Breeds Data Wrangling - Collect and Prepare Data and Setup for Training

October 2, 2018

1 Part 1: Data Wrangling with Breeds on CPU

2 Objective

Understand ways to find a data set and to prepare a data set for machine learning and training.

2.1 Activities

In this section of the training you will - Transfer a data set from the shared location on the server to your current directory. - View your initial data - Clean and normalize the data set - Organize the data into training and testing groups

3 Find a Data set

3.0.1 Research Existing Data Sets

Artificial intelligence projects depend upon data. When beginning a project, data scientists look for existing data sets that are similar to or match the given problem. This saves time and money, and leverages the work of others, building upon the body of knowledge for all future projects.

Typically you begin with a search engine query. For this project, we were looking for a data set with an unencumbered license.

This project starts with the Oxford IIIT Pet Data set http://www.robots.ox.ac.uk/~vgg/data/pets/ , a 37-category pet data set with roughly 200 images for each class. The images have a large variations in scale, pose, and lighting. All images have an associated ground truth annotation of breed, head region of interest (ROI), and pixel-level trimap segmentation.

3.0.2 Background

"The pet images were downloaded from Catster* and Dogster, *two social web sites dedicated to the collection and discussion of images of pets, from Flickr* groups, and from Google Images*. People uploading images to Catster and Dogster provide the breed information as well, and the Flickr groups are specific to each breed, which simplifies tagging. For each of the 37 breeds, about 2,000 – 2,500 images were downloaded from these data sources to form a pool of candidates for inclusion in the dataset. From this candidate list, images were dropped if any of the following conditions applied, as judged by the annotators: (i) the image was gray scale, (ii) another image

Fetch Data

View and Understand Your Data

portraying the same animal existed (which happens frequently in Flickr), (iii) the illumination was poor, (iv) the pet was not centered in the image, or (v) the pet was wearing clothes. The most common problem in all the data sources, however, was found to be errors in the breed labels. Thus labels were reviewed by the human annotators and fixed whenever possible. When fixing was not possible, for instance because the pet was a cross breed, the image was dropped."

From *Cats and Dogs*, http://www.robots.ox.ac.uk/~vgg/publications/2012/parkhi12a/parkhi12a.pdf

4 Fetch Your Data

4.0.1 Activity

Click the cell below and then click **Run**.

```
In [ ]: !rsync -r --progress --ignore-existing /data/aidata/breeds/original/ breeds/

        !echo "Done."
```

5 View the Baseline Data

Take a look at the images in your data set. This gives you some idea as to how much cleaning and normalizing will be required.

5.0.1 Activity

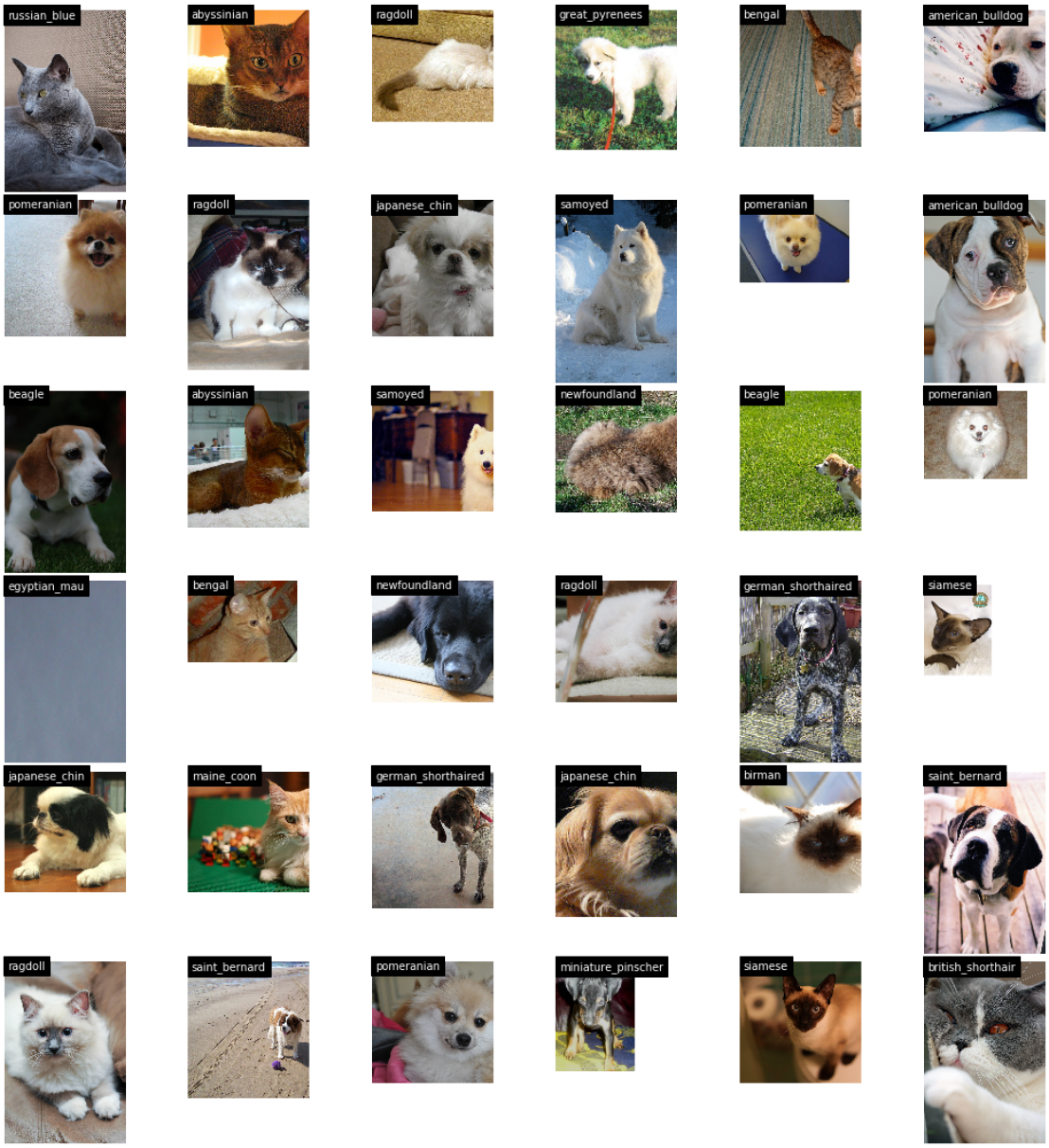
In the cell below, update the display_images function by changing the **numOfImages** parameter to a number from 1 to 5. Click **Save**, and then click **Run**.

*Hint: The display_images function sets a display grid showing NxN pet images. The default number of images is set to **?**. Change the **?** character to something greater than 1; for example, **numOfImages = 6**.*

```
In [1]: import os
        import glob
```

```
import re
import random
import matplotlib.pyplot as plt
import matplotlib.image as mpimg
%matplotlib inline

def get_category(file):
    m = re.search("\d", file, re.IGNORECASE)
    if m:
        return file[:m.start() - 1].lower().split("/")[1]

def display_images(file_names, numOfImages = 6):
    indicies = random.sample(range(len(file_names)), numOfImages * numOfImages)
    train_images = [file_names[i] for i in indicies]

    fig, axes = plt.subplots(nrows=numOfImages,ncols=numOfImages, figsize=(15,15), share
    for i,ax in enumerate(axes.flat):
        ax.get_xaxis().set_visible(False)
        ax.get_yaxis().set_visible(False)
        curr_i = train_images[i]
        imgplot = mpimg.imread(curr_i)
        ax.imshow(imgplot)
        ax.text(10,20,get_category(curr_i), fontdict={"backgroundcolor": "black","color"
        ax.axis('off')
    plt.tight_layout(h_pad=0, w_pad=0)


display_images(glob.glob('breeds/*.jpg'))

print("Done.")
```

```
Done.
```



6 Clean and Normalize the Data

Existing image recognition data-sets often include images of multiple dimensions, color mixed with black and white photos, maybe even line art plus photos. File names may follow multiple formats, and the subject matter within the images may be single, multiple, profile, straight-on face, back of head, surrounded by a complex background or more. Cleaning and normalizing the data means fixing the inconsistencies so that the machine processing can occur with minimal errors. Oftentimes data cleaning is tedious and requires significant time commitment. Data preprocessing techniques include: 1. Data cleaning Eliminates noise and resolves inconsistencies in the data. 2. Data integration Migrates data from various different sources into one coherent source, such as a

Clean and Normalize the Data

data warehouse. 3. Data transformation – Standardizes or normalizes any form of data. 4. Data reduction – Reduces the size of the data by aggregating it.

Another name for this effort is extract, transform, and load (ETL). This project required the team to normalize the file dimensions, file names and create a data layout expected by the framework.

It is common for the data cleanup tasks to be pared with framework and topology selection because different topologies expect different data layouts and formats. When experimenting with different topologies it might be necessary to have several copies of the data in various formats. Multiple copies of data-sets can take up a lot of space, so ensure you've got lots of storage and processing capability.

6.1 Activity

The code in the next cell performs some of the cleanup tasks. Review the code and notice that it is removing corrupt files, files with the wrong format, and files with incorrect metadata.

Click the cell below and then click **Run**.

```
In [2]: import cv2

        for file in glob.glob("breeds/*"):
            if not file.endswith(".jpg"):
                #Not ending in .jpg
                print("Deleting (.mat): " + file)
                os.remove(os.path.join(os.getcwd(), file))
            else:
                flags = cv2.IMREAD_COLOR
                im = cv2.imread(file, flags)

                if im is None:
                    #Can't read in image
                    print("Deleting (None): " + file)
                    os.remove(os.path.join(os.getcwd(), file))
                    continue
                elif len(im.shape) != 3:
                    #Wrong amount of channels
                    print("Deleting (len != 3): " + file)
                    os.remove(os.path.join(os.getcwd(), file))
                    continue
                elif im.shape[2] != 3:
```

```
                    #Wrong amount of channels
                    print("Deleting (shape[2] != 3): " + file)
                    os.remove(os.path.join(os.getcwd(), file))
                    continue

                with open(os.path.join(os.getcwd(), file), 'rb') as f:
                    check_chars = f.read()
                if check_chars[-2:] != b'\xff\xd9':
                    #Wrong ending metadata for jpg standard
                    print('Deleting (xd9): ' + file)
                    os.remove(os.path.join(os.getcwd(), file))
                elif check_chars[:4] != b'\xff\xd8\xff\xe0':
                    #Wrong Start Marker / JFIF Marker metadata for jpg standard
                    print('Deleting (xd8/xe0): ' + file)
                    os.remove(os.path.join(os.getcwd(), file))
                elif check_chars[6:10] != b'JFIF':
                    #Wrong Identifier metadata for jpg standard
                    print('Deleting (xd8/xe0): ' + file)
                    os.remove(os.path.join(os.getcwd(), file))
                elif "beagle_116.jpg" in file or "chihuahua_121.jpg" in file:
                    #Using EXIF Data to determine this
                    print('Deleting (corrupt jpeg data): ', file)
                    os.remove(os.path.join(os.getcwd(), file))


        print('Done.')
```

```
Deleting (xd8/xe0): breeds/Maine_Coon_10.jpg
Deleting (xd8/xe0): breeds/Bombay_88.jpg
Deleting (xd8/xe0): breeds/Egyptian_Mau_162.jpg
Deleting (None): breeds/Abyssinian_34.jpg
Deleting (xd8/xe0): breeds/Ragdoll_4.jpg
Deleting (xd9): breeds/Egyptian_Mau_138.jpg
Deleting (None): breeds/Egyptian_Mau_191.jpg
Deleting (None): breeds/Egyptian_Mau_167.jpg
Deleting (xd9): breeds/Egyptian_Mau_156.jpg
Deleting (xd8/xe0): breeds/Abyssinian_67.jpg
Deleting (None): breeds/Egyptian_Mau_177.jpg
Deleting (xd8/xe0): breeds/British_Shorthair_239.jpg
Deleting (xd8/xe0): breeds/Egyptian_Mau_19.jpg
Deleting (None): breeds/Egyptian_Mau_145.jpg
Deleting (corrupt jpeg data):  breeds/chihuahua_121.jpg
Deleting (xd8/xe0): breeds/Persian_5.jpg
Deleting (xd8/xe0): breeds/Persian_269.jpg
Deleting (xd8/xe0): breeds/Bombay_29.jpg
Deleting (xd8/xe0): breeds/Maine_Coon_213.jpg
Deleting (xd8/xe0): breeds/Russian_Blue_13.jpg
Deleting (xd8/xe0): breeds/Egyptian_Mau_8.jpg
```

```
Deleting (None): breeds/Egyptian_Mau_139.jpg
Deleting (xd8/xe0): breeds/Sphynx_222.jpg
Deleting (xd9): breeds/Egyptian_Mau_186.jpg
Deleting (xd8/xe0): breeds/Abyssinian_180.jpg
Deleting (xd8/xe0): breeds/miniature_pinscher_154.jpg
Deleting (xd8/xe0): breeds/Bombay_160.jpg
Deleting (.mat): breeds/Abyssinian_101.mat
Deleting (xd8/xe0): breeds/Abyssinian_170.jpg
Deleting (xd8/xe0): breeds/Sphynx_252.jpg
Deleting (corrupt jpeg data):  breeds/beagle_116.jpg
Deleting (xd9): breeds/Egyptian_Mau_14.jpg
Deleting (xd8/xe0): breeds/Egyptian_Mau_193.jpg
Deleting (xd8/xe0): breeds/Bengal_195.jpg
Deleting (xd9): breeds/Abyssinian_5.jpg
Deleting (xd8/xe0): breeds/Egyptian_Mau_160.jpg
Deleting (xd8/xe0): breeds/Bombay_19.jpg
Deleting (xd8/xe0): breeds/Bengal_146.jpg
Deleting (xd8/xe0): breeds/Egyptian_Mau_165.jpg
Deleting (.mat): breeds/Abyssinian_102.mat
Deleting (.mat): breeds/Abyssinian_100.mat
Done.
```

7 Augment Your Data

Most of the time you're cleaning data and removing noise. Since our app needs to work with images of wet, muddy, or injured animals, or perhaps blurry images because the animal is running away in fear, we actually need to ADD noise to the data-set.

We decided to add image noise by building a small program to flip, flop, blur, and extract color channels from the images in the dataset. These actions expanded our training data-set by 6x.

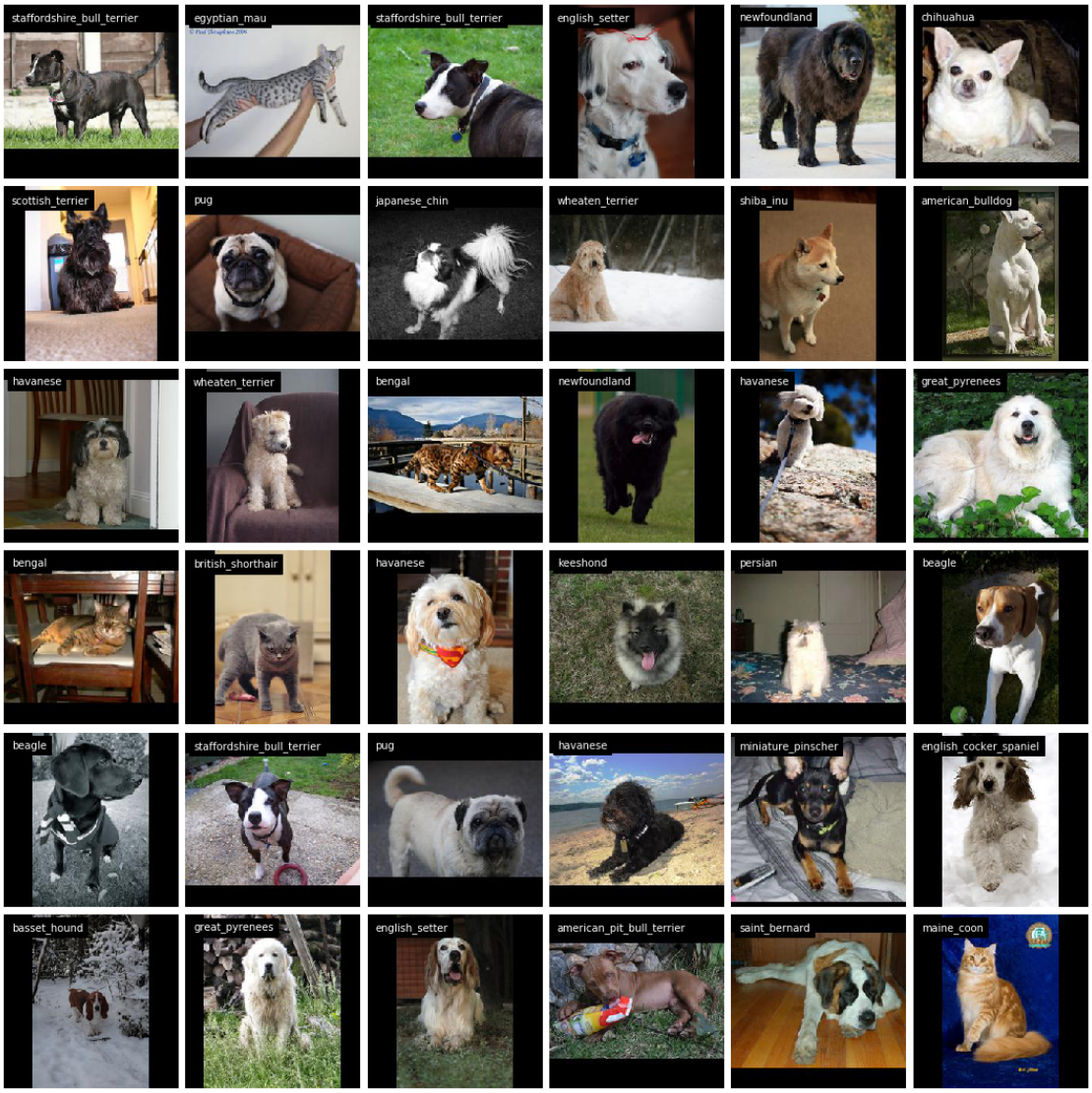
The cell below uses a parallel method to scale the image processing tasks to all available processors.

In [3]:
```
#%%bash

        #echo "Start resizing to 227x227"
        #parallel -j 200 convert {} -resize 227x227 -filter spline -unsharp 0x6+0.5+0 -backgroun
        #echo "Resizing done"

        #mkdir flop
        #echo "Start augmentation 1"
        #parallel -j 200 convert {} -flop flop/{.}-flop.jpg ::: *.jpg
        #echo "Finish augmetation 1"

        #mkdir flip
        #echo "Start augmentation 2"
        #parallel -j 200 convert {} -transverse -rotate 90 flip/{.}-flip.jpg ::: *.jpg
```

```
#echo "Finish augmetation 2"

#mkdir blur
#echo "Start augmentation 3"
#parallel -j 200 convert {} -blur 0x1 blur/{.}-blur.jpg ::: *.jpg
#echo "Finish augmetation 3"

#mkdir red
#echo "Start augmentation 4"
#parallel -j 200 convert {} -channel R -separate red/{.}-red.jpg ::: *.jpg
#echo "Finish augmetation 4"

#mkdir blue
#echo "Start augmentation 5"
#parallel -j 200 convert {} -channel B -separate blue/{.}-blue.jpg ::: *.jpg
#echo "Finish augmetation 5"

#mkdir green
#echo "Start augmentation 6"
#parallel -j 200 convert {} -channel G -separate green/{.}-green.jpg ::: *.jpg
#echo "Finish augmetation 6"

#echo "Copying augmented data to main folder"
#cp flop/* flip/* blur/* red/* blue/* green/* .

#echo "Augmentation done"

from multiprocessing import Pool
from PIL import Image
import sys

def resize_image(file, size=224):
    black_background = Image.new('RGB', (size, size), "black")
    img = Image.open(file)
    img.thumbnail((size,size))
    x, y = img.size
    black_background.paste(img, (int((size - x) / 2), int((size - y) / 2)))
    black_background.save(file)
    return black_background

pool = Pool(6)
for i, _ in enumerate(pool.map(resize_image, glob.glob("breeds/*"))):
    if i % 10 == 0:
        sys.stdout.write('\r{0} out of {1} processed'.format(i+1, len(glob.glob("breeds/

sys.stdout.write('\n')
sys.stdout.flush()
```

```
        display_images(glob.glob('breeds/*.jpg'))

        print("Done.")
```

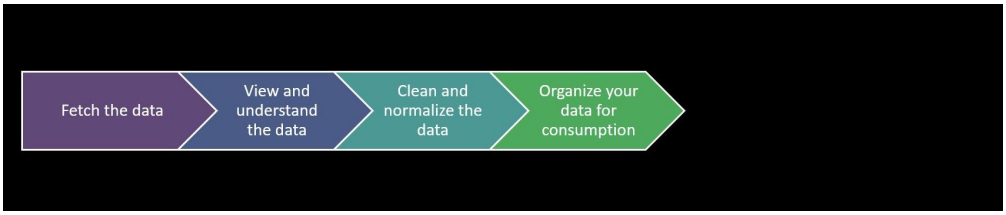
```
7351 out of 7352 processed
Done.
```



Organize Data for Consumption by TensorFlow*

The framework you choose for your project determines how you need to organize your data. After extensive experimentation we selected TensorFlow for this project. This section describes how to organize your data layers.

We are splitting the images into training and validation sets, with 80 percent of the images targeted for training and 20 percent of the images targeted for validation. Our data needs to be



Organize Data for Consumption by Framework

organized in a specific manner. That organization is to have each image in a folder that dictates which category it belongs to.

We'll create a train and a validation folder. Within those folders, we'll have directories with each category name and then the respective images within their category folder.

This next cell of code creates the data layout as expected.

7.0.1 Activity

In the cell below, set the **train_ratio** to **0.8** and then click **Run**.

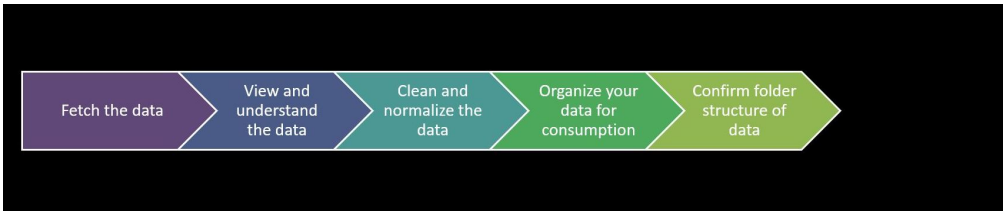
Hint: We set the train_ratio = ? to a value between 0 and 1 to define our train and validation split.

```
In [4]: import os
        import re
        import errno
        import math
        import sys

        def get_category(file):
            m = re.search("\d", file, re.IGNORECASE)
            if m:
                return file[:m.start() - 1].lower()

        def make_sure_path_exists(path):
            try:
                os.makedirs(path)
            except OSError as exception:
                if exception.errno != errno.EEXIST:
                    raise

        train_ratio = .8

        file_names = os.listdir('breeds')
        category_names = [ get_category(file) for file in file_names]
        category_names = [ name for name in category_names if name is not None ]
        category_names = sorted(list(set(category_names)))
        for category in category_names:
            make_sure_path_exists("breeds/train/" + str(category))
            make_sure_path_exists("breeds/validation/" + str(category))
```

```
        for idx, category in enumerate(category_names):
            category_list = []
            for file in file_names:
                if category.lower() in file.lower():
                    category_list.append(file)

            category_list = sorted(category_list)
            split_ratio = math.floor(len(category_list) * train_ratio)
            train_list = category_list[:split_ratio]
            validation_list = category_list[split_ratio:]
            for i, file in enumerate(train_list):
                os.rename("breeds/" + file, "breeds/train/" + str(category) + "/" + file)
                if i % 10 == 0:
                    sys.stdout.write('\r>> Moving train image %d to category folder %s' % (i+1,
                    sys.stdout.flush()

            sys.stdout.write('\n')
            sys.stdout.flush()

            for i, file in enumerate(validation_list):
                os.rename("breeds/" + file, "breeds/validation/" + str(category) + "/" + file)
                if i % 10 == 0:
                    sys.stdout.write('\r>> Moving validation image %d to category folder %s' % (
                    sys.stdout.flush()

            sys.stdout.write('\n')
            sys.stdout.flush()

        print("Done.")
```

```
>> Moving train image 151 to category folder abyssinian
>> Moving validation image 31 to category folder abyssinian
>> Moving train image 151 to category folder american_bulldog
>> Moving validation image 31 to category folder american_bulldog
>> Moving train image 151 to category folder american_pit_bull_terrier
>> Moving validation image 31 to category folder american_pit_bull_terrier
>> Moving train image 151 to category folder basset_hound
>> Moving validation image 31 to category folder basset_hound
>> Moving train image 151 to category folder beagle
>> Moving validation image 31 to category folder beagle
>> Moving train image 151 to category folder bengal
>> Moving validation image 31 to category folder bengal
>> Moving train image 151 to category folder birman
>> Moving validation image 31 to category folder birman
>> Moving train image 151 to category folder bombay
>> Moving validation image 31 to category folder bombay
>> Moving train image 151 to category folder boxer
```

```
>> Moving validation image 31 to category folder boxer
>> Moving train image 151 to category folder british_shorthair
>> Moving validation image 31 to category folder british_shorthair
>> Moving train image 151 to category folder chihuahua
>> Moving validation image 31 to category folder chihuahua
>> Moving train image 141 to category folder egyptian_mau
>> Moving validation image 31 to category folder egyptian_mau
>> Moving train image 151 to category folder english_cocker_spaniel
>> Moving validation image 31 to category folder english_cocker_spaniel
>> Moving train image 151 to category folder english_setter
>> Moving validation image 31 to category folder english_setter
>> Moving train image 151 to category folder german_shorthaired
>> Moving validation image 31 to category folder german_shorthaired
>> Moving train image 151 to category folder great_pyrenees
>> Moving validation image 31 to category folder great_pyrenees
>> Moving train image 151 to category folder havanese
>> Moving validation image 31 to category folder havanese
>> Moving train image 151 to category folder japanese_chin
>> Moving validation image 31 to category folder japanese_chin
>> Moving train image 151 to category folder keeshond
>> Moving validation image 31 to category folder keeshond
>> Moving train image 151 to category folder leonberger
>> Moving validation image 31 to category folder leonberger
>> Moving train image 151 to category folder maine_coon
>> Moving validation image 31 to category folder maine_coon
>> Moving train image 151 to category folder miniature_pinscher
>> Moving validation image 31 to category folder miniature_pinscher
>> Moving train image 151 to category folder newfoundland
>> Moving validation image 31 to category folder newfoundland
>> Moving train image 151 to category folder persian
>> Moving validation image 31 to category folder persian
>> Moving train image 151 to category folder pomeranian
>> Moving validation image 31 to category folder pomeranian
>> Moving train image 151 to category folder pug
>> Moving validation image 31 to category folder pug
>> Moving train image 151 to category folder ragdoll
>> Moving validation image 31 to category folder ragdoll
>> Moving train image 151 to category folder russian_blue
>> Moving validation image 31 to category folder russian_blue
>> Moving train image 151 to category folder saint_bernard
>> Moving validation image 31 to category folder saint_bernard
>> Moving train image 151 to category folder samoyed
>> Moving validation image 31 to category folder samoyed
>> Moving train image 151 to category folder scottish_terrier
>> Moving validation image 31 to category folder scottish_terrier
>> Moving train image 151 to category folder shiba_inu
>> Moving validation image 31 to category folder shiba_inu
>> Moving train image 151 to category folder siamese
```



Confirm Folder Structure is Correct

```
>> Moving validation image 31 to category folder siamese
>> Moving train image 151 to category folder sphynx
>> Moving validation image 31 to category folder sphynx
>> Moving train image 151 to category folder staffordshire_bull_terrier
>> Moving validation image 31 to category folder staffordshire_bull_terrier
>> Moving train image 151 to category folder wheaten_terrier
>> Moving validation image 31 to category folder wheaten_terrier
>> Moving train image 151 to category folder yorkshire_terrier
>> Moving validation image 31 to category folder yorkshire_terrier
Done.
```

8 Confirm Folder Structure is Correct

We have a sorted folder, 37 breeds folders, and pictures of those breeds within their respective folders.

8.0.1 Activity

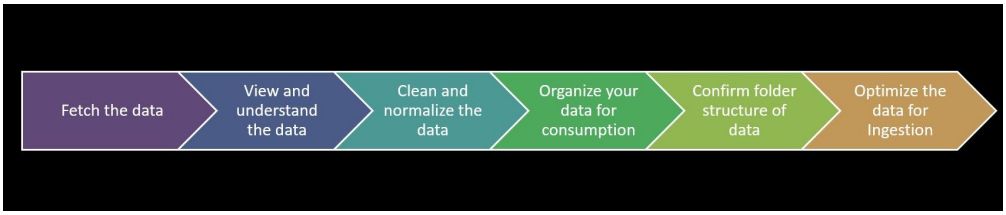
Click the cell below and then click **Run**.

```
In [ ]: for root, dirs, files in os.walk("breeds"):
            level = root.replace(os.getcwd(), '').count(os.sep)
            print('{0}{1}/'.format('    ' * level, os.path.basename(root)))
            for f in files[:5]:
                print('{0}{1}'.format('    ' * (level + 1), f))
        print("Done.")
```

9 Optimize Data for Ingestion

9.0.1 Data Input/Output

A TFRecords file represents a sequence of (binary) strings. The format is not random access, so it is suitable for streaming large amounts of data but not suitable if fast sharding or other non-sequential access is desired. See Data IO (Python Functions), https://www.tensorflow.org/api_guides/python/python_io#tfrecords_format_details



Optimize Data for Ingestion

9.0.2 Standard TensorFlow* Format

Another approach is to convert whatever data you have into a supported format. This approach makes it easier to mix and match data-sets and network architectures. The recommended format for TensorFlow is a TFRecords file containing tf.train.Example protocol buffers (which contain Features as a field). You write a little program that gets your data, stuffs it in an Example protocol buffer, serializes the protocol buffer to a string, and then writes the string to a TFRecords file using the tf.python_io.TFRecordWriter. See https://www.tensorflow.org/versions/r1.0/programmers_guide/reading_data#file_formats

9.0.3 Activity

When creating a TFRecord file you can split the dataset into shards. This can be especially beneficial if you have a particularly large dataset and don't want to end up with a single 1+GB file.

Below, we're creating shards of data based on the files into the number passed in **_NUM_SHARDS**.

In the cell below, set **_NUM_SHARDS** to a value between **1** and **5** and then click **Run**.

```
In [6]: import tensorflow as tf

        _NUM_SHARDS = 5
        _SHARD_NAME = "breeds"
        LABELS_FILENAME = 'labels.txt'

        class ImageReader(object):
            def __init__(self):
                # Initializes function that decodes RGB JPEG data.
                self._decode_jpeg_data = tf.placeholder(dtype=tf.string)
                self._decode_jpeg = tf.image.decode_jpeg(self._decode_jpeg_data, channels=3)

            def read_image_dims(self, sess, image_data):
                image = self.decode_jpeg(sess, image_data)
                return image.shape[0], image.shape[1]

            def decode_jpeg(self, sess, image_data):
                image = sess.run(self._decode_jpeg,
                                 feed_dict={self._decode_jpeg_data: image_data})
                assert len(image.shape) == 3
                assert image.shape[2] == 3
                return image
```

```
def int64_feature(values):
    if not isinstance(values, (tuple, list)):
        values = [values]
    return tf.train.Feature(int64_list=tf.train.Int64List(value=values))


def bytes_feature(values):
    return tf.train.Feature(bytes_list=tf.train.BytesList(value=[values]))

def image_to_tfexample(image_data, image_format, height, width, class_id):
    return tf.train.Example(features=tf.train.Features(feature={
      'image/encoded': bytes_feature(image_data),
      'image/format': bytes_feature(image_format),
      'image/class/label': int64_feature(class_id),
      'image/height': int64_feature(height),
      'image/width': int64_feature(width),
    }))

def write_label_file(labels_to_class_names, dataset_dir,
                     filename=LABELS_FILENAME):
    labels_filename = os.path.join(dataset_dir, filename)
    with tf.gfile.Open(labels_filename, 'w') as f:
        for label in labels_to_class_names:
            class_name = labels_to_class_names[label]
            f.write('%d:%s\n' % (label, class_name))


def _get_filenames_and_classes(dataset_dir, sorted_dir):
    breeds_root = os.path.join(dataset_dir, sorted_dir)
    directories = []
    class_names = []
    for filename in os.listdir(breeds_root):
        path = os.path.join(breeds_root, filename)
        if os.path.isdir(path):
            directories.append(path)
            class_names.append(filename)

    photo_filenames = []
    for directory in directories:
        for filename in os.listdir(directory):
            path = os.path.join(directory, filename)
            photo_filenames.append(path)

    return photo_filenames, sorted(class_names)


def _get_dataset_filename(dataset_dir, split_name, shard_id):
```

```
    output_filename = _SHARD_NAME + '_%s_%05d-of-%05d.tfrecord' % (
      split_name, shard_id, _NUM_SHARDS)
    return os.path.join(dataset_dir, output_filename)


def _convert_dataset(split_name, filenames, class_names_to_ids, dataset_dir):
    assert split_name in ['train', 'validation']

    num_per_shard = int(math.ceil(len(filenames) / float(_NUM_SHARDS)))

    with tf.Graph().as_default():
        image_reader = ImageReader()

        with tf.Session('') as sess:

            for shard_id in range(_NUM_SHARDS):
                output_filename = _get_dataset_filename(
                    dataset_dir, split_name, shard_id)

                with tf.python_io.TFRecordWriter(output_filename) as tfrecord_writer:
                    start_ndx = shard_id * num_per_shard
                    end_ndx = min((shard_id+1) * num_per_shard, len(filenames))
                    for i in range(start_ndx, end_ndx):
                        sys.stdout.write('\r>> Converting image %d/%d shard %d' % (
                            i+1, len(filenames), shard_id))
                        sys.stdout.flush()

                        # Read the filename:
                        image_data = tf.gfile.FastGFile(filenames[i], 'rb').read()
                        height, width = image_reader.read_image_dims(sess, image_data)

                        class_name = os.path.basename(os.path.dirname(filenames[i]))
                        class_id = class_names_to_ids[class_name]

                        example = image_to_tfexample(
                            image_data, b'jpg', height, width, class_id)
                        tfrecord_writer.write(example.SerializeToString())

    sys.stdout.write('\n')
    sys.stdout.flush()


def _dataset_exists(dataset_dir):
    for split_name in ['train', 'validation']:
        for shard_id in range(_NUM_SHARDS):
            output_filename = _get_dataset_filename(
              dataset_dir, split_name, shard_id)
            if not tf.gfile.Exists(output_filename):
```

```
                    return False
            return True

        print("Done.")
```

```
/home/zephyrie/anaconda3/envs/tf/lib/python3.6/site-packages/h5py/__init__.py:36: FutureWarning:
  from ._conv import register_converters as _register_converters
```

```
Done.
```

9.0.4 Activity

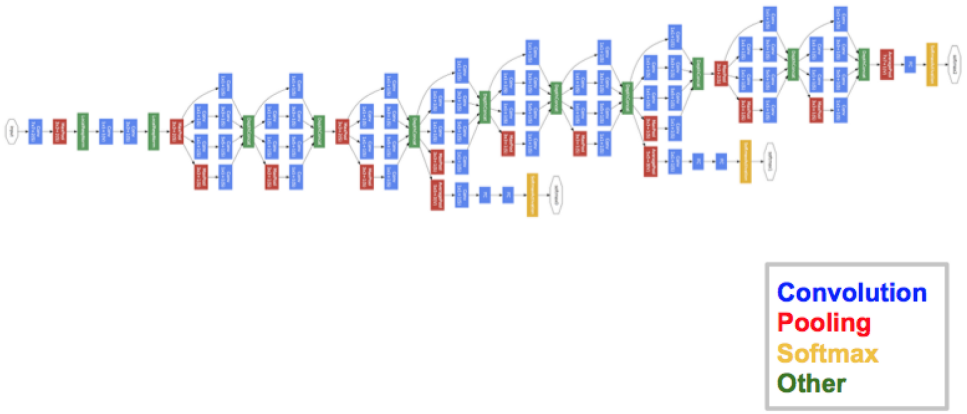
TensorFlow requires separate data sets for training and validation and that the data be stored in two separate records. Why separate image sets for training and validation? To prevent *overfitting*, which occurs when you train and test on the same images. You train on a set, then test on a new/different set to validate that the machine is truly learning to recognize the images.

Our records will contain the words **train** and **validation** in their path to distinguish between the two. We used the industry standard ratio of 80 percent train and 20 percent test/validation to split the data-set.

In the cell below, set the two function calls to **_convert_dataset_** first parameter to **"train"** and **"validation"** and then click **Run**.

*Hint: Look at the filenames being passed into the **_convert_dataset_** function and make sure you are matching that with the correct label you are replacing into the **?????**.*

```
In [7]: import random

        def run(dataset_dir):
            if not tf.gfile.Exists(dataset_dir):
                tf.gfile.MakeDirs(dataset_dir)

            if _dataset_exists(dataset_dir):
                print('Dataset files already exist. Exiting without re-creating them.')
                return

            train_photo_filenames, class_names = _get_filenames_and_classes(dataset_dir, "train"
            validation_photo_filenames, class_names = _get_filenames_and_classes(dataset_dir, "v
            class_names_to_ids = dict(zip(class_names, range(len(class_names))))

            # First, convert the training and validation sets.
            _convert_dataset("train", train_photo_filenames, class_names_to_ids,
                             dataset_dir)
            _convert_dataset("validation", validation_photo_filenames, class_names_to_ids,
                             dataset_dir)

            # Finally, write the labels file:
            labels_to_class_names = dict(zip(range(len(class_names)), class_names))
            write_label_file(labels_to_class_names, dataset_dir)
```



GoogLeNet

```
            print('\nFinished converting the Breeds dataset!')

        run('breeds')
```

```
>> Converting image 5877/5877 shard 4
>> Converting image 1475/1475 shard 4

Finished converting the Breeds dataset!
```

9.0.5 After All of This Data Wrangling We Can Actually Begin the Training Process

When we started this project, we always had an edge device in mind as our ultimate deployment platform. To that end we always considered three things when selecting our topology or network: time to train, size, and inference speed.

Time to Train: Depending on the number of layers and computation required, a network can take a significantly shorter or longer time to train. Computation time and programmer time are costly resources, so we wanted short training times.

Size: Since we're targeting an edge device and a an Intelő Movidius Neural Compute Stick stick we must consider the size of the network that is allowed in memory as well as supported networks.

Inference Speed: Typically the deeper and larger the network, the slower the inference speed. In our use case we are working with a live video stream; we want at least 10 frames per second on inference.

At this point we're going to continue with the TensorFlow framework plus the GoogLeNet Inception* v1 topology/network since we're currently working on a simpler dataset.

Download pre-trained model

10 Part 2: Training CatVsDog with TensorFlow and GoogLeNet Inception* v1 on CPU

11 Objective

Understand the stages of preparing for training using the TensorFlow framework and an GoogLeNet Inception v1 topology. You will initiate training and view a completed graph, and learn about the relationship between accuracy and loss.

12 Activities

In this section of the training you will - Download pretrained model - Clone TensorFlow/models Github* repo - Modify/add files within repo to add our dataset - Initiate training and review live training logs

12.0.1 Pretrained Models

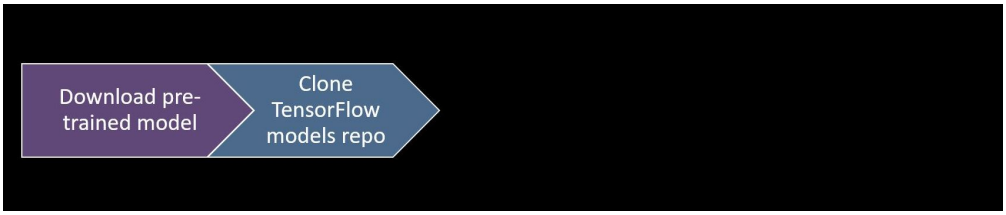
"Neural nets work best when they have many parameters, making them powerful function approximators. However, this means they must be trained on very large datasets. Because training models from scratch can be a very computationally intensive process requiring days or even weeks, we are using a pre-trained models provided by Google. This CNNs have been trained on the ILSVRC-2012-CLS image classification dataset." From https://github.com/tensorflow/models/tree/master/research/slim#pre-trained-models.

12.0.2 Activity

Click the cell below and then click **Run**.

```
In [8]: !wget http://download.tensorflow.org/models/inception_v1_2016_08_28.tar.gz
        !tar xf inception_v1_2016_08_28.tar.gz
        !rm -rf checkpoints
        !mkdir checkpoints
        !mv inception_v1.ckpt checkpoints
        !rm inception_v1_2016_08_28.tar.gz
        !echo "Done."
```

```
--2018-10-02 16:58:02--  http://download.tensorflow.org/models/inception_v1_2016_08_28.tar.gz
Resolving download.tensorflow.org (download.tensorflow.org)... 172.217.14.240, 2607:f8b0:400a:80
Connecting to download.tensorflow.org (download.tensorflow.org)|172.217.14.240|:80... connected.
```



Clone TensorFlow Models Repo

```
HTTP request sent, awaiting response... 200 OK
Length: 24642554 (24M) [application/x-tar]
Saving to: inception_v1_2016_08_28.tar.gz

inception_v1_2016_0 100%[===================>]  23.50M  94.5MB/s    in 0.2s

2018-10-02 16:58:03 (94.5 MB/s) - inception_v1_2016_08_28.tar.gz saved [24642554/24642554]

Done.
```

12.0.3 TensorFlow/Models

The TensorFlow team provides nice wrappers around a lot of functionality that needs to be done when training using TensorFlow. Below, we're going to pull in one of these repos directly so that we have access to those wrappers.

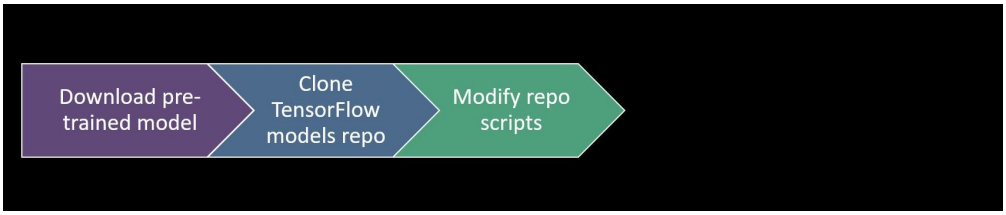
12.0.4 Activity

Click the cell below and then click **Run**.

```
In [9]: !git clone --depth 1 https://github.com/tensorflow/models
        !echo "Done."

Cloning into 'models'...
remote: Enumerating objects: 2973, done.
remote: Counting objects: 100% (2973/2973), done.
remote: Compressing objects: 100% (2574/2574), done.
remote: Total 2973 (delta 507), reused 1779 (delta 323), pack-reused 0
Receiving objects: 100% (2973/2973), 376.91 MiB | 27.55 MiB/s, done.
Resolving deltas: 100% (507/507), done.
Checking connectivity... done.
Done.
```

12.0.5 Adding to the Slim Datasets

To use the wrappers we're going to have to modify and add some existing code to the repo. Below we're overwriting the **dataset_factory.py** file with a slightly modified version that knows about our breeds dataset and an additional Python* import statement. We're also copying over



Modify Repo Scripts

breeds.py since this contains information specific to our dataset that will be utilized by the **dataset_factory**.

12.0.6 Activity

Click the cell below and then click **Run**.

```
In [10]: !cp breeds.py models/research/slim/datasets/breeds.py
         !cp dataset_factory_modified.py models/research/slim/datasets/dataset_factory.py
         !cp train_image_classifier_modified.py models/research/slim/train_image_classifier.py
         !echo "Done."

Done.
```

13 Start Training

Let's start training with TensorFlow.

CPUs, which includes Intelő Xeon Phi processors, achieve optimal performance when TensorFlow is built from source with all of the instructions supported by the target CPU.

Beyond using the latest instruction sets, Intel has added support for the Intelő Math Kernel Library for Deep Neural Networks (Intelő MKL-DNN) to TensorFlow. While the name is not completely accurate, these optimizations are often simply referred to as MKL or *TensorFlow with MKL*. TensorFlow with Intel MKL-DNN contains details on the Intelő MKL optimizations.

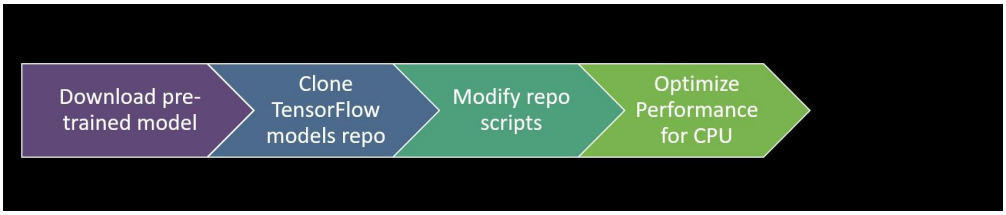
The two configurations listed below are used to optimize CPU performance by adjusting the thread pools.

- **intra_op_parallelism_threads**: Nodes that can use multiple threads to parallelize their execution will schedule the individual pieces into this pool.
- **inter_op_parallelism_threads**: All ready nodes are scheduled in this pool.

These configurations are set via the tf.ConfigProto and passed to tf.Session in the config attribute as shown in the snippet below. For both configuration options, if they are unset or set to zero, will default to the number of logical CPU cores. Testing has shown that the default is effective for systems ranging from one CPU with 4 cores to multiple CPUs with 70+ combined logical cores. A common alternative optimization is to set the number of threads in both pools equal to the number of physical cores rather than logical cores.

Intel MKL uses the following environment variables to tune performance:

KMP_BLOCKTIME - Sets the time, in milliseconds, that a thread should wait, after completing the execution of a parallel region, before sleeping.



Optimize Performance for CPU

KMP_AFFINITY - Enables the runtime library to bind threads to physical processing units.

KMP_SETTINGS - Enables (true) or disables (false) the printing of OpenMP* runtime library environment variables during program execution.

OMP_NUM_THREADS - Specifies the number of threads to use.

See *Optimizing for CPU*, https://www.tensorflow.org/performance/performance_guide#optimizing_for_cpu

Best Settings for Intelő Xeon Processor - 5th Generation (2 Socket -- 44 Cores)

Benchmark	Data Format	Inter_op	Intra_op	KMP_BLOCKTIME	Batch size
ConvNet- AlexnetNet	NCHW	1	44	30	2048
ConvNet-Googlenet V1	NCHW	2	44	1	256
ConvNet-VGG	NCHW	1	44	1	128

13.0.1 Activity

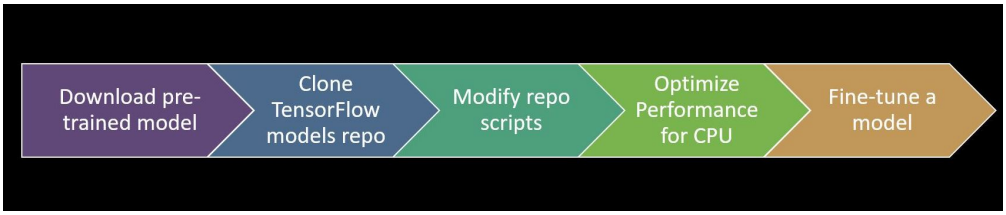
In the cell below, update **OMP_NUM_THREADS** to **"12"**, **KMP_BLOCKTIME** to **"1"**, and then click **Run**.

```
In [11]: import os
         import tensorflow as tf

         os.environ["KMP_BLOCKTIME"] = "1"
         os.environ["KMP_AFFINITY"] = "granularity=fine,compact,1,0"
         os.environ["KMP_SETTINGS"] = "0"
         os.environ["OMP_NUM_THREADS"] = "44"
         print("Done.")

Done.
```

13.0.2 Fine-Tuning a Model from an Existing Checkpoint

"Rather than training from scratch, we'll often want to start from a pre-trained model and fine-tune it. To indicate a checkpoint from which to fine-tune, we'll call training with the --checkpoint_path flag and assign it an absolute path to a checkpoint file.

When fine-tuning a model, we need to be careful about restoring checkpoint weights. In particular, when we fine-tune a model on a new task with a different number of output labels, we wont be able restore the final logits (classifier) layer. For this, we'll use the --checkpoint_exclude_scopes flag. This flag hinders certain variables from being loaded. When fine-tuning on a classification task using a different number of classes than the trained model, the new model will have a final 'logits' layer whose dimensions differ from the pre-trained model. For example, if fine-tuning an ImageNet-trained model on Flowers, the pre-trained logits layer will have dimensions [2048 x



Fine-Tune a Model

1001] but our new logits layer will have dimensions [2048 x 5]. Consequently, this flag indicates to TF-Slim to avoid loading these weights from the checkpoint.

Keep in mind that warm-starting from a checkpoint affects the model's weights only during the initialization of the model. Once a model has started training, a new checkpoint will be created in --train_dir. If the fine-tuning training is stopped and restarted, this new checkpoint will be the one from which weights are restored and not the --checkpoint_path. Consequently, the flags --checkpoint_path and --checkpoint_exclude_scopes are only used during the 0-th global step (model initialization). Typically for fine-tuning one only want train a sub-set of layers, so the flag --trainable_scopes allows to specify which subsets of layers should trained, the rest would remain frozen." See https://github.com/tensorflow/models/tree/master/research/slim#fine-tuning-a-model-from-an-existing-checkpoint.

13.0.3 Activity

In the cell below, update the **max_number_of_steps** parameter to a number between **500** and **1500**, the **intra_op** parameter to the number **12** and then click **Run**.

```
In [12]: !rm -rf train_dir
         !mkdir train_dir

         !python models/research/slim/train_image_classifier.py \
             --train_dir=train_dir \
             --dataset_name=breeds \
             --dataset_split_name=train \
             --clone_on_cpu=true \
             --dataset_dir=breeds \
             --model_name=inception_v1 \
             --checkpoint_path=checkpoints/inception_v1.ckpt \
             --checkpoint_exclude_scopes=InceptionV1/Logits \
             --trainable_scopes=InceptionV1/Logits \
             --max_number_of_steps=1500 \
             --learning_rate=0.01 \
             --batch_size=32 \
             --save_interval_secs=60 \
             --save_summaries_secs=60 \
             --inter_op=2 \
             --intra_op=44

         !echo "Done."
```

```
/home/zephyrie/anaconda3/envs/tf/lib/python3.6/site-packages/h5py/__init__.py:36: FutureWarning:
  from ._conv import register_converters as _register_converters
WARNING:tensorflow:From models/research/slim/train_image_classifier.py:406: create_global_step (
Instructions for updating:
Please switch to tf.train.create_global_step
WARNING:tensorflow:From models/research/slim/train_image_classifier.py:474: softmax_cross_entrop
Instructions for updating:
Use tf.losses.softmax_cross_entropy instead. Note that the order of the logits and labels argume
WARNING:tensorflow:From /home/zephyrie/anaconda3/envs/tf/lib/python3.6/site-packages/tensorflow/
Instructions for updating:

Future major versions of TensorFlow will allow gradients to flow
into the labels input on backprop by default.

See @{tf.nn.softmax_cross_entropy_with_logits_v2}.

WARNING:tensorflow:From /home/zephyrie/anaconda3/envs/tf/lib/python3.6/site-packages/tensorflow/
Instructions for updating:
Use tf.losses.compute_weighted_loss instead.
WARNING:tensorflow:From /home/zephyrie/anaconda3/envs/tf/lib/python3.6/site-packages/tensorflow/
Instructions for updating:
Use tf.losses.add_loss instead.
INFO:tensorflow:Fine-tuning from checkpoints/inception_v1.ckpt
WARNING:tensorflow:From /home/zephyrie/anaconda3/envs/tf/lib/python3.6/site-packages/tensorflow/
Instructions for updating:
Please switch to tf.train.MonitoredTrainingSession
2018-10-02 16:58:33.661333: I tensorflow/core/platform/cpu_feature_guard.cc:141] Your CPU suppor
INFO:tensorflow:Restoring parameters from checkpoints/inception_v1.ckpt
INFO:tensorflow:Running local_init_op.
INFO:tensorflow:Done running local_init_op.
INFO:tensorflow:Starting Session.
INFO:tensorflow:Saving checkpoint to path train_dir/model.ckpt
INFO:tensorflow:Starting Queues.
INFO:tensorflow:global_step/sec: 0
INFO:tensorflow:Recording summary at step 5.
INFO:tensorflow:global step 10: loss = 3.1051 (0.161 sec/step)
INFO:tensorflow:global step 20: loss = 1.7931 (0.145 sec/step)
INFO:tensorflow:global step 30: loss = 2.8675 (0.144 sec/step)
INFO:tensorflow:global step 40: loss = 2.6198 (0.144 sec/step)
INFO:tensorflow:global step 50: loss = 1.9375 (0.142 sec/step)
INFO:tensorflow:global step 60: loss = 2.4671 (0.143 sec/step)
INFO:tensorflow:global step 70: loss = 2.0565 (0.143 sec/step)
INFO:tensorflow:global step 80: loss = 2.4098 (0.151 sec/step)
INFO:tensorflow:global step 90: loss = 2.8326 (0.142 sec/step)
INFO:tensorflow:global step 100: loss = 2.1472 (0.144 sec/step)
INFO:tensorflow:global step 110: loss = 1.8979 (0.144 sec/step)
INFO:tensorflow:global step 120: loss = 2.0021 (0.143 sec/step)
INFO:tensorflow:global step 130: loss = 2.1512 (0.143 sec/step)
```

```
INFO:tensorflow:global step 140: loss = 1.7888 (0.142 sec/step)
INFO:tensorflow:global step 150: loss = 1.6095 (0.142 sec/step)
INFO:tensorflow:global step 160: loss = 1.2775 (0.146 sec/step)
INFO:tensorflow:global step 170: loss = 1.7342 (0.143 sec/step)
INFO:tensorflow:global step 180: loss = 0.8412 (0.142 sec/step)
INFO:tensorflow:global step 190: loss = 0.8679 (0.143 sec/step)
INFO:tensorflow:global step 200: loss = 1.4060 (0.148 sec/step)
INFO:tensorflow:global step 210: loss = 1.6273 (0.140 sec/step)
INFO:tensorflow:global step 220: loss = 0.7771 (0.137 sec/step)
INFO:tensorflow:global step 230: loss = 1.7468 (0.138 sec/step)
INFO:tensorflow:global step 240: loss = 2.0340 (0.138 sec/step)
INFO:tensorflow:global step 250: loss = 1.0088 (0.143 sec/step)
INFO:tensorflow:global step 260: loss = 1.6544 (0.134 sec/step)
INFO:tensorflow:global step 270: loss = 1.6340 (0.137 sec/step)
INFO:tensorflow:global step 280: loss = 1.5845 (0.136 sec/step)
INFO:tensorflow:global step 290: loss = 1.5778 (0.137 sec/step)
INFO:tensorflow:global step 300: loss = 1.5472 (0.133 sec/step)
INFO:tensorflow:global step 310: loss = 1.0936 (0.139 sec/step)
INFO:tensorflow:global step 320: loss = 1.2296 (0.137 sec/step)
INFO:tensorflow:global step 330: loss = 0.9902 (0.138 sec/step)
INFO:tensorflow:global step 340: loss = 1.6850 (0.138 sec/step)
INFO:tensorflow:global step 350: loss = 0.9676 (0.136 sec/step)
INFO:tensorflow:global step 360: loss = 1.1277 (0.135 sec/step)
INFO:tensorflow:Saving checkpoint to path train_dir/model.ckpt
INFO:tensorflow:global_step/sec: 6.19554
INFO:tensorflow:global step 370: loss = 1.2474 (0.179 sec/step)
INFO:tensorflow:Recording summary at step 375.
INFO:tensorflow:global step 380: loss = 0.8035 (0.150 sec/step)
INFO:tensorflow:global step 390: loss = 1.5594 (0.144 sec/step)
INFO:tensorflow:global step 400: loss = 1.0750 (0.142 sec/step)
INFO:tensorflow:global step 410: loss = 0.8108 (0.141 sec/step)
INFO:tensorflow:global step 420: loss = 1.2183 (0.139 sec/step)
INFO:tensorflow:global step 430: loss = 1.2743 (0.142 sec/step)
INFO:tensorflow:global step 440: loss = 1.5252 (0.141 sec/step)
INFO:tensorflow:global step 450: loss = 1.3485 (0.140 sec/step)
INFO:tensorflow:global step 460: loss = 1.0892 (0.140 sec/step)
INFO:tensorflow:global step 470: loss = 1.6199 (0.139 sec/step)
INFO:tensorflow:global step 480: loss = 1.4286 (0.142 sec/step)
INFO:tensorflow:global step 490: loss = 1.1253 (0.141 sec/step)
INFO:tensorflow:global step 500: loss = 1.9860 (0.141 sec/step)
INFO:tensorflow:global step 510: loss = 1.0477 (0.140 sec/step)
INFO:tensorflow:global step 520: loss = 1.0629 (0.141 sec/step)
INFO:tensorflow:global step 530: loss = 1.3634 (0.139 sec/step)
INFO:tensorflow:global step 540: loss = 1.4148 (0.140 sec/step)
INFO:tensorflow:global step 550: loss = 0.7942 (0.140 sec/step)
INFO:tensorflow:global step 560: loss = 0.9288 (0.142 sec/step)
INFO:tensorflow:global step 570: loss = 0.9254 (0.138 sec/step)
INFO:tensorflow:global step 580: loss = 1.2159 (0.137 sec/step)
```

```
INFO:tensorflow:global step 590: loss = 1.5387 (0.139 sec/step)
INFO:tensorflow:global step 600: loss = 1.2282 (0.138 sec/step)
INFO:tensorflow:global step 610: loss = 1.4907 (0.139 sec/step)
INFO:tensorflow:global step 620: loss = 1.6897 (0.142 sec/step)
INFO:tensorflow:global step 630: loss = 1.2541 (0.141 sec/step)
INFO:tensorflow:global step 640: loss = 0.9589 (0.141 sec/step)
INFO:tensorflow:global step 650: loss = 1.3972 (0.141 sec/step)
INFO:tensorflow:global step 660: loss = 1.4240 (0.140 sec/step)
INFO:tensorflow:global step 670: loss = 1.3885 (0.143 sec/step)
INFO:tensorflow:global step 680: loss = 0.9355 (0.139 sec/step)
INFO:tensorflow:global step 690: loss = 1.6813 (0.138 sec/step)
INFO:tensorflow:global step 700: loss = 1.6678 (0.140 sec/step)
INFO:tensorflow:global step 710: loss = 1.0907 (0.138 sec/step)
INFO:tensorflow:global step 720: loss = 2.7130 (0.140 sec/step)
INFO:tensorflow:global step 730: loss = 0.7533 (0.144 sec/step)
INFO:tensorflow:global step 740: loss = 1.1892 (0.141 sec/step)
INFO:tensorflow:global step 750: loss = 1.3621 (0.144 sec/step)
INFO:tensorflow:global step 760: loss = 0.9082 (0.136 sec/step)
INFO:tensorflow:Saving checkpoint to path train_dir/model.ckpt
INFO:tensorflow:global step 770: loss = 1.3040 (1.167 sec/step)
INFO:tensorflow:global_step/sec: 6.75657
INFO:tensorflow:global step 780: loss = 0.9775 (0.160 sec/step)
INFO:tensorflow:Recording summary at step 780.
INFO:tensorflow:global step 790: loss = 1.2563 (0.140 sec/step)
INFO:tensorflow:global step 800: loss = 1.4683 (0.141 sec/step)
INFO:tensorflow:global step 810: loss = 1.2757 (0.145 sec/step)
INFO:tensorflow:global step 820: loss = 0.9695 (0.141 sec/step)
INFO:tensorflow:global step 830: loss = 0.9695 (0.140 sec/step)
INFO:tensorflow:global step 840: loss = 1.2847 (0.141 sec/step)
INFO:tensorflow:global step 850: loss = 1.4966 (0.142 sec/step)
INFO:tensorflow:global step 860: loss = 0.5865 (0.138 sec/step)
INFO:tensorflow:global step 870: loss = 1.0782 (0.140 sec/step)
INFO:tensorflow:global step 880: loss = 1.2805 (0.139 sec/step)
INFO:tensorflow:global step 890: loss = 1.2885 (0.141 sec/step)
INFO:tensorflow:global step 900: loss = 0.9977 (0.136 sec/step)
INFO:tensorflow:global step 910: loss = 0.9235 (0.138 sec/step)
INFO:tensorflow:global step 920: loss = 1.9766 (0.143 sec/step)
INFO:tensorflow:global step 930: loss = 1.1444 (0.146 sec/step)
INFO:tensorflow:global step 940: loss = 1.9816 (0.143 sec/step)
INFO:tensorflow:global step 950: loss = 1.5270 (0.143 sec/step)
INFO:tensorflow:global step 960: loss = 1.6379 (0.137 sec/step)
INFO:tensorflow:global step 970: loss = 1.4791 (0.137 sec/step)
INFO:tensorflow:global step 980: loss = 1.6431 (0.139 sec/step)
INFO:tensorflow:global step 990: loss = 0.8863 (0.141 sec/step)
INFO:tensorflow:global step 1000: loss = 1.7450 (0.148 sec/step)
INFO:tensorflow:global step 1010: loss = 1.6122 (0.142 sec/step)
INFO:tensorflow:global step 1020: loss = 0.6652 (0.143 sec/step)
INFO:tensorflow:global step 1030: loss = 0.9683 (0.138 sec/step)
```

```
INFO:tensorflow:global step 1040: loss = 1.7766 (0.140 sec/step)
INFO:tensorflow:global step 1050: loss = 1.4346 (0.137 sec/step)
INFO:tensorflow:global step 1060: loss = 0.8594 (0.137 sec/step)
INFO:tensorflow:global step 1070: loss = 0.7697 (0.140 sec/step)
INFO:tensorflow:global step 1080: loss = 0.9524 (0.137 sec/step)
INFO:tensorflow:global step 1090: loss = 1.5209 (0.142 sec/step)
INFO:tensorflow:global step 1100: loss = 0.9119 (0.141 sec/step)
INFO:tensorflow:global step 1110: loss = 1.3779 (0.137 sec/step)
INFO:tensorflow:global step 1120: loss = 1.6174 (0.142 sec/step)
INFO:tensorflow:global step 1130: loss = 0.8075 (0.139 sec/step)
INFO:tensorflow:global step 1140: loss = 1.1304 (0.141 sec/step)
INFO:tensorflow:global step 1150: loss = 0.9756 (0.142 sec/step)
INFO:tensorflow:global step 1160: loss = 1.6580 (0.141 sec/step)
INFO:tensorflow:global step 1170: loss = 1.2977 (0.136 sec/step)
INFO:tensorflow:Saving checkpoint to path train_dir/model.ckpt
INFO:tensorflow:global step 1180: loss = 0.6770 (0.179 sec/step)
INFO:tensorflow:global_step/sec: 6.85608
INFO:tensorflow:global step 1190: loss = 0.8630 (0.163 sec/step)
INFO:tensorflow:Recording summary at step 1190.
INFO:tensorflow:global step 1200: loss = 1.4339 (0.142 sec/step)
INFO:tensorflow:global step 1210: loss = 1.2983 (0.141 sec/step)
INFO:tensorflow:global step 1220: loss = 1.4205 (0.142 sec/step)
INFO:tensorflow:global step 1230: loss = 0.9015 (0.142 sec/step)
INFO:tensorflow:global step 1240: loss = 1.2431 (0.137 sec/step)
INFO:tensorflow:global step 1250: loss = 1.2796 (0.136 sec/step)
INFO:tensorflow:global step 1260: loss = 1.1537 (0.137 sec/step)
INFO:tensorflow:global step 1270: loss = 1.1076 (0.141 sec/step)
INFO:tensorflow:global step 1280: loss = 1.6768 (0.137 sec/step)
INFO:tensorflow:global step 1290: loss = 1.1467 (0.142 sec/step)
INFO:tensorflow:global step 1300: loss = 1.0258 (0.142 sec/step)
INFO:tensorflow:global step 1310: loss = 0.8574 (0.140 sec/step)
INFO:tensorflow:global step 1320: loss = 1.0024 (0.137 sec/step)
INFO:tensorflow:global step 1330: loss = 0.8005 (0.142 sec/step)
INFO:tensorflow:global step 1340: loss = 1.7092 (0.150 sec/step)
INFO:tensorflow:global step 1350: loss = 1.5530 (0.144 sec/step)
INFO:tensorflow:global step 1360: loss = 0.9054 (0.138 sec/step)
INFO:tensorflow:global step 1370: loss = 0.4180 (0.142 sec/step)
INFO:tensorflow:global step 1380: loss = 1.3068 (0.146 sec/step)
INFO:tensorflow:global step 1390: loss = 0.5247 (0.141 sec/step)
INFO:tensorflow:global step 1400: loss = 2.7704 (0.141 sec/step)
INFO:tensorflow:global step 1410: loss = 1.1973 (0.146 sec/step)
INFO:tensorflow:global step 1420: loss = 1.3894 (0.140 sec/step)
INFO:tensorflow:global step 1430: loss = 0.6713 (0.144 sec/step)
INFO:tensorflow:global step 1440: loss = 1.6653 (0.147 sec/step)
INFO:tensorflow:global step 1450: loss = 1.1555 (0.143 sec/step)
INFO:tensorflow:global step 1460: loss = 1.1007 (0.143 sec/step)
INFO:tensorflow:global step 1470: loss = 0.9594 (0.145 sec/step)
INFO:tensorflow:global step 1480: loss = 0.9613 (0.143 sec/step)
```

Evaluate Your Checkpoint

```
INFO:tensorflow:global step 1490: loss = 1.2351 (0.143 sec/step)
INFO:tensorflow:global step 1500: loss = 1.0924 (0.142 sec/step)
INFO:tensorflow:Stopping Training.
INFO:tensorflow:Finished training! Saving model to disk.
Done.
```

14 Part 3: Evaluate, Freeze and Test Your Training Results

14.0.1 Evaluate Your Latest Training Checkpoint

Earlier we created a TFRecord file with our validation images. Below, we'll be using our validation set to determine our accuracy by running the eval_image_classifier script. It will give us the Accuracy and Recall for Top 5.

14.0.2 Activity

Click the cell below and then click **Run**.

```
In [13]: !rm -rf eval_dir
         !mkdir eval_dir
         !python models/research/slim/eval_image_classifier.py \
             --checkpoint_path=$(ls -t train_dir/model.ckpt* | head -1 | rev | cut -d '.' -f2- |
             --eval_dir=eval_dir \
             --dataset_dir=breeds \
             --dataset_name=breeds \
             --dataset_split_name=validation \
             --model_name=inception_v1

         !echo "Done."
```

```
/home/zephyrie/anaconda3/envs/tf/lib/python3.6/site-packages/h5py/__init__.py:36: FutureWarning:
  from ._conv import register_converters as _register_converters
WARNING:tensorflow:From models/research/slim/eval_image_classifier.py:94: get_or_create_global_s
Instructions for updating:
Please switch to tf.train.get_or_create_global_step
INFO:tensorflow:Scale of 0 disables regularizer.
WARNING:tensorflow:From models/research/slim/eval_image_classifier.py:161: streaming_accuracy (f
Instructions for updating:
```

```
Please switch to tf.metrics.accuracy. Note that the order of the labels and predictions argument
WARNING:tensorflow:From models/research/slim/eval_image_classifier.py:163: streaming_recall_at_k
Instructions for updating:
Please use `streaming_sparse_recall_at_k`, and reshape labels from [batch_size] to [batch_size,
WARNING:tensorflow:From /home/zephyrie/anaconda3/envs/tf/lib/python3.6/site-packages/tensorflow/
Instructions for updating:
Please switch to tf.metrics.mean
INFO:tensorflow:Evaluating train_dir/model.ckpt-1500
INFO:tensorflow:Starting evaluation at 2018-10-03-00:02:31
INFO:tensorflow:Graph was finalized.
2018-10-02 17:02:31.842295: I tensorflow/core/platform/cpu_feature_guard.cc:141] Your CPU suppor
2018-10-02 17:02:31.856482: I tensorflow/core/common_runtime/process_util.cc:69] Creating new th
INFO:tensorflow:Restoring parameters from train_dir/model.ckpt-1500
INFO:tensorflow:Running local_init_op.
INFO:tensorflow:Done running local_init_op.
INFO:tensorflow:Evaluation [1/15]
INFO:tensorflow:Evaluation [2/15]
INFO:tensorflow:Evaluation [3/15]
INFO:tensorflow:Evaluation [4/15]
INFO:tensorflow:Evaluation [5/15]
INFO:tensorflow:Evaluation [6/15]
INFO:tensorflow:Evaluation [7/15]
INFO:tensorflow:Evaluation [8/15]
INFO:tensorflow:Evaluation [9/15]
INFO:tensorflow:Evaluation [10/15]
INFO:tensorflow:Evaluation [11/15]
INFO:tensorflow:Evaluation [12/15]
INFO:tensorflow:Evaluation [13/15]
INFO:tensorflow:Evaluation [14/15]
INFO:tensorflow:Evaluation [15/15]
eval/Recall_5[0.979333341]
eval/Accuracy[0.757333338]
INFO:tensorflow:Finished evaluation at 2018-10-03-00:02:41
Done.
```

14.0.3 Export Your Inference Graph of Inception v1

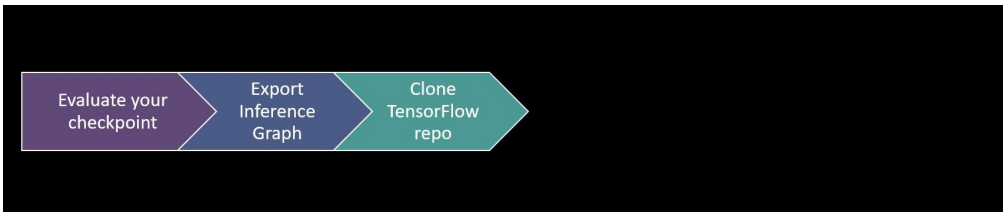
We want to export our inference graph of Inception v1 so we can use it later to create a frozen graph (.pb) file. Below, we'll run the export_inference_graph script that will take the inceptionv1 model and our dataset to create a .pb file. Passing in our dataset is important since it will make sure to create a final layer of 37 categories rather than the 1000 from ImageNet.

14.0.4 Activity

Click the cell below and then click **Run**.

```
In [14]: !python models/research/slim/export_inference_graph.py \
             --alsologtostderr \
```

Export Inference Graph



Clone TensorFlow Repo

```
    --model_name=inception_v1 \
    --image_size=224 \
    --batch_size=1 \
    --output_file=train_dir/inception_v1_inf_graph.pb \
    --dataset_name=breeds

!echo "Done."
```

```
/home/zephyrie/anaconda3/envs/tf/lib/python3.6/site-packages/h5py/__init__.py:36: FutureWarning:
  from ._conv import register_converters as _register_converters
INFO:tensorflow:Scale of 0 disables regularizer.
Done.
```

14.0.5 Clone the Main TensorfFow Repo

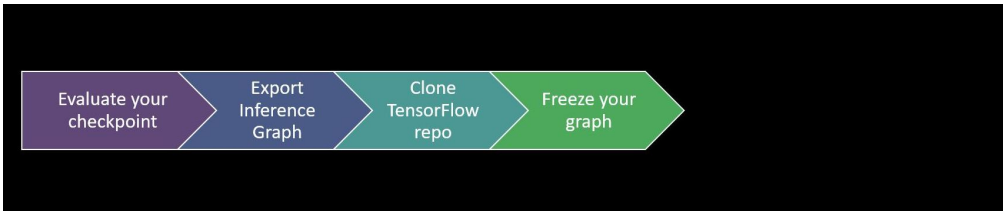
We're cloning the main TensorFlow/TensorFlow repository since it contains the script to create a frozen graph.

14.0.6 Activity

Click the cell below and then click **Run**.

```
In [15]: !git clone --depth 1 https://github.com/tensorflow/tensorflow.git

         !echo "Done."
```

```
Cloning into 'tensorflow'...
remote: Enumerating objects: 13632, done.
remote: Counting objects: 100% (13632/13632), done.
remote: Compressing objects: 100% (10925/10925), done.
remote: Total 13632 (delta 4300), reused 5065 (delta 2390), pack-reused 0
```



Freeze Your Graph

```
Receiving objects: 100% (13632/13632), 28.76 MiB | 10.77 MiB/s, done.
Resolving deltas: 100% (4300/4300), done.
Checking connectivity... done.
Done.
```

14.0.7 Freeze Your Graph

Freezing your graph will take the inference graph definition we created above and the latest checkpoint file that was created during training. It will merge these two into a single file for a convenient way to have the graph definition and weights for deployment.

14.0.8 Activity

Click the cell below and then click **Run**.

```
In [24]: !python tensorflow/tensorflow/python/tools/freeze_graph.py \
             --clear_devices=true \
             --input_graph=train_dir/inception_v1_inf_graph.pb \
             --input_checkpoint=$(ls -t train_dir/model.ckpt* | head -1 | rev | cut -d '.' -f2-
             --input_binary=true \
             --output_graph=train_dir/frozen_inception_v1.pb \
             --output_node_names=InceptionV1/Logits/Predictions/Reshape_1

         !echo "Done."
```

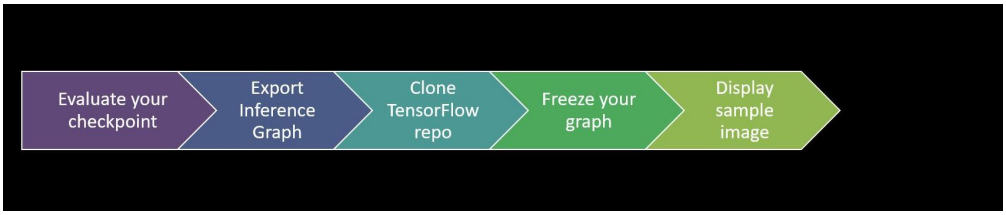
```
/home/zephyrie/anaconda3/envs/tf/lib/python3.6/site-packages/h5py/__init__.py:36: FutureWarning:
  from ._conv import register_converters as _register_converters
2018-10-02 17:13:59.157161: I tensorflow/core/platform/cpu_feature_guard.cc:141] Your CPU suppor
2018-10-02 17:13:59.173055: I tensorflow/core/common_runtime/process_util.cc:69] Creating new th
Done.
```

14.0.9 Look at a Sample Image

We're going to use this image to run through the network and see the results.

14.0.10 Activity

Click the cell below and then click **Run**.



Display a Sample Image

```
In [17]: from PIL import Image

         Image.open('breeds/train/maine_coon/Maine_Coon_100.jpg')
```

Out[17]:

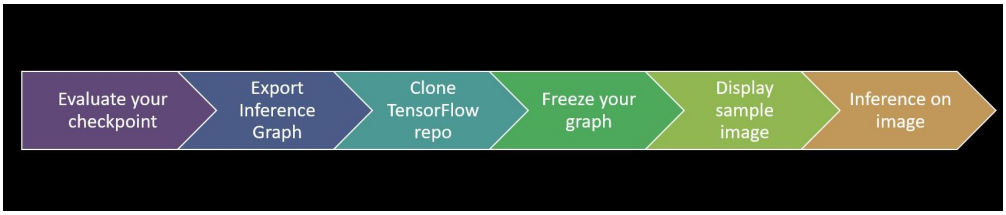
14.0.11 Inference on an Image

We can use the newly created frozen graph file to test a sample image. We're using the label_image script that takes an image, frozen graph, labels.txt files, and displays the top five probabilities for the given image.

14.0.12 Activity

Click the cell below and then click **Run**.

```
In [25]: !python tensorflow/tensorflow/examples/label_image/label_image.py \
             --image=breeds/train/maine_coon/Maine_Coon_100.jpg \
             --input_layer=input \
```



Inference on Image

```
    --input_height=224 \
    --input_width=224 \
    --output_layer=InceptionV1/Logits/Predictions/Reshape_1 \
    --graph=train_dir/frozen_inception_v1.pb \
    --labels=breeds/labels.txt

print("Done.")
```

```
/home/zephyrie/anaconda3/envs/tf/lib/python3.6/site-packages/h5py/__init__.py:36: FutureWarning:
  from ._conv import register_converters as _register_converters
2018-10-02 17:14:02.136814: I tensorflow/core/platform/cpu_feature_guard.cc:141] Your CPU suppor
2018-10-02 17:14:02.151017: I tensorflow/core/common_runtime/process_util.cc:69] Creating new th
20:maine_coon 0.88512963
0:abyssinian 0.04162266
9:british_shorthair 0.031994887
5:bengal 0.01510871
11:egyptian_mau 0.01088539
Done.
```

14.0.13 Summary

- Getting your dataset
- Sorting your dataset
- Generating TFRecord files
- Learning about fine-tuning and checkpoints
- Train your dataset with fine-tune checkpoint
- Evaluating your training
- Creating a frozen graph
- Using a frozen graph to test image classification

15 Part 4: Additional Fine Tuning (Optional)

15.0.1 Fine Tuning the Entire Network

We previously fine tuned only the final layer of the network. Now we're going to allow for all of the layers in the network to be trained but we're going to use a much lower learning rate. This will let the network narrow in and tune the remaining weights we didn't tune from the ImageNet

checkpoint. We'll want to make sure not to train too much though, or we might start to overfit, so we'll limit the steps to about 500-1500.

15.0.2 Activity

In the cell below, update the **max_number_of_steps** parameter to a number between **500** and **1500**, the **learning_rate** to **0.0001** and then click **Run**.

```
In [19]: !python models/research/slim/train_image_classifier.py \
             --train_dir=train_dir/all \
             --dataset_name=breeds \
             --dataset_split_name=train \
             --clone_on_cpu=true \
             --dataset_dir=breeds \
             --model_name=inception_v1 \
             --checkpoint_path=train_dir \
             --max_number_of_steps=1500 \
             --learning_rate=0.0001 \
             --learning_rate_decay_type=fixed \
             --batch_size=32 \
             --save_interval_secs=60 \
             --save_summaries_secs=60 \
             --inter_op=2 \
             --intra_op=12

         !echo "Done."

/home/zephyrie/anaconda3/envs/tf/lib/python3.6/site-packages/h5py/__init__.py:36: FutureWarning:
  from ._conv import register_converters as _register_converters
WARNING:tensorflow:From models/research/slim/train_image_classifier.py:406: create_global_step (
Instructions for updating:
Please switch to tf.train.create_global_step
WARNING:tensorflow:From models/research/slim/train_image_classifier.py:474: softmax_cross_entrop
Instructions for updating:
Use tf.losses.softmax_cross_entropy instead. Note that the order of the logits and labels argume
WARNING:tensorflow:From /home/zephyrie/anaconda3/envs/tf/lib/python3.6/site-packages/tensorflow/
Instructions for updating:

Future major versions of TensorFlow will allow gradients to flow
into the labels input on backprop by default.

See @{tf.nn.softmax_cross_entropy_with_logits_v2}.

WARNING:tensorflow:From /home/zephyrie/anaconda3/envs/tf/lib/python3.6/site-packages/tensorflow/
Instructions for updating:
Use tf.losses.compute_weighted_loss instead.
WARNING:tensorflow:From /home/zephyrie/anaconda3/envs/tf/lib/python3.6/site-packages/tensorflow/
Instructions for updating:
```

```
Use tf.losses.add_loss instead.
INFO:tensorflow:Fine-tuning from train_dir/model.ckpt-1500
WARNING:tensorflow:From /home/zephyrie/anaconda3/envs/tf/lib/python3.6/site-packages/tensorflow/
Instructions for updating:
Please switch to tf.train.MonitoredTrainingSession
2018-10-02 17:03:16.916050: I tensorflow/core/platform/cpu_feature_guard.cc:141] Your CPU suppor
INFO:tensorflow:Restoring parameters from train_dir/model.ckpt-1500
INFO:tensorflow:Running local_init_op.
INFO:tensorflow:Done running local_init_op.
INFO:tensorflow:Starting Session.
INFO:tensorflow:Saving checkpoint to path train_dir/all/model.ckpt
INFO:tensorflow:Starting Queues.
INFO:tensorflow:global_step/sec: 0
INFO:tensorflow:Recording summary at step 3.
INFO:tensorflow:global step 10: loss = 2.4591 (0.406 sec/step)
INFO:tensorflow:global step 20: loss = 3.3044 (0.401 sec/step)
INFO:tensorflow:global step 30: loss = 2.1207 (0.413 sec/step)
INFO:tensorflow:global step 40: loss = 2.1262 (0.415 sec/step)
INFO:tensorflow:global step 50: loss = 1.8909 (0.383 sec/step)
INFO:tensorflow:global step 60: loss = 1.6558 (0.457 sec/step)
INFO:tensorflow:global step 70: loss = 1.7278 (0.376 sec/step)
INFO:tensorflow:global step 80: loss = 1.6163 (0.374 sec/step)
INFO:tensorflow:global step 90: loss = 1.6921 (0.387 sec/step)
INFO:tensorflow:global step 100: loss = 2.3219 (0.383 sec/step)
INFO:tensorflow:global step 110: loss = 1.8419 (0.376 sec/step)
INFO:tensorflow:global step 120: loss = 1.7735 (0.387 sec/step)
INFO:tensorflow:Saving checkpoint to path train_dir/all/model.ckpt
INFO:tensorflow:global_step/sec: 2.16142
INFO:tensorflow:global step 130: loss = 1.8473 (0.891 sec/step)
INFO:tensorflow:Recording summary at step 133.
INFO:tensorflow:global step 140: loss = 1.1304 (0.385 sec/step)
INFO:tensorflow:global step 150: loss = 1.0472 (0.398 sec/step)
INFO:tensorflow:global step 160: loss = 1.8068 (0.390 sec/step)
INFO:tensorflow:global step 170: loss = 1.8953 (0.384 sec/step)
INFO:tensorflow:global step 180: loss = 2.6623 (0.380 sec/step)
INFO:tensorflow:global step 190: loss = 3.2296 (0.384 sec/step)
INFO:tensorflow:global step 200: loss = 2.8923 (0.387 sec/step)
INFO:tensorflow:global step 210: loss = 1.6469 (0.374 sec/step)
INFO:tensorflow:global step 220: loss = 1.3073 (0.373 sec/step)
INFO:tensorflow:global step 230: loss = 1.6738 (0.377 sec/step)
INFO:tensorflow:global step 240: loss = 1.2588 (0.375 sec/step)
INFO:tensorflow:global step 250: loss = 1.1937 (0.381 sec/step)
INFO:tensorflow:global step 260: loss = 1.1922 (0.371 sec/step)
INFO:tensorflow:global step 270: loss = 1.3525 (0.372 sec/step)
INFO:tensorflow:Saving checkpoint to path train_dir/all/model.ckpt
INFO:tensorflow:global step 280: loss = 0.9859 (0.485 sec/step)
INFO:tensorflow:Recording summary at step 283.
INFO:tensorflow:global_step/sec: 2.4456
```

```
INFO:tensorflow:global step 290: loss = 1.2808 (0.388 sec/step)
INFO:tensorflow:global step 300: loss = 0.9249 (0.386 sec/step)
INFO:tensorflow:global step 310: loss = 1.2622 (0.380 sec/step)
INFO:tensorflow:global step 320: loss = 1.6424 (0.381 sec/step)
INFO:tensorflow:global step 330: loss = 0.9829 (0.382 sec/step)
INFO:tensorflow:global step 340: loss = 1.2884 (0.384 sec/step)
INFO:tensorflow:global step 350: loss = 1.4252 (0.379 sec/step)
INFO:tensorflow:global step 360: loss = 2.0395 (0.401 sec/step)
INFO:tensorflow:global step 370: loss = 1.8464 (0.396 sec/step)
INFO:tensorflow:global step 380: loss = 1.5635 (0.382 sec/step)
INFO:tensorflow:global step 390: loss = 0.7707 (0.379 sec/step)
INFO:tensorflow:global step 400: loss = 2.1706 (0.381 sec/step)
INFO:tensorflow:global step 410: loss = 1.5936 (0.392 sec/step)
INFO:tensorflow:global step 420: loss = 1.3369 (0.376 sec/step)
INFO:tensorflow:Saving checkpoint to path train_dir/all/model.ckpt
INFO:tensorflow:global_step/sec: 2.57128
INFO:tensorflow:global step 430: loss = 1.6130 (0.488 sec/step)
INFO:tensorflow:Recording summary at step 434.
INFO:tensorflow:global step 440: loss = 1.1228 (0.378 sec/step)
INFO:tensorflow:global step 450: loss = 2.0455 (0.378 sec/step)
INFO:tensorflow:global step 460: loss = 1.2578 (0.377 sec/step)
INFO:tensorflow:global step 470: loss = 1.5018 (0.378 sec/step)
INFO:tensorflow:global step 480: loss = 1.2264 (0.382 sec/step)
INFO:tensorflow:global step 490: loss = 0.9648 (0.379 sec/step)
INFO:tensorflow:global step 500: loss = 1.7832 (0.384 sec/step)
INFO:tensorflow:global step 510: loss = 1.0977 (0.376 sec/step)
INFO:tensorflow:global step 520: loss = 1.2596 (0.378 sec/step)
INFO:tensorflow:global step 530: loss = 1.0970 (0.379 sec/step)
INFO:tensorflow:global step 540: loss = 1.4754 (0.378 sec/step)
INFO:tensorflow:global step 550: loss = 1.7514 (0.377 sec/step)
INFO:tensorflow:global step 560: loss = 1.2020 (0.382 sec/step)
INFO:tensorflow:global step 570: loss = 1.4522 (0.378 sec/step)
INFO:tensorflow:Saving checkpoint to path train_dir/all/model.ckpt
INFO:tensorflow:global_step/sec: 2.50428
INFO:tensorflow:global step 580: loss = 0.8886 (0.491 sec/step)
INFO:tensorflow:Recording summary at step 583.
INFO:tensorflow:global step 590: loss = 0.8185 (0.376 sec/step)
INFO:tensorflow:global step 600: loss = 0.9379 (0.376 sec/step)
INFO:tensorflow:global step 610: loss = 1.1677 (0.373 sec/step)
INFO:tensorflow:global step 620: loss = 1.2249 (0.379 sec/step)
INFO:tensorflow:global step 630: loss = 1.4523 (0.379 sec/step)
INFO:tensorflow:global step 640: loss = 1.6219 (0.383 sec/step)
INFO:tensorflow:global step 650: loss = 0.8298 (0.385 sec/step)
INFO:tensorflow:global step 660: loss = 1.0587 (0.390 sec/step)
INFO:tensorflow:global step 670: loss = 1.3759 (0.382 sec/step)
INFO:tensorflow:global step 680: loss = 1.3849 (0.381 sec/step)
INFO:tensorflow:global step 690: loss = 1.0809 (0.392 sec/step)
INFO:tensorflow:global step 700: loss = 1.6431 (0.384 sec/step)
```

```
INFO:tensorflow:global step 710: loss = 0.9590 (0.401 sec/step)
INFO:tensorflow:global step 720: loss = 1.1857 (0.381 sec/step)
INFO:tensorflow:Saving checkpoint to path train_dir/all/model.ckpt
INFO:tensorflow:global_step/sec: 2.526
INFO:tensorflow:global step 730: loss = 1.0281 (0.480 sec/step)
INFO:tensorflow:Recording summary at step 735.
INFO:tensorflow:global step 740: loss = 1.1714 (0.382 sec/step)
INFO:tensorflow:global step 750: loss = 1.7541 (0.376 sec/step)
INFO:tensorflow:global step 760: loss = 1.6374 (0.395 sec/step)
INFO:tensorflow:global step 770: loss = 1.2610 (0.386 sec/step)
INFO:tensorflow:global step 780: loss = 0.6405 (0.389 sec/step)
INFO:tensorflow:global step 790: loss = 1.1476 (0.376 sec/step)
INFO:tensorflow:global step 800: loss = 1.2253 (0.375 sec/step)
INFO:tensorflow:global step 810: loss = 0.9969 (0.382 sec/step)
INFO:tensorflow:global step 820: loss = 0.9858 (0.375 sec/step)
INFO:tensorflow:global step 830: loss = 1.3004 (0.379 sec/step)
INFO:tensorflow:global step 840: loss = 1.3735 (0.380 sec/step)
INFO:tensorflow:global step 850: loss = 0.9141 (0.380 sec/step)
INFO:tensorflow:global step 860: loss = 1.7246 (0.388 sec/step)
INFO:tensorflow:global step 870: loss = 1.5733 (0.378 sec/step)
INFO:tensorflow:Saving checkpoint to path train_dir/all/model.ckpt
INFO:tensorflow:global step 880: loss = 1.4940 (0.629 sec/step)
INFO:tensorflow:global_step/sec: 2.52963
INFO:tensorflow:Recording summary at step 887.
INFO:tensorflow:global step 890: loss = 1.1497 (0.382 sec/step)
INFO:tensorflow:global step 900: loss = 0.9529 (0.381 sec/step)
INFO:tensorflow:global step 910: loss = 1.2687 (0.383 sec/step)
INFO:tensorflow:global step 920: loss = 1.4682 (0.381 sec/step)
INFO:tensorflow:global step 930: loss = 1.0118 (0.388 sec/step)
INFO:tensorflow:global step 940: loss = 1.5101 (0.386 sec/step)
INFO:tensorflow:global step 950: loss = 0.7738 (0.395 sec/step)
INFO:tensorflow:global step 960: loss = 1.2322 (0.383 sec/step)
INFO:tensorflow:global step 970: loss = 1.6020 (0.384 sec/step)
INFO:tensorflow:global step 980: loss = 1.0699 (0.379 sec/step)
INFO:tensorflow:global step 990: loss = 1.6012 (0.402 sec/step)
INFO:tensorflow:global step 1000: loss = 1.3185 (0.376 sec/step)
INFO:tensorflow:global step 1010: loss = 0.5566 (0.388 sec/step)
INFO:tensorflow:global step 1020: loss = 0.9075 (0.383 sec/step)
INFO:tensorflow:Saving checkpoint to path train_dir/all/model.ckpt
INFO:tensorflow:global step 1030: loss = 1.2825 (0.516 sec/step)
INFO:tensorflow:global_step/sec: 2.50933
INFO:tensorflow:Recording summary at step 1038.
INFO:tensorflow:global step 1040: loss = 0.9061 (0.497 sec/step)
INFO:tensorflow:global step 1050: loss = 0.7411 (0.380 sec/step)
INFO:tensorflow:global step 1060: loss = 1.2406 (0.381 sec/step)
INFO:tensorflow:global step 1070: loss = 0.9659 (0.378 sec/step)
INFO:tensorflow:global step 1080: loss = 1.1839 (0.384 sec/step)
INFO:tensorflow:global step 1090: loss = 1.3277 (0.388 sec/step)
```

```
INFO:tensorflow:global step 1100: loss = 1.3188 (0.389 sec/step)
INFO:tensorflow:global step 1110: loss = 1.2447 (0.378 sec/step)
INFO:tensorflow:global step 1120: loss = 1.7153 (0.376 sec/step)
INFO:tensorflow:global step 1130: loss = 1.1349 (0.375 sec/step)
INFO:tensorflow:global step 1140: loss = 1.4405 (0.382 sec/step)
INFO:tensorflow:global step 1150: loss = 1.3433 (0.386 sec/step)
INFO:tensorflow:global step 1160: loss = 1.1267 (0.386 sec/step)
INFO:tensorflow:global step 1170: loss = 1.8910 (0.381 sec/step)
INFO:tensorflow:Saving checkpoint to path train_dir/all/model.ckpt
INFO:tensorflow:global step 1180: loss = 1.2208 (0.436 sec/step)
INFO:tensorflow:global_step/sec: 2.51927
INFO:tensorflow:Recording summary at step 1188.
INFO:tensorflow:global step 1190: loss = 1.6574 (0.387 sec/step)
INFO:tensorflow:global step 1200: loss = 1.1980 (0.388 sec/step)
INFO:tensorflow:global step 1210: loss = 1.2220 (0.378 sec/step)
INFO:tensorflow:global step 1220: loss = 0.9859 (0.382 sec/step)
INFO:tensorflow:global step 1230: loss = 1.8326 (0.383 sec/step)
INFO:tensorflow:global step 1240: loss = 1.8306 (0.385 sec/step)
INFO:tensorflow:global step 1250: loss = 0.8266 (0.380 sec/step)
INFO:tensorflow:global step 1260: loss = 1.0713 (0.383 sec/step)
INFO:tensorflow:global step 1270: loss = 1.3724 (0.372 sec/step)
INFO:tensorflow:global step 1280: loss = 1.0205 (0.376 sec/step)
INFO:tensorflow:global step 1290: loss = 0.9180 (0.378 sec/step)
INFO:tensorflow:global step 1300: loss = 1.4478 (0.375 sec/step)
INFO:tensorflow:global step 1310: loss = 1.0103 (0.379 sec/step)
INFO:tensorflow:global step 1320: loss = 1.3477 (0.377 sec/step)
INFO:tensorflow:global step 1330: loss = 0.8471 (0.385 sec/step)
INFO:tensorflow:Saving checkpoint to path train_dir/all/model.ckpt
INFO:tensorflow:global step 1340: loss = 1.1565 (0.490 sec/step)
INFO:tensorflow:Recording summary at step 1340.
INFO:tensorflow:global_step/sec: 2.47214
INFO:tensorflow:global step 1350: loss = 1.0413 (0.386 sec/step)
INFO:tensorflow:global step 1360: loss = 0.8194 (0.378 sec/step)
INFO:tensorflow:global step 1370: loss = 1.7339 (0.376 sec/step)
INFO:tensorflow:global step 1380: loss = 0.9989 (0.379 sec/step)
INFO:tensorflow:global step 1390: loss = 1.2054 (0.379 sec/step)
INFO:tensorflow:global step 1400: loss = 1.1661 (0.386 sec/step)
INFO:tensorflow:global step 1410: loss = 1.2447 (0.382 sec/step)
INFO:tensorflow:global step 1420: loss = 1.2160 (0.384 sec/step)
INFO:tensorflow:global step 1430: loss = 0.9722 (0.374 sec/step)
INFO:tensorflow:global step 1440: loss = 1.2179 (0.381 sec/step)
INFO:tensorflow:global step 1450: loss = 1.1450 (0.388 sec/step)
INFO:tensorflow:global step 1460: loss = 0.9736 (0.390 sec/step)
INFO:tensorflow:global step 1470: loss = 1.0105 (0.381 sec/step)
INFO:tensorflow:global step 1480: loss = 0.9292 (0.386 sec/step)
INFO:tensorflow:Saving checkpoint to path train_dir/all/model.ckpt
INFO:tensorflow:global step 1490: loss = 1.1913 (0.575 sec/step)
INFO:tensorflow:Recording summary at step 1491.
```

```
INFO:tensorflow:global step 1500: loss = 0.7949 (0.386 sec/step)
INFO:tensorflow:Stopping Training.
INFO:tensorflow:Finished training! Saving model to disk.
Done.
```

15.0.3 Activity

Click the cell below and then click **Run**.

```
In [20]: !python models/research/slim/eval_image_classifier.py \
             --checkpoint_path=$(ls -t train_dir/all/model.ckpt* | head -1 | rev | cut -d '.' -f
             --eval_dir=eval_dir/all \
             --dataset_dir=breeds \
             --dataset_name=breeds \
             --dataset_split_name=validation \
             --model_name=inception_v1

         !echo "Done."
```

```
/home/zephyrie/anaconda3/envs/tf/lib/python3.6/site-packages/h5py/__init__.py:36: FutureWarning:
  from ._conv import register_converters as _register_converters
WARNING:tensorflow:From models/research/slim/eval_image_classifier.py:94: get_or_create_global_s
Instructions for updating:
Please switch to tf.train.get_or_create_global_step
INFO:tensorflow:Scale of 0 disables regularizer.
WARNING:tensorflow:From models/research/slim/eval_image_classifier.py:161: streaming_accuracy (f
Instructions for updating:
Please switch to tf.metrics.accuracy. Note that the order of the labels and predictions argument
WARNING:tensorflow:From models/research/slim/eval_image_classifier.py:163: streaming_recall_at_k
Instructions for updating:
Please use `streaming_sparse_recall_at_k`, and reshape labels from [batch_size] to [batch_size,
WARNING:tensorflow:From /home/zephyrie/anaconda3/envs/tf/lib/python3.6/site-packages/tensorflow/
Instructions for updating:
Please switch to tf.metrics.mean
INFO:tensorflow:Evaluating train_dir/all/model.ckpt-1500
INFO:tensorflow:Starting evaluation at 2018-10-03-00:13:37
INFO:tensorflow:Graph was finalized.
2018-10-02 17:13:38.114175: I tensorflow/core/platform/cpu_feature_guard.cc:141] Your CPU suppor
2018-10-02 17:13:38.128318: I tensorflow/core/common_runtime/process_util.cc:69] Creating new th
INFO:tensorflow:Restoring parameters from train_dir/all/model.ckpt-1500
INFO:tensorflow:Running local_init_op.
INFO:tensorflow:Done running local_init_op.
INFO:tensorflow:Evaluation [1/15]
INFO:tensorflow:Evaluation [2/15]
INFO:tensorflow:Evaluation [3/15]
INFO:tensorflow:Evaluation [4/15]
INFO:tensorflow:Evaluation [5/15]
```

```
INFO:tensorflow:Evaluation [6/15]
INFO:tensorflow:Evaluation [7/15]
INFO:tensorflow:Evaluation [8/15]
INFO:tensorflow:Evaluation [9/15]
INFO:tensorflow:Evaluation [10/15]
INFO:tensorflow:Evaluation [11/15]
INFO:tensorflow:Evaluation [12/15]
INFO:tensorflow:Evaluation [13/15]
INFO:tensorflow:Evaluation [14/15]
INFO:tensorflow:Evaluation [15/15]
eval/Recall_5[0.992]
eval/Accuracy[0.888]
INFO:tensorflow:Finished evaluation at 2018-10-03-00:13:46
Done.
```

15.0.4 Activity

Click the cell below and then click **Run**.

```
In [21]: !python tensorflow/tensorflow/python/tools/freeze_graph.py \
             --clear_devices=true \
             --input_graph=train_dir/inception_v1_inf_graph.pb \
             --input_checkpoint=$(ls -t train_dir/all/model.ckpt* | head -1 | rev | cut -d '.' -
             --input_binary=true \
             --output_graph=train_dir/all/frozen_inception_v1.pb \
             --output_node_names=InceptionV1/Logits/Predictions/Reshape_1

         !echo "Done."
```

```
/home/zephyrie/anaconda3/envs/tf/lib/python3.6/site-packages/h5py/__init__.py:36: FutureWarning:
  from ._conv import register_converters as _register_converters
2018-10-02 17:13:50.843068: I tensorflow/core/platform/cpu_feature_guard.cc:141] Your CPU suppor
2018-10-02 17:13:50.855044: I tensorflow/core/common_runtime/process_util.cc:69] Creating new th
Done.
```

15.0.5 Activity

Click the cell below and then click **Run**.

```
In [22]: from PIL import Image

         Image.open('breeds/train/maine_coon/Maine_Coon_100.jpg')
```

```
   Out[22]:
```

15.0.6 Activity

Click the cell below and then click **Run**.

```
In [23]: !python tensorflow/tensorflow/examples/label_image/label_image.py \
             --image=breeds/train/maine_coon/Maine_Coon_100.jpg \
             --input_layer=input \
             --input_height=224 \
             --input_width=224 \
             --output_layer=InceptionV1/Logits/Predictions/Reshape_1 \
             --graph=train_dir/all/frozen_inception_v1.pb \
             --labels=breeds/labels.txt

         print("Done.")
```

```
/home/zephyrie/anaconda3/envs/tf/lib/python3.6/site-packages/h5py/__init__.py:36: FutureWarning:
  from ._conv import register_converters as _register_converters
2018-10-02 17:13:53.928068: I tensorflow/core/platform/cpu_feature_guard.cc:141] Your CPU suppor
2018-10-02 17:13:53.941772: I tensorflow/core/common_runtime/process_util.cc:69] Creating new th
20:maine_coon 0.9803002
5:bengal 0.0177954
11:egyptian_mau 0.0010098823
0:abyssinian 0.000502652
23:persian 0.0001829258
Done.
```

15.0.7 Resources

TensorFlow* Optimizations on Modern Intelő Architecture, https://software.intel.com/en-us/articles/tensorflow-optimizations-on-modern-intel-architecture

Intel Optimized TensorFlow Wheel Now Available, https://software.intel.com/en-us/articles/intel-optimized-tensorflow-wheel-now-available

Build and Install TensorFlow* on Intelő Architecture, https://software.intel.com/en-us/articles/build-and-install-tensorflow-on-intel-architecture

TensorFlow, https://www.tensorflow.org/

15.0.8 Case Studies

Manufacturing Package Fault Detection Using Deep Learning, https://software.intel.com/en-us/articles/manufacturing-package-fault-detection-using-deep-learning

Automatic Defect Inspection Using Deep Learning for Solar Farm, https://software.intel.com/en-us/articles/automatic-defect-inspection-using-deep-learning-for-solar-farm