset = dataset.with_options(ignore_order) # uses data as soon as it streams in, rather than in its original orde
    dataset = dataset.map(read_labeled_tfrecord if labeled else read_unlabeled_tfrecord, num_parallel_calls=AUTO)
    # returns a dataset of (image, label) pairs if labeled=True or (image, id) pairs if labeled=False
    return dataset

def data_augment(image, label):
    # data augmentation. Thanks to the dataset.prefetch(AUTO) statement in the next function (below),
    # this happens essentially for free on TPU. Data pipeline code is executed on the "CPU" part
    # of the TPU while the TPU itself is computing gradients.
    image = tf.image.random_flip_left_right(image)
    #image = tf.image.random_saturation(image, 0, 2)
    return image, label

def get_training_dataset():
    dataset = load_dataset(TRAINING_FILENAMES, labeled=True)
    dataset = dataset.map(data_augment, num_parallel_calls=AUTO)
    dataset = dataset.repeat() # the training dataset must repeat for several epochs
    dataset = dataset.shuffle(2048)
    dataset = dataset.batch(BATCH_SIZE)
    dataset = dataset.prefetch(AUTO) # prefetch next batch while training (autotune prefetch buffer size)
    return dataset

def get_validation_dataset(ordered=False):
    dataset = load_dataset(VALIDATION_FILENAMES, labeled=True, ordered=ordered)
    dataset = dataset.batch(BATCH_SIZE)
```

```
    dataset = dataset.cache()
    dataset = dataset.prefetch(AUTO) # prefetch next batch while training (autotune prefetch buffer size)
    return dataset

def get_test_dataset(ordered=False):
    dataset = load_dataset(TEST_FILENAMES, labeled=False, ordered=ordered)
    dataset = dataset.batch(BATCH_SIZE)
    dataset = dataset.prefetch(AUTO) # prefetch next batch while training (autotune prefetch buffer size)
    return dataset

def count_data_items(filenames):
    # the number of data items is written in the name of the .tfrec files, i.e. flowers00-230.tfrec = 230 data items
    n = [int(re.compile(r"-([0-9]*)\.").search(filename).group(1)) for filename in filenames]
    return np.sum(n)

NUM_TRAINING_IMAGES = count_data_items(TRAINING_FILENAMES)
NUM_VALIDATION_IMAGES = count_data_items(VALIDATION_FILENAMES)
NUM_TEST_IMAGES = count_data_items(TEST_FILENAMES)
STEPS_PER_EPOCH = NUM_TRAINING_IMAGES // BATCH_SIZE
VALIDATION_STEPS = -(-NUM_VALIDATION_IMAGES // BATCH_SIZE) # The "-(-//)" trick rounds up instead of down :-)
TEST_STEPS = -(-NUM_TEST_IMAGES // BATCH_SIZE)             # The "-(-//)" trick rounds up instead of down :-)
print('Dataset: {} training images, {} validation images, {} unlabeled test images'.format(NUM_TRAINING_IMAGES, NUM_VAL
```

```
Dataset: 12753 training images, 3712 validation images, 7382 unlabeled test images
```

Dataset visualizations

In [7]:

```
# data dump
print("Training data shapes:")
for image, label in get_training_dataset().take(3):
    print(image.numpy().shape, label.numpy().shape)
print("Training data label examples:", label.numpy())
print("Validation data shapes:")
for image, label in get_validation_dataset().take(3):
    print(image.numpy().shape, label.numpy().shape)
print("Validation data label examples:", label.numpy())
print("Test data shapes:")
for image, idnum in get_test_dataset().take(3):
    print(image.numpy().shape, idnum.numpy().shape)
print("Test data IDs:", idnum.numpy().astype('U')) # U=unicode string
```

```
Training data shapes:
(128, 512, 512, 3) (128,)
(128, 512, 512, 3) (128,)
(128, 512, 512, 3) (128,)
Training data label examples: [73 26 62 ... 88 50 46]
Validation data shapes:
(128, 512, 512, 3) (128,)
(128, 512, 512, 3) (128,)
(128, 512, 512, 3) (128,)
Validation data label examples: [103  53  49 ...  73  95  88]
Test data shapes:
(128, 512, 512, 3) (128,)
(128, 512, 512, 3) (128,)
(128, 512, 512, 3) (128,)
Test data IDs: ['b51c9863e' '92c33e670' '8756842ba' ... '2b782bb3f' '23253eaed' 'd25115245']
```

In [8]:
```
# Peek at training data
training_dataset = get_training_dataset()
training_dataset = training_dataset.unbatch().batch(20)
train_batch = iter(training_dataset)
```

In [9]:
```
# run this cell again for next set of images
display_batch_of_images(next(train_batch))
```


yellow iris
tree poppy
hard-leaved pocket orchid
daisy
azalea
common dandelion
wild geranium
primula
common tulip
spear thistle
sweet william
geranium
petunia
sunflower
buttercup
sword lily
wild rose
mexican petunia
wild geranium
iris

In [10]:
```
# peer at test data
test_dataset = get_test_dataset()
test_dataset = test_dataset.unbatch().batch(20)
test_batch = iter(test_dataset)
```

In [11]:
```
# run this cell again for next set of images
display_batch_of_images(next(test_batch))
```

Model

In [12]:
```
with strategy.scope():
    #img_adjust_layer = tf.keras.layers.Lambda(lambda data: tf.keras.applications.xception.preprocess_input(tf.cast(data
    #pretrained_model = tf.keras.applications.Xception(weights='imagenet', include_top=False)

    img_adjust_layer = tf.keras.layers.Lambda(lambda data: tf.keras.applications.vgg16.preprocess_input(tf.cast(data, t
    pretrained_model = tf.keras.applications.VGG16(weights='imagenet', include_top=False)

    pretrained_model.trainable = False # False = transfer learning, True = fine-tuning

    model = tf.keras.Sequential([
        img_adjust_layer,
        pretrained_model,
        tf.keras.layers.GlobalAveragePooling2D(),
        tf.keras.layers.Dense(len(CLASSES), activation='softmax')
    ])

model.compile(
    optimizer='adam',
    loss = 'sparse_categorical_crossentropy',
    metrics=['sparse_categorical_accuracy'],
    # NEW on TPU in TensorFlow 24: sending multiple batches to the TPU at once saves communications
    # overheads and allows the XLA compiler to unroll the loop on TPU and optimize hardware utilization.
    steps_per_execution=16
)
model.summary()
```

```
Downloading data from https://storage.googleapis.com/tensorflow/keras-applications/vgg16/vgg16_weights_tf_dim_ordering_
tf_kernels_notop.h5
58889256/58889256 [==============================] - 0s 0us/step
Model: "sequential"
_________________________________________________________________
 Layer (type)                Output Shape              Param #
=================================================================
 lambda (Lambda)             (None, 512, 512, 3)       0

 vgg16 (Functional)          (None, None, None, 512)   14714688

 global_average_pooling2d (G  (None, 512)              0
 lobalAveragePooling2D)

 dense (Dense)               (None, 104)               53352

=================================================================
Total params: 14,768,040
Trainable params: 53,352
Non-trainable params: 14,714,688
_________________________________________________________________
```

Training

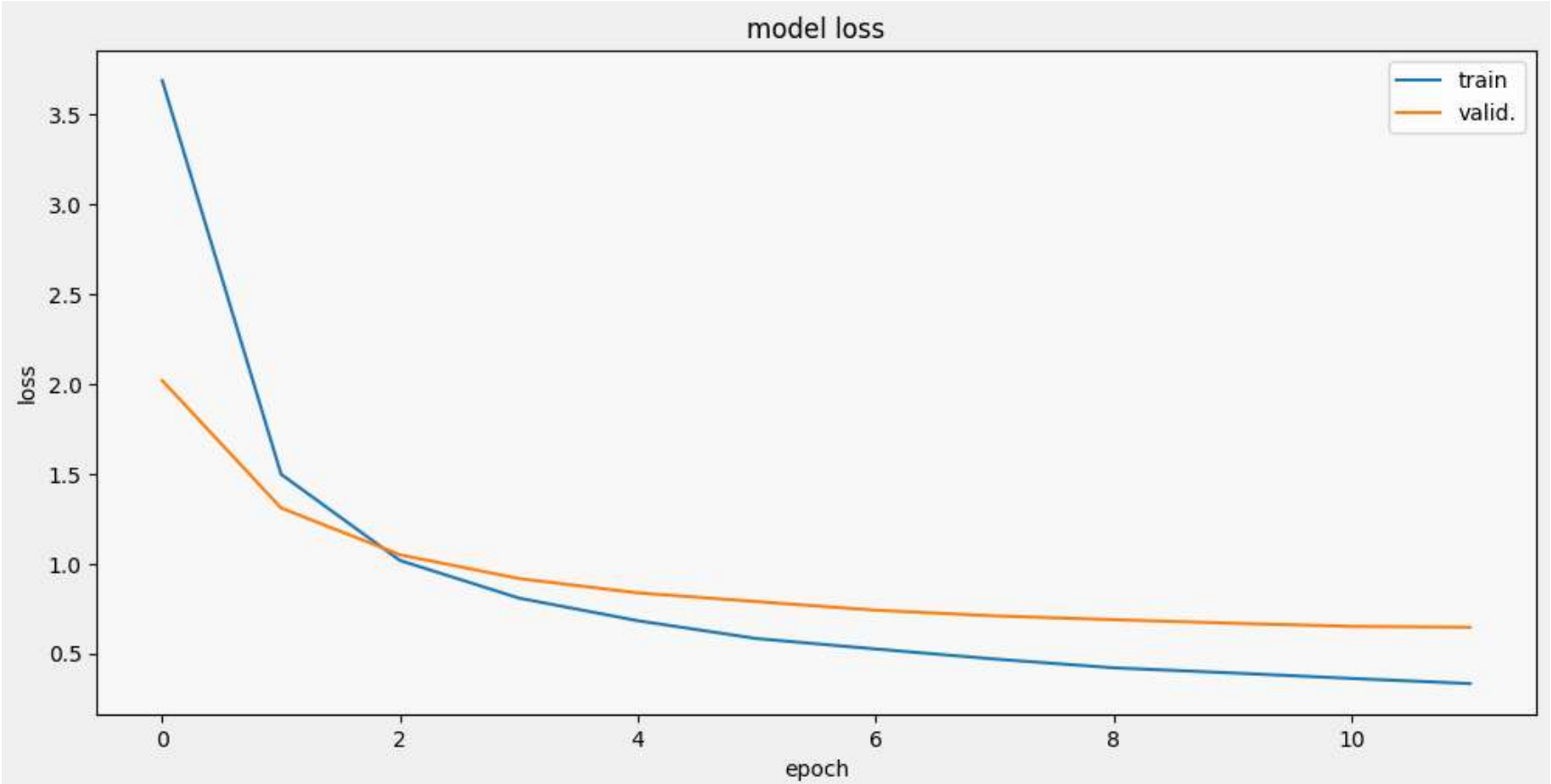
In [13]:
```
history = model.fit(get_training_dataset(), steps_per_epoch=STEPS_PER_EPOCH, epochs=EPOCHS,
                    validation_data=get_validation_dataset(), validation_steps=VALIDATION_STEPS)
```

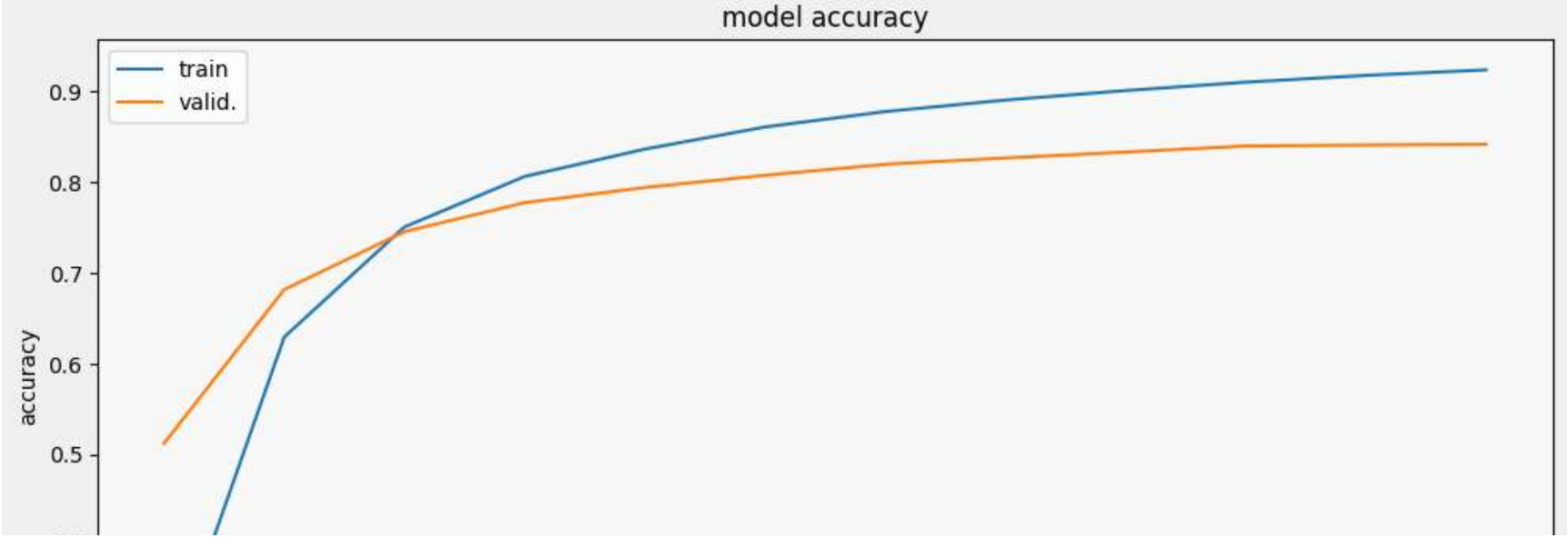
```
Epoch 1/12
99/99 [==============================] - 30s 303ms/step - loss: 3.6911 - sparse_categorical_accuracy: 0.2514 - val_los
s: 2.0196 - val_sparse_categorical_accuracy: 0.5116
Epoch 2/12
99/99 [==============================] - 12s 122ms/step - loss: 1.4973 - sparse_categorical_accuracy: 0.6289 - val_los
s: 1.3093 - val_sparse_categorical_accuracy: 0.6813
Epoch 3/12
99/99 [==============================] - 12s 122ms/step - loss: 1.0170 - sparse_categorical_accuracy: 0.7504 - val_los
s: 1.0488 - val_sparse_categorical_accuracy: 0.7452
Epoch 4/12
99/99 [==============================] - 12s 122ms/step - loss: 0.8074 - sparse_categorical_accuracy: 0.8061 - val_los
s: 0.9162 - val_sparse_categorical_accuracy: 0.7772
Epoch 5/12
99/99 [==============================] - 12s 124ms/step - loss: 0.6813 - sparse_categorical_accuracy: 0.8363 - val_los
s: 0.8364 - val_sparse_categorical_accuracy: 0.7939
Epoch 6/12
99/99 [==============================] - 12s 122ms/step - loss: 0.5818 - sparse_categorical_accuracy: 0.8607 - val_los
s: 0.7878 - val_sparse_categorical_accuracy: 0.8074
Epoch 7/12
99/99 [==============================] - 12s 122ms/step - loss: 0.5239 - sparse_categorical_accuracy: 0.8777 - val_los
s: 0.7399 - val_sparse_categorical_accuracy: 0.8195
Epoch 8/12
99/99 [==============================] - 12s 122ms/step - loss: 0.4678 - sparse_categorical_accuracy: 0.8904 - val_los
s: 0.7082 - val_sparse_categorical_accuracy: 0.8265
Epoch 9/12
99/99 [==============================] - 12s 122ms/step - loss: 0.4190 - sparse_categorical_accuracy: 0.9009 - val_los
s: 0.6872 - val_sparse_categorical_accuracy: 0.8330
Epoch 10/12
99/99 [==============================] - 12s 122ms/step - loss: 0.3903 - sparse_categorical_accuracy: 0.9101 - val_los
s: 0.6672 - val_sparse_categorical_accuracy: 0.8397
Epoch 11/12
99/99 [==============================] - 12s 122ms/step - loss: 0.3597 - sparse_categorical_accuracy: 0.9176 - val_los
s: 0.6498 - val_sparse_categorical_accuracy: 0.8408
Epoch 12/12
99/99 [==============================] - 12s 122ms/step - loss: 0.3311 - sparse_categorical_accuracy: 0.9237 - val_los
s: 0.6450 - val_sparse_categorical_accuracy: 0.8416
```

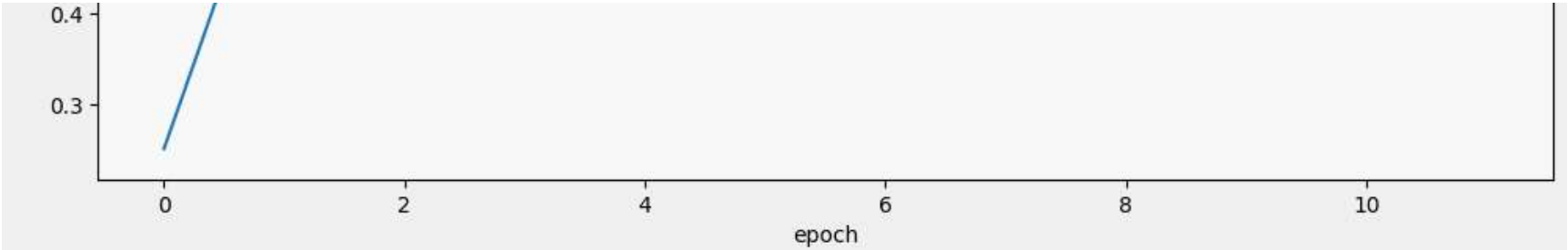
In [14]:

```
display_training_curves(history.history['loss'], history.history['val_loss'], 'loss', 211)
display_training_curves(history.history['sparse_categorical_accuracy'], history.history['val_sparse_categorical_accuracy
```

```
/tmp/ipykernel_15/3094237293.py:95: MatplotlibDeprecationWarning: Auto-removal of overlapping axes is deprecated since
3.6 and will be removed two minor releases later; explicitly call ax.remove() as needed.
  ax = plt.subplot(subplot)
```


model loss
train
valid.
loss
epoch
0
2
4
6
8
10
0.5
1.0
1.5
2.0
2.5
3.0
3.5


model accuracy
train
valid.
accuracy
0.5
0.6
0.7
0.8
0.9



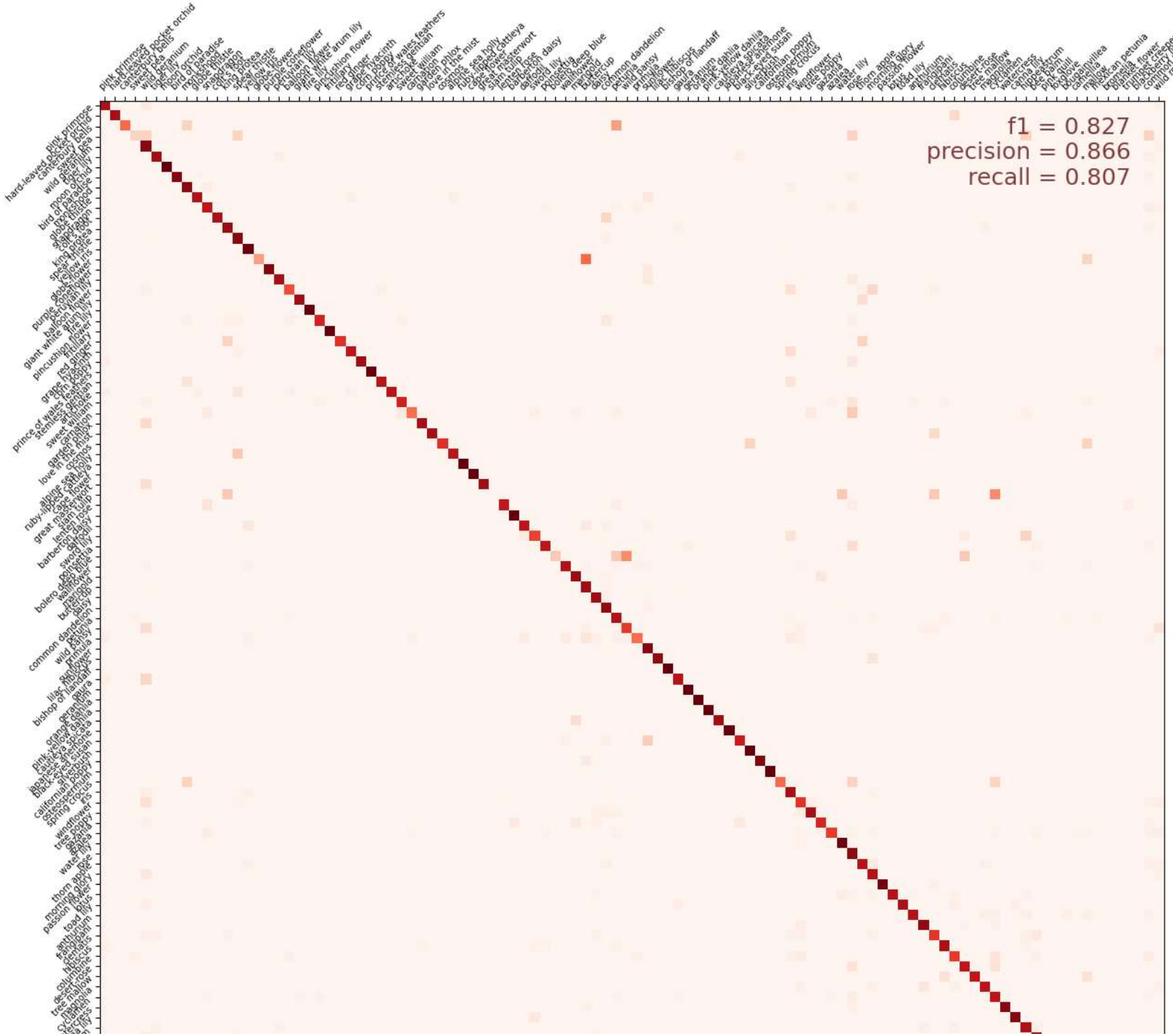
Confusion matrix

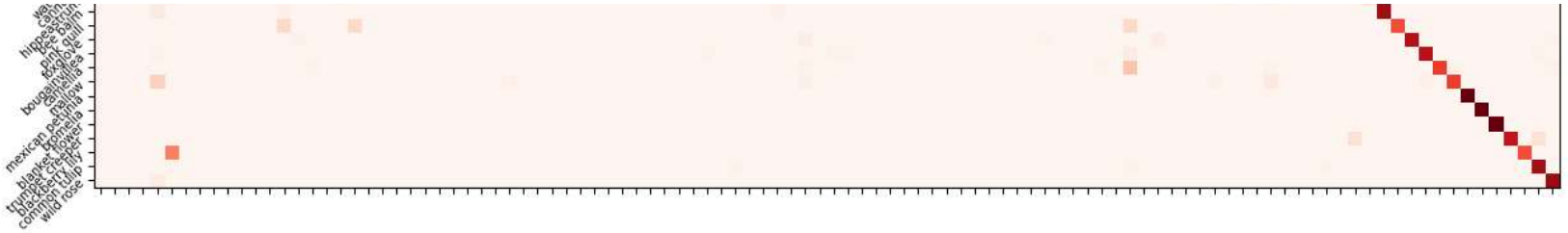
In [15]:
```
cmdataset = get_validation_dataset(ordered=True) # since we are splitting the dataset and iterating separately on images
images_ds = cmdataset.map(lambda image, label: image)
labels_ds = cmdataset.map(lambda image, label: label).unbatch()
cm_correct_labels = next(iter(labels_ds.batch(NUM_VALIDATION_IMAGES))).numpy() # get everything as one batch
cm_probabilities = model.predict(images_ds, steps=VALIDATION_STEPS)
cm_predictions = np.argmax(cm_probabilities, axis=-1)
print("Correct   labels: ", cm_correct_labels.shape, cm_correct_labels)
print("Predicted labels: ", cm_predictions.shape, cm_predictions)
```

```
2023-10-22 21:41:40.661302: E tensorflow/core/grappler/optimizers/meta_optimizer.cc:954] model_pruner failed: INVALID_A
RGUMENT: Graph does not contain terminal node AssignAddVariableOp.
2023-10-22 21:41:40.855424: E tensorflow/core/grappler/optimizers/meta_optimizer.cc:954] model_pruner failed: INVALID_A
RGUMENT: Graph does not contain terminal node AssignAddVariableOp.
29/29 [==============================] - 17s 593ms/step
Correct   labels:  (3712,) [74 82 62 ... 67 50 53]
Predicted labels:  (3712,) [74 82 53 ... 67 50 53]
```

In [16]:
```
cmat = confusion_matrix(cm_correct_labels, cm_predictions, labels=range(len(CLASSES)))
score = f1_score(cm_correct_labels, cm_predictions, labels=range(len(CLASSES)), average='macro')
precision = precision_score(cm_correct_labels, cm_predictions, labels=range(len(CLASSES)), average='macro')
recall = recall_score(cm_correct_labels, cm_predictions, labels=range(len(CLASSES)), average='macro')
cmat = (cmat.T / cmat.sum(axis=1)).T # normalized
display_confusion_matrix(cmat, score, precision, recall)
print('f1 score: {:.3f}, precision: {:.3f}, recall: {:.3f}'.format(score, precision, recall))
```

```
/usr/local/lib/python3.8/site-packages/sklearn/metrics/_classification.py:1469: UndefinedMetricWarning: Precision is il
l-defined and being set to 0.0 in labels with no predicted samples. Use `zero_division` parameter to control this behav
ior.
  _warn_prf(average, modifier, msg_start, len(result))
```


f1 = 0.827
precision = 0.866
recall = 0.807



f1 score: 0.827, precision: 0.866, recall: 0.807

Predictions

In [17]:
```
test_ds = get_test_dataset(ordered=True) # since we are splitting the dataset and iterating separately on images and ids

print('Computing predictions...')
test_images_ds = test_ds.map(lambda image, idnum: image)
probabilities = model.predict(test_images_ds, steps=TEST_STEPS)
predictions = np.argmax(probabilities, axis=-1)
print(predictions)

print('Generating submission.csv file...')
test_ids_ds = test_ds.map(lambda image, idnum: idnum).unbatch()
test_ids = next(iter(test_ids_ds.batch(NUM_TEST_IMAGES))).numpy().astype('U') # all in one batch
np.savetxt('submission.csv', np.rec.fromarrays([test_ids, predictions]), fmt=['%s', '%d'], delimiter=',', header='id,la
!head submission.csv
```

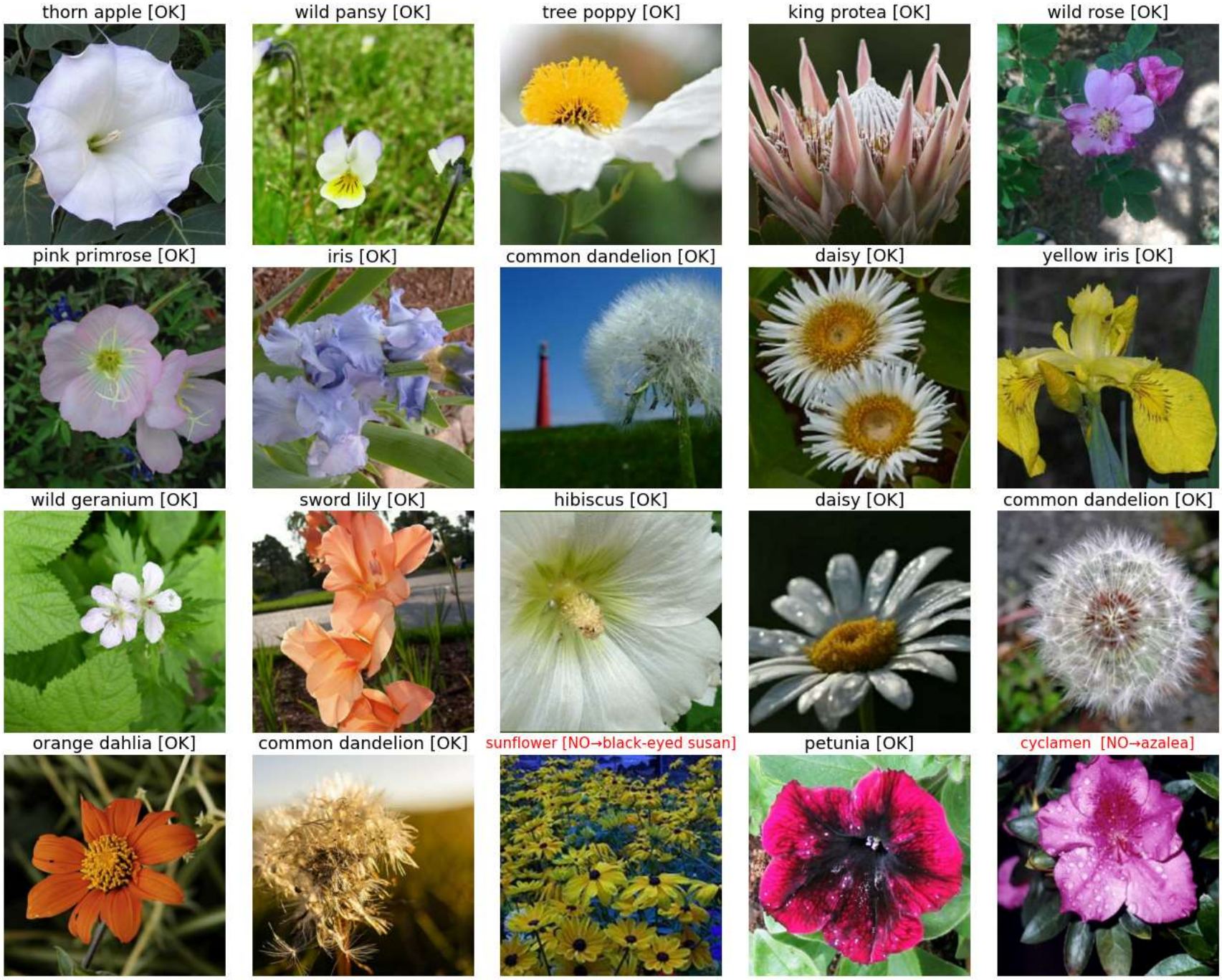
```
Computing predictions...
58/58 [==============================] - 15s 258ms/step
[ 14  83 103 ...  49  47 103]
Generating submission.csv file...
id,label
0b9afbdf2,14
c37a6f3e9,83
00e4f514e,103
1c4736dea,28
252d840db,67
dfc9c6a23,103
53cfc6586,48
541c4d41e,102
59d1b6146,70
```

Visual validation

In [18]:
```
dataset = get_validation_dataset()
dataset = dataset.unbatch().batch(20)
batch = iter(dataset)
```

In [19]:
```
# run this cell again for next set of images
images, labels = next(batch)
probabilities = model.predict(tf.cast(images, tf.float32))
predictions = np.argmax(probabilities, axis=-1)
display_batch_of_images((images, labels), predictions)
```

```
2023-10-22 21:42:18.746861: E tensorflow/core/grappler/optimizers/meta_optimizer.cc:954] model_pruner failed: INVALID_A
RGUMENT: Graph does not contain terminal node AssignAddVariableOp.
2023-10-22 21:42:18.939716: E tensorflow/core/grappler/optimizers/meta_optimizer.cc:954] model_pruner failed: INVALID_A
RGUMENT: Graph does not contain terminal node AssignAddVariableOp.
1/1 [==============================] - 9s 9s/step
```


thorn apple [OK]
wild pansy [OK]
tree poppy [OK]
king protea [OK]
wild rose [OK]
pink primrose [OK]
iris [OK]
common dandelion [OK]
daisy [OK]
yellow iris [OK]
wild geranium [OK]
sword lily [OK]
hibiscus [OK]
daisy [OK]
common dandelion [OK]
orange dahlia [OK]
common dandelion [OK]
sunflower [NO→black-eyed susan]
petunia [OK]
cyclamen [NO→azalea]