**Practical Assignment**

**Objective: - Image Classification with Tiny ImageNet**

Stanford prepared the Tiny ImageNet dataset for their CS231ncourse. The dataset spans 200 image classes with 500 training examples per class. The dataset also has 50 validation and 50 test examples per class. The images are down-sampled to 64x64 pixels vs. 256x256 for the original ImageNet. The full ImageNet dataset also has 1000 classes. Tiny ImageNet is large enough to be a challenging and realistic problem. But not so big as to require days of training before you see results.

**Dataset Link: -**

[**http://cs231n.stanford.edu/tiny-imagenet-200.zip**](http://cs231n.stanford.edu/tiny-imagenet-200.zip)

**Task: -** Create a Web Application using Flask. Use the end user should be able to upload an image and get results with the prediction score. Use any CNN architecture launched after 2016.

**Deployment: -** Any Free Platform(Try to look out for free options.)

**Assignment Submission: -** Only submit the hosted app link.

**Quick Tour of Repository**

### Python Files

**logistic\_regression.py**

It is good practice to build a simple baseline to start. This baseline gets reaches around 3% top-1 classification accuracy (random guessing = 0.5%).

"""

Tiny ImageNet Model: Logistic Regression

Written by Patrick Coady (pcoady@alum.mit.edu)

Logistic regression NN baseline.

"""

import tensorflow as tf

def logistic\_regression(training\_batch, config):

"""Baseline logistic regression

Args:

training\_batch: batch of images (N, 64, 64, 3)

config: training configuration object

Returns:

logits: class prediction scores

"""

img = tf.cast(training\_batch, tf.float32)

out = (img - 128.0) / 128.0

out = tf.reshape(out, (config.batch\_size, -1))

with tf.variable\_scope('logistic\_r',

initializer=tf.random\_normal\_initializer(stddev=0.1 / (56 \* 56 \* 3) \*\* 0.5),

dtype=tf.float32):

w = tf.get\_variable(shape=(56 \* 56 \* 3, 200), name='W')

b = tf.get\_variable(shape=(1, 200), name='b')

logits = tf.matmul(out, w) + b

tf.summary.histogram('logits', logits)

l2\_loss = tf.nn.l2\_loss(w)

tf.add\_to\_collection(tf.GraphKeys.REGULARIZATION\_LOSSES, l2\_loss)

return logits

**single\_layer\_nn.py**

Another simple baseline. A neural net with a single hidden layer: 1024 hidden units with ReLU activations. Reaches about 8% accuracy with minimal tuning effort

"""

Tiny ImageNet Model: Single Layer NN

Written by Patrick Coady (pcoady@alum.mit.edu)

Single-layer NN baseline.

"""

import tensorflow as tf

def single\_layer\_nn(training\_batch, config):

"""Baseline single layer NN

Args:

training\_batch: batch of images (N, 56, 56, 3)

config: training configuration object

Returns:

logits: class prediction scores

"""

img = tf.cast(training\_batch, tf.float32)

out = (img - 128.0) / 128.0

x = tf.reshape(out, (config.batch\_size, -1))

with tf.variable\_scope('hid1',

initializer=tf.random\_normal\_initializer(stddev=0.1 /

(56 \* 56 \* 3) \*\* 0.5),

dtype=tf.float32):

w1 = tf.get\_variable(shape=(56 \* 56 \* 3, 1024), name='W1')

b1 = tf.get\_variable(shape=(1, 1024), name='b')

h1 = tf.nn.relu(tf.matmul(x, w1) + b1)

tf.summary.histogram('hid1', h1)

with tf.variable\_scope('output',

initializer=tf.random\_normal\_initializer(stddev=0.1 /

1024 \*\* 0.5),

dtype=tf.float32):

w2 = tf.get\_variable(shape=(1024, 200), name='W2')

b2 = tf.get\_variable(shape=(1, 200), name='b2')

logits = tf.matmul(h1, w2) + b2

tf.summary.histogram('logits', logits)

l2\_loss = (tf.nn.l2\_loss(w1) + tf.nn.l2\_loss(w2))

tf.add\_to\_collection(tf.GraphKeys.REGULARIZATION\_LOSSES, l2\_loss)

return logits

**vgg\_16.py**

[This paper](https://arxiv.org/pdf/1409.1556.pdf) by Karen Simonyan and Andrew Zisserman introduced the VGG-16 architecture. The authors reached state-of-the-art performance using only a deep stack of 3x3xC filters and max-pooling layers. Because Tiny ImageNet has much lower resolution than the original ImageNet data, I removed the last max-pool layer and the last three convolution layers. With a little tuning, this model reaches 52% top-1 accuracy and 77% top-5 accuracy.

To keep it fair, I didn't use any pre-trained VGG-16 layers and only trained using the Tiny ImageNet examples.

"""

Tiny ImageNet Model

Written by Patrick Coady (pcoady@alum.mit.edu)

Architecture is based on VGG-16 model, but the final pool-conv-conv-conv-pool

layers were discarded. The input to the network is a 56x56 RGB crop (versus

224x224 crop for the original VGG-16 model). L2 regularization is applied to

all layer weights. And dropout is applied to the first 2 fully-connected

layers.

1. conv-conv-maxpool

2. conv-conv-maxpool

3. conv-conv-maxpool

4. conv-conv-conv-maxpool

4. fc-4096 (ReLU)

5. fc-2048 (ReLU)

6. fc-200

7. softmax

"""

import tensorflow as tf

import numpy as np

def conv\_2d(inputs, filters, kernel\_size, name=None):

"""3x3 conv layer: ReLU + (1, 1) stride + He initialization"""

# He initialization = normal dist with stdev = sqrt(2.0/fan-in)

stddev = np.sqrt(2 / (np.prod(kernel\_size) \* int(inputs.shape[3])))

out = tf.layers.conv2d(inputs, filters=filters, kernel\_size=kernel\_size,

padding='same', activation=tf.nn.relu,

kernel\_initializer=tf.random\_normal\_initializer(stddev=stddev),

kernel\_regularizer=tf.contrib.layers.l2\_regularizer(1.0),

name=name)

tf.summary.histogram('act' + name, out)

return out

def dense\_relu(inputs, units, name=None):

"""3x3 conv layer: ReLU + He initialization"""

# He initialization: normal dist with stdev = sqrt(2.0/fan-in)

stddev = np.sqrt(2 / int(inputs.shape[1]))

out = tf.layers.dense(inputs, units, activation=tf.nn.relu,

kernel\_initializer=tf.random\_normal\_initializer(stddev=stddev),

kernel\_regularizer=tf.contrib.layers.l2\_regularizer(1.0),

name=name)

tf.summary.histogram('act' + name, out)

return out

def dense(inputs, units, name=None):

"""3x3 conv layer: ReLU + He initialization"""

# He initialization: normal dist with stdev = sqrt(2.0/fan-in)

stddev = np.sqrt(2 / int(inputs.shape[1]))

out = tf.layers.dense(inputs, units,

kernel\_initializer=tf.random\_normal\_initializer(stddev=stddev),

kernel\_regularizer=tf.contrib.layers.l2\_regularizer(1.0),

name=name)

tf.summary.histogram('act' + name, out)

return out

def vgg\_16(training\_batch, config):

"""VGG-like conv-net

Args:

training\_batch: batch of images (N, 56, 56, 3)

config: training configuration object

Returns:

class prediction scores

"""

img = tf.cast(training\_batch, tf.float32)

out = (img - 128.0) / 128.0

tf.summary.histogram('img', training\_batch)

# (N, 56, 56, 3)

out = conv\_2d(out, 64, (3, 3), 'conv1\_1')

out = conv\_2d(out, 64, (3, 3), 'conv1\_2')

out = tf.layers.max\_pooling2d(out, (2, 2), (2, 2), name='pool1')

# (N, 28, 28, 64)

out = conv\_2d(out, 128, (3, 3), 'conv2\_1')

out = conv\_2d(out, 128, (3, 3), 'conv2\_2')

out = tf.layers.max\_pooling2d(out, (2, 2), (2, 2), name='pool2')

# (N, 14, 14, 128)

out = conv\_2d(out, 256, (3, 3), 'conv3\_1')

out = conv\_2d(out, 256, (3, 3), 'conv3\_2')

out = conv\_2d(out, 256, (3, 3), 'conv3\_3')

out = tf.layers.max\_pooling2d(out, (2, 2), (2, 2), name='pool3')

# (N, 7, 7, 256)

out = conv\_2d(out, 512, (3, 3), 'conv4\_1')

out = conv\_2d(out, 512, (3, 3), 'conv4\_2')

out = conv\_2d(out, 512, (3, 3), 'conv4\_3')

# fc1: flatten -> fully connected layer

# (N, 7, 7, 512) -> (N, 25088) -> (N, 4096)

out = tf.contrib.layers.flatten(out)

out = dense\_relu(out, 4096, 'fc1')

out = tf.nn.dropout(out, config.dropout\_keep\_prob)

# fc2

# (N, 4096) -> (N, 2048)

out = dense\_relu(out, 2048, 'fc2')

out = tf.nn.dropout(out, config.dropout\_keep\_prob)

# softmax

# (N, 2048) -> (N, 200)

logits = dense(out, 200, 'fc3')

return logits

**input\_pipe.py**

* Load JPEGs (using Tiny ImageNet directory structure)
* Load labels and build label integer-to-text dictionary
* QueueRunner to feed GPU
  + including data augmentation (i.e. various image distortions)

"""

Tiny ImageNet: Input Pipeline

Written by Patrick Coady (pcoady@alum.mit.edu)

Reads in jpegs, distorts images (flips, translations, hue and

saturation) and builds QueueRunners to keep the GPU well-fed. Uses

specific directory and file naming structure from data download

link below.

Also builds dictionary between label integer and human-readable

class names.

Get data here:

https://tiny-imagenet.herokuapp.com/

"""

import glob

import re

import tensorflow as tf

import random

import numpy as np

def load\_filenames\_labels(mode):

"""Gets filenames and labels

Args:

mode: 'train' or 'val'

(Directory structure and file naming different for

train and val datasets)

Returns:

list of tuples: (jpeg filename with path, label)

"""

label\_dict, class\_description = build\_label\_dicts()

filenames\_labels = []

if mode == 'train':

filenames = glob.glob('../tiny-imagenet-200/train/\*/images/\*.JPEG')

for filename in filenames:

match = re.search(r'n\d+', filename)

label = str(label\_dict[match.group()])

filenames\_labels.append((filename, label))

elif mode == 'val':

with open('../tiny-imagenet-200/val/val\_annotations.txt', 'r') as f:

for line in f.readlines():

split\_line = line.split('\t')

filename = '../tiny-imagenet-200/val/images/' + split\_line[0]

label = str(label\_dict[split\_line[1]])

filenames\_labels.append((filename, label))

return filenames\_labels

def build\_label\_dicts():

"""Build look-up dictionaries for class label, and class description

Class labels are 0 to 199 in the same order as

tiny-imagenet-200/wnids.txt. Class text descriptions are from

tiny-imagenet-200/words.txt

Returns:

tuple of dicts

label\_dict:

keys = synset (e.g. "n01944390")

values = class integer {0 .. 199}

class\_desc:

keys = class integer {0 .. 199}

values = text description from words.txt

"""

label\_dict, class\_description = {}, {}

with open('../tiny-imagenet-200/wnids.txt', 'r') as f:

for i, line in enumerate(f.readlines()):

synset = line[:-1] # remove \n

label\_dict[synset] = i

with open('../tiny-imagenet-200/words.txt', 'r') as f:

for i, line in enumerate(f.readlines()):

synset, desc = line.split('\t')

desc = desc[:-1] # remove \n

if synset in label\_dict:

class\_description[label\_dict[synset]] = desc

return label\_dict, class\_description

def read\_image(filename\_q, mode):

"""Load next jpeg file from filename / label queue

Randomly applies distortions if mode == 'train' (including a

random crop to [56, 56, 3]). Standardizes all images.

Args:

filename\_q: Queue with 2 columns: filename string and label string.

filename string is relative path to jpeg file. label string is text-

formatted integer between '0' and '199'

mode: 'train' or 'val'

Returns:

[img, label]:

img = tf.uint8 tensor [height, width, channels] (see tf.image.decode.jpeg())

label = tf.unit8 target class label: {0 .. 199}

"""

item = filename\_q.dequeue()

filename = item[0]

label = item[1]

file = tf.read\_file(filename)

img = tf.image.decode\_jpeg(file, channels=3)

# image distortions: left/right, random hue, random color saturation

if mode == 'train':

img = tf.random\_crop(img, np.array([56, 56, 3]))

img = tf.image.random\_flip\_left\_right(img)

# val accuracy improved without random hue

# img = tf.image.random\_hue(img, 0.05)

img = tf.image.random\_saturation(img, 0.5, 2.0)

else:

img = tf.image.crop\_to\_bounding\_box(img, 4, 4, 56, 56)

label = tf.string\_to\_number(label, tf.int32)

label = tf.cast(label, tf.uint8)

return [img, label]

def batch\_q(mode, config):

"""Return batch of images using filename Queue

Args:

mode: 'train' or 'val'

config: training configuration object

Returns:

imgs: tf.uint8 tensor [batch\_size, height, width, channels]

labels: tf.uint8 tensor [batch\_size,]

"""

filenames\_labels = load\_filenames\_labels(mode)

random.shuffle(filenames\_labels)

filename\_q = tf.train.input\_producer(filenames\_labels,

num\_epochs=config.num\_epochs,

shuffle=True)

# 2 read\_image threads to keep batch\_join queue full:

return tf.train.batch\_join([read\_image(filename\_q, mode) for i in range(2)],

config.batch\_size, shapes=[(56, 56, 3), ()],

capacity=2048)

**train.py**

Trains models and monitors validation accuracy. The training loop has learning rate control and terminates training when progress stops. I take full advantage of TensorBoard by saving histograms of all weights, activations, and also learning curves.

Training is built to run fast on GPU by running the data pipeline on the CPU and model training on the GPU. It is straightforward to train a different model by changing 'model\_name' in TrainConfig class.

"""

Tiny ImageNet: Training

Written by Patrick Coady (pcoady@alum.mit.edu)

Train model specified in class static var: TrainConfig.model\_name.

Key Features:

1. Saves key operations and variables for viewing in TensorBoard

2. Training control:

a) Learning rate decreases based on validation accuracy trend

b) Training terminates based on validation accuracy trend

4. Basic user interface to:

a) Name training runs / directories

b) Saves copy of .py files in training result directory

c) Resume training from checkpoint

"""

from vgg\_16 import \*

# from logistic\_regression import \*

# from single\_layer\_nn import \*

from metrics import \*

from losses import \*

from input\_pipe import \*

from datetime import datetime

import numpy as np

import os

import shutil

import glob

class TrainConfig(object):

"""Training configuration"""

batch\_size = 64

num\_epochs = 50

summary\_interval = 250

eval\_interval = 2000 # must be integer multiple of summary\_interval

lr = 0.01 # learning rate

reg = 5e-4 # regularization

momentum = 0.9

dropout\_keep\_prob = 0.5

model\_name = 'vgg\_16' # choose model

model = staticmethod(globals()[model\_name]) # gets model by name

class TrainControl(object):

"""Basic training control

Decreases learning rate (lr), terminates training after 3 lr decreases

Track validation accuracy, decrease lr by 1/5th when:

1. validation accuracy worsens

2. less than 0.2% absolute improvement last 3 iterations

"""

def \_\_init\_\_(self, lr):

self.val\_accs = []

self.lr = lr

self.num\_lr\_updates = 0

self.lr\_factor = 1/5

def add\_val\_acc(self, loss):

self.val\_accs.append(loss)

def update\_lr(self, sess):

if len(self.val\_accs) < 3:

return

decrease = False

# decrease LR if validation acc worsens

if self.val\_accs[-1] < max(self.val\_accs):

decrease = True

avg\_2 = (self.val\_accs[-2] + self.val\_accs[-3]) / 2

# decrease LR if validation accuracy doesn't improve by 0.2% (absolute)

if abs(self.val\_accs[-1] - avg\_2) < 0.002:

decrease = True

if decrease:

old\_lr = sess.run(self.lr)

self.lr.load(old\_lr \* self.lr\_factor)

self.num\_lr\_updates += 1

print('\*\*\* New learning rate: {}'.format(old\_lr \* self.lr\_factor))

def done(self):

if self.num\_lr\_updates > 3: # terminate training after 3 lr decreases

return True

else:

return False

def optimizer(loss, config):

"""Add training operation, global\_step and learning rate variable to Graph

Args:

loss: model loss tensor

config: training configuration object

Returns:

(train\_op, global\_step, lr)

"""

lr = tf.Variable(config.lr, trainable=False, dtype=tf.float32)

global\_step = tf.Variable(0, trainable=False, name='global\_step')

optim = tf.train.MomentumOptimizer(lr, config.momentum,

use\_nesterov=True)

train\_op = optim.minimize(loss, global\_step=global\_step)

return train\_op, global\_step, lr

def get\_logdir():

"""Return unique logdir based on datetime"""

now = datetime.utcnow().strftime("%m%d%H%M%S")

logdir = "run-{}/".format(now)

return logdir

def model(mode, config):

"""Pull it all together: input queue, inference model and loss functions

Args:

mode: 'train' or 'val'

config: model configuration object

Returns:

loss and accuracy tensors

"""

# preprocess images on cpu - send to gpu as uint8 for speed

with tf.device('/cpu:0'):

imgs, labels = batch\_q(mode, config)

logits = config.model(imgs, config)

softmax\_ce\_loss(logits, labels)

acc = accuracy(logits, labels)

total\_loss = tf.add\_n(tf.get\_collection(tf.GraphKeys.LOSSES), name='total\_loss')

total\_loss += tf.add\_n(tf.get\_collection(tf.GraphKeys.REGULARIZATION\_LOSSES),

name='total\_loss') \* config.reg

for l2 in tf.get\_collection(tf.GraphKeys.REGULARIZATION\_LOSSES):

# add l2 loss histograms to TensorBoard and cleanup var names

name = 'l2\_loss\_' + l2.name.split('/')[0]

tf.summary.histogram(name, l2)

return total\_loss, acc

def evaluate(ckpt):

"""Load checkpoint and run on validation set"""

config = TrainConfig()

config.dropout\_keep\_prob = 1.0 # disable dropout for validation

config.num\_epochs = 1

accs, losses = [], []

with tf.Graph().as\_default():

loss, acc = model('val', config)

saver = tf.train.Saver()

init = tf.group(tf.global\_variables\_initializer(),

tf.local\_variables\_initializer())

with tf.Session() as sess:

init.run()

saver.restore(sess, ckpt)

coord = tf.train.Coordinator()

threads = tf.train.start\_queue\_runners(sess=sess, coord=coord)

try:

while not coord.should\_stop():

step\_loss, step\_acc = sess.run([loss, acc])

accs.append(step\_acc)

losses.append(step\_loss)

except tf.errors.OutOfRangeError as e:

coord.request\_stop(e)

finally:

coord.request\_stop()

coord.join(threads)

mean\_loss, mean\_acc = np.mean(losses), np.mean(accs)

print('Validation. Loss: {:.3f}, Accuracy: {:.4f}'.

format(mean\_loss, mean\_acc))

return mean\_loss, mean\_acc

def options(config):

"""Get user input on training options"""

q = input('Enter a short configuration name [default = "default"]: ')

if len(q) == 0:

q = 'default'

config.config\_name = q

# tensorboard and checkpoint log directory names

ckpt\_path = 'checkpoints/' + config.model\_name + '/' + config.config\_name

tflog\_path = ('tf\_logs/' + config.model\_name + '/' +

config.config\_name + '/' + get\_logdir())

checkpoint = None

# TODO: spaghetti mess, clean up:

if not os.path.isdir(ckpt\_path):

os.makedirs(ckpt\_path)

filenames = glob.glob('\*.py')

for filename in filenames:

shutil.copy(filename, ckpt\_path)

return False, ckpt\_path, tflog\_path, checkpoint

else:

filenames = glob.glob('\*.py')

for filename in filenames:

shutil.copy(filename, ckpt\_path)

while True:

q1 = input('Continue previous training? [Y/n]: ')

if len(q1) == 0 or q1 == 'n' or q1 == 'Y':

break

if q1 == 'n':

return False, ckpt\_path, tflog\_path, checkpoint

else:

q2 = input('Enter checkpoint name [defaults to most recent]: ')

if len(q2) == 0:

checkpoint = tf.train.latest\_checkpoint(ckpt\_path)

else:

checkpoint = ckpt\_path + '/' + q2

return True, ckpt\_path, tflog\_path, checkpoint

def train():

"""Build Graph, launch session and train."""

config = TrainConfig()

continue\_train, ckpt\_path, tflog\_path, checkpoint = options(config)

g = tf.Graph()

with g.as\_default():

loss, acc = model('train', config)

train\_op, g\_step, lr = optimizer(loss, config)

controller = TrainControl(lr)

# put variables in graph to hold validation acc and loss for TensorBoard viewing

val\_acc = tf.Variable(0.0, trainable=False)

val\_loss = tf.Variable(0.0, trainable=False)

tf.summary.scalar('val\_loss', val\_loss)

tf.summary.scalar('val\_accuracy', val\_acc)

init = tf.group(tf.global\_variables\_initializer(),

tf.local\_variables\_initializer())

# histograms of all variables to TensorBoard

[tf.summary.histogram(v.name.replace(':', '\_'), v)

for v in tf.trainable\_variables()]

# next line only needed for batch normalization (updates beta and gamma)

extra\_update\_ops = tf.get\_collection(tf.GraphKeys.UPDATE\_OPS)

summ = tf.summary.merge\_all()

saver = tf.train.Saver(max\_to\_keep=1)

writer = tf.summary.FileWriter(tflog\_path, g)

with tf.Session() as sess:

init.run()

if continue\_train:

saver.restore(sess, checkpoint)

coord = tf.train.Coordinator()

threads = tf.train.start\_queue\_runners(sess=sess, coord=coord)

try:

losses, accs = [], [] # hold running averages for test loss/acc

while not coord.should\_stop():

step\_loss, \_, step, step\_acc, \_\_ = sess.run([loss, train\_op,

g\_step, acc, extra\_update\_ops])

losses.append(step\_loss)

accs.append(step\_acc)

if step % config.eval\_interval == 0:

ckpt = saver.save(sess, ckpt\_path + '/model', step)

mean\_loss, mean\_acc = evaluate(ckpt)

val\_acc.load(mean\_acc)

val\_loss.load(mean\_loss)

controller.add\_val\_acc(mean\_acc)

controller.update\_lr(sess)

if controller.done():

break

if step % config.summary\_interval == 0:

writer.add\_summary(sess.run(summ), step)

print('Iteration: {}, Loss: {:.3f}, Accuracy: {:.4f}'.

format(step, np.mean(losses), np.mean(accs)))

losses, accs = [], []

except tf.errors.OutOfRangeError as e:

coord.request\_stop(e)

finally:

coord.request\_stop()

coord.join(threads)

if \_\_name\_\_ == "\_\_main\_\_":

train()

**losses.py**

Contains three loss functions:

1. Cross-entropy loss
2. Smoothed cross-entropy loss (add small, non-zero, probability to all classes)
3. SVM (works, but never got great performance)

"""

Tiny ImageNet: Loss Functions

"""

import tensorflow as tf

def softmax\_ce\_loss(logits, labels):

"""Softmax + cross-entropy loss

Args:

logits: logits (N, C) C = number of classes

labels: tf.uint8 labels {0 .. 199}

Returns:

losses: mean cross entropy loss

"""

labels = tf.cast(labels, tf.int32)

ce\_loss = tf.losses.sparse\_softmax\_cross\_entropy(labels,

logits,

weights=1.0)

tf.summary.scalar('loss', ce\_loss)

def softmax\_smooth\_ce\_loss(logits, labels):

"""Softmax + cross-entropy loss with label smoothing

Args:

logits: logits (N, C) C = number of classes

labels: tf.uint8 labels {0 .. 199}

Returns:

losses: mean cross entropy loss

"""

labels = tf.cast(labels, tf.int32)

ohe = tf.one\_hot(labels, 200, dtype=tf.int32)

ce\_loss = tf.losses.softmax\_cross\_entropy(ohe,

logits,

label\_smoothing=0.1)

tf.summary.scalar('loss', ce\_loss)

def svm\_loss(logits, labels):

"""SVM loss: one-vs-all

Args:

logits: logits (N, C) C = number of classes

labels: tf.uint8 labels {0 .. 199}

Returns:

losses: mean cross entropy loss

"""

c = 1.0

labels = tf.cast(labels, tf.int32)

ohe = tf.one\_hot(labels, 200, dtype=tf.float32, on\_value=-200.0, off\_value=1.0)

tf.summary.histogram('svm\_mat\_b4\_shift', ohe)

svm\_mat = 1.0 + ohe \* logits

tf.summary.histogram('logits', logits)

tf.summary.histogram('svm\_mat\_b4\_clip', svm\_mat)

svm\_mat = tf.maximum(svm\_mat, 0.0)

tf.summary.histogram('svm\_mat', svm\_mat)

svm\_l = c \* tf.reduce\_mean(svm\_mat)

tf.add\_to\_collection(tf.GraphKeys.LOSSES, svm\_l)

tf.summary.scalar('loss', svm\_l)

**metrics.py**

Measures % accuracy.

"""

Tiny ImageNet: Performance Metrics

"""

import tensorflow as tf

def accuracy(logits, labels):

"""Return batch accuracy

Args:

logits: logits (N, C) C = number of classes

labels: tf.uint8 labels {0 .. 199}

Returns:

classification accuracy

"""

labels = tf.cast(labels, tf.int64)

pred = tf.argmax(logits, axis=1)

acc = tf.contrib.metrics.accuracy(pred, labels)

tf.summary.scalar('acc', acc)

return acc

### Notebooks

**predict\_and\_saliency.ipynb**

This short notebook randomly selects ten images from the validation set and displays the top-5 predictions vs. the "gold" label. The notebook also displays saliency maps next to each image so you can see where the model is "looking" as it makes decisions.

This notebook predicts the top-5 most likely labels for a random selection of images. The human labels, along with the top-5 model predictions, are displayed below each picture. Also, a "saliency map" is displayed next to each image. The saliency map highlights areas that were most important in making the top prediction.

It is worth noting that a human labeler would have a difficult time correctly identifying many of these down-sampled images. The human labelers had the advantage of 4x higher resolution images to make their predictions (256x256 vs. 64x64 images). So, the model performance is quite impressive considering the low-resolution images.

The last cell may be re-run multiple times to explore a new selection of pictures.

Python Notebook by Patrick Coady: [Learning Artificial Intelligence](https://learningai.io/)

**from** train **import** **\***

**import** tensorflow **as** tf

**import** glob

**import** matplotlib.pyplot **as** plt

**import** random

**import** scipy.ndimage

**%matplotlib** inline

plt**.**rcParams['figure.figsize'] **=** (10, 6)

**class** TrainConfig(object):

"""Training configuration"""

dropout\_keep\_prob **=** 1.0

model\_name **=** 'vgg\_16' *# choose model*

model **=** staticmethod(globals()[model\_name])

config\_name **=** 'no\_hue' *# choose training run*

**def** predict(imgs, config):

"""Load most recent checkpoint, make predictions, compute saliency maps"""

g **=** tf**.**Graph()

**with** g**.**as\_default():

imgs\_ph **=** tf**.**placeholder(dtype**=**tf**.**uint8, shape**=**(**None**, 56, 56, 3))

logits **=** config**.**model(imgs\_ph, config)

top\_pred **=** tf**.**reduce\_max(logits, axis**=**1)

top\_5 **=** tf**.**nn**.**top\_k(logits, k**=**5, sorted**=True**)

*# can't calculate gradient to integer, get float32 version of image:*

float\_img **=** g**.**get\_tensor\_by\_name('Cast:0')

*# calc gradient of top predicted class to image:*

grads **=** tf**.**gradients(top\_pred, float\_img)

saver **=** tf**.**train**.**Saver()

**with** tf**.**Session() **as** sess:

path **=** 'checkpoints/' **+** config**.**model\_name **+** '/' **+** config**.**config\_name

saver**.**restore(sess, tf**.**train**.**latest\_checkpoint(path))

feed\_dict **=** {imgs\_ph: imgs}

top\_5\_np, grads\_np **=** sess**.**run([top\_5, grads], feed\_dict**=**feed\_dict)

**return** top\_5\_np, grads\_np

*# get label integer -> text description dictionary*

label\_dict, class\_description **=** build\_label\_dicts()

**for** i **in** range(len(class\_description)):

class\_description[i] **=** class\_description[i]**.**split(',')[0]

N **=** 10 *# number of validation examples to view*

filenames\_labels **=** load\_filenames\_labels('val')

pick\_N **=** random**.**sample(filenames\_labels, N)

imgs **=** np**.**zeros((N, 64, 64, 3), dtype**=**np**.**uint8)

labels **=** []

**for** i, filename\_label **in** enumerate(pick\_N):

imgs[i, :, :, :] **=** scipy**.**ndimage**.**imread(filename\_label[0], mode**=**'RGB')

labels**.**append(class\_description[int(filename\_label[1])])

imgs **=** imgs[:, 4:60, 4:60, :] *# take center crop of images*

config **=** TrainConfig()

top\_5, sal\_imgs **=** predict(imgs, config)

top\_5 **=** top\_5[1] *# 2nd element of list are class predictions*

sal\_imgs **=** sal\_imgs[0] *# 1st element of list are saliency maps*

*# root-sum-square RGB channels of generated saliency heat map*

sal\_imgs **=** np**.**sqrt(np**.**sum(np**.**square(sal\_imgs), axis**=**3))

**for** idx, filename **in** enumerate(pick\_N):

plt**.**subplot(121)

plt**.**imshow(imgs[idx, :, :, :], interpolation**=**'none')

*# next 5 lines get rid of all white space when saving .png*

plt**.**gca()**.**set\_axis\_off()

plt**.**subplots\_adjust(top **=** 1, bottom **=** 0, right **=** 1, left **=** 0,

hspace **=** 0, wspace **=** 2)

plt**.**margins(0,0)

plt**.**gca()**.**xaxis**.**set\_major\_locator(plt**.**NullLocator())

plt**.**gca()**.**yaxis**.**set\_major\_locator(plt**.**NullLocator())

plt**.**subplot(122)

plt**.**imshow(sal\_imgs[idx, :, :], interpolation**=**'none')

*# next 5 lines get rid of all white space when saving .png*

plt**.**gca()**.**set\_axis\_off()

plt**.**subplots\_adjust(top **=** 1, bottom **=** 0, right **=** 1, left **=** 0,

hspace **=** 0, wspace **=** 0.1)

plt**.**margins(0,0)

plt**.**gca()**.**xaxis**.**set\_major\_locator(plt**.**NullLocator())

plt**.**gca()**.**yaxis**.**set\_major\_locator(plt**.**NullLocator())

plt**.**savefig('plots/pred\_sal'**+**str(idx)**+**'.png', bbox\_inches**=**'tight',

pad\_inches**=**0.0, dpi**=**64)

plt**.**show()

print('Actual label: ' **+** labels[idx])

print('Top 5 predictions:')

preds **=** map(**lambda** x: class\_description[x], top\_5[idx])

print([x **for** x **in** preds])

print('\n')

**kernel\_viz.ipynb**

Visualize input kernels (aka filters) of first two conv layers. The receptive field is only 7x7 after two 3x3 layers, but the results are still interesting.

# Tiny ImageNet: Visualize Layers

Simple routine to visualize the first stack of 3x3 kernels (conv1\_1 + conv2\_2) before max-pooling. We see the typical patterns: horizontal, vertical and diagonal stripes, and various color spots. It is interesting (and reassuring) to see that two slightly different models learn similar filter kernels.

Here is the basic procedure:

1. Load a trained model
2. Apply an even gray image to the input (i.e. all 128s)
3. Take the gradient of a conv2\_2 output (choosing a center "pixel") vs. input image b. Repeat for each of the 64 filters
4. Crop the image gradient: it is mostly zeros except near the receptive field
5. Scale the gradients to fill 0-255 range in RGB
6. Arrange the 64 gradient crops into a single 8x8 image array and plot

Python Notebook by Patrick Coady: [Learning Artificial Intelligence](https://learningai.io/)

**from** train **import** **\***

**import** tensorflow **as** tf

**import** matplotlib.pyplot **as** plt

**import** scipy.ndimage

**%matplotlib** inline

plt**.**rcParams['figure.figsize'] **=** (10, 6)

**class** TrainConfig(object):

"""Training configuration"""

dropout\_keep\_prob **=** 1.0

model\_name **=** 'vgg\_16' *# choose model*

model **=** staticmethod(globals()[model\_name])

config\_name **=** 'no\_hue' *# choose training run*

**def** img\_grad(config):

"""find gradient from pixel to img"""

**with** tf**.**Graph()**.**as\_default():

g **=** tf**.**get\_default\_graph()

img **=** tf**.**Variable(np**.**zeros((1, 56, 56, 3), dtype**=**np**.**uint8) **+** 128,

trainable**=False**,

dtype**=**tf**.**uint8,

collections**=**[tf**.**GraphKeys**.**LOCAL\_VARIABLES])

logits **=** config**.**model(img, config)

pixels **=** g**.**get\_tensor\_by\_name('conv1\_2/BiasAdd:0')

float\_img **=** g**.**get\_tensor\_by\_name('Cast:0')

grads **=** []

**for** i **in** range(64):

grads**.**append(tf**.**gradients(pixels[0, 28, 28, i], float\_img))

saver **=** tf**.**train**.**Saver()

init **=** tf**.**group(tf**.**global\_variables\_initializer(),

tf**.**local\_variables\_initializer())

**with** tf**.**Session() **as** sess:

init**.**run()

path **=** 'checkpoints/' **+** config**.**model\_name **+** '/' **+** config**.**config\_name

saver**.**restore(sess, tf**.**train**.**latest\_checkpoint(path))

results **=** sess**.**run(grads)

**return** results

config **=** TrainConfig()

result **=** img\_grad(config)

INFO:tensorflow:Restoring parameters from checkpoints/vgg\_16/no\_hue/model-44000

composite **=** np**.**zeros((7**\***8, 7**\***8, 3), dtype**=**np**.**uint8)

**for** i **in** range(8):

**for** j **in** range(8):

idx **=** i **\*** 8 **+** j

crop **=** np**.**squeeze(result[idx])[25:32, 25:32, :]

crop **=** crop **/** np**.**max([**-**np**.**min(crop), np**.**max(crop)])

crop **=** (crop **\*** 127 **+** 128)**.**astype(np**.**uint8)

composite[(i**\***7):(i**\***7**+**7), (j**\***7):(j**\***7**+**7), :] **=** crop

plt**.**imshow(composite, interpolation**=**'none')

*# next 5 lines get rid of all white space when saving .png*

plt**.**gca()**.**set\_axis\_off()

plt**.**subplots\_adjust(top **=** 1, bottom **=** 0, right **=** 1, left **=** 0,

hspace **=** 0, wspace **=** 0)

plt**.**margins(0,0)

plt**.**gca()**.**xaxis**.**set\_major\_locator(plt**.**NullLocator())

plt**.**gca()**.**yaxis**.**set\_major\_locator(plt**.**NullLocator())

plt**.**savefig('plots/kernel\_viz\_1.png', bbox\_inches**=**'tight',

pad\_inches**=**0.0, dpi**=**64)

plt**.**show()

**kernel\_viz\_conv4.ipynb**

Same as kernel\_viz.ipynb except visualizes after 4th conv layer.

# Tiny ImageNet: Visualize Layers

Similar to the kernel\_viz.ipynb notebook, but attempts visualize after the 4th convolution (before the 2nd max-pool operation). Again, many common filter patterns are seen, but in general the results are noiser. The colors seem dominated by blues and oranges, which I don't have a good explanation for.

These results exhibit [Checkerboard Artifacts](http://distill.pub/2016/deconv-checkerboard/).

Python Notebook by Patrick Coady: [Learning Artificial Intelligence](https://learningai.io/)

**from** train **import** **\***

**import** tensorflow **as** tf

**import** matplotlib.pyplot **as** plt

**import** scipy.ndimage

**%matplotlib** inline

plt**.**rcParams['figure.figsize'] **=** (10, 6)

**class** TrainConfig(object):

"""Training configuration"""

dropout\_keep\_prob **=** 1.0

model\_name **=** 'vgg\_16' *# choose model*

model **=** staticmethod(globals()[model\_name])

config\_name **=** 'no\_hue' *# choose training run*

**def** img\_grad(config):

"""find gradient from pixel to img"""

**with** tf**.**Graph()**.**as\_default():

g **=** tf**.**get\_default\_graph()

img **=** tf**.**Variable(np**.**zeros((1, 56, 56, 3), dtype**=**np**.**uint8) **+** 128,

trainable**=False**,

dtype**=**tf**.**uint8,

collections**=**[tf**.**GraphKeys**.**LOCAL\_VARIABLES])

logits **=** config**.**model(img, config)

pixels **=** g**.**get\_tensor\_by\_name('conv2\_2/BiasAdd:0')

float\_img **=** g**.**get\_tensor\_by\_name('Cast:0')

grads **=** []

**for** i **in** range(64):

grads**.**append(tf**.**gradients(pixels[0, 14, 14, i], float\_img))

saver **=** tf**.**train**.**Saver()

init **=** tf**.**group(tf**.**global\_variables\_initializer(),

tf**.**local\_variables\_initializer())

**with** tf**.**Session() **as** sess:

init**.**run()

path **=** 'checkpoints/' **+** config**.**model\_name **+** '/' **+** config**.**config\_name

saver**.**restore(sess, tf**.**train**.**latest\_checkpoint(path))

results **=** sess**.**run(grads)

**return** results

config **=** TrainConfig()

result **=** img\_grad(config)

composite **=** np**.**zeros((13**\***8, 13**\***8, 3), dtype**=**np**.**uint8)

**for** i **in** range(8):

**for** j **in** range(8):

idx **=** i **\*** 8 **+** j

crop **=** np**.**squeeze(result[idx])[22:35, 22:35, :]

crop **=** crop **/** np**.**max([**-**np**.**min(crop), np**.**max(crop)])

crop **=** (crop **\*** 127 **+** 128)**.**astype(np**.**uint8)

composite[(i**\***13):(i**\***13**+**13), (j**\***13):(j**\***13**+**13), :] **=** crop

plt**.**imshow(composite, interpolation**=**'none')

*# next 5 lines get rid of all white space when saving .png*

plt**.**gca()**.**set\_axis\_off()

plt**.**subplots\_adjust(top **=** 1, bottom **=** 0, right **=** 1, left **=** 0,

hspace **=** 0, wspace **=** 0)

plt**.**margins(0,0)

plt**.**gca()**.**xaxis**.**set\_major\_locator(plt**.**NullLocator())

plt**.**gca()**.**yaxis**.**set\_major\_locator(plt**.**NullLocator())

plt**.**savefig('plots/kernel\_viz\_2.png', bbox\_inches**=**'tight',

pad\_inches**=**0.0, dpi**=**64)

plt**.**show()

**val\_accuracy.ipynb**

This notebook loads a model and calculates the validation set accuracy. It also computes the accuracy when predictions from 5 different crops x 2 flips are averaged: about a 3% accuracy improvement. This notebook runs slowly because it loops through the validation images one-by-one: It was not worth the extra effort to write efficiently. Premature optimization is the root of all evil. -Donald KnuthTiny ImageNet: Validation Accuracy

Calculate top-1 and top-5 validation accuracy. 2 methods:

1. Feed validation image into model with no distortions
2. Feed validation image in with 10 permuations of l/r flip and 5 crops

The validation set was used because labels weren't available for the test set.

Note: This runs very slowly. The images are loaded one-by-one in a loop and fed to the model using a feed\_dict. During training, a much more efficient pipeline is used: distortions and QueueRunner on CPU and model on the GPU.

Python Notebook by Patrick Coady: [*Learning Artificial Intelligence*](https://learningai.io/)

**from** train **import** **\***

**import** tensorflow **as** tf

**import** glob

**import** matplotlib.pyplot **as** plt

**import** random

**import** scipy.ndimage

**import** scipy.misc

**%matplotlib** inline

plt**.**rcParams['figure.figsize'] **=** (10, 6)

**class** TrainConfig(object):

"""Training configuration"""

dropout\_keep\_prob **=** 1.0 *# disable dropout for inference*

model\_name **=** 'vgg\_16' *# choose model*

model **=** staticmethod(globals()[model\_name])

config\_name **=** 'no\_hue' *# choose training run*

**def** accuracy(config):

"""Load most recent checkpoint and make prediction"""

**with** tf**.**Graph()**.**as\_default():

img\_ph **=** tf**.**placeholder(dtype**=**tf**.**uint8, shape**=**(**None**, 64, 64, 3))

img **=** tf**.**image**.**crop\_to\_bounding\_box(img\_ph, 4, 4, 56, 56)

logits **=** config**.**model(img, config)

top\_5 **=** tf**.**nn**.**top\_k(logits, k**=**5, sorted**=True**)

saver **=** tf**.**train**.**Saver()

**with** tf**.**Session() **as** sess:

filenames\_labels **=** load\_filenames\_labels('val')

path **=** 'checkpoints/' **+** config**.**model\_name **+** '/' **+** config**.**config\_name

saver**.**restore(sess, tf**.**train**.**latest\_checkpoint(path))

count, correct1, correct5 **=** (0, 0, 0)

**for** filename, label **in** filenames\_labels:

count **+=** 1

np\_img **=** scipy**.**ndimage**.**imread(filename, mode**=**'RGB')

np\_img **=** np\_img[np**.**newaxis, :, :, :]

feed\_dict **=** {img\_ph: np\_img}

top\_vals, top\_idx **=** sess**.**run(top\_5, feed\_dict**=**feed\_dict)

**if** top\_idx[0][0] **==** int(label):

correct1 **+=** 1

**if** int(label) **in** top\_idx[0]:

correct5 **+=** 1

**return** correct1 **/** count, correct5 **/** count

**def** accuracy\_10\_crop(config):

"""Load most recent checkpoint and make prediction"""

**with** tf**.**Graph()**.**as\_default():

img\_ph **=** tf**.**placeholder(dtype**=**tf**.**uint8, shape**=**(64, 64, 3))

crops **=** [(0,0), (0,8), (8,0), (8,8), (4,4)]

img\_list **=** []

**for** crop **in** crops: *# 5 crops \* 2 flip l-r*

x, y **=** crop

img **=** tf**.**image**.**crop\_to\_bounding\_box(img\_ph, x, y, 56, 56)

img\_list**.**append(img)

img **=** tf**.**image**.**flip\_left\_right(img)

img\_list**.**append(img)

img **=** tf**.**stack(img\_list)

ps **=** tf**.**nn**.**softmax(config**.**model(img, config))

ps **=** tf**.**reduce\_prod(ps, axis**=**0, keep\_dims**=True**)

top\_k **=** tf**.**nn**.**top\_k(ps, k**=**5, sorted**=True**)

saver **=** tf**.**train**.**Saver()

**with** tf**.**Session() **as** sess:

filenames\_labels **=** load\_filenames\_labels('val')

path **=** 'checkpoints/' **+** config**.**model\_name **+** '/' **+** config**.**config\_name

saver**.**restore(sess, tf**.**train**.**latest\_checkpoint(path))

count, correct1, correct5 **=** (0, 0, 0)

**for** filename, label **in** filenames\_labels:

count **+=** 1

np\_img **=** scipy**.**ndimage**.**imread(filename, mode**=**'RGB')

feed\_dict **=** {img\_ph: np\_img}

top\_vals, top\_idx **=** sess**.**run(top\_k, feed\_dict**=**feed\_dict)

**if** top\_idx[0][0] **==** int(label):

correct1 **+=** 1

**if** int(label) **in** top\_idx[0]:

correct5 **+=** 1

**return** correct1 **/** count, correct5 **/** count

config **=** TrainConfig()

acc **=** accuracy(config)

print('Top-1 accuracy (center crop): {}, Top-5 accuracy: {}'**.**format(acc[0], acc[1]))

acc **=** accuracy\_10\_crop(config)

print('Top-1 accuracy (5-crops + l/r flip): {}, Top-5 accuracy: {}'**.**format(acc[0], acc[1]))

## Results Summary

**'baseline' performance**  
Top-1 accuracy (center crop): 0.4929, Top-5 accuracy: 0.7478  
Top-1 accuracy (5-crops + l/r flip): 0.522, Top-5 accuracy: 0.7723

**smoothed cross-entropy**  
Top-1 accuracy (center crop): 0.4958, Top-5 accuracy: 0.7512  
Top-1 accuracy (5-crops + l/r flip): 0.5197, Top-5 accuracy: 0.7695

There is no significant improvement from using smoothed cross-entropy loss.

## Ablation with Image Distortions Removed

### No Flip

Top-1 accuracy (center crop): 0.4613, Top-5 accuracy: 0.7156  
Top-1 accuracy (5-crops + l/r flip): 0.5127, Top-5 accuracy: 0.7572

### No Crop

Top-1 accuracy (center crop): 0.4441, Top-5 accuracy: 0.6986  
Top-1 accuracy (5-crops + l/r flip): 0.4979, Top-5 accuracy: 0.7494

### No Saturation

Top-1 accuracy (center crop): 0.5254, Top-5 accuracy: 0.7691  
Top-1 accuracy (5-crops + l/r flip): 0.5525, Top-5 accuracy: 0.7895

### No Hue

Top-1 accuracy (center crop): 0.5286, Top-5 accuracy: 0.771  
Top-1 accuracy (5-crops + l/r flip): 0.5644, Top-5 accuracy: 0.7929

### No Saturation / No Hue

Top-1 accuracy (center crop): 0.5195, Top-5 accuracy: 0.7621  
Top-1 accuracy (5-crops + l/r flip): 0.5482, Top-5 accuracy: 0.7817

# View TensorFlow Image Distortions

It is standard practice to "augment" training data with distorted images: strerching, cropping, flipping, saturation and hue. TensorFlow has several built-in image distortions. This is a short notebook to view images with these distortions. This is useful for selecting reasonable ranges for random distortions.

Python Notebook by Patrick Coady: [Learning Artificial Intelligence](https://learningai.io/)

**import** tensorflow **as** tf

**import** glob

**import** matplotlib.pyplot **as** plt

**import** random

**import** scipy.ndimage

**%matplotlib** inline

plt**.**rcParams['figure.figsize'] **=** (10, 6)

**def** distort(filename):

"""Apply image distortions"""

**with** tf**.**Graph()**.**as\_default():

file **=** tf**.**read\_file(filename)

img **=** tf**.**image**.**decode\_jpeg(file, 3)

img **=** tf**.**image**.**adjust\_saturation(img, 0.5)

*# img = tf.image.adjust\_hue(img, -0.05)*

**with** tf**.**Session() **as** sess:

dist\_img **=** sess**.**run(img)

**return** dist\_img

filenames **=** glob**.**glob('../tiny-imagenet-200/test/images/\*.JPEG')

pick\_8 **=** random**.**sample(filenames, 8)

count **=** 0

**for** filename **in** pick\_8:

count **+=** 1

plt**.**subplot(4, 4, count)

img **=** scipy**.**ndimage**.**imread(filename)

plt**.**imshow(img)

plt**.**axis('off')

img\_distort **=** distort(filename)

count **+=** 1

plt**.**subplot(4, 4, count)

plt**.**imshow(img\_distort)

plt**.**axis('off')

plt**.**show()