FASHION LENS

Introduction

TensorFlow is a free and open-source software library for dataflow and differentiable programming across a range of tasks. It is a symbolic math library, and is also used for machine learning applications such as neural networks.

It is used for both research and production at Google.

TensorFlow was developed by the Google Brain team for internal Google use. It was released under the Apache 2.0 open-source license on November 9, 2015.

Frameworks

TensorFlow is Google Brain's second-generation system. Version 1.0.0 was released on February 11, 2017. While the reference implementation runs on single devices, TensorFlow can run on multiple CPUs and GPUs (with optional CUDA and SYCL extensions for general-purpose computing on graphics processing units). TensorFlow is available on 64-bit Linux, macOS, Windows, and mobile computing platforms including Android and iOS.

Its flexible architecture allows for the easy deployment of computation across a variety of platforms (CPUs, GPUs, TPUs), and from desktops to clusters of servers to mobile and edge devices.

TensorFlow computations are expressed as stateful dataflow graphs. The name TensorFlow derives from the operations that such neural networks perform on multidimensional data arrays, which are referred to as tensors. During the Google I/O Conference in June 2016, Jeff Dean stated that 1,500 repositories on GitHub mentioned TensorFlow, of which only 5 were from Google.

In May 2017, Google announced a software stack specifically for mobile development, TensorFlow Lite. In January 2019, TensorFlow team released a developer preview of the mobile GPU inference engine with OpenGL ES 3.1 Compute Shaders on Android devices and Metal Compute Shaders on iOS devices.

Project Process

I will be using transfer learning, which means starting with a model that has been already trained on another problem and retrain it on a similar problem. Deep learning from scratch can take days, but transfer learning can be done in short order.

Going to use a model trained on the ImageNet Large Visual Recognition Challenge dataset. These models can differentiate between 1,000 different classes, like Dalmatian or dishwasher. You will have a choice of model architectures, so you can determine the right tradeoff between speed, size and accuracy for your problem.

We will use this same model, but retrain it to tell apart a small number of classes based on our own examples.

IMPORT PACKAGES

In [1]:
```
from __future__ import absolute_import
from __future__ import division
from __future__ import print_function

import argparse
import collections
from datetime import datetime
import hashlib
import os.path
import random
import re
import sys
import tarfile

from tensorflow.python.framework import graph_util
from tensorflow.python.framework import tensor_shape
from tensorflow.python.platform import gfile
from tensorflow.python.util import compat

import numpy as np
from six.moves import urllib
import tensorflow as tf

FLAGS = None
```

PRE-PROCESSING IMAGES

In [2]:
```
MAX_NUM_IMAGES_PER_CLASS = 2 ** 27 - 1  # ~134M
```

In [3]:
```
def create_image_lists(image_dir, testing_percentage, validation_pe
rcentage):
  """Builds a list of training images from the file system.
```

```
  Analyzes the sub folders in the image directory, splits them into
stable
  training, testing, and validation sets, and returns a data struct
ure
  describing the lists of images for each label and their paths.

  Args:
    image_dir: String path to a folder containing subfolders of ima
ges.
    testing_percentage: Integer percentage of the images to reserve
for tests.
    validation_percentage: Integer percentage of images reserved fo
r validation.

  Returns:
    A dictionary containing an entry for each label subfolder, with
images split
    into training, testing, and validation sets within each label.
  """
  if not gfile.Exists(image_dir):
    tf.logging.error("Image directory '" + image_dir + "' not found
.")
    return None
  result = collections.OrderedDict()
  sub_dirs = [
    os.path.join(image_dir,item)
    for item in gfile.ListDirectory(image_dir)]
  sub_dirs = sorted(item for item in sub_dirs
                    if gfile.IsDirectory(item))
  for sub_dir in sub_dirs:
    extensions = ['jpg', 'jpeg', 'JPG', 'JPEG']
    file_list = []
    dir_name = os.path.basename(sub_dir)
    if dir_name == image_dir:
      continue
    tf.logging.info("Looking for images in '" + dir_name + "'")
    for extension in extensions:
      file_glob = os.path.join(image_dir, dir_name, '*.' + extensio
n)
      file_list.extend(gfile.Glob(file_glob))
    if not file_list:
      tf.logging.warning('No files found')
      continue
    if len(file_list) < 20:
      tf.logging.warning(
          'WARNING: Folder has less than 20 images, which may cause
issues.')
    elif len(file_list) > MAX_NUM_IMAGES_PER_CLASS:
      tf.logging.warning(
          'WARNING: Folder {} has more than {} images. Some images
will '
          'never be selected.'.format(dir_name, MAX_NUM_IMAGES_PER_
CLASS))
    label_name = re.sub(r'[^a-z0-9]+', ' ', dir_name.lower())
    training_images = []
    testing_images = []
```

```
    validation_images = []
    for file_name in file_list:
      base_name = os.path.basename(file_name)
      # We want to ignore anything after '_nohash_' in the file name when
      # deciding which set to put an image in, the data set creator has a way of
      # grouping photos that are close variations of each other. For example
      # this is used in the plant disease data set to group multiple pictures of
      # the same leaf.
      hash_name = re.sub(r'_nohash_.*$', '', file_name)
      # This looks a bit magical, but we need to decide whether this file should
      # go into the training, testing, or validation sets, and we want to keep
      # existing files in the same set even if more files are subsequently
      # added.
      # To do that, we need a stable way of deciding based on just the file name
      # itself, so we do a hash of that and then use that to generate a
      # probability value that we use to assign it.
      hash_name_hashed = hashlib.sha1(compat.as_bytes(hash_name)).hexdigest()
      percentage_hash = ((int(hash_name_hashed, 16) %
                          (MAX_NUM_IMAGES_PER_CLASS + 1)) *
                         (100.0 / MAX_NUM_IMAGES_PER_CLASS))
      if percentage_hash < validation_percentage:
        validation_images.append(base_name)
      elif percentage_hash < (testing_percentage + validation_percentage):
        testing_images.append(base_name)
      else:
        training_images.append(base_name)
    result[label_name] = {
        'dir': dir_name,
        'training': training_images,
        'testing': testing_images,
        'validation': validation_images,
    }
  return result
```

In [4]:
```
def get_image_path(image_lists, label_name, index, image_dir, categ
ory):
  """"Returns a path to an image for a label at the given index.

  Args:
    image_lists: Dictionary of training images for each label.
    label_name: Label string we want to get an image for.
    index: Int offset of the image we want. This will be moduloed b
y the
    available number of images for the label, so it can be arbitrar
ily large.
    image_dir: Root folder string of the subfolders containing the
training
    images.
    category: Name string of set to pull images from - training, te
sting, or
    validation.

  Returns:
    File system path string to an image that meets the requested pa
rameters.

  """
  if label_name not in image_lists:
    tf.logging.fatal('Label does not exist %s.', label_name)
  label_lists = image_lists[label_name]
  if category not in label_lists:
    tf.logging.fatal('Category does not exist %s.', category)
  category_list = label_lists[category]
  if not category_list:
    tf.logging.fatal('Label %s has no images in the category %s.',
                     label_name, category)
  mod_index = index % len(category_list)
  base_name = category_list[mod_index]
  sub_dir = label_lists['dir']
  full_path = os.path.join(image_dir, sub_dir, base_name)
  return full_path
```

In [5]:
```
def get_bottleneck_path(image_lists, label_name, index, bottleneck_
dir,
                        category, architecture):
  """"Returns a path to a bottleneck file for a label at the given
index.

  Args:
    image_lists: Dictionary of training images for each label.
    label_name: Label string we want to get an image for.
    index: Integer offset of the image we want. This will be modulo
ed by the
    available number of images for the label, so it can be arbitrar
ily large.
    bottleneck_dir: Folder string holding cached files of bottlenec
k values.
    category: Name string of set to pull images from - training, te
sting, or
    validation.
    architecture: The name of the model architecture.

  Returns:
    File system path string to an image that meets the requested pa
rameters.
  """
  return get_image_path(image_lists, label_name, index, bottleneck_
dir,
                        category) + '_' + architecture + '.txt'
```

In [6]:
```
def create_model_graph(model_info, model_dir):
  """"Creates a graph from saved GraphDef file and returns a Graph
object.

  Args:
    model_info: Dictionary containing information about the model a
rchitecture.

  Returns:
    Graph holding the trained Inception network, and various tensor
s we'll be
    manipulating.
  """
  with tf.Graph().as_default() as graph:
    model_path = os.path.join(model_dir, model_info['model_file_nam
e'])
    with gfile.FastGFile(model_path, 'rb') as f:
      graph_def = tf.GraphDef()
      graph_def.ParseFromString(f.read())
      bottleneck_tensor, resized_input_tensor = (tf.import_graph_de
f(
          graph_def,
          name='',
          return_elements=[
              model_info['bottleneck_tensor_name'],
              model_info['resized_input_tensor_name'],
          ]))
  return graph, bottleneck_tensor, resized_input_tensor
```

In [7]:
```
def run_bottleneck_on_image(sess, image_data, image_data_tensor,
                            decoded_image_tensor, resized_input_ten
sor,
                            bottleneck_tensor):
  """Runs inference on an image to extract the 'bottleneck' summary
layer.

  Args:
    sess: Current active TensorFlow Session.
    image_data: String of raw JPEG data.
    image_data_tensor: Input data layer in the graph.
    decoded_image_tensor: Output of initial image resizing and  pre
processing.
    resized_input_tensor: The input node of the recognition graph.
    bottleneck_tensor: Layer before the final softmax.

  Returns:
    Numpy array of bottleneck values.
  """
  # First decode the JPEG image, resize it, and rescale the pixel v
alues.
  resized_input_values = sess.run(decoded_image_tensor,
                                  {image_data_tensor: image_data})
  # Then run it through the recognition network.
  bottleneck_values = sess.run(bottleneck_tensor,
                               {resized_input_tensor: resized_input
_values})
  bottleneck_values = np.squeeze(bottleneck_values)
  return bottleneck_values
```

In [8]:
```
def maybe_download_and_extract(data_url):
  """Download and extract model tar file.

  If the pretrained model we're using doesn't already exist, this f
unction
  downloads it from the TensorFlow.org website and unpacks it into
a directory.

  Args:
    data_url: Web location of the tar file containing the pretraine
d model.
  """
  dest_directory = "model_dir"
  if not os.path.exists(dest_directory):
    os.makedirs(dest_directory)
  filename = data_url.split('/')[-1]
  filepath = os.path.join(dest_directory, filename)
  if not os.path.exists(filepath):

    def _progress(count, block_size, total_size):
      sys.stdout.write('\r>> Downloading %s %.1f%%' %
                       (filename,
                        float(count * block_size) / float(total_siz
e) * 100.0))
      sys.stdout.flush()

    filepath, _ = urllib.request.urlretrieve(data_url, filepath, _p
rogress)
    print()
    statinfo = os.stat(filepath)
    tf.logging.info('Successfully downloaded', filename, statinfo.s
t_size,
                    'bytes.')
  tarfile.open(filepath, 'r:gz').extractall(dest_directory)
```

In [9]:
```
def ensure_dir_exists(dir_name):
  """Makes sure the folder exists on disk.

  Args:
    dir_name: Path string to the folder we want to create.
  """
  if not os.path.exists(dir_name):
    os.makedirs(dir_name)


bottleneck_path_2_bottleneck_values = {}
```

In [10]:
```
def create_bottleneck_file(bottleneck_path, image_lists, label_name
, index,
                           image_dir, category, sess, jpeg_data_ten
sor,
                           decoded_image_tensor, resized_input_tens
or,
                           bottleneck_tensor):
  """Create a single bottleneck file."""
  tf.logging.info('Creating bottleneck at ' + bottleneck_path)
  image_path = get_image_path(image_lists, label_name, index,
                              image_dir, category)
  if not gfile.Exists(image_path):
    tf.logging.fatal('File does not exist %s', image_path)
  image_data = gfile.FastGFile(image_path, 'rb').read()
  try:
    bottleneck_values = run_bottleneck_on_image(
        sess, image_data, jpeg_data_tensor, decoded_image_tensor,
        resized_input_tensor, bottleneck_tensor)
  except Exception as e:
    raise RuntimeError('Error during processing file %s (%s)' % (im
age_path,
                                                                  st
r(e)))
  bottleneck_string = ','.join(str(x) for x in bottleneck_values)
  with open(bottleneck_path, 'w') as bottleneck_file:
    bottleneck_file.write(bottleneck_string)
```

In [11]:
```
def get_or_create_bottleneck(sess, image_lists, label_name, index,
image_dir,
                             category, bottleneck_dir, jpeg_data_te
nsor,
                             decoded_image_tensor, resized_input_te
nsor,
                             bottleneck_tensor, architecture):
  """Retrieves or calculates bottleneck values for an image.

  If a cached version of the bottleneck data exists on-disk, return
that,
  otherwise calculate the data and save it to disk for future use.

  Args:
    sess: The current active TensorFlow Session.
    image_lists: Dictionary of training images for each label.
    label_name: Label string we want to get an image for.
    index: Integer offset of the image we want. This will be modulo
-ed by the
    available number of images for the label, so it can be arbitrar
ily large.
    image_dir: Root folder string  of the subfolders containing the
training
    images.
    category: Name string of which  set to pull images from - train
ing, testing,
    or validation.
    bottleneck_dir: Folder string holding cached files of bottlenec
k values.
```

```
    jpeg_data_tensor: The tensor to feed loaded jpeg data into.
    decoded_image_tensor: The output of decoding and resizing the i
mage.
    resized_input_tensor: The input node of the recognition graph.
    bottleneck_tensor: The output tensor for the bottleneck values.
    architecture: The name of the model architecture.

  Returns:
    Numpy array of values produced by the bottleneck layer for the
image.
  """
  label_lists = image_lists[label_name]
  sub_dir = label_lists['dir']
  sub_dir_path = os.path.join(bottleneck_dir, sub_dir)
  ensure_dir_exists(sub_dir_path)
  bottleneck_path = get_bottleneck_path(image_lists, label_name, in
dex,
                                        bottleneck_dir, category, a
rchitecture)
  if not os.path.exists(bottleneck_path):
    create_bottleneck_file(bottleneck_path, image_lists, label_name
, index,
                           image_dir, category, sess, jpeg_data_ten
sor,
                           decoded_image_tensor, resized_input_tens
or,
                           bottleneck_tensor)
  with open(bottleneck_path, 'r') as bottleneck_file:
    bottleneck_string = bottleneck_file.read()
  did_hit_error = False
  try:
    bottleneck_values = [float(x) for x in bottleneck_string.split(
',')]
  except ValueError:
    tf.logging.warning('Invalid float found, recreating bottleneck'
)
    did_hit_error = True
  if did_hit_error:
    create_bottleneck_file(bottleneck_path, image_lists, label_name
, index,
                           image_dir, category, sess, jpeg_data_ten
sor,
                           decoded_image_tensor, resized_input_tens
or,
                           bottleneck_tensor)
    with open(bottleneck_path, 'r') as bottleneck_file:
      bottleneck_string = bottleneck_file.read()
    # Allow exceptions to propagate here, since they shouldn't happ
en after a
    # fresh creation
    bottleneck_values = [float(x) for x in bottleneck_string.split(
',')]
  return bottleneck_values
```

In [12]:
```
def cache_bottlenecks(sess, image_lists, image_dir, bottleneck_dir,
                      jpeg_data_tensor, decoded_image_tensor,
                      resized_input_tensor, bottleneck_tensor, arch
itecture):
  """Ensures all the training, testing, and validation bottlenecks
are cached.

  Because we're likely to read the same image multiple times (if th
ere are no
  distortions applied during training) it can speed things up a lot
if we
  calculate the bottleneck layer values once for each image during
  preprocessing, and then just read those cached values repeatedly
during
  training. Here we go through all the images we've found, calculat
e those
  values, and save them off.

  Args:
    sess: The current active TensorFlow Session.
    image_lists: Dictionary of training images for each label.
    image_dir: Root folder string of the subfolders containing the
training
    images.
    bottleneck_dir: Folder string holding cached files of bottlenec
k values.
    jpeg_data_tensor: Input tensor for jpeg data from file.
    decoded_image_tensor: The output of decoding and resizing the i
mage.
    resized_input_tensor: The input node of the recognition graph.
    bottleneck_tensor: The penultimate output layer of the graph.
    architecture: The name of the model architecture.

  Returns:
    Nothing.
  """
  how_many_bottlenecks = 0
  ensure_dir_exists(bottleneck_dir)
  for label_name, label_lists in image_lists.items():
    for category in ['training', 'testing', 'validation']:
      category_list = label_lists[category]
      for index, unused_base_name in enumerate(category_list):
        get_or_create_bottleneck(
            sess, image_lists, label_name, index, image_dir, catego
ry,
            bottleneck_dir, jpeg_data_tensor, decoded_image_tensor,
            resized_input_tensor, bottleneck_tensor, architecture)

        how_many_bottlenecks += 1
        if how_many_bottlenecks % 100 == 0:
          tf.logging.info(
              str(how_many_bottlenecks) + ' bottleneck files create
d.')
```

In [13]:
```
def get_random_cached_bottlenecks(sess, image_lists, how_many, cate
gory,
```

```
                                  bottleneck_dir, image_dir, jpeg_d
ata_tensor,
                                  decoded_image_tensor, resized_inp
ut_tensor,
                                  bottleneck_tensor, architecture):
  """Retrieves bottleneck values for cached images.

  If no distortions are being applied, this function can retrieve t
he cached
  bottleneck values directly from disk for images. It picks a rando
m set of
  images from the specified category.

  Args:
    sess: Current TensorFlow Session.
    image_lists: Dictionary of training images for each label.
    how_many: If positive, a random sample of this size will be cho
sen.
    If negative, all bottlenecks will be retrieved.
    category: Name string of which set to pull from - training, tes
ting, or
    validation.
    bottleneck_dir: Folder string holding cached files of bottlenec
k values.
    image_dir: Root folder string of the subfolders containing the
training
    images.
    jpeg_data_tensor: The layer to feed jpeg image data into.
    decoded_image_tensor: The output of decoding and resizing the i
mage.
    resized_input_tensor: The input node of the recognition graph.
    bottleneck_tensor: The bottleneck output layer of the CNN graph
.
    architecture: The name of the model architecture.

  Returns:
    List of bottleneck arrays, their corresponding ground truths, a
nd the
    relevant filenames.
  """
  class_count = len(image_lists.keys())
  bottlenecks = []
  ground_truths = []
  filenames = []
  if how_many >= 0:
    # Retrieve a random sample of bottlenecks.
    for unused_i in range(how_many):
      label_index = random.randrange(class_count)
      label_name = list(image_lists.keys())[label_index]
      image_index = random.randrange(MAX_NUM_IMAGES_PER_CLASS + 1)
      image_name = get_image_path(image_lists, label_name, image_in
dex,
                                  image_dir, category)
      bottleneck = get_or_create_bottleneck(
          sess, image_lists, label_name, image_index, image_dir, ca
tegory,
          bottleneck_dir, jpeg_data_tensor, decoded_image_tensor,
```

```
            resized_input_tensor, bottleneck_tensor, architecture)
        ground_truth = np.zeros(class_count, dtype=np.float32)
        ground_truth[label_index] = 1.0
        bottlenecks.append(bottleneck)
        ground_truths.append(ground_truth)
        filenames.append(image_name)
  else:
    # Retrieve all bottlenecks.
    for label_index, label_name in enumerate(image_lists.keys()):
      for image_index, image_name in enumerate(
          image_lists[label_name][category]):
        image_name = get_image_path(image_lists, label_name, image_
index,
                                    image_dir, category)
        bottleneck = get_or_create_bottleneck(
            sess, image_lists, label_name, image_index, image_dir,
category,
            bottleneck_dir, jpeg_data_tensor, decoded_image_tensor,
            resized_input_tensor, bottleneck_tensor, architecture)
        ground_truth = np.zeros(class_count, dtype=np.float32)
        ground_truth[label_index] = 1.0
        bottlenecks.append(bottleneck)
        ground_truths.append(ground_truth)
        filenames.append(image_name)
  return bottlenecks, ground_truths, filenames
```

In [14]:

```
def get_random_distorted_bottlenecks(
    sess, image_lists, how_many, category, image_dir, input_jpeg_te
nsor,
    distorted_image, resized_input_tensor, bottleneck_tensor):
  """Retrieves bottleneck values for training images, after distort
ions.

  If we're training with distortions like crops, scales, or flips,
we have to
  recalculate the full model for every image, and so we can't use c
ached
  bottleneck values. Instead we find random images for the requeste
d category,
  run them through the distortion graph, and then the full graph to
get the
  bottleneck results for each.

  Args:
    sess: Current TensorFlow Session.
    image_lists: Dictionary of training images for each label.
    how_many: The integer number of bottleneck values to return.
    category: Name string of which set of images to fetch - trainin
g, testing,
    or validation.
    image_dir: Root folder string of the subfolders containing the
training
    images.
    input_jpeg_tensor: The input layer we feed the image data to.
    distorted_image: The output node of the distortion graph.
    resized_input_tensor: The input node of the recognition graph.
    bottleneck_tensor: The bottleneck output layer of the CNN graph
```

```
.

  Returns:
    List of bottleneck arrays and their corresponding ground truths
.
  """
  class_count = len(image_lists.keys())
  bottlenecks = []
  ground_truths = []
  for unused_i in range(how_many):
    label_index = random.randrange(class_count)
    label_name = list(image_lists.keys())[label_index]
    image_index = random.randrange(MAX_NUM_IMAGES_PER_CLASS + 1)
    image_path = get_image_path(image_lists, label_name, image_inde
x, image_dir,
                                category)
    if not gfile.Exists(image_path):
      tf.logging.fatal('File does not exist %s', image_path)
    jpeg_data = gfile.FastGFile(image_path, 'rb').read()
    # Note that we materialize the distorted_image_data as a numpy
array before
    # sending running inference on the image. This involves 2 memor
y copies and
    # might be optimized in other implementations.
    distorted_image_data = sess.run(distorted_image,
                                    {input_jpeg_tensor: jpeg_data})
    bottleneck_values = sess.run(bottleneck_tensor,
                                 {resized_input_tensor: distorted_i
mage_data})
    bottleneck_values = np.squeeze(bottleneck_values)
    ground_truth = np.zeros(class_count, dtype=np.float32)
    ground_truth[label_index] = 1.0
    bottlenecks.append(bottleneck_values)
    ground_truths.append(ground_truth)
  return bottlenecks, ground_truths
```

In [15]:

```
def should_distort_images(flip_left_right, random_crop, random_scal
e,
                          random_brightness):
  """Whether any distortions are enabled, from the input flags.

  Args:
    flip_left_right: Boolean whether to randomly mirror images hori
zontally.
    random_crop: Integer percentage setting the total margin used a
round the
    crop box.
    random_scale: Integer percentage of how much to vary the scale
by.
    random_brightness: Integer range to randomly multiply the pixel
values by.

  Returns:
    Boolean value indicating whether any distortions should be appl
ied.
  """
  return (flip_left_right or (random_crop != 0) or (random_scale !=
0) or
          (random_brightness != 0))
```

In [16]:

```
def add_input_distortions(flip_left_right, random_crop, random_scal
e,
                          random_brightness, input_width, input_hei
ght,
                          input_depth, input_mean, input_std):
  """Creates the operations to apply the specified distortions.

  During training it can help to improve the results if we run the
images
  through simple distortions like crops, scales, and flips. These r
eflect the
  kind of variations we expect in the real world, and so can help t
rain the
  model to cope with natural data more effectively. Here we take th
e supplied
  parameters and construct a network of operations to apply them to
an image.

  Cropping
  ~~~~~~~~

  Cropping is done by placing a bounding box at a random position i
n the full
  image. The cropping parameter controls the size of that box relat
ive to the
  input image. If it's zero, then the box is the same size as the i
nput and no
  cropping is performed. If the value is 50%, then the crop box wil
l be half the
  width and height of the input. In a diagram it looks like this:

  <       width         >
```

```
  +---------------------+
  |                     |
  |   width - crop%     |
  |    <      >         |
  |    +------+         |
  |    |      |         |
  |    |      |         |
  |    |      |         |
  |    +------+         |
  |                     |
  |                     |
  +---------------------+

  Scaling
  ~~~~~~~

  Scaling is a lot like cropping, except that the bounding box is a
lways
  centered and its size varies randomly within the given range. For
example if
  the scale percentage is zero, then the bounding box is the same s
ize as the
  input and no scaling is applied. If it's 50%, then the bounding b
ox will be in
  a random range between half the width and height and full size.

  Args:
    flip_left_right: Boolean whether to randomly mirror images hori
zontally.
    random_crop: Integer percentage setting the total margin used a
round the
    crop box.
    random_scale: Integer percentage of how much to vary the scale
by.
    random_brightness: Integer range to randomly multiply the pixel
values by.
    graph.
    input_width: Horizontal size of expected input image to model.
    input_height: Vertical size of expected input image to model.
    input_depth: How many channels the expected input image should
have.
    input_mean: Pixel value that should be zero in the image for th
e graph.
    input_std: How much to divide the pixel values by before recogn
ition.

  Returns:
    The jpeg input layer and the distorted result tensor.
  """

  jpeg_data = tf.placeholder(tf.string, name='DistortJPGInput')
  decoded_image = tf.image.decode_jpeg(jpeg_data, channels=input_de
pth)
  decoded_image_as_float = tf.cast(decoded_image, dtype=tf.float32)
  decoded_image_4d = tf.expand_dims(decoded_image_as_float, 0)
  margin_scale = 1.0 + (random_crop / 100.0)
  resize_scale = 1.0 + (random_scale / 100.0)
```

```
  margin_scale_value = tf.constant(margin_scale)
  resize_scale_value = tf.random_uniform(tensor_shape.scalar(),
                                         minval=1.0,
                                         maxval=resize_scale)
  scale_value = tf.multiply(margin_scale_value, resize_scale_value)
  precrop_width = tf.multiply(scale_value, input_width)
  precrop_height = tf.multiply(scale_value, input_height)
  precrop_shape = tf.stack([precrop_height, precrop_width])
  precrop_shape_as_int = tf.cast(precrop_shape, dtype=tf.int32)
  precropped_image = tf.image.resize_bilinear(decoded_image_4d,
                                              precrop_shape_as_int)
  precropped_image_3d = tf.squeeze(precropped_image, squeeze_dims=[
0])
  cropped_image = tf.random_crop(precropped_image_3d,
                                 [input_height, input_width, input_
depth])
  if flip_left_right:
    flipped_image = tf.image.random_flip_left_right(cropped_image)
  else:
    flipped_image = cropped_image
  brightness_min = 1.0 - (random_brightness / 100.0)
  brightness_max = 1.0 + (random_brightness / 100.0)
  brightness_value = tf.random_uniform(tensor_shape.scalar(),
                                       minval=brightness_min,
                                       maxval=brightness_max)
  brightened_image = tf.multiply(flipped_image, brightness_value)
  offset_image = tf.subtract(brightened_image, input_mean)
  mul_image = tf.multiply(offset_image, 1.0 / input_std)
  distort_result = tf.expand_dims(mul_image, 0, name='DistortResult
')
  return jpeg_data, distort_result
```

In [17]:

```
def variable_summaries(var):
  """Attach a lot of summaries to a Tensor (for TensorBoard visuali
zation)."""
  with tf.name_scope('summaries'):
    mean = tf.reduce_mean(var)
    tf.summary.scalar('mean', mean)
    with tf.name_scope('stddev'):
      stddev = tf.sqrt(tf.reduce_mean(tf.square(var - mean)))
    tf.summary.scalar('stddev', stddev)
    tf.summary.scalar('max', tf.reduce_max(var))
    tf.summary.scalar('min', tf.reduce_min(var))
    tf.summary.histogram('histogram', var)
```

In [18]:

```
def add_final_training_ops(class_count, final_tensor_name, bottlene
ck_tensor,
                           bottleneck_tensor_size):
  """Adds a new softmax and fully-connected layer for training.

  We need to retrain the top layer to identify our new classes, so
this function
  adds the right operations to the graph, along with some variables
to hold the
  weights, and then sets up all the gradients for the backward pass
.
```

```
  The set up for the softmax and fully-connected layers is based on
:

https://www.tensorflow.org/versions/master/tutorials/mnist/beginner
s/index.html

  Args:
    class_count: Integer of how many categories of things we're try
ing to
    recognize.
    final_tensor_name: Name string for the new final node that prod
uces results.
    bottleneck_tensor: The output of the main CNN graph.
    bottleneck_tensor_size: How many entries in the bottleneck vect
or.

  Returns:
    The tensors for the training and cross entropy results, and ten
sors for the
    bottleneck input and ground truth input.
  """
  with tf.name_scope('input'):
    bottleneck_input = tf.placeholder_with_default(
        bottleneck_tensor,
        shape=[None, bottleneck_tensor_size],
        name='BottleneckInputPlaceholder')

    ground_truth_input = tf.placeholder(tf.float32,
                                        [None, class_count],
                                        name='GroundTruthInput')

    # Organizing the following ops as `final_training_ops` so they'
re easier
  # to see in TensorBoard
  layer_name = 'final_training_ops'
  with tf.name_scope(layer_name):
    with tf.name_scope('weights'):
      initial_value = tf.truncated_normal(
          [bottleneck_tensor_size, class_count], stddev=0.001)

      layer_weights = tf.Variable(initial_value, name='final_weight
s')

      variable_summaries(layer_weights)
    with tf.name_scope('biases'):
      layer_biases = tf.Variable(tf.zeros([class_count]), name='fin
al_biases')
      variable_summaries(layer_biases)
    with tf.name_scope('Wx_plus_b'):
      logits = tf.matmul(bottleneck_input, layer_weights) + layer_b
iases
      tf.summary.histogram('pre_activations', logits)

  final_tensor = tf.nn.softmax(logits, name=final_tensor_name)
  tf.summary.histogram('activations', final_tensor)
```

```
  with tf.name_scope('cross_entropy'):
    cross_entropy = tf.nn.softmax_cross_entropy_with_logits(
        labels=ground_truth_input, logits=logits)
    with tf.name_scope('total'):
      cross_entropy_mean = tf.reduce_mean(cross_entropy)
  tf.summary.scalar('cross_entropy', cross_entropy_mean)

  with tf.name_scope('train'):
    optimizer = tf.train.GradientDescentOptimizer(learning_rate)
    train_step = optimizer.minimize(cross_entropy_mean)

  return (train_step, cross_entropy_mean, bottleneck_input, ground_
truth_input,
          final_tensor)
```

In [19]:
```
def add_evaluation_step(result_tensor, ground_truth_tensor):
  """Inserts the operations we need to evaluate the accuracy of our
results.

  Args:
    result_tensor: The new final node that produces results.
    ground_truth_tensor: The node we feed ground truth data
    into.

  Returns:
    Tuple of (evaluation step, prediction).
  """
  with tf.name_scope('accuracy'):
    with tf.name_scope('correct_prediction'):
      prediction = tf.argmax(result_tensor, 1)
      correct_prediction = tf.equal(
          prediction, tf.argmax(ground_truth_tensor, 1))
    with tf.name_scope('accuracy'):
      evaluation_step = tf.reduce_mean(tf.cast(correct_prediction,
tf.float32))
  tf.summary.scalar('accuracy', evaluation_step)
  return evaluation_step, prediction
```

In [20]:
```
def save_graph_to_file(sess, graph, graph_file_name):
  output_graph_def = graph_util.convert_variables_to_constants(
      sess, graph.as_graph_def(), [final_tensor_name])
  with gfile.FastGFile(graph_file_name, 'wb') as f:
    f.write(output_graph_def.SerializeToString())
  return
```

In [21]:
```
def prepare_file_system():
    # Setup the directory we'll write summaries to for TensorBoard
    summaries_dir = "retrain_logs"
    intermediate_store_frequency = 0 # How many steps to store inte
rmediate graph. If "0" then will not store!

    if tf.gfile.Exists(summaries_dir):
        tf.gfile.DeleteRecursively(summaries_dir)
    tf.gfile.MakeDirs(summaries_dir)

    if intermediate_store_frequency > 0:
        ensure_dir_exists(intermediate_output_graphs_dir)
    return
```

In [22]:
```
def create_model_info(architecture):
  """Given the name of a model architecture, returns information ab
out it.

  There are different base image recognition pretrained models that
can be
  retrained using transfer learning, and this function translates f
rom the name
  of a model to the attributes that are needed to download and trai
n with it.

  Args:
    architecture: Name of a model architecture.

  Returns:
    Dictionary of information about the model, or None if the name
isn't
    recognized

  Raises:
    ValueError: If architecture name is unknown.
  """
  architecture = architecture.lower()
  if architecture == 'inception_v3':
    # pylint: disable=line-too-long
    data_url = 'http://download.tensorflow.org/models/image/imagene
t/inception-2015-12-05.tgz'
    # pylint: enable=line-too-long
    bottleneck_tensor_name = 'pool_3/_reshape:0'
    bottleneck_tensor_size = 2048
    input_width = 299
    input_height = 299
    input_depth = 3
    resized_input_tensor_name = 'Mul:0'
    model_file_name = 'classify_image_graph_def.pb'
    input_mean = 128
    input_std = 128
  elif architecture.startswith('mobilenet_'):
    parts = architecture.split('_')
    if len(parts) != 3 and len(parts) != 4:
      tf.logging.error("Couldn't understand architecture name '%s'"
,
```

```
                              architecture)
      return None
    version_string = parts[1]
    if (version_string != '1.0' and version_string != '0.75' and
        version_string != '0.50' and version_string != '0.25'):
      tf.logging.error(
          """"The Mobilenet version should be '1.0', '0.75', '0.50'
, or '0.25',
  but found '%s' for architecture '%s'""",
          version_string, architecture)
      return None
    size_string = parts[2]
    if (size_string != '224' and size_string != '192' and
        size_string != '160' and size_string != '128'):
      tf.logging.error(
          """The Mobilenet input size should be '224', '192', '160'
, or '128',
 but found '%s' for architecture '%s'""",
          size_string, architecture)
      return None
    if len(parts) == 3:
      is_quantized = False
    else:
      if parts[3] != 'quantized':
        tf.logging.error(
            "Couldn't understand architecture suffix '%s' for '%s'"
, parts[3],
            architecture)
        return None
      is_quantized = True
    data_url = 'http://download.tensorflow.org/models/mobilenet_v1_
'
    data_url += version_string + '_' + size_string + '_frozen.tgz'
    bottleneck_tensor_name = 'MobilenetV1/Predictions/Reshape:0'
    bottleneck_tensor_size = 1001
    input_width = int(size_string)
    input_height = int(size_string)
    input_depth = 3
    resized_input_tensor_name = 'input:0'
    if is_quantized:
      model_base_name = 'quantized_graph.pb'
    else:
      model_base_name = 'frozen_graph.pb'
    model_dir_name = 'mobilenet_v1_' + version_string + '_' + size_
string
    model_file_name = os.path.join(model_dir_name, model_base_name)
    input_mean = 127.5
    input_std = 127.5
  else:
    tf.logging.error("Couldn't understand architecture name '%s'",
architecture)
    raise ValueError('Unknown architecture', architecture)

  return {
      'data_url': data_url,
      'bottleneck_tensor_name': bottleneck_tensor_name,
      'bottleneck_tensor_size': bottleneck_tensor_size,
```

```
        'input_width': input_width,
        'input_height': input_height,
        'input_depth': input_depth,
        'resized_input_tensor_name': resized_input_tensor_name,
        'model_file_name': model_file_name,
        'input_mean': input_mean,
        'input_std': input_std,
    }
```

In [23]:
```
def add_jpeg_decoding(input_width, input_height, input_depth, input
_mean,
                      input_std):
  """Adds operations that perform JPEG decoding and resizing to the
graph..

  Args:
    input_width: Desired width of the image fed into the recognizer
graph.
    input_height: Desired width of the image fed into the recognize
r graph.
    input_depth: Desired channels of the image fed into the recogni
zer graph.
    input_mean: Pixel value that should be zero in the image for th
e graph.
    input_std: How much to divide the pixel values by before recogn
ition.

  Returns:
    Tensors for the node to feed JPEG data into, and the output of
the
      preprocessing steps.
  """
  jpeg_data = tf.placeholder(tf.string, name='DecodeJPGInput')
  decoded_image = tf.image.decode_jpeg(jpeg_data, channels=input_de
pth)
  decoded_image_as_float = tf.cast(decoded_image, dtype=tf.float32)
  decoded_image_4d = tf.expand_dims(decoded_image_as_float, 0)
  resize_shape = tf.stack([input_height, input_width])
  resize_shape_as_int = tf.cast(resize_shape, dtype=tf.int32)
  resized_image = tf.image.resize_bilinear(decoded_image_4d,
                                           resize_shape_as_int)
  offset_image = tf.subtract(resized_image, input_mean)
  mul_image = tf.multiply(offset_image, 1.0 / input_std)
  return jpeg_data, mul_image
```

In [24]:
```
# Needed to make sure the logging output is visible.
# See https://github.com/tensorflow/tensorflow/issues/3047
tf.logging.set_verbosity(tf.logging.INFO)
```

In [25]:
```
# Prepare necessary directories that can be used during training
prepare_file_system()
```

RETRAINING THE NETWORK

Configure your MobileNet

MobileNet is a a small efficient convolutional neural network. "Convolutional" just means that the same calculations are performed at each location in the image.

The MobileNet is configurable in two ways:

- Input image resolution: 128,160,192, or 224px. Unsurprisingly, feeding in a higher resolution image takes more processing time, but results in better classification accuracy.
- The relative size of the model as a fraction of the largest MobileNet: 1.0, 0.75, 0.50, or 0.25.

I will use image resolution as (h:224px X w:224px) & relative size as 0.5 for this project.

In [26]:
```
# Gather information about the model architecture we'll be using.
architecture = "mobilenet_0.50_224"

model_info = create_model_info(architecture)
```

In [27]:
```
# Set up the pre-trained graph.
data_url = "../tf_files/Clothes/"
model_dir = "model_dir/"

maybe_download_and_extract(model_info['data_url'])

graph, bottleneck_tensor, resized_image_tensor = (create_model_grap
h(model_info, model_dir))
```

```
WARNING:tensorflow:From <ipython-input-6-f6601b6ba390>:13: FastGFi
le.__init__ (from tensorflow.python.platform.gfile) is deprecated
and will be removed in a future version.
Instructions for updating:
Use tf.gfile.GFile.
```

In [28]:

```
# Look at the folder structure, and create lists of all the images.
testing_percentage = 10
validation_percentage = 10
image_dir = "../tf_files/Clothes/"

image_lists = create_image_lists(image_dir, testing_percentage, val
idation_percentage)
class_count = len(image_lists.keys())

if class_count == 0:
    tf.logging.error('No valid folders of images found at ' + image
_dir)

if class_count == 1:
    tf.logging.error('Only one valid folder of images found at ' +
image_dir + ' - multiple classes are needed for classification.')
```

```
INFO:tensorflow:Looking for images in 'Blazer'
INFO:tensorflow:Looking for images in 'Cap'
INFO:tensorflow:Looking for images in 'Chino'
INFO:tensorflow:Looking for images in 'Denim Shirt'
INFO:tensorflow:Looking for images in 'Dress'
INFO:tensorflow:Looking for images in 'Hoodie'
INFO:tensorflow:Looking for images in 'Jacket'
INFO:tensorflow:Looking for images in 'Jeans'
INFO:tensorflow:Looking for images in 'Kurta'
INFO:tensorflow:Looking for images in 'Leather Jacket'
INFO:tensorflow:Looking for images in 'Polo Shirt'
INFO:tensorflow:Looking for images in 'Shirt'
INFO:tensorflow:Looking for images in 'Shorts'
INFO:tensorflow:Looking for images in 'Suit'
INFO:tensorflow:Looking for images in 'Sweater'
INFO:tensorflow:Looking for images in 'T-Shirt'
```

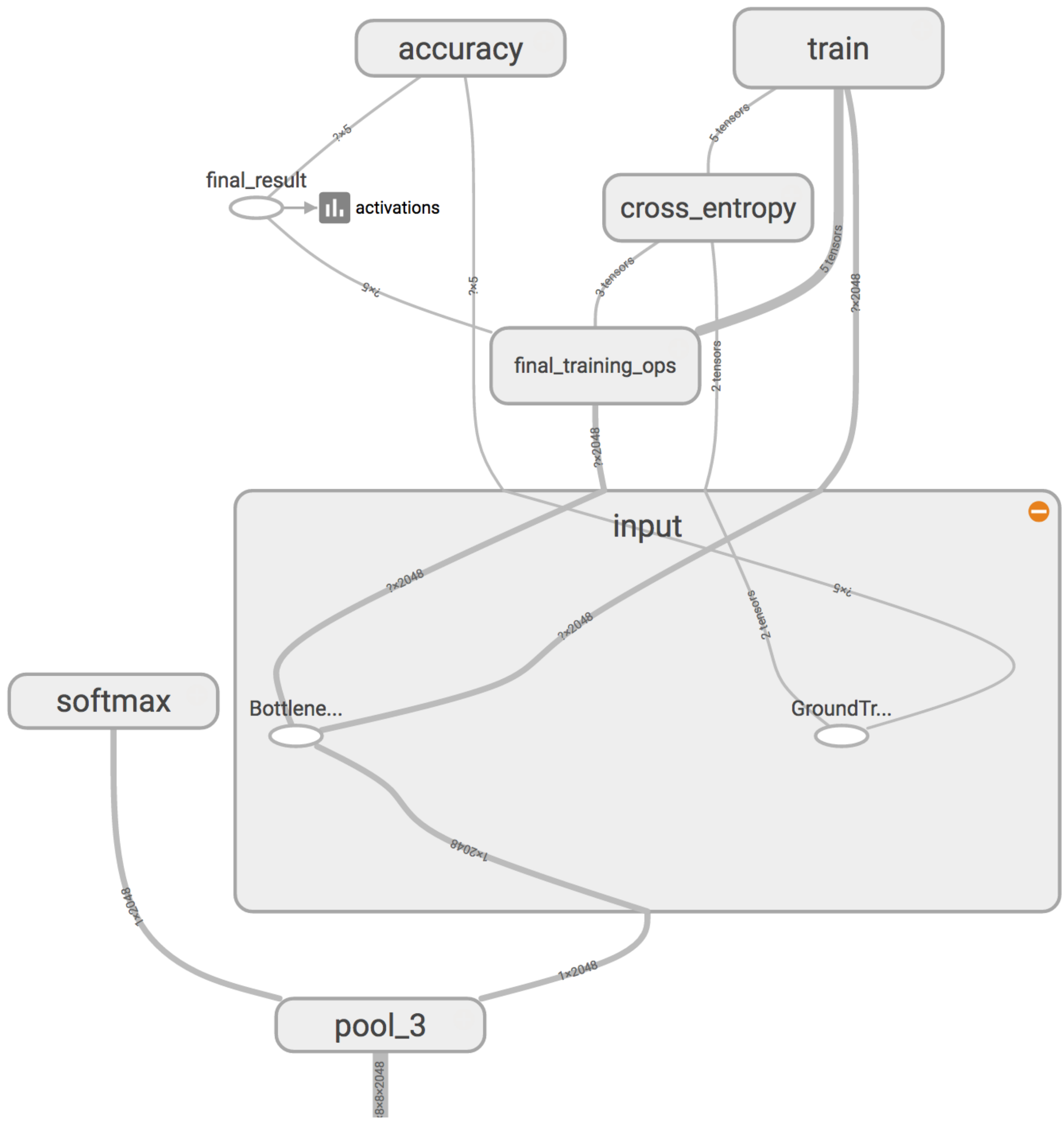
In [29]:

```
# Set parameters for applying any distortions.
do_distort_images = should_distort_images(
    flip_left_right=False,
    random_crop=0,
    random_scale=0,
    random_brightness=0)
```

Bottlenecks

What's a bottleneck?

These models are made up of many layers stacked on top of each other, a simplified picture of Inception V3 from TensorBoard, is shown above (all the details are available in this paper, with a complete picture on page 6). These layers are pre-trained and are already very valuable at finding and summarizing information that will help classify most images. For this codelab, you are training only the last layer (final_training_ops in the figure below). While all the previous layers retain their already-trained state.



In the above figure, the node labeled "softmax", on the left side, is the output layer of the original model. While all the nodes to the right of the "softmax" were added by the retraining script.

In [30]:
```
# Hyperparameters
image_dir = "../tf_files/Clothes/"
bottleneck_dir = "bottleneck_dir"
how_many_training_steps = 1000
train_batch_size = 100
validation_batch_size = 100
test_batch_size = 100
eval_step_interval = 10
intermediate_store_frequency = 0
intermediate_output_graphs_dir = "intermediate_graphs"
final_tensor_name="final_result"
learning_rate = 0.01
summaries_dir = "retrain_logs"
```

```
output_labels = 'output_labels.txt'
output_graph = 'output_graph.pb'
print_misclassified_test_images = False

with tf.Session(graph=graph) as sess:
    # Set up the image decoding sub-graph.
    jpeg_data_tensor, decoded_image_tensor = add_jpeg_decoding(
        model_info['input_width'], model_info['input_height'],
        model_info['input_depth'], model_info['input_mean'],
        model_info['input_std'])

    if do_distort_images:
      # We will be applying distortions, so setup the operations we
'll need.
      (distorted_jpeg_data_tensor,
       distorted_image_tensor) = add_input_distortions(
           flip_left_right, random_crop, random_scale,
           random_brightness, model_info['input_width'],
           model_info['input_height'], model_info['input_depth'],
           model_info['input_mean'], model_info['input_std'])
    else:
      # We'll make sure we've calculated the 'bottleneck' image sum
maries and
      # cached them on disk.
      cache_bottlenecks(sess, image_lists, image_dir,
                        bottleneck_dir, jpeg_data_tensor,
                        decoded_image_tensor, resized_image_tensor,
                        bottleneck_tensor, architecture)

    # Add the new layer that we'll be training.
    (train_step, cross_entropy, bottleneck_input, ground_truth_inpu
t,
     final_tensor) = add_final_training_ops(
         len(image_lists.keys()), final_tensor_name, bottleneck_ten
sor,
         model_info['bottleneck_tensor_size'])

    # Create the operations we need to evaluate the accuracy of our
new layer.
    evaluation_step, prediction = add_evaluation_step(
        final_tensor, ground_truth_input)

    # Merge all the summaries and write them out to the summaries_d
ir
    merged = tf.summary.merge_all()
    train_writer = tf.summary.FileWriter(summaries_dir + '/train',
sess.graph)

    validation_writer = tf.summary.FileWriter(summaries_dir + '/val
idation')

    # Set up all our weights to their initial default values.
    init = tf.global_variables_initializer()
    sess.run(init)

    # Run the training for as many cycles as requested on the comma
nd line.
```

```
    for i in range(how_many_training_steps):
      # Get a batch of input bottleneck values, either calculated f
resh every
      # time with distortions applied, or from the cache stored on
disk.
      if do_distort_images:
        (train_bottlenecks,
         train_ground_truth) = get_random_distorted_bottlenecks(
             sess, image_lists, train_batch_size, 'training',
             image_dir, distorted_jpeg_data_tensor,
             distorted_image_tensor, resized_image_tensor, bottlene
ck_tensor)
      else:
        (train_bottlenecks,
         train_ground_truth, _) = get_random_cached_bottlenecks(
             sess, image_lists, train_batch_size, 'training',
             bottleneck_dir, image_dir, jpeg_data_tensor,
             decoded_image_tensor, resized_image_tensor, bottleneck
_tensor,
             architecture)

      # Feed the bottlenecks and ground truth into the graph, and r
un a training
      # step. Capture training summaries for TensorBoard with the `
merged` op.
      train_summary, _ = sess.run(
          [merged, train_step],
          feed_dict={bottleneck_input: train_bottlenecks,
                     ground_truth_input: train_ground_truth})
      train_writer.add_summary(train_summary, i)

      # Every so often, print out how well the graph is training.
      is_last_step = (i + 1 == how_many_training_steps)
      if (i % eval_step_interval) == 0 or is_last_step:
        train_accuracy, cross_entropy_value = sess.run(
            [evaluation_step, cross_entropy],
            feed_dict={bottleneck_input: train_bottlenecks,
                       ground_truth_input: train_ground_truth})
        tf.logging.info('%s: Step %d: Train accuracy = %.1f%%' %
                        (datetime.now(), i, train_accuracy * 100))
        tf.logging.info('%s: Step %d: Cross entropy = %f' %
                        (datetime.now(), i, cross_entropy_value))
        validation_bottlenecks, validation_ground_truth, _ = (
            get_random_cached_bottlenecks(
                sess, image_lists, validation_batch_size, 'validati
on',
                bottleneck_dir, image_dir, jpeg_data_tensor,
                decoded_image_tensor, resized_image_tensor, bottlen
eck_tensor,
                architecture))
        # Run a validation step and capture training summaries for
TensorBoard
        # with the `merged` op.
        validation_summary, validation_accuracy = sess.run(
            [merged, evaluation_step],
            feed_dict={bottleneck_input: validation_bottlenecks,
                       ground_truth_input: validation_ground_truth}
```

```
)
        validation_writer.add_summary(validation_summary, i)
        tf.logging.info('%s: Step %d: Validation accuracy = %.1f%%
(N=%d)' %
                        (datetime.now(), i, validation_accuracy * 1
00,
                         len(validation_bottlenecks)))

      # Store intermediate results
      intermediate_frequency = intermediate_store_frequency

      if (intermediate_frequency > 0 and (i % intermediate_frequenc
y == 0)
          and i > 0):
        intermediate_file_name = (intermediate_output_graphs_dir +
                                  'intermediate_' + str(i) + '.pb')
        tf.logging.info('Save intermediate result to : ' +
                        intermediate_file_name)
        save_graph_to_file(sess, graph, intermediate_file_name)

    # We've completed all our training, so run a final test evaluat
ion on
    # some new images we haven't used before.
    test_bottlenecks, test_ground_truth, test_filenames = (
        get_random_cached_bottlenecks(
            sess, image_lists, test_batch_size, 'testing',
            bottleneck_dir, image_dir, jpeg_data_tensor,
            decoded_image_tensor, resized_image_tensor, bottleneck_
tensor,
            architecture))
    test_accuracy, predictions = sess.run(
        [evaluation_step, prediction],
        feed_dict={bottleneck_input: test_bottlenecks,
                   ground_truth_input: test_ground_truth})
    tf.logging.info('Final test accuracy = %.1f%% (N=%d)' %
                    (test_accuracy * 100, len(test_bottlenecks)))

    if print_misclassified_test_images:
      tf.logging.info('=== MISCLASSIFIED TEST IMAGES ===')
      for i, test_filename in enumerate(test_filenames):
        if predictions[i] != test_ground_truth[i].argmax():
          tf.logging.info('%70s  %s' %
                          (test_filename,
                           list(image_lists.keys())[predictions[i]]
))

    # Write out the trained graph and labels with the weights store
d as
    # constants.
    save_graph_to_file(sess, graph, output_graph)
    with gfile.FastGFile(output_labels, 'w') as f:
      f.write('\n'.join(image_lists.keys()) + '\n')
```

```
INFO:tensorflow:100 bottleneck files created.
INFO:tensorflow:200 bottleneck files created.
INFO:tensorflow:300 bottleneck files created.
INFO:tensorflow:400 bottleneck files created.
```

```
INFO:tensorflow:500 bottleneck files created.
INFO:tensorflow:600 bottleneck files created.
INFO:tensorflow:700 bottleneck files created.
INFO:tensorflow:800 bottleneck files created.
INFO:tensorflow:900 bottleneck files created.
INFO:tensorflow:1000 bottleneck files created.
INFO:tensorflow:1100 bottleneck files created.
INFO:tensorflow:1200 bottleneck files created.
INFO:tensorflow:1300 bottleneck files created.
INFO:tensorflow:1400 bottleneck files created.
INFO:tensorflow:1500 bottleneck files created.
INFO:tensorflow:1600 bottleneck files created.
INFO:tensorflow:1700 bottleneck files created.
INFO:tensorflow:1800 bottleneck files created.
INFO:tensorflow:1900 bottleneck files created.
INFO:tensorflow:2000 bottleneck files created.
INFO:tensorflow:2100 bottleneck files created.
INFO:tensorflow:2200 bottleneck files created.
INFO:tensorflow:2300 bottleneck files created.
INFO:tensorflow:2400 bottleneck files created.
INFO:tensorflow:2500 bottleneck files created.
INFO:tensorflow:2600 bottleneck files created.
INFO:tensorflow:2700 bottleneck files created.
INFO:tensorflow:2800 bottleneck files created.
INFO:tensorflow:2900 bottleneck files created.
INFO:tensorflow:3000 bottleneck files created.
INFO:tensorflow:3100 bottleneck files created.
INFO:tensorflow:3200 bottleneck files created.
INFO:tensorflow:3300 bottleneck files created.
INFO:tensorflow:3400 bottleneck files created.
INFO:tensorflow:3500 bottleneck files created.
INFO:tensorflow:3600 bottleneck files created.
INFO:tensorflow:3700 bottleneck files created.
INFO:tensorflow:3800 bottleneck files created.
INFO:tensorflow:3900 bottleneck files created.
INFO:tensorflow:4000 bottleneck files created.
INFO:tensorflow:4100 bottleneck files created.
INFO:tensorflow:4200 bottleneck files created.
INFO:tensorflow:4300 bottleneck files created.
INFO:tensorflow:4400 bottleneck files created.
INFO:tensorflow:4500 bottleneck files created.
INFO:tensorflow:4600 bottleneck files created.
INFO:tensorflow:4700 bottleneck files created.
INFO:tensorflow:4800 bottleneck files created.
INFO:tensorflow:4900 bottleneck files created.
INFO:tensorflow:5000 bottleneck files created.
INFO:tensorflow:5100 bottleneck files created.
INFO:tensorflow:5200 bottleneck files created.
INFO:tensorflow:5300 bottleneck files created.
INFO:tensorflow:5400 bottleneck files created.
INFO:tensorflow:5500 bottleneck files created.
INFO:tensorflow:5600 bottleneck files created.
INFO:tensorflow:5700 bottleneck files created.
INFO:tensorflow:5800 bottleneck files created.
INFO:tensorflow:5900 bottleneck files created.
INFO:tensorflow:6000 bottleneck files created.
INFO:tensorflow:6100 bottleneck files created.
```

```
INFO:tensorflow:6200 bottleneck files created.
INFO:tensorflow:6300 bottleneck files created.
INFO:tensorflow:6400 bottleneck files created.
INFO:tensorflow:6500 bottleneck files created.
INFO:tensorflow:6600 bottleneck files created.
INFO:tensorflow:6700 bottleneck files created.
INFO:tensorflow:6800 bottleneck files created.
INFO:tensorflow:6900 bottleneck files created.
INFO:tensorflow:7000 bottleneck files created.
INFO:tensorflow:7100 bottleneck files created.
INFO:tensorflow:7200 bottleneck files created.
INFO:tensorflow:7300 bottleneck files created.
INFO:tensorflow:7400 bottleneck files created.
INFO:tensorflow:7500 bottleneck files created.
INFO:tensorflow:7600 bottleneck files created.
INFO:tensorflow:7700 bottleneck files created.
INFO:tensorflow:7800 bottleneck files created.
INFO:tensorflow:7900 bottleneck files created.
INFO:tensorflow:8000 bottleneck files created.
INFO:tensorflow:8100 bottleneck files created.
INFO:tensorflow:8200 bottleneck files created.
INFO:tensorflow:8300 bottleneck files created.
INFO:tensorflow:8400 bottleneck files created.
INFO:tensorflow:8500 bottleneck files created.
INFO:tensorflow:8600 bottleneck files created.
INFO:tensorflow:8700 bottleneck files created.
INFO:tensorflow:8800 bottleneck files created.
INFO:tensorflow:8900 bottleneck files created.
INFO:tensorflow:9000 bottleneck files created.
INFO:tensorflow:9100 bottleneck files created.
INFO:tensorflow:9200 bottleneck files created.
INFO:tensorflow:9300 bottleneck files created.
INFO:tensorflow:9400 bottleneck files created.
INFO:tensorflow:9500 bottleneck files created.
INFO:tensorflow:9600 bottleneck files created.
INFO:tensorflow:9700 bottleneck files created.
INFO:tensorflow:9800 bottleneck files created.
INFO:tensorflow:9900 bottleneck files created.
INFO:tensorflow:10000 bottleneck files created.
INFO:tensorflow:10100 bottleneck files created.
INFO:tensorflow:10200 bottleneck files created.
INFO:tensorflow:10300 bottleneck files created.
INFO:tensorflow:10400 bottleneck files created.
INFO:tensorflow:10500 bottleneck files created.
INFO:tensorflow:10600 bottleneck files created.
INFO:tensorflow:10700 bottleneck files created.
INFO:tensorflow:10800 bottleneck files created.
INFO:tensorflow:10900 bottleneck files created.
INFO:tensorflow:11000 bottleneck files created.
INFO:tensorflow:11100 bottleneck files created.
INFO:tensorflow:11200 bottleneck files created.
INFO:tensorflow:11300 bottleneck files created.
INFO:tensorflow:11400 bottleneck files created.
INFO:tensorflow:11500 bottleneck files created.
INFO:tensorflow:11600 bottleneck files created.
INFO:tensorflow:11700 bottleneck files created.
INFO:tensorflow:11800 bottleneck files created.
```

```
INFO:tensorflow:11900 bottleneck files created.
INFO:tensorflow:12000 bottleneck files created.
INFO:tensorflow:12100 bottleneck files created.
INFO:tensorflow:12200 bottleneck files created.
INFO:tensorflow:12300 bottleneck files created.
INFO:tensorflow:12400 bottleneck files created.
INFO:tensorflow:12500 bottleneck files created.
INFO:tensorflow:12600 bottleneck files created.
INFO:tensorflow:12700 bottleneck files created.
WARNING:tensorflow:From /anaconda3/lib/python3.6/site-packages/ten
sorflow/python/framework/op_def_library.py:263: colocate_with (fro
m tensorflow.python.framework.ops) is deprecated and will be remov
ed in a future version.
Instructions for updating:
Colocations handled automatically by placer.
WARNING:tensorflow:From <ipython-input-18-06d43fde6b60>:56: softma
x_cross_entropy_with_logits (from tensorflow.python.ops.nn_ops) is
deprecated and will be removed in a future version.
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`.

INFO:tensorflow:2019-04-19 16:22:34.621011: Step 0: Train accuracy
= 13.0%
INFO:tensorflow:2019-04-19 16:22:34.622101: Step 0: Cross entropy
= 3.061954
INFO:tensorflow:2019-04-19 16:22:35.042847: Step 0: Validation acc
uracy = 14.0% (N=100)
INFO:tensorflow:2019-04-19 16:22:35.793847: Step 10: Train accurac
y = 33.0%
INFO:tensorflow:2019-04-19 16:22:35.794574: Step 10: Cross entropy
= 5.970832
INFO:tensorflow:2019-04-19 16:22:35.866127: Step 10: Validation ac
curacy = 26.0% (N=100)
INFO:tensorflow:2019-04-19 16:22:36.608350: Step 20: Train accurac
y = 47.0%
INFO:tensorflow:2019-04-19 16:22:36.609042: Step 20: Cross entropy
= 2.818956
INFO:tensorflow:2019-04-19 16:22:36.682358: Step 20: Validation ac
curacy = 42.0% (N=100)
INFO:tensorflow:2019-04-19 16:22:37.397043: Step 30: Train accurac
y = 49.0%
INFO:tensorflow:2019-04-19 16:22:37.397732: Step 30: Cross entropy
= 2.694890
INFO:tensorflow:2019-04-19 16:22:37.470440: Step 30: Validation ac
curacy = 32.0% (N=100)
INFO:tensorflow:2019-04-19 16:22:38.216148: Step 40: Train accurac
y = 47.0%
INFO:tensorflow:2019-04-19 16:22:38.217453: Step 40: Cross entropy
= 1.861224
INFO:tensorflow:2019-04-19 16:22:38.331172: Step 40: Validation ac
curacy = 43.0% (N=100)
INFO:tensorflow:2019-04-19 16:22:39.092034: Step 50: Train accurac
y = 47.0%
```

```
INFO:tensorflow:2019-04-19 16:22:39.092847: Step 50: Cross entropy
= 2.080700
INFO:tensorflow:2019-04-19 16:22:39.169914: Step 50: Validation ac
curacy = 36.0% (N=100)
INFO:tensorflow:2019-04-19 16:22:39.911146: Step 60: Train accurac
y = 63.0%
INFO:tensorflow:2019-04-19 16:22:39.911856: Step 60: Cross entropy
= 1.402826
INFO:tensorflow:2019-04-19 16:22:39.992399: Step 60: Validation ac
curacy = 51.0% (N=100)
INFO:tensorflow:2019-04-19 16:22:40.704541: Step 70: Train accurac
y = 49.0%
INFO:tensorflow:2019-04-19 16:22:40.705791: Step 70: Cross entropy
= 2.175385
INFO:tensorflow:2019-04-19 16:22:40.775881: Step 70: Validation ac
curacy = 43.0% (N=100)
INFO:tensorflow:2019-04-19 16:22:41.490153: Step 80: Train accurac
y = 58.0%
INFO:tensorflow:2019-04-19 16:22:41.490872: Step 80: Cross entropy
= 1.719470
INFO:tensorflow:2019-04-19 16:22:41.560025: Step 80: Validation ac
curacy = 48.0% (N=100)
INFO:tensorflow:2019-04-19 16:22:42.287766: Step 90: Train accurac
y = 51.0%
INFO:tensorflow:2019-04-19 16:22:42.288470: Step 90: Cross entropy
= 1.705877
INFO:tensorflow:2019-04-19 16:22:42.368376: Step 90: Validation ac
curacy = 43.0% (N=100)
INFO:tensorflow:2019-04-19 16:22:43.102003: Step 100: Train accura
cy = 65.0%
INFO:tensorflow:2019-04-19 16:22:43.102722: Step 100: Cross entrop
y = 1.176083
INFO:tensorflow:2019-04-19 16:22:43.171548: Step 100: Validation a
ccuracy = 54.0% (N=100)
INFO:tensorflow:2019-04-19 16:22:43.817913: Step 110: Train accura
cy = 77.0%
INFO:tensorflow:2019-04-19 16:22:43.818642: Step 110: Cross entrop
y = 1.078799
INFO:tensorflow:2019-04-19 16:22:43.888779: Step 110: Validation a
ccuracy = 48.0% (N=100)
INFO:tensorflow:2019-04-19 16:22:44.540810: Step 120: Train accura
cy = 63.0%
INFO:tensorflow:2019-04-19 16:22:44.541730: Step 120: Cross entrop
y = 1.124328
INFO:tensorflow:2019-04-19 16:22:44.620807: Step 120: Validation a
ccuracy = 60.0% (N=100)
INFO:tensorflow:2019-04-19 16:22:45.333577: Step 130: Train accura
cy = 61.0%
INFO:tensorflow:2019-04-19 16:22:45.334273: Step 130: Cross entrop
y = 1.288900
INFO:tensorflow:2019-04-19 16:22:45.399620: Step 130: Validation a
ccuracy = 53.0% (N=100)
INFO:tensorflow:2019-04-19 16:22:46.066121: Step 140: Train accura
cy = 71.0%
INFO:tensorflow:2019-04-19 16:22:46.066818: Step 140: Cross entrop
y = 0.845092
INFO:tensorflow:2019-04-19 16:22:46.140289: Step 140: Validation a
```

```
ccuracy = 64.0% (N=100)
INFO:tensorflow:2019-04-19 16:22:46.810528: Step 150: Train accura
cy = 69.0%
INFO:tensorflow:2019-04-19 16:22:46.811406: Step 150: Cross entrop
y = 1.379404
INFO:tensorflow:2019-04-19 16:22:46.875973: Step 150: Validation a
ccuracy = 61.0% (N=100)
INFO:tensorflow:2019-04-19 16:22:47.516614: Step 160: Train accura
cy = 64.0%
INFO:tensorflow:2019-04-19 16:22:47.517501: Step 160: Cross entrop
y = 0.988481
INFO:tensorflow:2019-04-19 16:22:47.585937: Step 160: Validation a
ccuracy = 58.0% (N=100)
INFO:tensorflow:2019-04-19 16:22:48.270022: Step 170: Train accura
cy = 69.0%
INFO:tensorflow:2019-04-19 16:22:48.270817: Step 170: Cross entrop
y = 1.490823
INFO:tensorflow:2019-04-19 16:22:48.337132: Step 170: Validation a
ccuracy = 61.0% (N=100)
INFO:tensorflow:2019-04-19 16:22:49.006550: Step 180: Train accura
cy = 63.0%
INFO:tensorflow:2019-04-19 16:22:49.007254: Step 180: Cross entrop
y = 1.167296
INFO:tensorflow:2019-04-19 16:22:49.079963: Step 180: Validation a
ccuracy = 55.0% (N=100)
INFO:tensorflow:2019-04-19 16:22:49.758225: Step 190: Train accura
cy = 77.0%
INFO:tensorflow:2019-04-19 16:22:49.759275: Step 190: Cross entrop
y = 0.748049
INFO:tensorflow:2019-04-19 16:22:49.824492: Step 190: Validation a
ccuracy = 72.0% (N=100)
INFO:tensorflow:2019-04-19 16:22:50.500612: Step 200: Train accura
cy = 74.0%
INFO:tensorflow:2019-04-19 16:22:50.501382: Step 200: Cross entrop
y = 0.937247
INFO:tensorflow:2019-04-19 16:22:50.573595: Step 200: Validation a
ccuracy = 63.0% (N=100)
INFO:tensorflow:2019-04-19 16:22:51.250296: Step 210: Train accura
cy = 73.0%
INFO:tensorflow:2019-04-19 16:22:51.250987: Step 210: Cross entrop
y = 0.956763
INFO:tensorflow:2019-04-19 16:22:51.319636: Step 210: Validation a
ccuracy = 60.0% (N=100)
INFO:tensorflow:2019-04-19 16:22:51.994690: Step 220: Train accura
cy = 67.0%
INFO:tensorflow:2019-04-19 16:22:51.995391: Step 220: Cross entrop
y = 1.060545
INFO:tensorflow:2019-04-19 16:22:52.061403: Step 220: Validation a
ccuracy = 61.0% (N=100)
INFO:tensorflow:2019-04-19 16:22:52.710160: Step 230: Train accura
cy = 67.0%
INFO:tensorflow:2019-04-19 16:22:52.710877: Step 230: Cross entrop
y = 0.962022
INFO:tensorflow:2019-04-19 16:22:52.778727: Step 230: Validation a
ccuracy = 58.0% (N=100)
INFO:tensorflow:2019-04-19 16:22:53.424280: Step 240: Train accura
cy = 70.0%
```

```
INFO:tensorflow:2019-04-19 16:22:53.424947: Step 240: Cross entrop
y = 1.020624
INFO:tensorflow:2019-04-19 16:22:53.493984: Step 240: Validation a
ccuracy = 62.0% (N=100)
INFO:tensorflow:2019-04-19 16:22:54.114153: Step 250: Train accura
cy = 71.0%
INFO:tensorflow:2019-04-19 16:22:54.114864: Step 250: Cross entrop
y = 1.046482
INFO:tensorflow:2019-04-19 16:22:54.182880: Step 250: Validation a
ccuracy = 60.0% (N=100)
INFO:tensorflow:2019-04-19 16:22:54.836279: Step 260: Train accura
cy = 72.0%
INFO:tensorflow:2019-04-19 16:22:54.836967: Step 260: Cross entrop
y = 1.159551
INFO:tensorflow:2019-04-19 16:22:54.908544: Step 260: Validation a
ccuracy = 61.0% (N=100)
INFO:tensorflow:2019-04-19 16:22:55.537605: Step 270: Train accura
cy = 70.0%
INFO:tensorflow:2019-04-19 16:22:55.538301: Step 270: Cross entrop
y = 0.994852
INFO:tensorflow:2019-04-19 16:22:55.605295: Step 270: Validation a
ccuracy = 72.0% (N=100)
INFO:tensorflow:2019-04-19 16:22:56.244780: Step 280: Train accura
cy = 68.0%
INFO:tensorflow:2019-04-19 16:22:56.245588: Step 280: Cross entrop
y = 0.785224
INFO:tensorflow:2019-04-19 16:22:56.307007: Step 280: Validation a
ccuracy = 48.0% (N=100)
INFO:tensorflow:2019-04-19 16:22:56.963394: Step 290: Train accura
cy = 72.0%
INFO:tensorflow:2019-04-19 16:22:56.964116: Step 290: Cross entrop
y = 0.837372
INFO:tensorflow:2019-04-19 16:22:57.030457: Step 290: Validation a
ccuracy = 59.0% (N=100)
INFO:tensorflow:2019-04-19 16:22:57.699055: Step 300: Train accura
cy = 78.0%
INFO:tensorflow:2019-04-19 16:22:57.699729: Step 300: Cross entrop
y = 0.653664
INFO:tensorflow:2019-04-19 16:22:57.771122: Step 300: Validation a
ccuracy = 66.0% (N=100)
INFO:tensorflow:2019-04-19 16:22:58.404866: Step 310: Train accura
cy = 74.0%
INFO:tensorflow:2019-04-19 16:22:58.405585: Step 310: Cross entrop
y = 0.891529
INFO:tensorflow:2019-04-19 16:22:58.484672: Step 310: Validation a
ccuracy = 62.0% (N=100)
INFO:tensorflow:2019-04-19 16:22:59.140213: Step 320: Train accura
cy = 67.0%
INFO:tensorflow:2019-04-19 16:22:59.140898: Step 320: Cross entrop
y = 1.207222
INFO:tensorflow:2019-04-19 16:22:59.206804: Step 320: Validation a
ccuracy = 66.0% (N=100)
INFO:tensorflow:2019-04-19 16:22:59.830683: Step 330: Train accura
cy = 75.0%
INFO:tensorflow:2019-04-19 16:22:59.831528: Step 330: Cross entrop
y = 0.942613
INFO:tensorflow:2019-04-19 16:22:59.892697: Step 330: Validation a
```

```
ccuracy = 54.0% (N=100)
INFO:tensorflow:2019-04-19 16:23:00.501672: Step 340: Train accura
cy = 71.0%
INFO:tensorflow:2019-04-19 16:23:00.502428: Step 340: Cross entrop
y = 1.114180
INFO:tensorflow:2019-04-19 16:23:00.566323: Step 340: Validation a
ccuracy = 64.0% (N=100)
INFO:tensorflow:2019-04-19 16:23:01.168111: Step 350: Train accura
cy = 73.0%
INFO:tensorflow:2019-04-19 16:23:01.168812: Step 350: Cross entrop
y = 0.943369
INFO:tensorflow:2019-04-19 16:23:01.231134: Step 350: Validation a
ccuracy = 72.0% (N=100)
INFO:tensorflow:2019-04-19 16:23:01.825483: Step 360: Train accura
cy = 78.0%
INFO:tensorflow:2019-04-19 16:23:01.826369: Step 360: Cross entrop
y = 0.686456
INFO:tensorflow:2019-04-19 16:23:01.890009: Step 360: Validation a
ccuracy = 70.0% (N=100)
INFO:tensorflow:2019-04-19 16:23:02.486034: Step 370: Train accura
cy = 65.0%
INFO:tensorflow:2019-04-19 16:23:02.486742: Step 370: Cross entrop
y = 1.698057
INFO:tensorflow:2019-04-19 16:23:02.550818: Step 370: Validation a
ccuracy = 58.0% (N=100)
INFO:tensorflow:2019-04-19 16:23:03.128962: Step 380: Train accura
cy = 71.0%
INFO:tensorflow:2019-04-19 16:23:03.129665: Step 380: Cross entrop
y = 0.992413
INFO:tensorflow:2019-04-19 16:23:03.189417: Step 380: Validation a
ccuracy = 70.0% (N=100)
INFO:tensorflow:2019-04-19 16:23:03.766007: Step 390: Train accura
cy = 84.0%
INFO:tensorflow:2019-04-19 16:23:03.766796: Step 390: Cross entrop
y = 0.670173
INFO:tensorflow:2019-04-19 16:23:03.826535: Step 390: Validation a
ccuracy = 65.0% (N=100)
INFO:tensorflow:2019-04-19 16:23:04.394713: Step 400: Train accura
cy = 70.0%
INFO:tensorflow:2019-04-19 16:23:04.395582: Step 400: Cross entrop
y = 0.997186
INFO:tensorflow:2019-04-19 16:23:04.458617: Step 400: Validation a
ccuracy = 57.0% (N=100)
INFO:tensorflow:2019-04-19 16:23:05.037170: Step 410: Train accura
cy = 86.0%
INFO:tensorflow:2019-04-19 16:23:05.037892: Step 410: Cross entrop
y = 0.494253
INFO:tensorflow:2019-04-19 16:23:05.099990: Step 410: Validation a
ccuracy = 66.0% (N=100)
INFO:tensorflow:2019-04-19 16:23:05.655968: Step 420: Train accura
cy = 71.0%
INFO:tensorflow:2019-04-19 16:23:05.656985: Step 420: Cross entrop
y = 1.435556
INFO:tensorflow:2019-04-19 16:23:05.715607: Step 420: Validation a
ccuracy = 66.0% (N=100)
INFO:tensorflow:2019-04-19 16:23:06.269585: Step 430: Train accura
cy = 64.0%
```

```
INFO:tensorflow:2019-04-19 16:23:06.270314: Step 430: Cross entrop
y = 0.933639
INFO:tensorflow:2019-04-19 16:23:06.329115: Step 430: Validation a
ccuracy = 63.0% (N=100)
INFO:tensorflow:2019-04-19 16:23:06.888705: Step 440: Train accura
cy = 78.0%
INFO:tensorflow:2019-04-19 16:23:06.889522: Step 440: Cross entrop
y = 1.054687
INFO:tensorflow:2019-04-19 16:23:06.951761: Step 440: Validation a
ccuracy = 64.0% (N=100)
INFO:tensorflow:2019-04-19 16:23:07.496110: Step 450: Train accura
cy = 78.0%
INFO:tensorflow:2019-04-19 16:23:07.496914: Step 450: Cross entrop
y = 1.018594
INFO:tensorflow:2019-04-19 16:23:07.549541: Step 450: Validation a
ccuracy = 65.0% (N=100)
INFO:tensorflow:2019-04-19 16:23:08.104181: Step 460: Train accura
cy = 68.0%
INFO:tensorflow:2019-04-19 16:23:08.104821: Step 460: Cross entrop
y = 1.577670
INFO:tensorflow:2019-04-19 16:23:08.167025: Step 460: Validation a
ccuracy = 61.0% (N=100)
INFO:tensorflow:2019-04-19 16:23:08.766593: Step 470: Train accura
cy = 71.0%
INFO:tensorflow:2019-04-19 16:23:08.767334: Step 470: Cross entrop
y = 1.936523
INFO:tensorflow:2019-04-19 16:23:08.826052: Step 470: Validation a
ccuracy = 61.0% (N=100)
INFO:tensorflow:2019-04-19 16:23:09.359435: Step 480: Train accura
cy = 75.0%
INFO:tensorflow:2019-04-19 16:23:09.360134: Step 480: Cross entrop
y = 0.681986
INFO:tensorflow:2019-04-19 16:23:09.414236: Step 480: Validation a
ccuracy = 70.0% (N=100)
INFO:tensorflow:2019-04-19 16:23:09.956726: Step 490: Train accura
cy = 71.0%
INFO:tensorflow:2019-04-19 16:23:09.957829: Step 490: Cross entrop
y = 1.089501
INFO:tensorflow:2019-04-19 16:23:10.015367: Step 490: Validation a
ccuracy = 69.0% (N=100)
INFO:tensorflow:2019-04-19 16:23:10.564535: Step 500: Train accura
cy = 74.0%
INFO:tensorflow:2019-04-19 16:23:10.565438: Step 500: Cross entrop
y = 0.845896
INFO:tensorflow:2019-04-19 16:23:10.630128: Step 500: Validation a
ccuracy = 63.0% (N=100)
INFO:tensorflow:2019-04-19 16:23:11.177526: Step 510: Train accura
cy = 64.0%
INFO:tensorflow:2019-04-19 16:23:11.178254: Step 510: Cross entrop
y = 1.158226
INFO:tensorflow:2019-04-19 16:23:11.238216: Step 510: Validation a
ccuracy = 55.0% (N=100)
INFO:tensorflow:2019-04-19 16:23:11.799027: Step 520: Train accura
cy = 76.0%
INFO:tensorflow:2019-04-19 16:23:11.799764: Step 520: Cross entrop
y = 0.994964
INFO:tensorflow:2019-04-19 16:23:11.869502: Step 520: Validation a
```

```
ccuracy = 55.0% (N=100)
INFO:tensorflow:2019-04-19 16:23:12.444082: Step 530: Train accura
cy = 67.0%
INFO:tensorflow:2019-04-19 16:23:12.444836: Step 530: Cross entrop
y = 0.788251
INFO:tensorflow:2019-04-19 16:23:12.509681: Step 530: Validation a
ccuracy = 68.0% (N=100)
INFO:tensorflow:2019-04-19 16:23:13.105740: Step 540: Train accura
cy = 84.0%
INFO:tensorflow:2019-04-19 16:23:13.106507: Step 540: Cross entrop
y = 0.357493
INFO:tensorflow:2019-04-19 16:23:13.166747: Step 540: Validation a
ccuracy = 76.0% (N=100)
INFO:tensorflow:2019-04-19 16:23:13.816167: Step 550: Train accura
cy = 83.0%
INFO:tensorflow:2019-04-19 16:23:13.816968: Step 550: Cross entrop
y = 0.541631
INFO:tensorflow:2019-04-19 16:23:13.886190: Step 550: Validation a
ccuracy = 65.0% (N=100)
INFO:tensorflow:2019-04-19 16:23:14.451239: Step 560: Train accura
cy = 77.0%
INFO:tensorflow:2019-04-19 16:23:14.451964: Step 560: Cross entrop
y = 0.639227
INFO:tensorflow:2019-04-19 16:23:14.507766: Step 560: Validation a
ccuracy = 68.0% (N=100)
INFO:tensorflow:2019-04-19 16:23:15.082511: Step 570: Train accura
cy = 70.0%
INFO:tensorflow:2019-04-19 16:23:15.083256: Step 570: Cross entrop
y = 0.806414
INFO:tensorflow:2019-04-19 16:23:15.148618: Step 570: Validation a
ccuracy = 66.0% (N=100)
INFO:tensorflow:2019-04-19 16:23:15.763760: Step 580: Train accura
cy = 77.0%
INFO:tensorflow:2019-04-19 16:23:15.764468: Step 580: Cross entrop
y = 0.723219
INFO:tensorflow:2019-04-19 16:23:15.828468: Step 580: Validation a
ccuracy = 58.0% (N=100)
INFO:tensorflow:2019-04-19 16:23:16.429737: Step 590: Train accura
cy = 77.0%
INFO:tensorflow:2019-04-19 16:23:16.430494: Step 590: Cross entrop
y = 0.669706
INFO:tensorflow:2019-04-19 16:23:16.494066: Step 590: Validation a
ccuracy = 71.0% (N=100)
INFO:tensorflow:2019-04-19 16:23:17.155851: Step 600: Train accura
cy = 64.0%
INFO:tensorflow:2019-04-19 16:23:17.156763: Step 600: Cross entrop
y = 1.023656
INFO:tensorflow:2019-04-19 16:23:17.224602: Step 600: Validation a
ccuracy = 58.0% (N=100)
INFO:tensorflow:2019-04-19 16:23:17.858732: Step 610: Train accura
cy = 75.0%
INFO:tensorflow:2019-04-19 16:23:17.859717: Step 610: Cross entrop
y = 0.744248
INFO:tensorflow:2019-04-19 16:23:17.933926: Step 610: Validation a
ccuracy = 66.0% (N=100)
INFO:tensorflow:2019-04-19 16:23:18.583887: Step 620: Train accura
cy = 74.0%
```

```
INFO:tensorflow:2019-04-19 16:23:18.584582: Step 620: Cross entrop
y = 0.942072
INFO:tensorflow:2019-04-19 16:23:18.646154: Step 620: Validation a
ccuracy = 66.0% (N=100)
INFO:tensorflow:2019-04-19 16:23:19.273459: Step 630: Train accura
cy = 80.0%
INFO:tensorflow:2019-04-19 16:23:19.274182: Step 630: Cross entrop
y = 0.640617
INFO:tensorflow:2019-04-19 16:23:19.333281: Step 630: Validation a
ccuracy = 74.0% (N=100)
INFO:tensorflow:2019-04-19 16:23:19.947050: Step 640: Train accura
cy = 75.0%
INFO:tensorflow:2019-04-19 16:23:19.947747: Step 640: Cross entrop
y = 0.990996
INFO:tensorflow:2019-04-19 16:23:20.015726: Step 640: Validation a
ccuracy = 60.0% (N=100)
INFO:tensorflow:2019-04-19 16:23:20.632183: Step 650: Train accura
cy = 72.0%
INFO:tensorflow:2019-04-19 16:23:20.632878: Step 650: Cross entrop
y = 1.168968
INFO:tensorflow:2019-04-19 16:23:20.689326: Step 650: Validation a
ccuracy = 68.0% (N=100)
INFO:tensorflow:2019-04-19 16:23:21.281094: Step 660: Train accura
cy = 76.0%
INFO:tensorflow:2019-04-19 16:23:21.281782: Step 660: Cross entrop
y = 0.773897
INFO:tensorflow:2019-04-19 16:23:21.339984: Step 660: Validation a
ccuracy = 77.0% (N=100)
INFO:tensorflow:2019-04-19 16:23:21.926816: Step 670: Train accura
cy = 75.0%
INFO:tensorflow:2019-04-19 16:23:21.927528: Step 670: Cross entrop
y = 0.680679
INFO:tensorflow:2019-04-19 16:23:21.992113: Step 670: Validation a
ccuracy = 74.0% (N=100)
INFO:tensorflow:2019-04-19 16:23:22.585953: Step 680: Train accura
cy = 79.0%
INFO:tensorflow:2019-04-19 16:23:22.586644: Step 680: Cross entrop
y = 0.738183
INFO:tensorflow:2019-04-19 16:23:22.661842: Step 680: Validation a
ccuracy = 71.0% (N=100)
INFO:tensorflow:2019-04-19 16:23:23.276436: Step 690: Train accura
cy = 77.0%
INFO:tensorflow:2019-04-19 16:23:23.277224: Step 690: Cross entrop
y = 0.727368
INFO:tensorflow:2019-04-19 16:23:23.339538: Step 690: Validation a
ccuracy = 76.0% (N=100)
INFO:tensorflow:2019-04-19 16:23:23.979166: Step 700: Train accura
cy = 77.0%
INFO:tensorflow:2019-04-19 16:23:23.979969: Step 700: Cross entrop
y = 0.759917
INFO:tensorflow:2019-04-19 16:23:24.049450: Step 700: Validation a
ccuracy = 71.0% (N=100)
INFO:tensorflow:2019-04-19 16:23:24.700168: Step 710: Train accura
cy = 77.0%
INFO:tensorflow:2019-04-19 16:23:24.700859: Step 710: Cross entrop
y = 0.833123
INFO:tensorflow:2019-04-19 16:23:24.771051: Step 710: Validation a
```

```
ccuracy = 66.0% (N=100)
INFO:tensorflow:2019-04-19 16:23:25.402086: Step 720: Train accura
cy = 79.0%
INFO:tensorflow:2019-04-19 16:23:25.402760: Step 720: Cross entrop
y = 0.910988
INFO:tensorflow:2019-04-19 16:23:25.466920: Step 720: Validation a
ccuracy = 72.0% (N=100)
INFO:tensorflow:2019-04-19 16:23:26.120265: Step 730: Train accura
cy = 74.0%
INFO:tensorflow:2019-04-19 16:23:26.120934: Step 730: Cross entrop
y = 0.803555
INFO:tensorflow:2019-04-19 16:23:26.187276: Step 730: Validation a
ccuracy = 68.0% (N=100)
INFO:tensorflow:2019-04-19 16:23:26.839986: Step 740: Train accura
cy = 75.0%
INFO:tensorflow:2019-04-19 16:23:26.840696: Step 740: Cross entrop
y = 1.122094
INFO:tensorflow:2019-04-19 16:23:26.910158: Step 740: Validation a
ccuracy = 65.0% (N=100)
INFO:tensorflow:2019-04-19 16:23:27.558743: Step 750: Train accura
cy = 83.0%
INFO:tensorflow:2019-04-19 16:23:27.560535: Step 750: Cross entrop
y = 0.737640
INFO:tensorflow:2019-04-19 16:23:27.632186: Step 750: Validation a
ccuracy = 69.0% (N=100)
INFO:tensorflow:2019-04-19 16:23:28.261408: Step 760: Train accura
cy = 80.0%
INFO:tensorflow:2019-04-19 16:23:28.262120: Step 760: Cross entrop
y = 0.706354
INFO:tensorflow:2019-04-19 16:23:28.336478: Step 760: Validation a
ccuracy = 66.0% (N=100)
INFO:tensorflow:2019-04-19 16:23:29.017443: Step 770: Train accura
cy = 79.0%
INFO:tensorflow:2019-04-19 16:23:29.018174: Step 770: Cross entrop
y = 0.500313
INFO:tensorflow:2019-04-19 16:23:29.083264: Step 770: Validation a
ccuracy = 67.0% (N=100)
INFO:tensorflow:2019-04-19 16:23:29.741646: Step 780: Train accura
cy = 76.0%
INFO:tensorflow:2019-04-19 16:23:29.742365: Step 780: Cross entrop
y = 0.924634
INFO:tensorflow:2019-04-19 16:23:29.807875: Step 780: Validation a
ccuracy = 65.0% (N=100)
INFO:tensorflow:2019-04-19 16:23:30.464603: Step 790: Train accura
cy = 68.0%
INFO:tensorflow:2019-04-19 16:23:30.465335: Step 790: Cross entrop
y = 0.981651
INFO:tensorflow:2019-04-19 16:23:30.534354: Step 790: Validation a
ccuracy = 62.0% (N=100)
INFO:tensorflow:2019-04-19 16:23:31.208256: Step 800: Train accura
cy = 68.0%
INFO:tensorflow:2019-04-19 16:23:31.208944: Step 800: Cross entrop
y = 1.254277
INFO:tensorflow:2019-04-19 16:23:31.277988: Step 800: Validation a
ccuracy = 51.0% (N=100)
INFO:tensorflow:2019-04-19 16:23:31.929800: Step 810: Train accura
cy = 70.0%
```

```
INFO:tensorflow:2019-04-19 16:23:31.930505: Step 810: Cross entrop
y = 0.758481
INFO:tensorflow:2019-04-19 16:23:32.000630: Step 810: Validation a
ccuracy = 72.0% (N=100)
INFO:tensorflow:2019-04-19 16:23:32.647575: Step 820: Train accura
cy = 79.0%
INFO:tensorflow:2019-04-19 16:23:32.648251: Step 820: Cross entrop
y = 0.721066
INFO:tensorflow:2019-04-19 16:23:32.719359: Step 820: Validation a
ccuracy = 68.0% (N=100)
INFO:tensorflow:2019-04-19 16:23:33.392154: Step 830: Train accura
cy = 75.0%
INFO:tensorflow:2019-04-19 16:23:33.392996: Step 830: Cross entrop
y = 0.832695
INFO:tensorflow:2019-04-19 16:23:33.476192: Step 830: Validation a
ccuracy = 64.0% (N=100)
INFO:tensorflow:2019-04-19 16:23:34.157655: Step 840: Train accura
cy = 85.0%
INFO:tensorflow:2019-04-19 16:23:34.158421: Step 840: Cross entrop
y = 0.595535
INFO:tensorflow:2019-04-19 16:23:34.234679: Step 840: Validation a
ccuracy = 67.0% (N=100)
INFO:tensorflow:2019-04-19 16:23:34.856031: Step 850: Train accura
cy = 77.0%
INFO:tensorflow:2019-04-19 16:23:34.856926: Step 850: Cross entrop
y = 0.993486
INFO:tensorflow:2019-04-19 16:23:34.924331: Step 850: Validation a
ccuracy = 63.0% (N=100)
INFO:tensorflow:2019-04-19 16:23:35.604663: Step 860: Train accura
cy = 77.0%
INFO:tensorflow:2019-04-19 16:23:35.605351: Step 860: Cross entrop
y = 0.818102
INFO:tensorflow:2019-04-19 16:23:35.680052: Step 860: Validation a
ccuracy = 75.0% (N=100)
INFO:tensorflow:2019-04-19 16:23:36.347821: Step 870: Train accura
cy = 74.0%
INFO:tensorflow:2019-04-19 16:23:36.348774: Step 870: Cross entrop
y = 1.220001
INFO:tensorflow:2019-04-19 16:23:36.415417: Step 870: Validation a
ccuracy = 71.0% (N=100)
INFO:tensorflow:2019-04-19 16:23:37.080998: Step 880: Train accura
cy = 69.0%
INFO:tensorflow:2019-04-19 16:23:37.081694: Step 880: Cross entrop
y = 0.883006
INFO:tensorflow:2019-04-19 16:23:37.156941: Step 880: Validation a
ccuracy = 71.0% (N=100)
INFO:tensorflow:2019-04-19 16:23:37.805704: Step 890: Train accura
cy = 82.0%
INFO:tensorflow:2019-04-19 16:23:37.806400: Step 890: Cross entrop
y = 0.709523
INFO:tensorflow:2019-04-19 16:23:37.870440: Step 890: Validation a
ccuracy = 69.0% (N=100)
INFO:tensorflow:2019-04-19 16:23:38.543251: Step 900: Train accura
cy = 77.0%
INFO:tensorflow:2019-04-19 16:23:38.543949: Step 900: Cross entrop
y = 1.080867
INFO:tensorflow:2019-04-19 16:23:38.617923: Step 900: Validation a
```

```
ccuracy = 68.0% (N=100)
INFO:tensorflow:2019-04-19 16:23:39.296248: Step 910: Train accura
cy = 64.0%
INFO:tensorflow:2019-04-19 16:23:39.297283: Step 910: Cross entrop
y = 1.254931
INFO:tensorflow:2019-04-19 16:23:39.365501: Step 910: Validation a
ccuracy = 59.0% (N=100)
INFO:tensorflow:2019-04-19 16:23:40.038755: Step 920: Train accura
cy = 72.0%
INFO:tensorflow:2019-04-19 16:23:40.039442: Step 920: Cross entrop
y = 0.893486
INFO:tensorflow:2019-04-19 16:23:40.104628: Step 920: Validation a
ccuracy = 74.0% (N=100)
INFO:tensorflow:2019-04-19 16:23:40.723413: Step 930: Train accura
cy = 81.0%
INFO:tensorflow:2019-04-19 16:23:40.724099: Step 930: Cross entrop
y = 0.609884
INFO:tensorflow:2019-04-19 16:23:40.795786: Step 930: Validation a
ccuracy = 76.0% (N=100)
INFO:tensorflow:2019-04-19 16:23:41.425433: Step 940: Train accura
cy = 77.0%
INFO:tensorflow:2019-04-19 16:23:41.426208: Step 940: Cross entrop
y = 0.643411
INFO:tensorflow:2019-04-19 16:23:41.494840: Step 940: Validation a
ccuracy = 75.0% (N=100)
INFO:tensorflow:2019-04-19 16:23:42.154237: Step 950: Train accura
cy = 81.0%
INFO:tensorflow:2019-04-19 16:23:42.155308: Step 950: Cross entrop
y = 0.532883
INFO:tensorflow:2019-04-19 16:23:42.219952: Step 950: Validation a
ccuracy = 68.0% (N=100)
INFO:tensorflow:2019-04-19 16:23:42.829749: Step 960: Train accura
cy = 80.0%
INFO:tensorflow:2019-04-19 16:23:42.830448: Step 960: Cross entrop
y = 0.573259
INFO:tensorflow:2019-04-19 16:23:42.896827: Step 960: Validation a
ccuracy = 62.0% (N=100)
INFO:tensorflow:2019-04-19 16:23:43.533685: Step 970: Train accura
cy = 81.0%
INFO:tensorflow:2019-04-19 16:23:43.534363: Step 970: Cross entrop
y = 0.607507
INFO:tensorflow:2019-04-19 16:23:43.599430: Step 970: Validation a
ccuracy = 63.0% (N=100)
INFO:tensorflow:2019-04-19 16:23:44.227920: Step 980: Train accura
cy = 76.0%
INFO:tensorflow:2019-04-19 16:23:44.229031: Step 980: Cross entrop
y = 0.717568
INFO:tensorflow:2019-04-19 16:23:44.295992: Step 980: Validation a
ccuracy = 65.0% (N=100)
INFO:tensorflow:2019-04-19 16:23:44.905825: Step 990: Train accura
cy = 86.0%
INFO:tensorflow:2019-04-19 16:23:44.906535: Step 990: Cross entrop
y = 0.538438
INFO:tensorflow:2019-04-19 16:23:44.970004: Step 990: Validation a
ccuracy = 69.0% (N=100)
INFO:tensorflow:2019-04-19 16:23:45.529867: Step 999: Train accura
cy = 81.0%
```

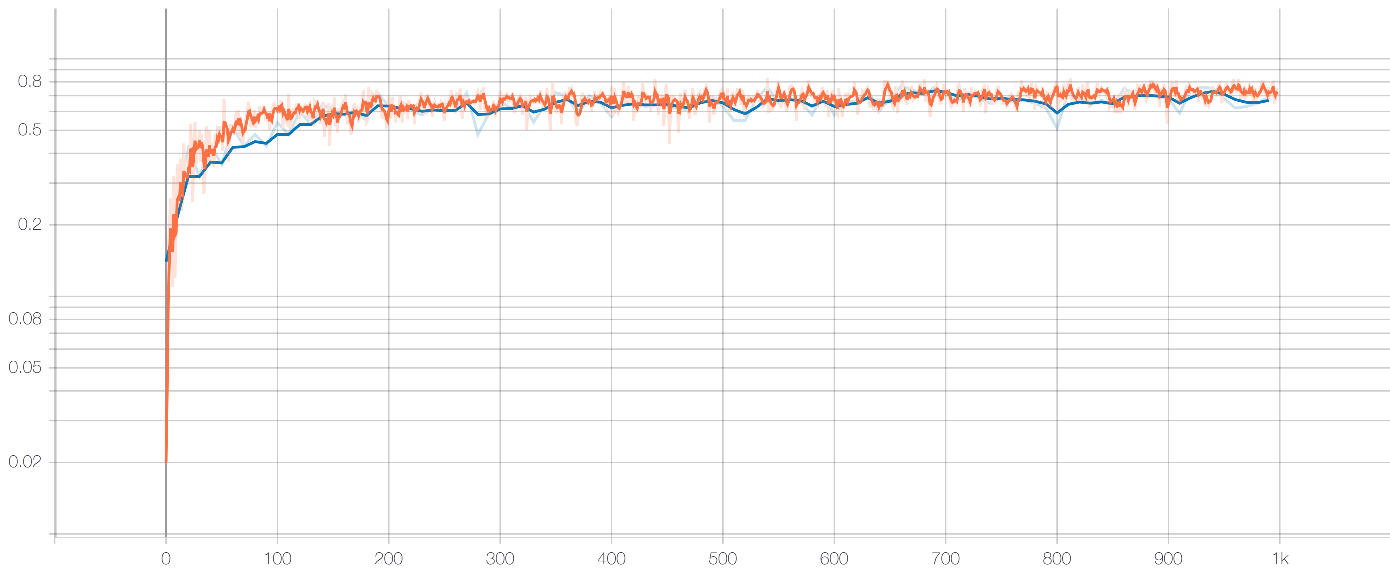
```
INFO:tensorflow:2019-04-19 16:23:45.530610: Step 999: Cross entrop
y = 0.593006
INFO:tensorflow:2019-04-19 16:23:45.594530: Step 999: Validation a
ccuracy = 57.0% (N=100)
INFO:tensorflow:Final test accuracy = 68.0% (N=100)
WARNING:tensorflow:From <ipython-input-20-50a32f7ca1b2>:3: convert
_variables_to_constants (from tensorflow.python.framework.graph_ut
il_impl) is deprecated and will be removed in a future version.
Instructions for updating:
Use tf.compat.v1.graph_util.convert_variables_to_constants
WARNING:tensorflow:From /anaconda3/lib/python3.6/site-packages/ten
sorflow/python/framework/graph_util_impl.py:245: extract_sub_graph
(from tensorflow.python.framework.graph_util_impl) is deprecated a
nd will be removed in a future version.
Instructions for updating:
Use tf.compat.v1.graph_util.extract_sub_graph
INFO:tensorflow:Froze 2 variables.
INFO:tensorflow:Converted 2 variables to const ops.
```

ANALYSE RESULTS

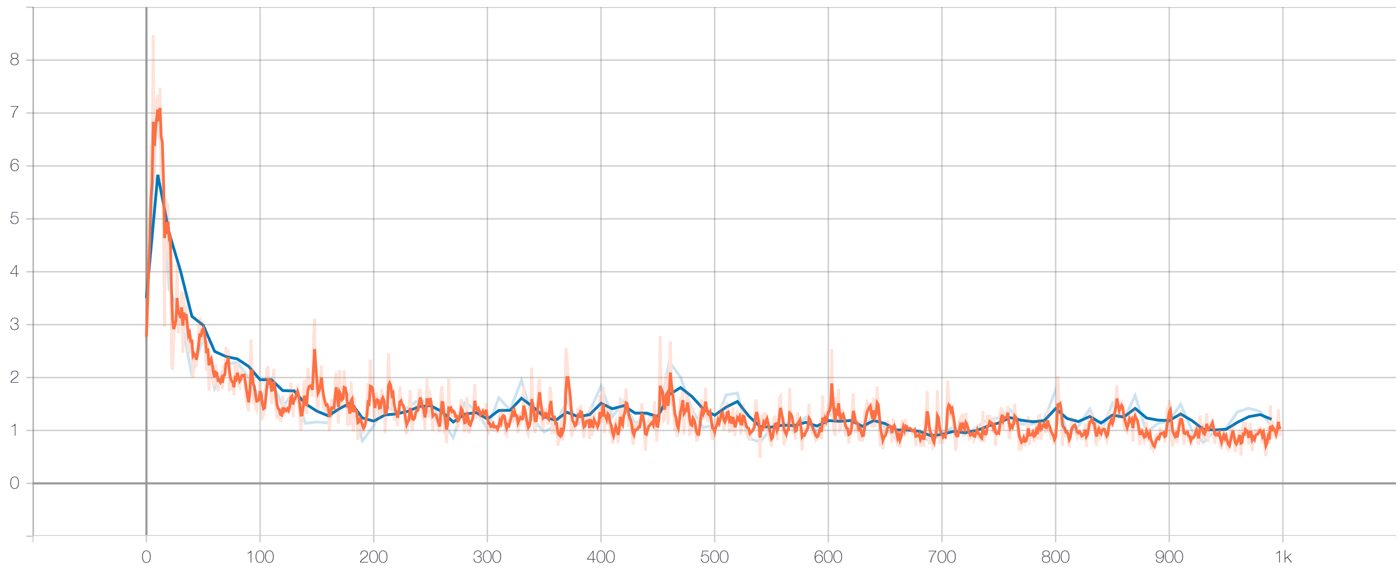
As it trains, you'll see a series of step outputs, each one showing training accuracy, validation accuracy, and the cross entropy:

- The training accuracy shows the percentage of the images used in the current training batch that were labeled with the correct class.
- The validation accuracy is the precision (percentage of correctly-labelled images) on a randomly-selected group of images from a different set.
- Cross entropy is a loss function that gives a glimpse into how well the learning process is progressing. (Lower numbers are better.)

TRAINING ACCURACY : 81%



CROSS ENTROPY: LOWER NUMBERS ARE BETTER!



Two lines are shown. The orange line shows the accuracy of the model on the training data. While the blue line shows the accuracy on the test set (which was not used for training). This is a much better measure of the true performance of the network. If the training accuracy continues to rise while the validation accuracy decreases then the model is said to be "overfitting". Overfitting is when the model begins to memorize the training set instead of understanding general patterns in the data.

As the process continues, you should see the reported accuracy improve. After all the training steps are complete, the script runs a final test accuracy evaluation on a set of images that are kept separate from the training and validation pictures. This test evaluation provides the best estimate of how the trained model will perform on the classification task.

You should see an accuracy value of between 85% and 99%, though the exact value will vary from run to run since there's randomness in the training process. (If you are only training on two classes, you should expect higher accuracy.) This number value indicates the percentage of the images in the test set that are given the correct label after the model is fully trained.

In [31]:
```
import tensorflow as tf

img = tf.placeholder(name="img", dtype=tf.float32, shape=(1, 224, 2
24, 3))
val = img + tf.constant([1., 2., 3.]) + tf.constant([1., 4., 4.])
out = tf.identity(val, name="out")
with tf.Session() as sess:
  tflite_model = tf.contrib.lite.toco_convert(sess.graph_def, [img]
, [out])
  open("fashion.tflite", "wb").write(tflite_model)
```

```
WARNING: The TensorFlow contrib module will not be included in Ten
sorFlow 2.0.
For more information, please see:
  * https://github.com/tensorflow/community/blob/master/rfcs/20180
907-contrib-sunset.md
  * https://github.com/tensorflow/addons
If you depend on functionality not listed there, please file an is
sue.

WARNING:tensorflow:From <ipython-input-31-f1ea6ee101b7>:7: toco_co
nvert (from tensorflow.lite.python.convert) is deprecated and will
be removed in a future version.
Instructions for updating:
Use `lite.TFLiteConverter` instead.
```