In [ ]:

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

```
Requirement already satisfied: tensorflow-addons in /usr/local/lib/python3.7/dist-package
s (0.17.0)
Requirement already satisfied: typeguard>=2.7 in /usr/local/lib/python3.7/dist-packages (
from tensorflow-addons) (2.7.1)
Requirement already satisfied: packaging in /usr/local/lib/python3.7/dist-packages (from
tensorflow-addons) (21.3)
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)
```

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
import tensorflow_addons as tfa
```

In [ ]:

```
# load the data from Tensorflow
tfds.disable_progress_bar()
(train, test),info = tfds.load(
    "stanford_dogs",
    split=["train", "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 test samples: %d" % tf.data.experimental.cardinality(test))
```

```
Number of training samples: 12000
Number of test samples: 8580
```

In [ ]:

```
print(info)
```

```
tfds.core.DatasetInfo(
    name='stanford_dogs',
```

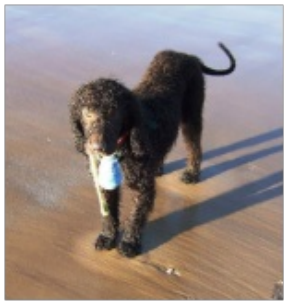
```
    version=0.2.0,
    description='The Stanford Dogs dataset contains images of 120 breeds of dogs from aro
und
the world. This dataset has been built using images and annotation from
ImageNet for the task of fine-grained image categorization. There are
20,580 images, out of which 12,000 are used for training and 8580 for
testing. Class labels and bounding box annotations are provided
for all the 12,000 images.',
    homepage='http://vision.stanford.edu/aditya86/ImageNetDogs/main.html',
    features=FeaturesDict({
        'image': Image(shape=(None, None, 3), dtype=tf.uint8),
        'image/filename': Text(shape=(), dtype=tf.string),
        'label': ClassLabel(shape=(), dtype=tf.int64, num_classes=120),
        'objects': Sequence({
            'bbox': BBoxFeature(shape=(4,), dtype=tf.float32),
        }),
    }),
    total_num_examples=20580,
    splits={
        'test': 8580,
        'train': 12000,
    },
    supervised_keys=('image', 'label'),
    citation="""@inproceedings{KhoslaYaoJayadevaprakashFeiFei_FGVC2011,
    author = "Aditya Khosla and Nityananda Jayadevaprakash and Bangpeng Yao and
              Li Fei-Fei",
    title = "Novel Dataset for Fine-Grained Image Categorization",
    booktitle = "First Workshop on Fine-Grained Visual Categorization,
                 IEEE Conference on Computer Vision and Pattern Recognition",
    year = "2011",
    month = "June",
    address = "Colorado Springs, CO",
    }
    @inproceedings{imagenet_cvpr09,
            AUTHOR = {Deng, J. and Dong, W. and Socher, R. and Li, L.-J. and
                      Li, K. and Fei-Fei, L.},
            TITLE = {{ImageNet: A Large-Scale Hierarchical Image Database}},
            BOOKTITLE = {CVPR09},
            YEAR = {2009},
            BIBSOURCE = "http://www.image-net.org/papers/imagenet_cvpr09.bib"}""",
    redistribution_info=,
)
```

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")
```



70



9

13

In [ ]:

```
#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 [ ]:

```
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 [ ]:

```
#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 [ ]:

```
#set up base model
input_shape = (224,224,3)
num_classes=120

base_model = applications.resnet_v2.ResNet101V2(input_shape=input_shape, include_top=Fal
se, 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)
```

In [ ]:

```
# compile the model
# Optimizer: Nesterov's Accelerated Gradient
alpha = 0.0001 #weight decay
momentum = 0
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 [ ]:

```
#train the model
```

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

```
Epoch 1/15
188/188 [==============================] - 140s 613ms/step - loss: 2.9108 - sparse_catego
rical_accuracy: 0.3708 - val_loss: 1.3919 - val_sparse_categorical_accuracy: 0.5963
Epoch 2/15
188/188 [==============================] - 131s 698ms/step - loss: 1.0210 - sparse_catego
rical_accuracy: 0.7369 - val_loss: 1.0962 - val_sparse_categorical_accuracy: 0.6922
Epoch 3/15
188/188 [==============================] - 111s 590ms/step - loss: 0.5373 - sparse_catego
rical_accuracy: 0.8737 - val_loss: 1.0512 - val_sparse_categorical_accuracy: 0.6974
Epoch 4/15
188/188 [==============================] - 111s 591ms/step - loss: 0.3173 - sparse_catego
rical_accuracy: 0.9373 - val_loss: 1.0014 - val_sparse_categorical_accuracy: 0.7101
Epoch 5/15
188/188 [==============================] - 111s 592ms/step - loss: 0.1968 - sparse_catego
rical_accuracy: 0.9689 - val_loss: 1.0008 - val_sparse_categorical_accuracy: 0.7100
Epoch 6/15
188/188 [==============================] - 111s 593ms/step - loss: 0.1318 - sparse_catego
rical_accuracy: 0.9837 - val_loss: 0.9681 - val_sparse_categorical_accuracy: 0.7210
Epoch 7/15
188/188 [==============================] - 132s 700ms/step - loss: 0.0906 - sparse_catego
rical_accuracy: 0.9912 - val_loss: 0.9837 - val_sparse_categorical_accuracy: 0.7157
Epoch 8/15
188/188 [==============================] - 112s 593ms/step - loss: 0.0646 - sparse_catego
rical_accuracy: 0.9956 - val_loss: 0.9954 - val_sparse_categorical_accuracy: 0.7189
Epoch 9/15
188/188 [==============================] - 111s 592ms/step - loss: 0.0510 - sparse_catego
rical_accuracy: 0.9974 - val_loss: 0.9973 - val_sparse_categorical_accuracy: 0.7168
Epoch 10/15
188/188 [==============================] - 131s 699ms/step - loss: 0.0410 - sparse_catego
rical_accuracy: 0.9987 - val_loss: 0.9950 - val_sparse_categorical_accuracy: 0.7224
Epoch 11/15
188/188 [==============================] - 131s 699ms/step - loss: 0.0361 - sparse_catego
rical_accuracy: 0.9982 - val_loss: 1.0009 - val_sparse_categorical_accuracy: 0.7219
Epoch 12/15
188/188 [==============================] - 111s 591ms/step - loss: 0.0328 - sparse_catego
rical_accuracy: 0.9985 - val_loss: 1.0111 - val_sparse_categorical_accuracy: 0.7203
Epoch 13/15
188/188 [==============================] - 111s 593ms/step - loss: 0.0288 - sparse_catego
rical_accuracy: 0.9993 - val_loss: 1.0065 - val_sparse_categorical_accuracy: 0.7234
Epoch 14/15
188/188 [==============================] - 111s 592ms/step - loss: 0.0273 - sparse_catego
rical_accuracy: 0.9994 - val_loss: 1.0090 - val_sparse_categorical_accuracy: 0.7227
Epoch 15/15
188/188 [==============================] - 131s 699ms/step - loss: 0.0253 - sparse_catego
rical_accuracy: 0.9996 - val_loss: 1.0151 - val_sparse_categorical_accuracy: 0.7216
```

In [ ]:

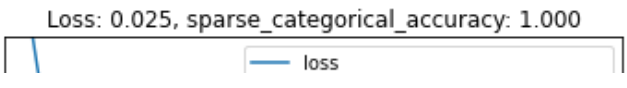
```
#Define plot function
def plot_loss_accuracy(history):
    historydf = pd.DataFrame(history.history, index=history.epoch)
    plt.figure(figsize=(8, 6))
    historydf.plot(ylim=(0, max(1, historydf.values.max())))
    loss = history.history['loss'][-1]
    acc = history.history['sparse_categorical_accuracy'][-1]
    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))
```

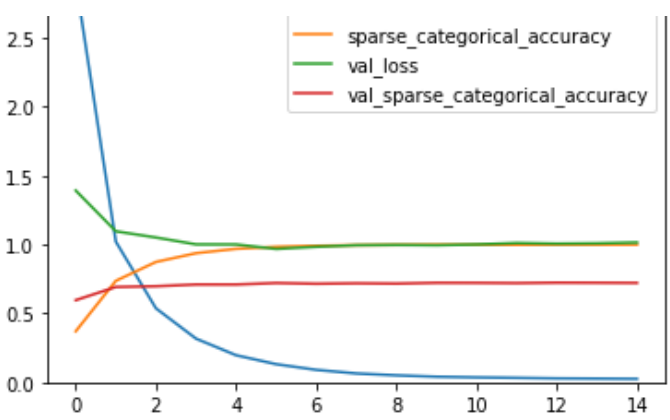
In [ ]:

```
plot_loss_accuracy(history)
```

```
Validation Error: 0.278

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




sparse_categorical_accuracy
val_loss
val_sparse_categorical_accuracy
2.5
2.0
1.5
1.0
0.5
0.0
0
2
4
6
8
10
12
14