Transfer learning and fine tuning for Oxford Pets dataset

In [ ]:

```
#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
```

In [ ]:

```
# load the data from Tensorflow
tfds.disable_progress_bar()
(train, validation, test),info = tfds.load(
    "oxford_iiit_pet",
    split=["train", "test[:50%]","test[50%:100%]"],
    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_iiit_pet/3.2.0 (download: 773.52 MiB, generated: 774.69 MiB, total: 1.51 GiB) to /root/tensorflow_datasets/oxford_iiit_pet/3.2.0...
Shuffling and writing examples to /root/tensorflow_datasets/oxford_iiit_pet/3.2.0.incompleteOP0UKO/oxford_iiit_pet-train.tfrecord
Shuffling and writing examples to /root/tensorflow_datasets/oxford_iiit_pet/3.2.0.incompleteOP0UKO/oxford_iiit_pet-test.tfrecord
Dataset oxford_iiit_pet downloaded and prepared to /root/tensorflow_datasets/oxford_iiit_pet/3.2.0. Subsequent calls will reuse this data.
Number of training samples: 3680
Number of validation samples: 1834
Number of test samples: 1835
```

In [ ]:

```
# check the metadata
print(info)
```

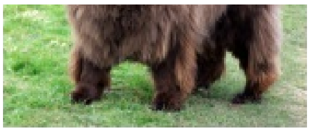
```
tfds.core.DatasetInfo(
```

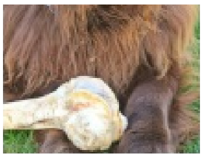
```
    name='oxford_iiit_pet',
    version=3.2.0,
    description='The Oxford-IIIT pet dataset is a 37 category pet image dataset with roug
hly 200
images for each class. The images have large variations in scale, pose and
lighting. All images have an associated ground truth annotation of breed.',
    homepage='http://www.robots.ox.ac.uk/~vgg/data/pets/',
    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=37),
        'segmentation_mask': Image(shape=(None, None, 1), dtype=tf.uint8),
        'species': ClassLabel(shape=(), dtype=tf.int64, num_classes=2),
    }),
    total_num_examples=7349,
    splits={
        'test': 3669,
        'train': 3680,
    },
    supervised_keys=('image', 'label'),
    citation="""@InProceedings{parkhi12a,
      author       = "Parkhi, O. M. and Vedaldi, A. and Zisserman, A. and Jawahar, C.~V."
,
      title        = "Cats and Dogs",
      booktitle    = "IEEE Conference on Computer Vision and Pattern Recognition",
      year         = "2012",
    }""",
    redistribution_info=,
)
```

show and check the images

In [ ]:

```
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")
```





Standardizing the data

In [ ]:

```
#resize the images

size = (224, 224)

train = train.map(lambda x, y: (tf.image.resize(x, size), y))
validation = validation.map(lambda x, y: (tf.image.resize(x, size), y))
test = test.map(lambda x, y: (tf.image.resize(x, size), y))
```

In [ ]:

```
#data augmentation #need to run cell first

size = (224, 224,3)

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

In [ ]:

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

batch_size = 64

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

DenseNet201

In [ ]:

```
#set up base model
input_shape = (224,224,3)
num_classes=37

base_model = applications.DenseNet201(input_shape=input_shape, include_top=False, weight
s='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/densen
et/densenet201_weights_tf_dim_ordering_tf_kernels_notop.h5
74842112/74836368 [==============================] - 0s 0us/step
74850304/74836368 [==============================] - 0s 0us/step
```

In [ ]:

```
# compile the model
model.compile(optimizer=SGD(momentum=0.9, learning_rate=0.001),
              loss=['sparse_categorical_crossentropy'], metrics=['sparse_categorical_acc
uracy'])
```

In [ ]:

```
#train the model
```

```
history = model.fit(train, epochs=20, validation_data=validation)
loss, accuracy  = model.evaluate(test, verbose=False)
```

```
Epoch 1/20
58/58 [==============================] - 71s 717ms/step - loss: 2.4363 - sparse_categoric
al_accuracy: 0.4505 - val_loss: 0.8022 - val_sparse_categorical_accuracy: 0.8157
Epoch 2/20
58/58 [==============================] - 36s 622ms/step - loss: 0.5502 - sparse_categoric
al_accuracy: 0.9120 - val_loss: 0.4641 - val_sparse_categorical_accuracy: 0.8931
Epoch 3/20
58/58 [==============================] - 36s 621ms/step - loss: 0.2748 - sparse_categoric
al_accuracy: 0.9603 - val_loss: 0.3943 - val_sparse_categorical_accuracy: 0.9095
Epoch 4/20
58/58 [==============================] - 36s 620ms/step - loss: 0.1751 - sparse_categoric
al_accuracy: 0.9802 - val_loss: 0.3580 - val_sparse_categorical_accuracy: 0.9106
Epoch 5/20
58/58 [==============================] - 36s 622ms/step - loss: 0.1215 - sparse_categoric
al_accuracy: 0.9916 - val_loss: 0.3351 - val_sparse_categorical_accuracy: 0.9149
Epoch 6/20
58/58 [==============================] - 36s 620ms/step - loss: 0.0891 - sparse_categoric
al_accuracy: 0.9965 - val_loss: 0.3183 - val_sparse_categorical_accuracy: 0.9133
Epoch 7/20
58/58 [==============================] - 36s 623ms/step - loss: 0.0670 - sparse_categoric
al_accuracy: 0.9984 - val_loss: 0.3098 - val_sparse_categorical_accuracy: 0.9144
Epoch 8/20
58/58 [==============================] - 36s 622ms/step - loss: 0.0528 - sparse_categoric
al_accuracy: 0.9997 - val_loss: 0.3019 - val_sparse_categorical_accuracy: 0.9144
Epoch 9/20
58/58 [==============================] - 36s 621ms/step - loss: 0.0429 - sparse_categoric
al_accuracy: 0.9997 - val_loss: 0.2957 - val_sparse_categorical_accuracy: 0.9188
Epoch 10/20
58/58 [==============================] - 36s 622ms/step - loss: 0.0353 - sparse_categoric
al_accuracy: 1.0000 - val_loss: 0.2926 - val_sparse_categorical_accuracy: 0.9160
Epoch 11/20
58/58 [==============================] - 36s 621ms/step - loss: 0.0304 - sparse_categoric
al_accuracy: 1.0000 - val_loss: 0.2904 - val_sparse_categorical_accuracy: 0.9177
Epoch 12/20
58/58 [==============================] - 36s 621ms/step - loss: 0.0262 - sparse_categoric
al_accuracy: 1.0000 - val_loss: 0.2895 - val_sparse_categorical_accuracy: 0.9182
Epoch 13/20
58/58 [==============================] - 36s 621ms/step - loss: 0.0228 - sparse_categoric
al_accuracy: 1.0000 - val_loss: 0.2877 - val_sparse_categorical_accuracy: 0.9177
Epoch 14/20
58/58 [==============================] - 36s 620ms/step - loss: 0.0207 - sparse_categoric
al_accuracy: 1.0000 - val_loss: 0.2869 - val_sparse_categorical_accuracy: 0.9177
Epoch 15/20
58/58 [==============================] - 36s 620ms/step - loss: 0.0184 - sparse_categoric
al_accuracy: 1.0000 - val_loss: 0.2868 - val_sparse_categorical_accuracy: 0.9177
Epoch 16/20
58/58 [==============================] - 36s 620ms/step - loss: 0.0167 - sparse_categoric
al_accuracy: 1.0000 - val_loss: 0.2862 - val_sparse_categorical_accuracy: 0.9171
Epoch 17/20
58/58 [==============================] - 36s 623ms/step - loss: 0.0151 - sparse_categoric
al_accuracy: 1.0000 - val_loss: 0.2858 - val_sparse_categorical_accuracy: 0.9177
Epoch 18/20
58/58 [==============================] - 36s 623ms/step - loss: 0.0140 - sparse_categoric
al_accuracy: 1.0000 - val_loss: 0.2869 - val_sparse_categorical_accuracy: 0.9177
Epoch 19/20
58/58 [==============================] - 36s 622ms/step - loss: 0.0129 - sparse_categoric
al_accuracy: 1.0000 - val_loss: 0.2863 - val_sparse_categorical_accuracy: 0.9188
Epoch 20/20
58/58 [==============================] - 36s 622ms/step - loss: 0.0120 - sparse_categoric
al_accuracy: 1.0000 - val_loss: 0.2867 - val_sparse_categorical_accuracy: 0.9171
```

In [ ]:

```
#Define plot function
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]
    plt.title('Loss: %.3f, sparse_categorical_accuracy: %.3f' % (loss, acc))
```

In [ ]:

```
# Print the result
plot_loss_accuracy(history)
print(f'Test loss: {loss:.3}')
print(f'Test accuracy: {accuracy:.3}')
```

```
Test loss: 0.247
Test accuracy: 0.92

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