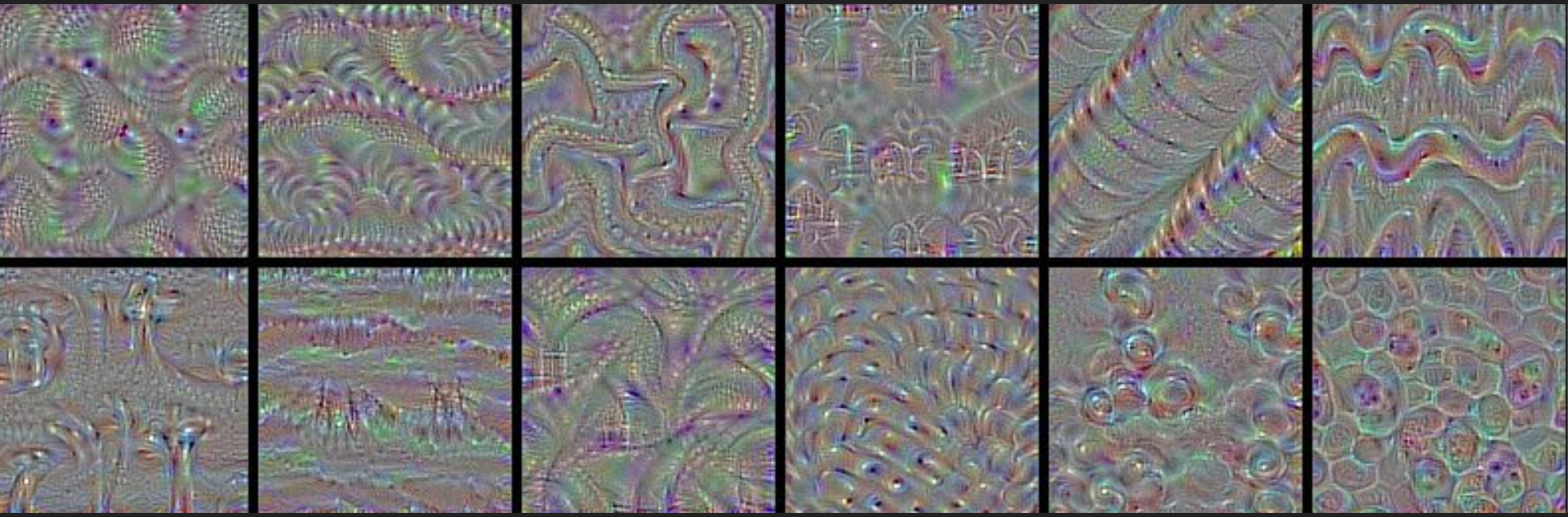
Neural Networks with Keras 2.0 and Tensorflow

Rebecca Ruppel
Github: github.com/reba84



What Goes into Building a Neural Network?

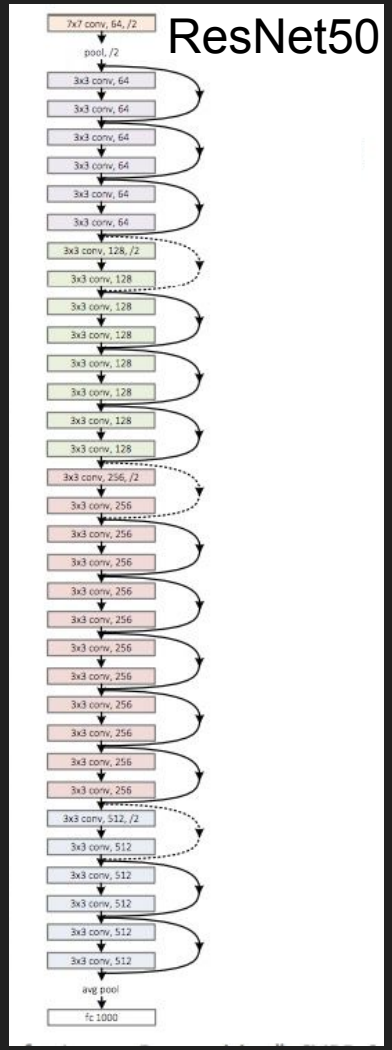
Data
(Testing/Training Sets,
Preprocessing, Batch)

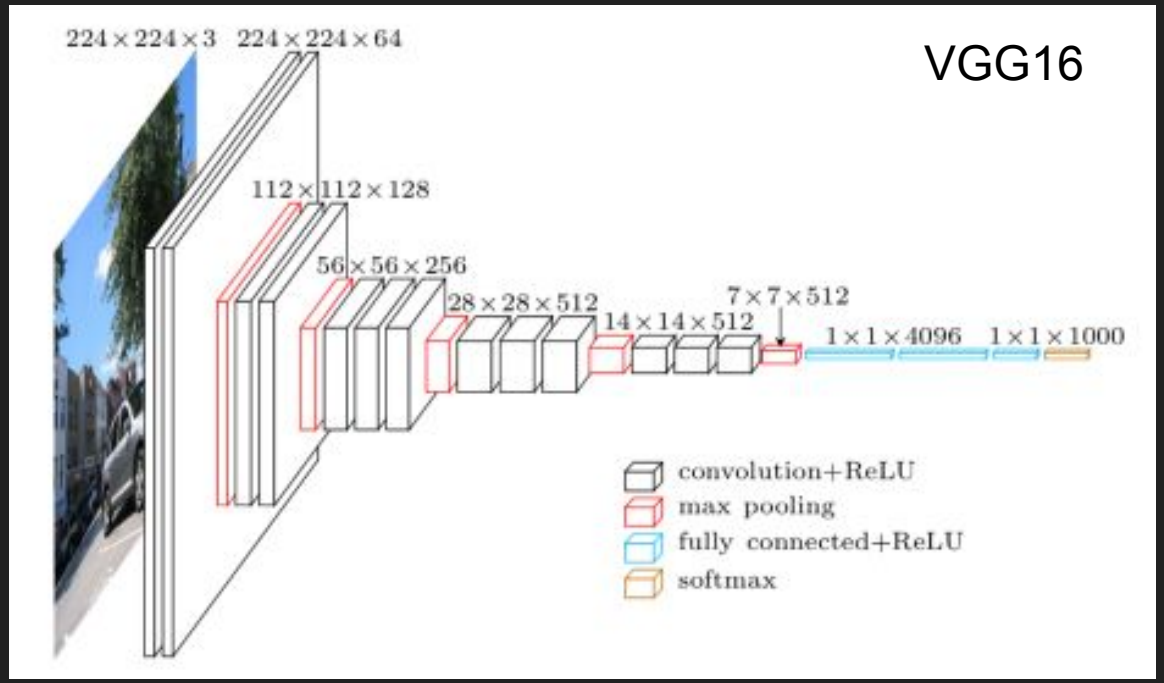
What Goes into Building a Neural Network?

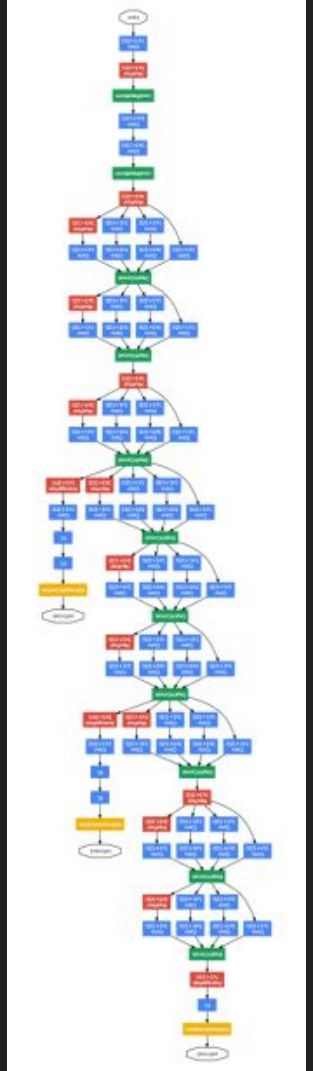
Data
(Testing/Training Sets, Preprocessing, Batch)

Layers
(Convolutional, LTSM, RNN, Fully Connected)

Neural Network Structures







What Goes into Building a Neural Network?

Data
(Testing/Training Sets, Preprocessing, Batch)

Layers
(Convolutional, LTSM, RNN, Fully Connected)

Training
(Data feeding, Optimizers)

What Goes into Building a Neural Network?

Data
(Testing/Training Sets, Preprocessing, Batch)

Layers
(Convolutional, LTSM, RNN, Fully Connected)

Training
(Data feeding, Optimizers)

Predictions
(Run stats here)

Neural Networks with Keras 2.0 and Tensorflow

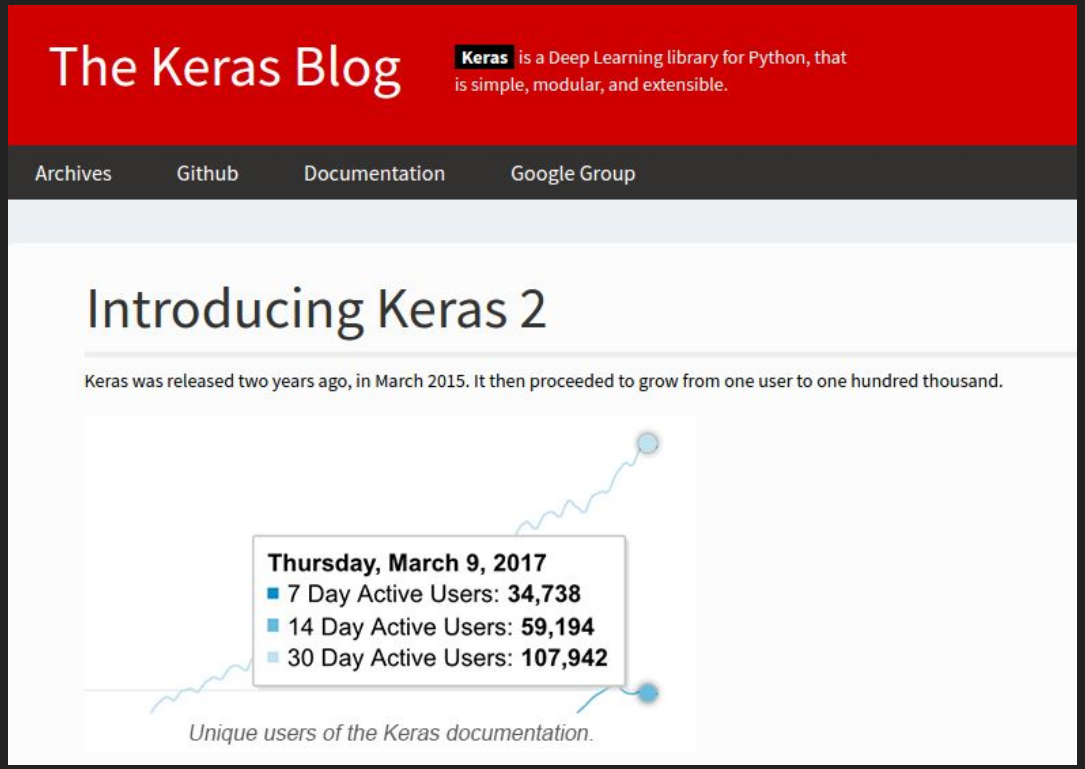
Keras became part of core Tensorflow in March...

What is Tensorflow?

Computational Graph

GPU Architecture

What is Keras?



High level API for:
Tensorflow
Theano
CNTK
More to come….

What Can Keras Do?

```
model = Sequential()
model.add(Conv2D(32, (3, 3), input_shape=(3, 150, 150)))
model.add(Activation('relu'))
model.add(MaxPooling2D(pool_size=(2, 2)))

model.add(Conv2D(32, (3, 3)))
model.add(Activation('relu'))
model.add(MaxPooling2D(pool_size=(2, 2)))

model.add(Conv2D(64, (3, 3)))
model.add(Activation('relu'))
model.add(MaxPooling2D(pool_size=(2, 2)))
```

Build a network
(9 layers)

```
model.add(Flatten())  # this converts our 3D feature maps to 1D feature vectors
model.add(Dense(64))
model.add(Activation('relu'))
model.add(Dropout(0.5))
model.add(Dense(1))
model.add(Activation('sigmoid'))

model.compile(loss='binary_crossentropy',
              optimizer='rmsprop',
              metrics=['accuracy'])
```

Define optimizing and
loss functions

What Can Keras Do?

Preprocess Data

```
batch_size = 16

# this is the augmentation configuration we will use for training
train_datagen = ImageDataGenerator(
        rescale=1./255,
        shear_range=0.2,
        zoom_range=0.2,
        horizontal_flip=True)
```

Data Loading

```
train_generator = train_datagen.flow_from_directory(
        'data/train',  # this is the target directory
        target_size=(150, 150),  # all images will be resized to 150x150
        batch_size=batch_size,
        class_mode='binary')  # since we use binary_crossentropy loss, we need binary labels
```

Data Feeding and Training

```
model.fit_generator(
        train_generator,
        steps_per_epoch=2000 // batch_size,
        epochs=50,
        validation_data=validation_generator,
        validation_steps=800 // batch_size)
model.save_weights('first_try.h5')  # always save your weights after training or during training
```

Tensorflow Neural Network

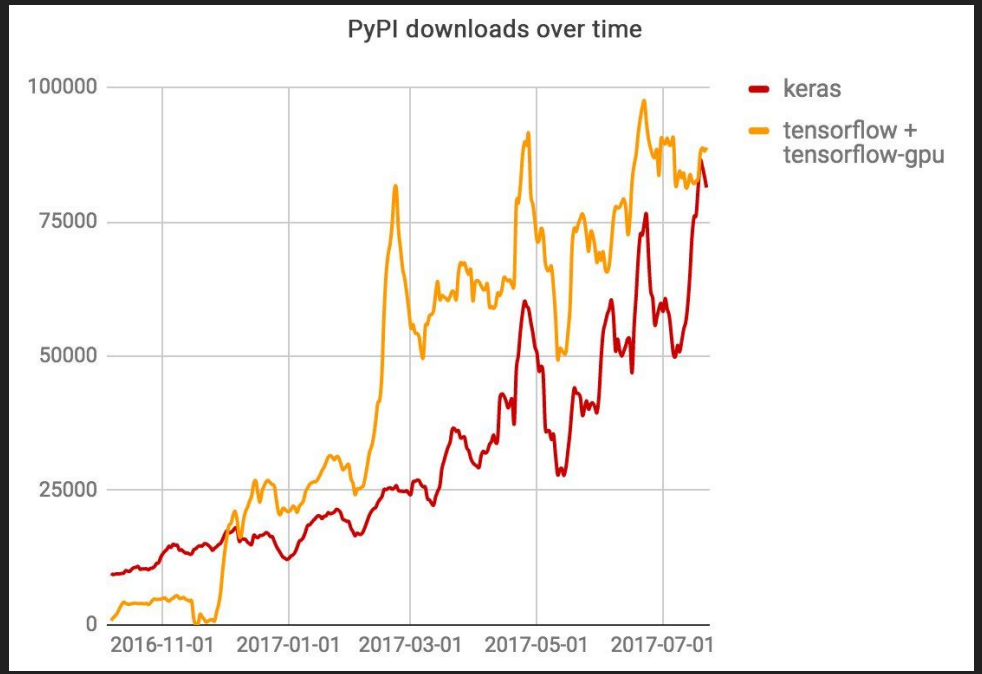
```python
def deepnn(x):
  # grayscale -- it would be 3 for an RGB image, 4 for RGBA, etc.
  x_image = tf.reshape(x, [-1, 28, 28, 1])

  # First convolutional layer - maps one grayscale image to 32 feature maps.
  W_conv1 = weight_variable([5, 5, 1, 32])
  b_conv1 = bias_variable([32])
  h_conv1 = tf.nn.relu(conv2d(x_image, W_conv1) + b_conv1)

  # Pooling layer - downsamples by 2X.
  h_pool1 = max_pool_2x2(h_conv1)

  # Second convolutional layer -- maps 32 feature maps to 64.
  W_conv2 = weight_variable([5, 5, 32, 64])
  b_conv2 = bias_variable([64])
  h_conv2 = tf.nn.relu(conv2d(h_pool1, W_conv2) + b_conv2)

  # Second pooling layer.
  h_pool2 = max_pool_2x2(h_conv2)

  # Fully connected layer 1 -- after 2 round of downsampling, our 28x28 image
  # is down to 7x7x64 feature maps -- maps this to 1024 features.
  W_fc1 = weight_variable([7 * 7 * 64, 1024])
  b_fc1 = bias_variable([1024])

  h_pool2_flat = tf.reshape(h_pool2, [-1, 7*7*64])
  h_fc1 = tf.nn.relu(tf.matmul(h_pool2_flat, W_fc1) + b_fc1)

  # Dropout - controls the complexity of the model, prevents co-adaptation of
  # features.
  keep_prob = tf.placeholder(tf.float32)
  h_fc1_drop = tf.nn.dropout(h_fc1, keep_prob)

  # Map the 1024 features to 10 classes, one for each digit
  W_fc2 = weight_variable([1024, 10])
  b_fc2 = bias_variable([10])

  y_conv = tf.matmul(h_fc1_drop, W_fc2) + b_fc2
  return y_conv, keep_prob


def conv2d(x, W):
  """conv2d returns a 2d convolution layer with full stride."""
  return tf.nn.conv2d(x, W, strides=[1, 1, 1, 1], padding='SAME')


def max_pool_2x2(x):
  """max_pool_2x2 downsamples a feature map by 2X."""
  return tf.nn.max_pool(x, ksize=[1, 2, 2, 1],
```

```python
def weight_variable(shape):
  """weight_variable generates a weight variable of a given shape."""
  initial = tf.truncated_normal(shape, stddev=0.1)
  return tf.Variable(initial)


def bias_variable(shape):
  """bias_variable generates a bias variable of a given shape."""
  initial = tf.constant(0.1, shape=shape)
  return tf.Variable(initial)


def main(_):
  # Import data
  mnist = input_data.read_data_sets(FLAGS.data_dir, one_hot=True)

  # Create the model
  x = tf.placeholder(tf.float32, [None, 784])

  # Define loss and optimizer
  y_ = tf.placeholder(tf.float32, [None, 10])

  # Build the graph for the deep net
  y_conv, keep_prob = deepnn(x)

  cross_entropy = tf.reduce_mean(tf.nn.softmax_cross_entropy_with_logits(labels=y_, logits=y_conv))
  train_step = tf.train.AdamOptimizer(1e-4).minimize(cross_entropy)
  correct_prediction = tf.equal(tf.argmax(y_conv, 1), tf.argmax(y_, 1))
  accuracy = tf.reduce_mean(tf.cast(correct_prediction, tf.float32))

  with tf.Session() as sess:
    sess.run(tf.global_variables_initializer())
    for i in range(1000):
      batch = mnist.train.next_batch(50)
      if i % 100 == 0:
        train_accuracy = accuracy.eval(feed_dict={
            x: batch[0], y_: batch[1], keep_prob: 1.0})
        print('step %d, training accuracy %g' % (i, train_accuracy))
      train_step.run(feed_dict={x: batch[0], y_: batch[1], keep_prob: 0.5})

    print('test accuracy %g' % accuracy.eval(feed_dict={
        x: mnist.test.images, y_: mnist.test.labels, keep_prob: 1.0}))

if __name__ == '__main__':
  parser = argparse.ArgumentParser()
  parser.add_argument('--data_dir', type=str,
                      default='/tmp/tensorflow/mnist/input_data',
                      help='Directory for storing input data')
  FLAGS, unparsed = parser.parse_known_args()
  tf.app.run(main=main, argv=[sys.argv[0]] + unparsed)
```

Complicated but very flexible!

5 layers

Neural Networks with Keras 2.0 and Tensorflow



new stars from 2017-04-20 to 2017-07-06		
#1:	7929	tensorflow/tensorflow
#2:	2465	fchollet/keras
#3:	1894	caffe2/caffe2
#4:	1526	BVLC/caffe
#5:	1250	pytorch/pytorch
#6:	1233	Microsoft/CNTK
#7:	979	dmlc/mxnet
#8:	709	deepmind/sonnet
#9:	690	tflearn/tflearn
#10:	485	deeplearning4j/deeplearning4j
#11:	458	Theano/Theano
#12:	452	davisking/dlib
#13:	341	torch/torch7
#14:	303	baidu/paddle
#15:	243	pfnet/chainer

aggregate activity from 2017-04-20 to 2017-07-06		
#1:	37.70	tensorflow/tensorflow
#2:	13.53	fchollet/keras
#3:	8.15	dmlc/mxnet
#4:	7.46	caffe2/caffe2
#5:	7.38	pytorch/pytorch
#6:	5.88	BVLC/caffe
#7:	5.71	baidu/paddle
#8:	4.57	Microsoft/CNTK
#9:	4.26	deeplearning4j/deeplearning4j
#10:	2.25	tflearn/tflearn
#11:	2.06	davisking/dlib
#12:	1.93	Theano/Theano
#13:	1.75	pfnet/chainer
#14:	1.52	NVIDIA/DIGITS
#15:	1.46	clab/dynet

Graphs from Francois Chollet's twitter

What Goes into Building a Neural Network?

Data
(Testing/Training Sets, Preprocessing, Batch)

Layers
(Convolutional, LTSM, RNN, Fully Connected)

Training
(Data feeding, Optimizers)

Predictions
(Run stats here)

Image Preprocessing

scikit-image
image processing in python

pillow

OpenCV

Useful Keras Features: Image Preprocessing

ImageDataGenerator

```
keras.preprocessing.image.ImageDataGenerator(featurewise_center=False,
    samplewise_center=False,
    featurewise_std_normalization=False,
    samplewise_std_normalization=False,
    zca_whitening=False,
    zca_epsilon=1e-6,
    rotation_range=0.,
    width_shift_range=0.,
    height_shift_range=0.,
    shear_range=0.,
    zoom_range=0.,
    channel_shift_range=0.,
    fill_mode='nearest',
    cval=0.,
    horizontal_flip=False,
    vertical_flip=False,
    rescale=None,
    preprocessing_function=None,
    data_format=K.image_data_format())
```

Useful Keras Features: Image Preprocessing

ImageDataGenerator

```
keras.preprocessing.image.ImageDataGenerator(featurewise_center=False,
    samplewise_center=False,
    featurewise_std_normalization=False,
    samplewise_std_normalization=False,
    zca_whitening=False,
    zca_epsilon=1e-6,
    rotation_range=0.,
    width_shift_range=0.,
    height_shift_range=0.,
    shear_range=0.,
    zoom_range=0.,
    channel_shift_range=0.,
    fill_mode='nearest',
    cval=0.,
    horizontal_flip=False,
    vertical_flip=False,
    rescale=None,
    preprocessing_function=None,
    data_format=K.image_data_format())
```

This has a method that batches data from directories...

Useful Keras Features: Data Loading

Keras Documentation

Search docs

Home

Getting started

- Guide to the Sequential model
- Guide to the Functional API
- FAQ

Models

- About Keras models
- Sequential
- Model (functional API)

Layers

- About Keras layers
- Core Layers
- Convolutional Layers

Example of using `.flow_from_directory(directory)`:

```
train_datagen = ImageDataGenerator(
        rescale=1./255,
        shear_range=0.2,
        zoom_range=0.2,
        horizontal_flip=True)

test_datagen = ImageDataGenerator(rescale=1./255)

train_generator = train_datagen.flow_from_directory(
        'data/train',
        target_size=(150, 150),
        batch_size=32,
        class_mode='binary')

validation_generator = test_datagen.flow_from_directory(
        'data/validation',
        target_size=(150, 150),
        batch_size=32,
        class_mode='binary')

model.fit_generator(
        train_generator,
        steps_per_epoch=2000,
        epochs=50,
        validation_data=validation_generator,
        validation_steps=800)
```

Keras Features: Data Loading

IMAGENET

Total number of synsets: 21841
Total number of images: 14,197,122

Loading Data in Tensorflow

```
import tensorflow as tf
import numpy as np
import threading

# Generating some simple data
r = np.arange(0.0,100003.0)
raw_data = np.dstack((r,r,r,r))[0]
raw_target = np.array([[1,0,0]] * 100003)

# are used to feed data into our queue
queue_input_data = tf.placeholder(tf.float32, shape=[20, 4])
queue_input_target = tf.placeholder(tf.float32, shape=[20, 3])

queue = tf.FIFOQueue(capacity=50, dtypes=[tf.float32, tf.float32], shapes=

enqueue_op = queue.enqueue_many([queue_input_data, queue_input_target])
dequeue_op = queue.dequeue()

# tensorflow recommendation:
# capacity = min_after_dequeue + (num_threads + a small safety margin) *
data_batch, target_batch = tf.train.batch(dequeue_op, batch_size=1, capac
# use this to shuffle batches:
# data_batch, target_batch = tf.train.shuffle_batch(dequeue_op, batch_size

def enqueue(sess):
  """ Iterates over our data puts small junks into our queue."""
  under = 0
  max = len(raw_data)
  while True:
    print("starting to write into queue")
    upper = under + 20
    print("try to enqueue ", under, " to ", upper)
    if upper <= max:
      curr_data = raw_data[under:upper]
      curr_target = raw_target[under:upper]
      under = upper
    else:
      rest = upper - max
      curr_data = np.concatenate((raw_data[under:max], raw_data[0:rest]))
      curr_target = np.concatenate((raw_target[under:max], raw_target[0:re
      under = rest
```

```
    sess.run(enqueue_op, feed_dict={queue_input_data: curr_data,
                                    queue_input_target: curr_target})
    print("added to the queue")
  print("finished enqueueing")

# start the threads for our FIFOQueue and batch
sess = tf.Session()
enqueue_thread = threading.Thread(target=enqueue, args=[sess])
enqueue_thread.isDaemon()
enqueue_thread.start()

coord = tf.train.Coordinator()
threads = tf.train.start_queue_runners(coord=coord, sess=sess)

# Fetch the data from the pipeline and put it where it belongs (into your
for i in range(5):
  run_options = tf.RunOptions(timeout_in_ms=4000)
  curr_data_batch, curr_target_batch = sess.run([data_batch, target_batch
  print(curr_data_batch)

# shutdown everything to avoid zombies
sess.run(queue.close(cancel_pending_enqueues=True))
coord.request_stop()
coord.join(threads)
sess.close()
```

Code from: http://ischlag.github.io/

Also complicated but very flexible!

Useful Keras Features: Data Loading

Keras Documentation

Search docs

Home

Getting started

Guide to the Sequential model

Guide to the Functional API

FAQ

Models

About Keras models

Sequential

Model (functional API)

Layers

About Keras layers

Core Layers

Convolutional Layers

Example of using `.flow_from_directory(directory)`:

```
train_datagen = ImageDataGenerator(
        rescale=1./255,
        shear_range=0.2,
        zoom_range=0.2,
        horizontal_flip=True)

test_datagen = ImageDataGenerator(rescale=1./255)

train_generator = train_datagen.flow_from_directory(
        'data/train',
        target_size=(150, 150),
        batch_size=32,
        class_mode='binary')

validation_generator = test_datagen.flow_from_directory(
        'data/validation',
        target_size=(150, 150),
        batch_size=32,
        class_mode='binary')

model.fit_generator(
        train_generator,
        steps_per_epoch=2000,
        epochs=50,
        validation_data=validation_generator,
        validation_steps=800)
```

Useful Keras Features: Data Loading

Keras Documentation

Search docs

Home

Getting started

Guide to the Sequential model

Guide to the Functional API

FAQ

Models

About Keras models

Sequential

Model (functional API)

Layers

About Keras layers

Core Layers

Convolutional Layers

Example of using `.flow_from_directory(directory)`:

```
train_datagen = ImageDataGenerator(
        rescale=1./255,
        shear_range=0.2,
        zoom_range=0.2,
        horizontal_flip=True)

test_datagen = ImageDataGenerator(rescale=1./255)

train_generator = train_datagen.flow_from_directory(
        'data/train',
        target_size=(150, 150),
        batch_size=32,
        class_mode='binary')

validation_generator = test_datagen.flow_from_directory(
        'data/validation',
        target_size=(150, 150),
        batch_size=32,
        class_mode='binary')

model.fit_generator(
        train_generator,
        steps_per_epoch=2000,
        epochs=50,
        validation_data=validation_generator,
        validation_steps=800)
```

What if you aren’t working with data that is in their “white list" of file types?

Solution: Write custom generator

Which function, `fit` or `train_on_batch` is more appropriate for this situation?

4

joelthchao commented on May 12, 2016 • edited

Contributor

I think the best way to fit large data which can not fit into memory is writing a customize generator. It yields (images, labels) with samples in size of batch. You can design your own loading mechanism to better fit with your memory requirement and use `fit_generator` like the example.

```
def generate_arrays_from_file(path):
    while 1:
        f = open(path)
        for line in f:
            # create numpy arrays of input data
            # and labels, from each line in the file
            x, y = process_line(line)
            img = load_images(x)
            yield (img, y)
        f.close()

model.fit_generator(generate_arrays_from_file('/my_file.txt'),
        samples_per_epoch=10000, nb_epoch=10)
```

23

My solution: Edit Image Processing Class

```
974         white_list_formats = {'png', 'jpg', 'jpeg', 'bmp'}
975
976         # first, count the number of samples and classes
977         self.samples = 0
978
979         if not classes:
980             classes = []
981             for subdir in sorted(os.listdir(directory)):
982                 if os.path.isdir(os.path.join(directory, subdir)):
983
```

Easy to understand comments

```
997         # second, build an index of the images in the different class subfolders
998         results = []
999
1000        self.filenames = []
1001        self.classes = np.zeros((self.samples,), dtype='int32')
1002        i = 0
1003        for dirpath in (os.path.join(directory, subdir) for subdir in classes):
1004            results.append(pool.apply_async(_list_valid_filenames_in_directory,
1005                                            (dirpath, white_list_formats,
1006                                             self.class_indices, follow_links)))
1007        for res in results:
1008            classes, filenames = res.get()
1009            self.classes[i:i + len(classes)] = classes
1010            self.filenames += filenames
1011            i += len(classes)
1012        pool.close()
1013        pool.join()
1014        super(DirectoryIterator, self).__init__(self.samples, batch_size, shuffle, seed)
```

Code with nice variable names and clear structure

My solution: Edit Image Processing Class

Keras code

```
        # The transformation of images is not under thread lock
        # so it can be done in parallel
        batch_x = np.zeros((current_batch_size,) + self.image_shape, dtype=K.floatx())
        grayscale = self.color_mode == 'grayscale'
        # build batch of image data
        for i, j in enumerate(index_array):
            fname = self.filenames[j]
            img = load_img(os.path.join(self.directory, fname),
                           grayscale=grayscale,
                           target_size=self.target_size)
            x = img_to_array(img, data_format=self.data_format)
            x = self.image_data_generator.random_transform(x)
            x = self.image_data_generator.standardize(x)
            batch_x[i] = x
        # optionally save augmented images to disk for debugging
```

Code I wrote

```
        # The transformation of images is not under thread lock
        # so it can be done in parallel
        batch_x = np.zeros((current_batch_size,) + self.image_shape, dtype=K.floatx())
        grayscale = self.color_mode == 'grayscale'
        # build batch of image data
        for i, j in enumerate(index_array):
            fname = self.filenames[j]

            if self.is_matrix:
                x = load_mat(os.path.join(self.directory, fname))
            else:
                img = load_img(os.path.join(self.directory, fname),
                               grayscale=grayscale,
                               target_size=self.target_size)
                x = img_to_array(img, data_format=self.data_format)
                x = self.image_data_generator.random_transform(x)
                x = self.image_data_generator.standardize(x)
            batch_x[i] = x
        # optionally save augmented images to disk for debugging purposes
        if self.save_to_dir:
```

Keras Community

Keras “Guiding Principles”

- User Friendliness
- Modularity
- Easy extensibility
- Work with Python

Keras Community

For Beginners

ImageDataGenerator object is not an iterator #7235

Closed **NikhilShaw** opened this issue on Jul 4 · 2 comments

NikhilShaw commented on Jul 4

Jul 4, 2017, 12:52 PM CDT

```
Using TensorFlow backend.
Epoch 1/10
Exception in thread Thread-1:
Traceback (most recent call last):
  File "/usr/lib/python2.7/threading.py", line 801, in __bootstrap_inner
    self.run()
```

Assignees

No one assigned

Labels

None yet

Or Advanced Users

Using data tensors as data sources: action plan #7503

Open **fchollet** opened this issue 5 days ago · 3 comments

fchollet commented 5 days ago

Owner

We want to add the ability to feed TensorFlow data tensors (e.g. input queues) into Keras models. A few days ago I met with **@athundt** and we discussed his previous efforts to make it happen. Here is how we will handle it:

First step

The following API:

```
# Get data tensors
data_tensor, target_tensor = ...
```

Keras Community

For Beginners

ImageDataGenerator object is not an iterator #7235

Closed **NikhilShaw** opened this issue on Jul 4 · 2 c

NikhilShaw commented on Jul 4

Jul 4, 2017, 12:5

```
Using TensorFlow backend.
Epoch 1/10
Exception in thread Thread-1:
Traceback (most recent call last):
  File "/usr/lib/python2.7/threading.py
    self.run()
```

```
    # compiling the model
    model.compile(optimizer='adam', loss='categorical_crossentropy', metrics=['accuracy'])
    return model

new_model=create_model()
new_model.fit_generator(datagen, steps_per_epoch= len(x_train), epochs=10)
```

Dref360 commented on Jul 4 Contributor

If you follow the documentation : https://keras.io/preprocessing/image/
You should use `flow`.

NikhilShaw commented on Jul 4

@**Dref360** Thanks, worked for me. I should have read documentation more properly.

NikhilShaw closed this on Jul 4

Keras Community

Or Advanced Users

Using data tensors as data sources: action plan #7503

Open **fchollet** opened this issue 5 days ago · 3 comments

Dref360 commented 4 days ago

Contributor

Step 3 seems really "hacky". Could we ask the TF team if they are willing to handle feeding placeholder with Tensors?

For step 2, I was away for a while so I didn't keep up with **@athundt** 's PR. But since the `data_tensor` is already there, I see no problem doing : `model.compile(y=target_tensor, optimizer='sgd', loss='mse')` .

Would save one compilation, if you've already talked about it in the PR, ignore this.

Keras Community

Or Advanced Users

Using data tensors as data sources: action plan #7503

Open **fchollet** opened this issue 5 days ago · 3 comments

Dref360 commented 4 days ago

Contributor

Step 3 se

Tensors?

For step

already tl

`loss='ms`

Would sa

TimZaman commented 3 days ago • edited

@Dref360

> Step 3 seems really "hacky".

Yes, it's a bit dirty. But I think Keras's API's do allow us to clean up the graph-surgery mess quite easily, in a way that it's a hack in principle, but it's a great one. We'll see when we get there.

> Could we ask the TF team if they are willing to handle feeding placeholder with Tensors?

We did; issue: tensorflow/tensorflow#10837

```
model.compile(y=target_tensor, optimizer='sgd', loss='mse').
```

On first glance, that sounds pretty sane to me! I don't recall anyone suggesting this?

1

What will tensorflow.contrib.keras be like?

```python
# ==============================================================================
"""Keras initializer classes (soon to be replaced with core TF initializers)
"""
from __future__ import absolute_import
from __future__ import division
from __future__ import print_function

import six

from tensorflow.contrib.keras.python.keras.utils.generic_utils import deserialize_keras_object
```

```python
# ==============================================================================
"""Keras optimizer classes (will eventually be replaced with core optimizers).
"""
from __future__ import absolute_import
from __future__ import division
from __future__ import print_function

import copy

import six
from six.moves import zip  # pylint: disable=redefined-builtin
```

Summary

- Keras as a high level API to simplify neural network development and iterative development
  - Data Preprocessing and batching
  - Defining layers
  - Training
  - Predicting
- Tensorflow is fast and is a backend. Also, option for advanced development
- Keras in Tensorflow will hopefully provide the best of both worlds

Rebecca Ruppel
Github: github.com/reba84

Neural Networks with Keras 2.0 and Tensorflow

Rebecca Ruppel
Github: github.com/reba84