In [1]:

```
!pip install tensorflow-addons
```

```
Looking in indexes: https://pypi.org/simple, https://us-python.pkg.dev/colab-wheels/publi
c/simple/
Collecting tensorflow-addons
  Downloading tensorflow_addons-0.17.0-cp37-cp37m-manylinux_2_17_x86_64.manylinux2014_x86
_64.whl (1.1 MB)
     |████████████████████████████████| 1.1 MB 4.8 MB/s
Requirement already satisfied: packaging in /usr/local/lib/python3.7/dist-packages (from
tensorflow-addons) (21.3)
Requirement already satisfied: typeguard>=2.7 in /usr/local/lib/python3.7/dist-packages (
from tensorflow-addons) (2.7.1)
Requirement already satisfied: pyparsing!=3.0.5,>=2.0.2 in /usr/local/lib/python3.7/dist-
packages (from packaging->tensorflow-addons) (3.0.9)
Installing collected packages: tensorflow-addons
Successfully installed tensorflow-addons-0.17.0
```

In [2]:

```
#General imports
from __future__ import print_function
import os
import pandas as pd
import numpy as np
import matplotlib.pyplot as plt
import progressbar
from sklearn.model_selection import train_test_split
from sklearn.preprocessing import normalize
from sklearn.decomposition import PCA

# Keras imports
import keras
from tensorflow.keras import layers

from keras.preprocessing.image import load_img
from keras.models import Sequential, Model
from tensorflow.keras.optimizers import *
from keras.utils.np_utils import to_categorical
import keras.backend as K

# application (model) imports
from tensorflow.keras import applications
#from keras.applications.inception_v3 import preprocess_input
from keras.layers import Dense

import tensorflow as tf
import tensorflow_datasets as tfds
import tensorflow_addons as tfa
```

In [3]:

```
# load the data from Tensorflow
tfds.disable_progress_bar()
(train, validation, test),info = tfds.load(
    "oxford_flowers102",
    split=["train", "validation", "test"],
    as_supervised=True,  # Include labels
    with_info = True,
    try_gcs = True,
)

print("Number of training samples: %d" % tf.data.experimental.cardinality(train))
print("Number of validation samples: %d" % tf.data.experimental.cardinality(validation))
print("Number of test samples: %d" % tf.data.experimental.cardinality(test))
```

```
Downloading and preparing dataset oxford_flowers102/2.1.1 (download: 328.90 MiB, generate
d: 331.34 MiB, total: 660.25 MiB) to /root/tensorflow_datasets/oxford_flowers102/2.1.1...
```

```
Shuffling and writing examples to /root/tensorflow_datasets/oxford_flowers102/2.1.1.incom
pleteMECMLO/oxford_flowers102-train.tfrecord
Shuffling and writing examples to /root/tensorflow_datasets/oxford_flowers102/2.1.1.incom
pleteMECMLO/oxford_flowers102-test.tfrecord
Shuffling and writing examples to /root/tensorflow_datasets/oxford_flowers102/2.1.1.incom
pleteMECMLO/oxford_flowers102-validation.tfrecord
Dataset oxford_flowers102 downloaded and prepared to /root/tensorflow_datasets/oxford_flo
wers102/2.1.1. Subsequent calls will reuse this data.
Number of training samples: 1020
Number of validation samples: 1020
Number of test samples: 6149
```

In [4]:

```python
print(info)
```

```
tfds.core.DatasetInfo(
    name='oxford_flowers102',
    version=2.1.1,
    description='The Oxford Flowers 102 dataset is a consistent of 102 flower categories
commonly occurring
in the United Kingdom. Each class consists of between 40 and 258 images. The images have
large scale, pose and light variations. In addition, there are categories that have large
variations within the category and several very similar categories.

The dataset is divided into a training set, a validation set and a test set.
The training set and validation set each consist of 10 images per class (totalling 1020 i
mages each).
The test set consists of the remaining 6149 images (minimum 20 per class).',
    homepage='https://www.robots.ox.ac.uk/~vgg/data/flowers/102/',
    features=FeaturesDict({
        'file_name': Text(shape=(), dtype=tf.string),
        'image': Image(shape=(None, None, 3), dtype=tf.uint8),
        'label': ClassLabel(shape=(), dtype=tf.int64, num_classes=102),
    }),
    total_num_examples=8189,
    splits={
        'test': 6149,
        'train': 1020,
        'validation': 1020,
    },
    supervised_keys=('image', 'label'),
    citation="""@InProceedings{Nilsback08,
       author = "Nilsback, M-E. and Zisserman, A.",
       title = "Automated Flower Classification over a Large Number of Classes",
       booktitle = "Proceedings of the Indian Conference on Computer Vision, Graphics and
Image Processing",
       year = "2008",
       month = "Dec"
    }""",
    redistribution_info=,
)
```

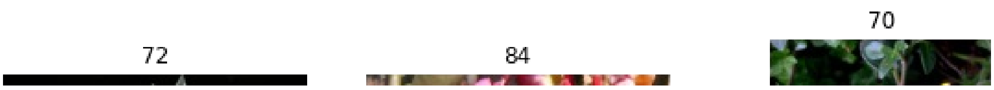
In [5]:

```python
#combine training and validation data as training data

train = train.concatenate(validation)
```

In [6]:

```python
plt.figure(figsize=(10, 10))
for i, (image, label) in enumerate(train.take(9)):
    ax = plt.subplot(3, 3, i + 1)
    plt.imshow(image)
    plt.title(int(label))
    plt.axis("off")
```



In [7]:

```
#resize the augmented images to 224 * 224

size = (224, 224)

train = train.map(lambda x, y: (tf.image.resize(x, size), y))
test = test.map(lambda x, y: (tf.image.resize(x, size), y))
#we don't need to resize the test image
```

In [8]:

```
size = (224, 224,3)

train = train.map(lambda x, y: (tf.image.random_flip_left_right(x), y))
train = train.map(lambda x, y: (tf.image.random_crop(x, size), y))
#train = train.map(lambda x, y: (tf.image.random_saturation(x, size), y))
```

In [9]:

```
#batch the data and use caching & prefetching to optimize loading speed

batch_size = 64

train = train.batch(batch_size).prefetch(buffer_size=10)
test = test.batch(batch_size).prefetch(buffer_size=10)
```

In [10]:

```
#set up base model
input_shape = (224,224,3)
num_classes=102

base_model = applications.resnet_v2.ResNet101V2(input_shape=input_shape, include_top=False, weights='imagenet', pooling='avg')

x = base_model.output #We use Keras Functional API here
predictions = Dense(num_classes, activation='softmax')(x)
```

```
model = Model(inputs = base_model.input, outputs=predictions)
```

```
Downloading data from https://storage.googleapis.com/tensorflow/keras-applications/resnet
/resnet101v2_weights_tf_dim_ordering_tf_kernels_notop.h5
171319296/171317808 [==============================] - 1s 0us/step
171327488/171317808 [==============================] - 1s 0us/step
```

In [11]:

```
# compile the model
# Optimizer: Nesterov's Accelerated Gradient
alpha = 0.0001 #weight decay
momentum = 0.9
model.compile(optimizer=tfa.optimizers.SGDW(momentum = momentum, weight_decay = alpha, n
esterov=True, learning_rate=0.01),
              loss=['sparse_categorical_crossentropy'], metrics=['sparse_categorical_acc
uracy'])
```

In [12]:

```
#train the model
history = model.fit(train, epochs=15, validation_data= test, batch_size=256)
```

```
Epoch 1/15
32/32 [==============================] - 57s 1s/step - loss: 2.7031 - sparse_categorical_
accuracy: 0.4294 - val_loss: 7.3995 - val_sparse_categorical_accuracy: 0.1226
Epoch 2/15
32/32 [==============================] - 30s 958ms/step - loss: 0.4003 - sparse_categoric
al_accuracy: 0.9005 - val_loss: 3.0069 - val_sparse_categorical_accuracy: 0.4558
Epoch 3/15
32/32 [==============================] - 30s 958ms/step - loss: 0.0585 - sparse_categoric
al_accuracy: 0.9897 - val_loss: 1.7892 - val_sparse_categorical_accuracy: 0.6221
Epoch 4/15
32/32 [==============================] - 30s 961ms/step - loss: 0.0182 - sparse_categoric
al_accuracy: 0.9971 - val_loss: 0.9182 - val_sparse_categorical_accuracy: 0.7845
Epoch 5/15
32/32 [==============================] - 30s 961ms/step - loss: 0.0075 - sparse_categoric
al_accuracy: 0.9995 - val_loss: 0.6394 - val_sparse_categorical_accuracy: 0.8379
Epoch 6/15
32/32 [==============================] - 30s 959ms/step - loss: 0.0053 - sparse_categoric
al_accuracy: 0.9995 - val_loss: 0.6844 - val_sparse_categorical_accuracy: 0.8235
Epoch 7/15
32/32 [==============================] - 36s 1s/step - loss: 0.0046 - sparse_categorical_
accuracy: 0.9995 - val_loss: 0.5130 - val_sparse_categorical_accuracy: 0.8644
Epoch 8/15
32/32 [==============================] - 30s 957ms/step - loss: 0.0037 - sparse_categoric
al_accuracy: 0.9995 - val_loss: 0.4754 - val_sparse_categorical_accuracy: 0.8720
Epoch 9/15
32/32 [==============================] - 30s 957ms/step - loss: 0.0022 - sparse_categoric
al_accuracy: 1.0000 - val_loss: 0.4576 - val_sparse_categorical_accuracy: 0.8754
Epoch 10/15
32/32 [==============================] - 30s 958ms/step - loss: 0.0018 - sparse_categoric
al_accuracy: 1.0000 - val_loss: 0.4507 - val_sparse_categorical_accuracy: 0.8774
Epoch 11/15
32/32 [==============================] - 30s 958ms/step - loss: 0.0016 - sparse_categoric
al_accuracy: 1.0000 - val_loss: 0.4489 - val_sparse_categorical_accuracy: 0.8787
Epoch 12/15
32/32 [==============================] - 30s 958ms/step - loss: 0.0015 - sparse_categoric
al_accuracy: 1.0000 - val_loss: 0.4500 - val_sparse_categorical_accuracy: 0.8784
Epoch 13/15
32/32 [==============================] - 30s 957ms/step - loss: 0.0014 - sparse_categoric
al_accuracy: 1.0000 - val_loss: 0.4501 - val_sparse_categorical_accuracy: 0.8785
Epoch 14/15
32/32 [==============================] - 30s 956ms/step - loss: 0.0013 - sparse_categoric
al_accuracy: 1.0000 - val_loss: 0.4496 - val_sparse_categorical_accuracy: 0.8798
Epoch 15/15
32/32 [==============================] - 30s 958ms/step - loss: 0.0013 - sparse_categoric
al_accuracy: 1.0000 - val_loss: 0.4503 - val_sparse_categorical_accuracy: 0.8797
```

In [13]:

```
#Define plot function
```

```python
def plot_loss_accuracy(history):
    historydf = pd.DataFrame(history.history, index=history.epoch)
    plt.figure(figsize=(8, 6))
    historydf.plot(ylim=(0, max(1, historydf.values.max())))
    loss = history.history['loss'][-1]
    acc = history.history['sparse_categorical_accuracy'][-1]
    val_acc = history.history['val_sparse_categorical_accuracy'][-1]
    val_error = (1 - val_acc)
    plt.title('Loss: %.3f, sparse_categorical_accuracy: %.3f' % (loss, acc))
    print('Validation Error: %.3f' % (val_error))
```

In [14]:

```python
plot_loss_accuracy(history)
```

```
Validation Error: 0.120

<Figure size 576x432 with 0 Axes>
```