UCLOUD

专业云计算服务商

深度学习TensorFlow下的计算机视觉

http://hacker.duanshishi.com
https://github.com/burness

Burness Duan

UCloud

OUTLINE

1. TensorFlow Features

2. TensorFlow And Deep CV

3. TFLearn Introduction

TensorFlow Features

1. Programming Model and Basic Concepts
2. Implementation
3. Extensions
4. Optimizations

Abadi M, Agarwal A, Barham P, et al. TensorFlow: Large-scale machine learning on heterogeneous systems, 2015[J]. Software available from tensorflow. org, 2015, 1.

Programming Model and Basic Concepts

```
import tensorflow as tf

b = tf.Variable(tf.zeros([100]))                    # 100-d vector, init to zeroes
W = tf.Variable(tf.random_uniform([784,100],-1,1))  # 784x100 matrix w/rnd vals
x = tf.placeholder(name="x")                        # Placeholder for input
relu = tf.nn.relu(tf.matmul(W, x) + b)              # Relu(Wx+b)
C = [...]                                           # Cost computed as a function
                                                    # of Relu

s = tf.Session()
for step in xrange(0, 10):
  input = ...construct 100-D input array ...        # Create 100-d vector for input
  result = s.run(C, feed_dict={x: input})           # Fetch cost, feeding x=input
  print step, result
```

Figure 1: Example TensorFlow code fragment

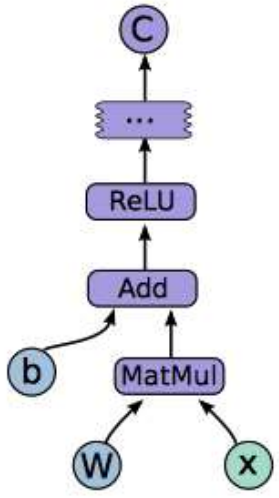


Figure 2: Corresponding computation graph for Figure 1

Programming Model and Basic Concepts

1. Operations and Kernels
2. Sessions
3. Variables

Programming Model and Basic Concepts

Operation: An abstract computation, can have attributes
Kernel: A particular implementation of an operation that can be run a particular type of device(CPU or GPU)
Sessions: Interact with the TensorFlow system, Run the full graph or a few distinct subgraphs
Variables: A speical operation returns a handle to a persistent mutable tensor

https://www.tensorflow.org/versions/r0.11/how_tos/adding_an_op/index.html

Implementation

1. Single-Device Execution
2. Multi-Device Execution
    a) Node Placement
    b) Cross-Device Communication
3. Distributed Execution

Single-Device Execution

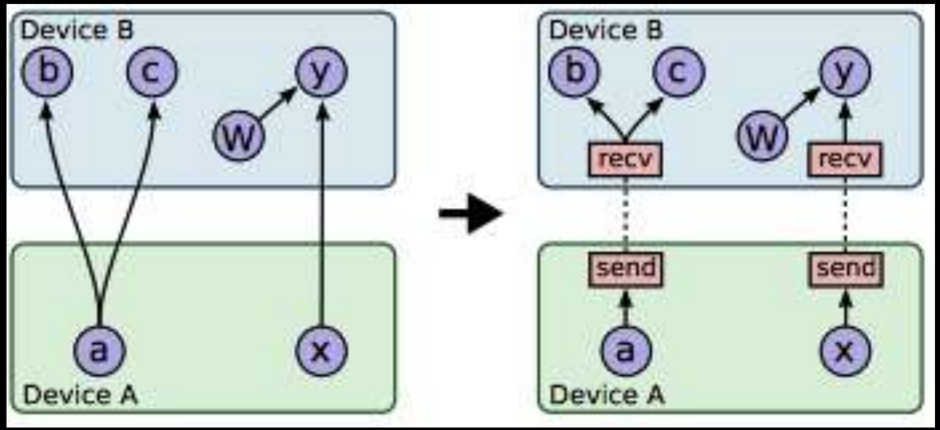
The nodes of the graph are executed in an order that respects the dependencies between the nodes

Multi-Device Execution

Node Placement： Based on heuristics associated with different operation types or is measured based on an actual set of placement decisions for earlier extentions of the graph

Multi-Device Execution

Cross-Device Communication:



Distributed Execution

Similar to multi-device execution, After device placement, a subgraph is created per device. Send/Receive node communicate across woker processes use remote communication such as TCP or RDMA.

Distributed TensorFlow Example

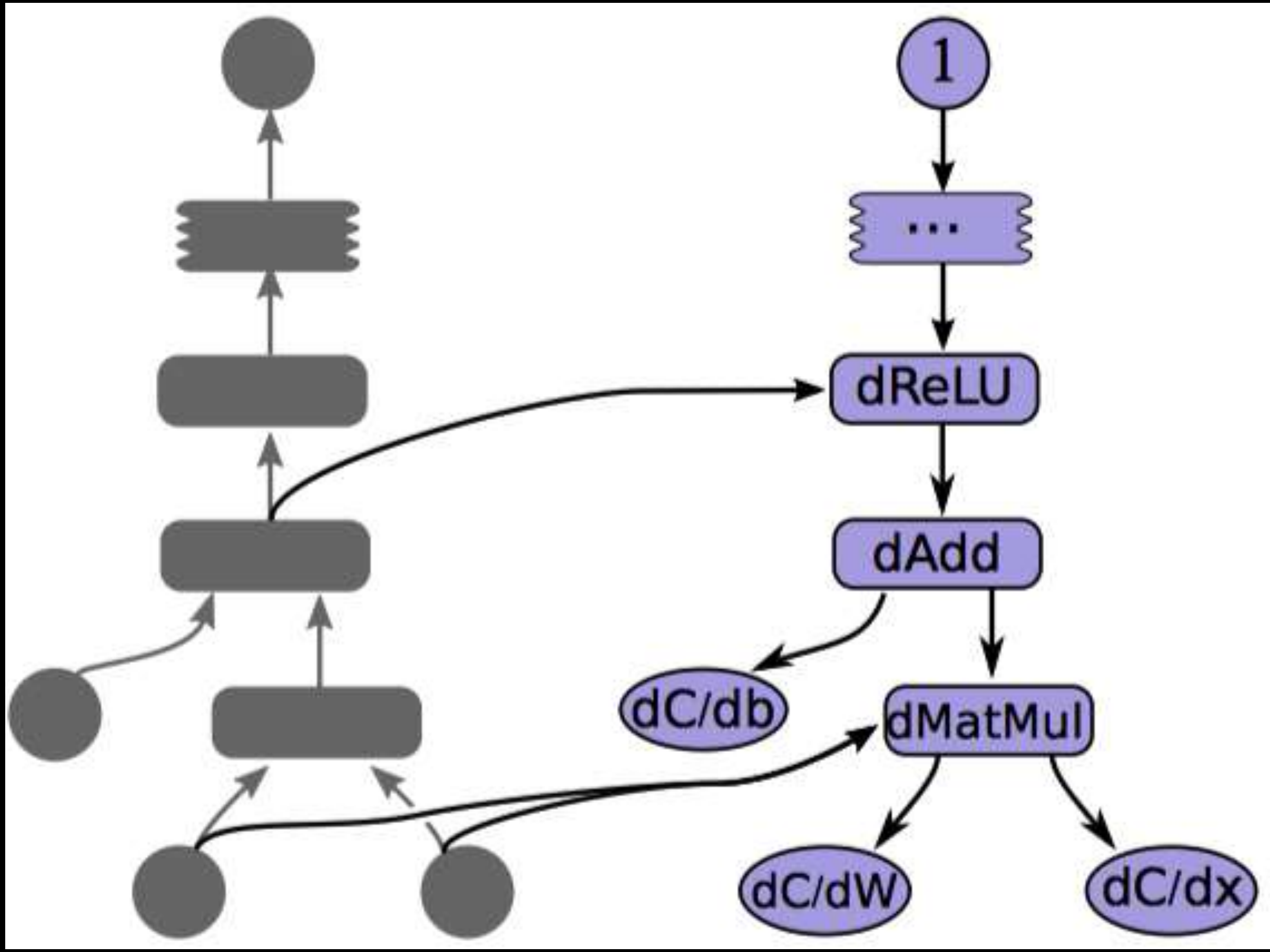
Distributed_tensorflow.py

```
# On ps0.example.com:
$ python trainer.py \
     --ps_hosts=ps0.example.com:2222,ps1.example.com:2222 \
     --worker_hosts=worker0.example.com:2222,worker1.example.com:2222 \
     --job_name=ps --task_index=0
# On ps1.example.com:
$ python trainer.py \
     --ps_hosts=ps0.example.com:2222,ps1.example.com:2222 \
     --worker_hosts=worker0.example.com:2222,worker1.example.com:2222 \
     --job_name=ps --task_index=1
# On worker0.example.com:
$ python trainer.py \
     --ps_hosts=ps0.example.com:2222,ps1.example.com:2222 \
     --worker_hosts=worker0.example.com:2222,worker1.example.com:2222 \
     --job_name=worker --task_index=0
# On worker1.example.com:
$ python trainer.py \
     --ps_hosts=ps0.example.com:2222,ps1.example.com:2222 \
     --worker_hosts=worker0.example.com:2222,worker1.example.com:2222 \
     --job_name=worker --task_index=1
```

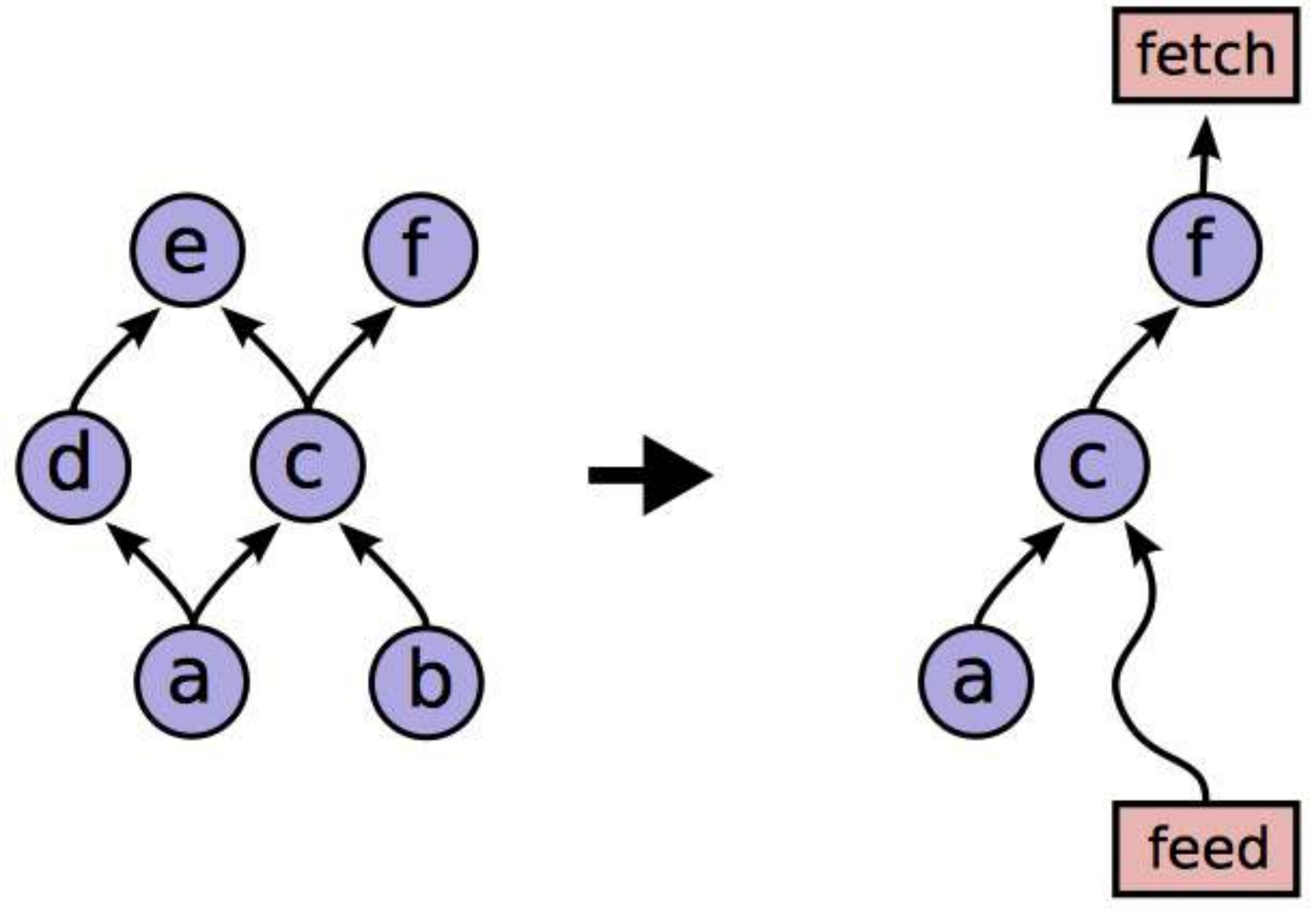
Extensions

1. Gradient Computation ⟷ TF hold a lot of GPU memmory
2. Partial Execution ⟷ Run the nodes with the deps of the out tensor
3. Device Constraints ⟷ You can define which nodes run on the specify devices
4. Control Flow ⟷ Allow us to skip the execution of an subgraph based on the Value of a boolean tensor
5. Input Operations ⟷ From the storage system to the worker
6. Queues ⟷ Allow the inpu data to be prefeteched from disk files while a previous batch of data is being processed
7. Containers ⟷ Share state even across completely disjoint computation graphs

Gradient Computation



Partial Execution



Device Constraints

Client can control the placement of nodes on devices by providing partial constraints for a node about Which devices it can execute on.

Control Flow

High-level programming constructs such if-conditionals and while-loops can be easily compiled Into dataflow graph with these control flow operators.

Input Operations

An special node in graph: efficient mechanism used for training large-scale machine learning models

Queues

Allow input data to prefetched from disk files while a previous batch of data is still being processed by the model

Containers

Allow input data to prefetched from disk files while a previous batch of data is still being processed by the model

Optimizations

1. Common Subexpression Elimination
2. Asynchronous Kernels
3. Optimized Libraries for Kernel Implementation

Common Subexpression Elimination

Convert the redundant copies of the same computation to just a single one

Asynchronous Kernels

Non-blocking kernels， avoid tying up an execution thread for unbounded periods of time while waiting for I/O or other events.

Optimized Libs For Kernel Implementations

Eigen\BLAS\cuBLAS, GPU CNN Kernel: cuda-convnet, cuDNN

TensorFlow And Deep CV

1.Image Classification

2.Neural Style

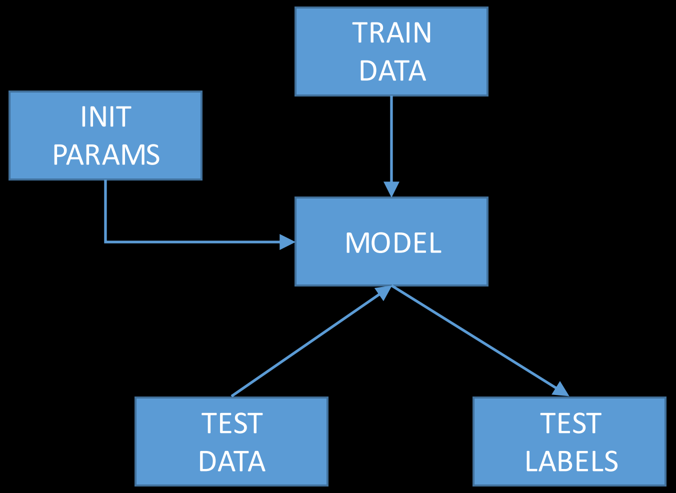
3.Txt2Img, img2txt

Image Classification

1. Training from scratch
2. Retrain from pre-trained model
3. Load model and frozen some layers' weights and retrain the other layers

Training from scratch

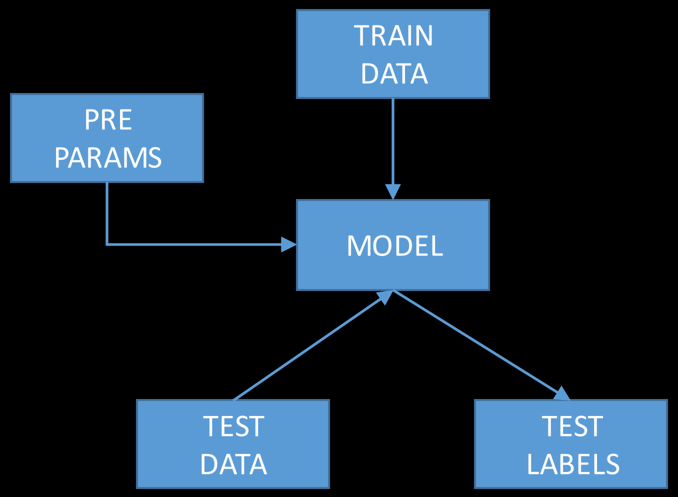
```
mnist = input_data.read_data_sets(FLAGS.data_dir, one_hot=True)
x = tf.placeholder(tf.float32, [None, 784])
W = tf.Variable(tf.zeros([784, 10]))
b = tf.Variable(tf.zeros([10]))
y = tf.matmul(x, W) + b
y_ = tf.placeholder(tf.float32, [None, 10])
cross_entropy = tf.reduce_mean(tf.nn.softmax_cross_entropy_with_logits(y, y_))
train_step = tf.train.GradientDescentOptimizer(0.5).minimize(cross_entropy)
sess = tf.InteractiveSession()
tf.initialize_all_variables().run()
for _ in range(1000):
   batch_xs, batch_ys =mnist.train.next_batch(100)
   sess.run(train_step, feed_dict={x: batch_xs, y_: batch_ys})
   correct_prediction = tf.equal(tf.argmax(y, 1), tf.argmax(y_, 1))
   accuracy = tf.reduce_mean(tf.cast(correct_prediction, tf.float32))
   print(sess.run(accuracy, feed_dict={x: mnist.test.images,  y_: mnist.test.labels}))
```



专业云计算服务商

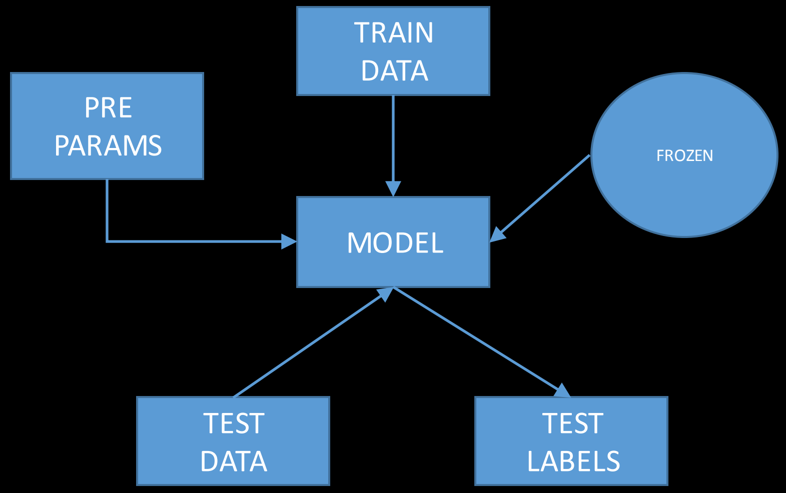
Retrain from pre-trained model

1. Load pre params from a model file(pb， ckpt)
2. Change your model to the new task(class num)
3. Fit the train data to update the all weights
4. Fit your test data to inference



Retrain from pre-trained model (frozen)

1. Load pre params from a model file(pb)
2. Frozen your layers' weight
3. Change the network's class num
4. Fit your train data to update the unfrozen layers' weight
5. Fit your test data to inference



https://github.com/spark-mler/WorkWithTensorflow/tree/master/cv_bot/models/pretrain_inference

Neural Style

UCLOUD
专业云计算服务商

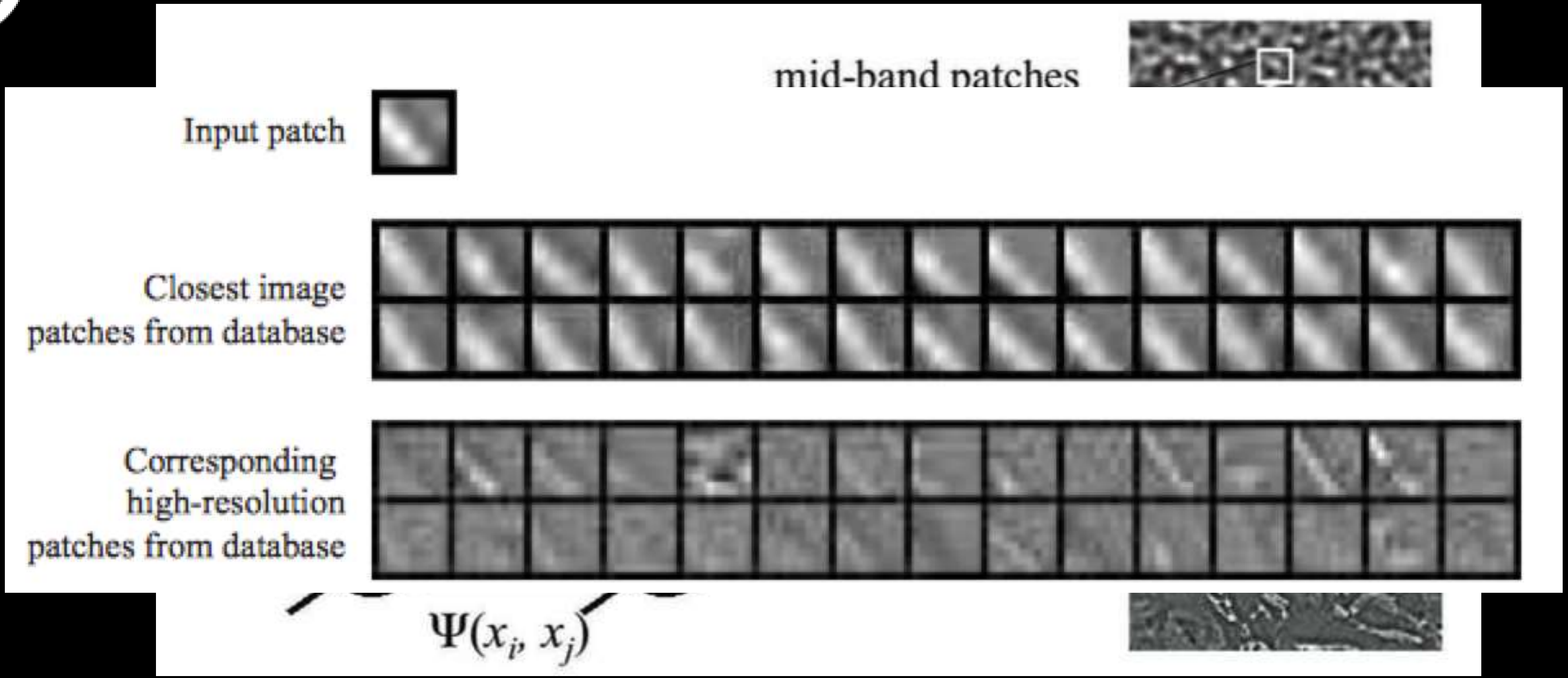
1. MRF-Based

2. CNN-Based

3. MRF and CNN-Based

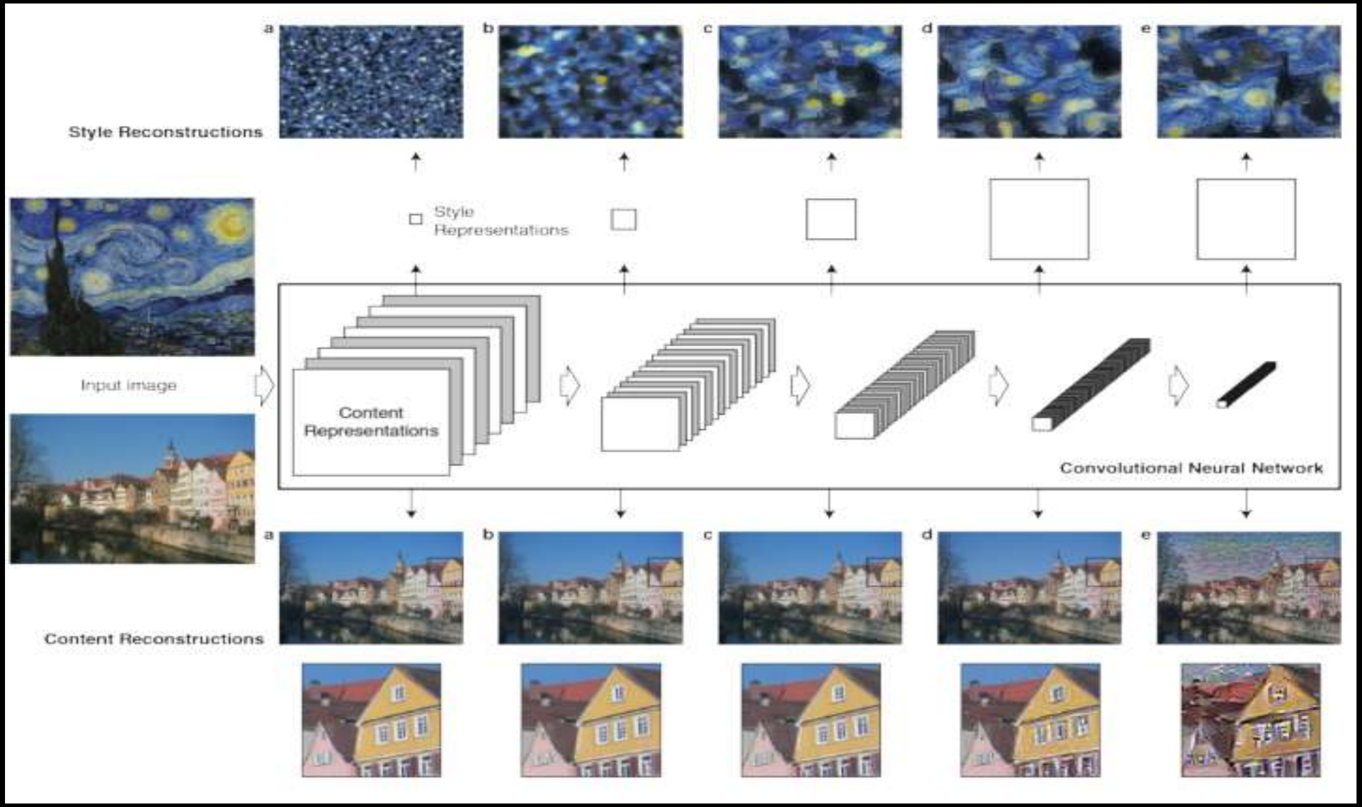
4. Fast Neural Style

MRF-Based



Freeman W T, Liu C. Markov random fields for super-resolution and texture synthesis[J]. Advances in Markov Random Fields for Vision and Image Processing, 2011, 1: 155-165.
Efros A A, Leung T K. Texture synthesis by non-parametric sampling[C]//Computer Vision, 1999. The Proceedings of the Seventh IEEE International Conference on. IEEE, 1999, 2: 1033-1038.

CNN-Based



Gatys L A, Ecker A S, Bethge M. A neural algorithm of artistic style[J]. arXiv preprint arXiv:1508.06576, 2015.

https://github.com/anishathalye/neural-style

https://github.com/jcjohnson/neural-style

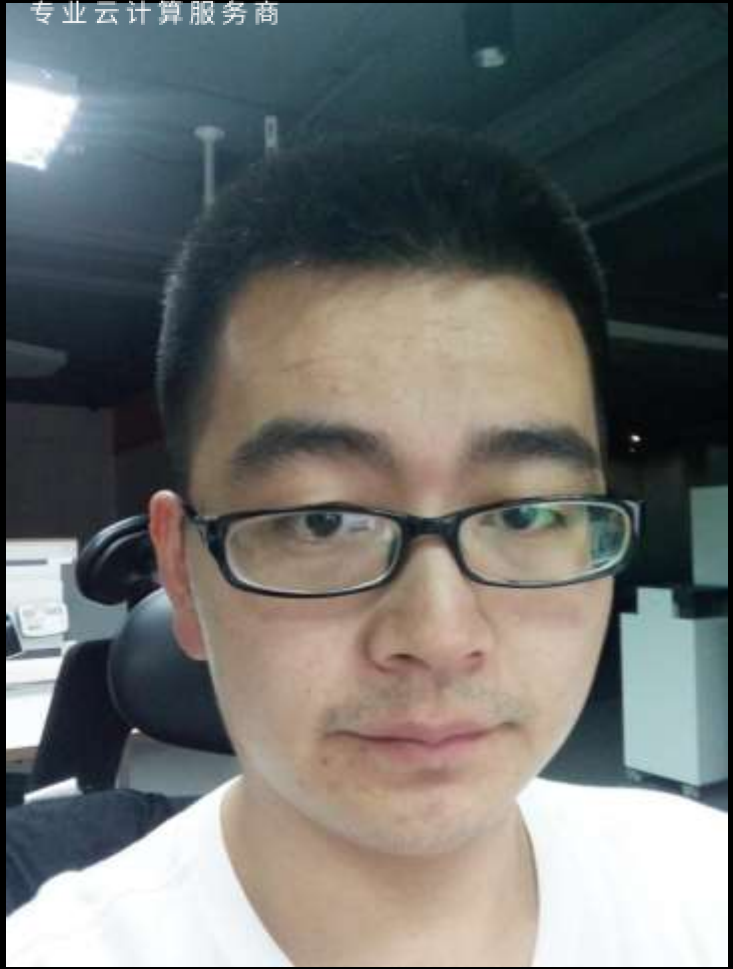
MRF And CNN-Based

$$
\mathbf{x} = \arg\min_{x} E_s(\Phi(\mathbf{x}), \Phi(\mathbf{x}_s)) + \alpha_1 E_c(\Phi(\mathbf{x}), \Phi(\mathbf{x}_c)) + \alpha_2 \Upsilon(\mathbf{x})
$$

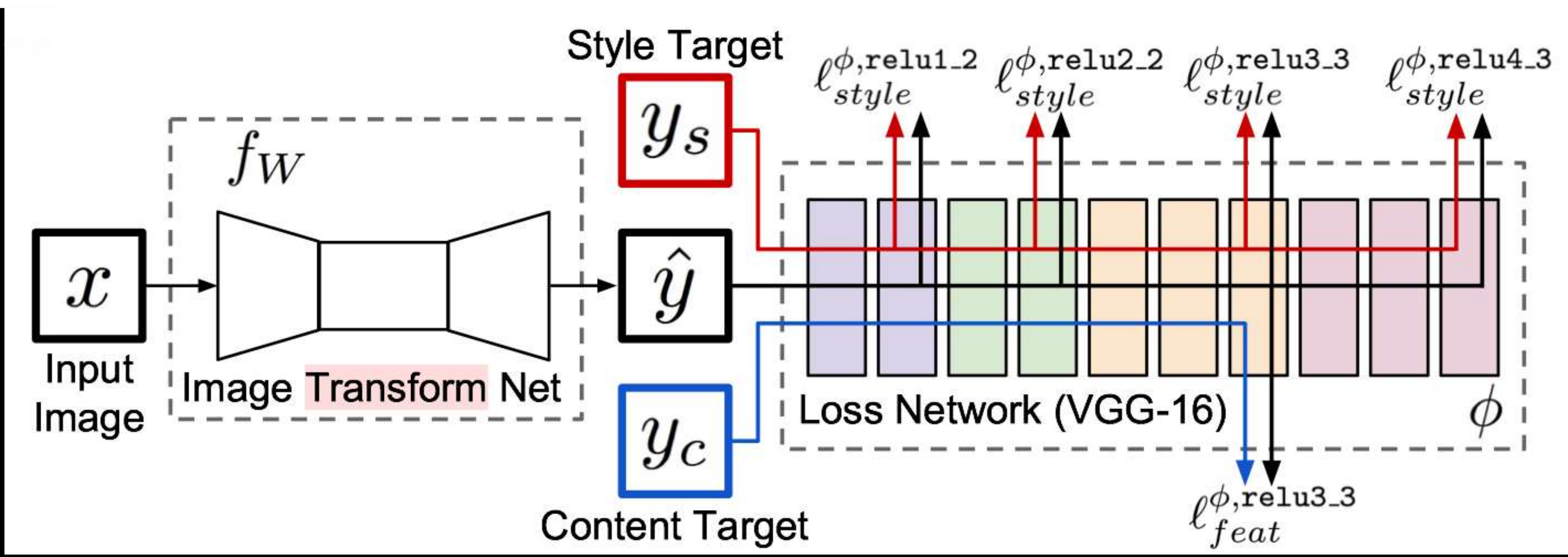
Li C, Wand M. Combining Markov Random Fields and Convolutional Neural Networks for Image Synthesis[J]. arXiv preprint arXiv:1601.04589, 2016
https://github.com/chuanli11/CNNMRF

MRF And CNN-Based

UCLOUD
专业云计算服务商



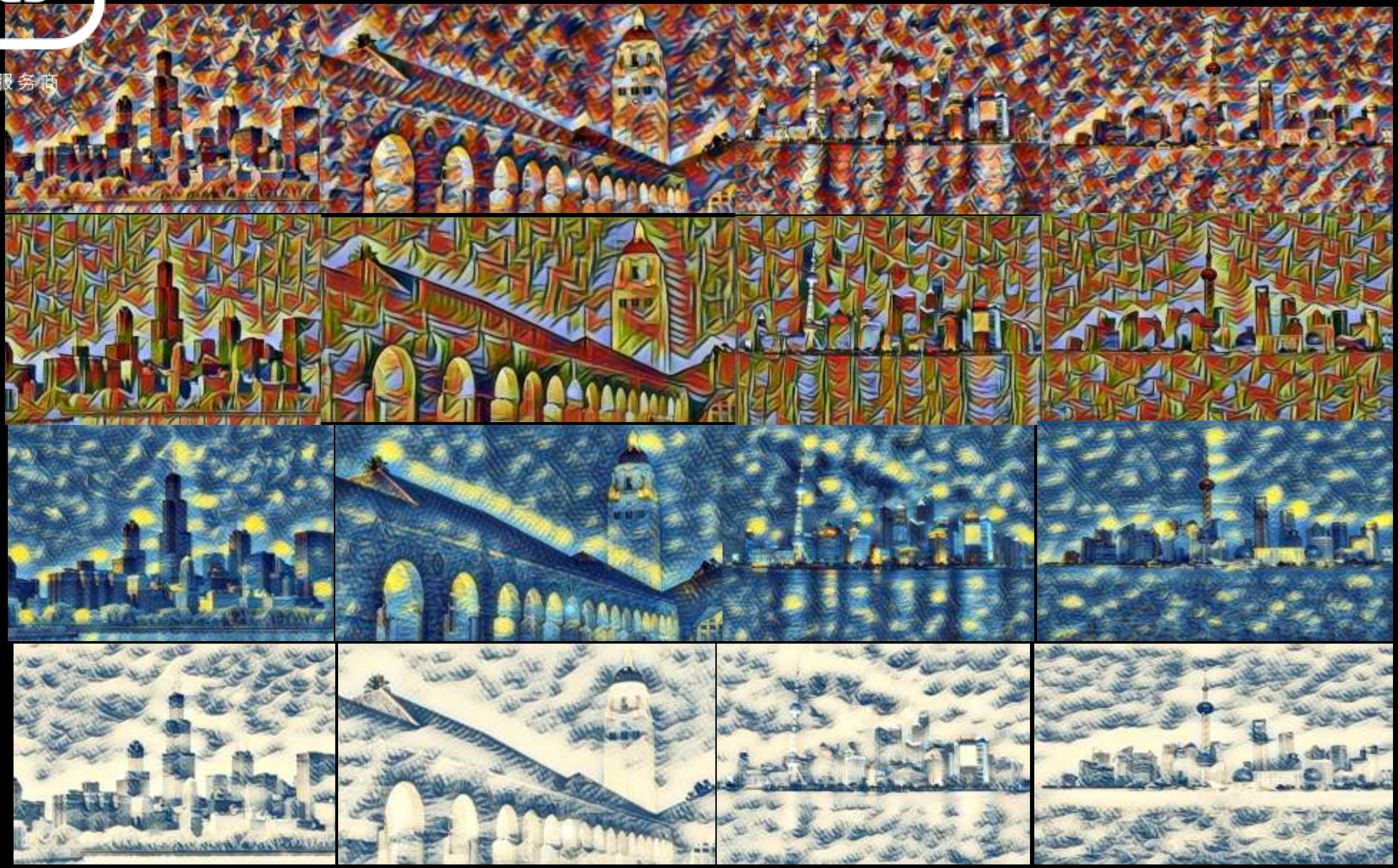
Fast Neural Style



Johnson J, Alahi A, Fei-Fei L. Perceptual losses for real-time style transfer and super-resolution[J]. arXiv preprint arXiv:1603.08155, 2016.

https://github.com/burness/neural_style_tensorflow/tree/master/fast_neural_style

Fast Neural Style



Text⇔Image

1.Text-to-Image

2.Image-to-Text

Text-to-Image

专业云计算服务商

Generative Adversarial Network

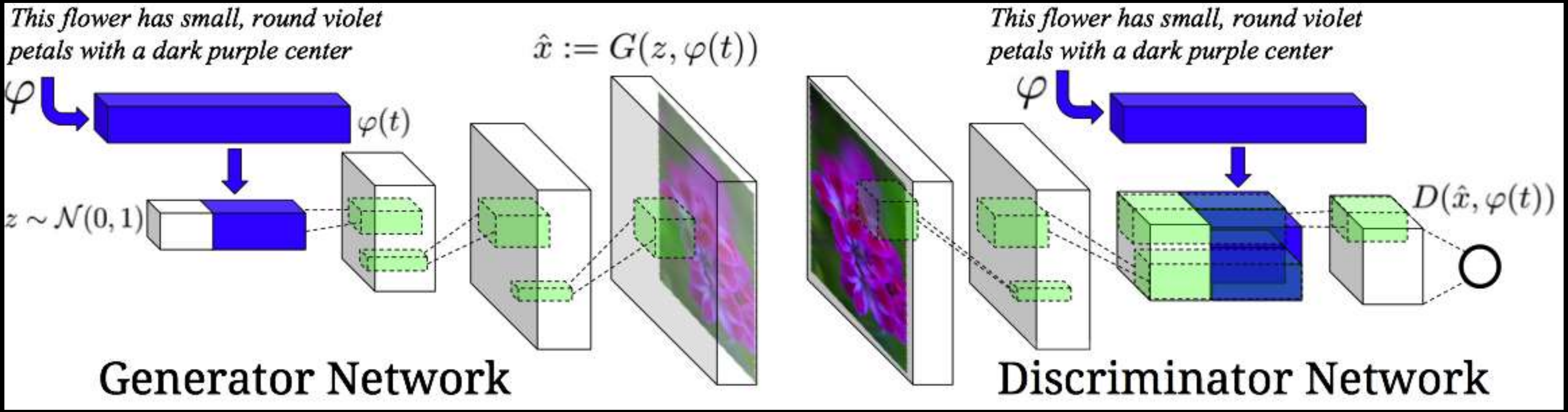
$$\min_G \max_D V(D,G) = \mathbb{E}_{x\sim p_{data}(x)}[\log D(x)] + \mathbb{E}_{x\sim p_z(z)}[\log(1 - D(G(z)))]$$

Generator G and a discriminator D compete in a two-player minimax game
G: fool the discriminator
D: Distinguish real training data from synthetic images

Reed S, Akata Z, Yan X, et al. Generative adversarial text to image synthesis[J]. arXiv preprint arXiv:1605.05396, 2016.
https://github.com/paarthneekhara/text-to-image
http://www.jianshu.com/p/db87c51de510

Text-to-Image

Generative Adversarial Network



G: $\mathbb{R}^Z \times \mathbb{R}^T \rightarrow \mathbb{R}^D$
D: $\mathbb{R}^T \times \mathbb{R}^D \rightarrow \{0,1\}$

Text-to-Image

Results

the flower shown has yellow anther red pistil and bright red petals

this flower has petals that are yellow, white and purple and has dark lines

the petals on this flower are white with a yellow center

this flower has a lot of small round pink petals

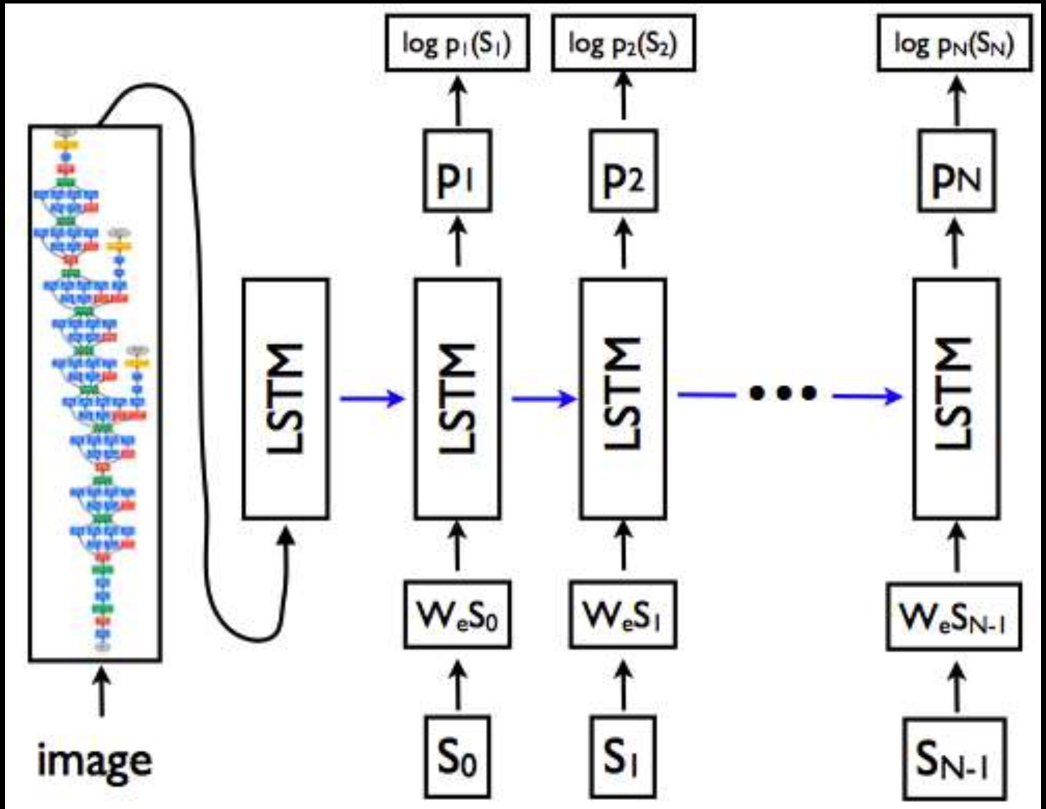
this flower is orange in color, and has petals that are ruffled and rounded

the flower has yellow petals and the center of it is brown

Image-to-Text

Vinyals O, Toshev A, Bengio S, et al. Show and Tell: Lessons learned from the 2015 MSCOCO Image Captioning Challenge[J]. 2016.
https://github.com/tensorflow/models/tree/master/im2txt
https://github.com/tensorflow/models/issues/480
https://github.com/tensorflow/models/pull/485/commits/c6a4f783080c5310ce0e3244daa31af57df12def

Image-to-Text



$$\theta^\star = \arg\max_\theta \sum_{(I,S)} \log p(S|I;\theta)$$

$$\log p(S|I) = \sum_{t=0}^{N} \log p(S_t|I, S_0, \ldots, S_{t-1})$$

$$\begin{aligned} x_{-1} &= \mathrm{CNN}(I) \\ x_t &= W_e S_t, \quad t \in \{0 \ldots N-1\} \\ p_{t+1} &= \mathrm{LSTM}(x_t), \quad t \in \{0 \ldots N-1\} \end{aligned}$$

$$L(I,S) = -\sum_{t=1}^{N} \log p_t(S_t)$$

Ioffe S, Szegedy C. Batch normalization: Accelerating deep network training by reducing internal covariate shift[J]. arXiv preprint arXiv:1502.03167, 2015.
Deep Visual-Semantic Alignments for Generating Image Descriptions
Andrej Karpathy, Li Fei-Fei


```
Captions for image 54.jpg:
  0) a group of people standing next to each other . (p=0.002586)
  1) a group of people standing in a room . (p=0.000496)
  2) a group of people standing next to each other in a room . (p=0.000119)
amax@am[redacted]ensorflow/models/im2txt$
```


```
Captions for image nba_jandunks_01.jpg:
  0) a group of young men playing a game of basketball . (p=0.005181)
  1) a group of men playing a game of basketball . (p=0.004607)
  2) a group of men playing basketball on a court (p=0.000636)
```

TFLearn Introduction

```
with tf.name_scope('conv1'):
    W = tf.Variable(tf.random_normal([5, 5, 1, 32]), dtype=tf.float32, name='Weights')
    b = tf.Variable(tf.random_normal([32]), dtype=tf.float32, name='biases')
    x = tf.nn.conv2d(x, W, strides=[1, 1, 1, 1], padding='SAME')
    x = tf.add_bias(W, b)
    x = tf.nn.relu(x)
```

```
tflearn.conv_2d(x, 32, 5, activation='relu', name='conv1')
```

TFLearn Example

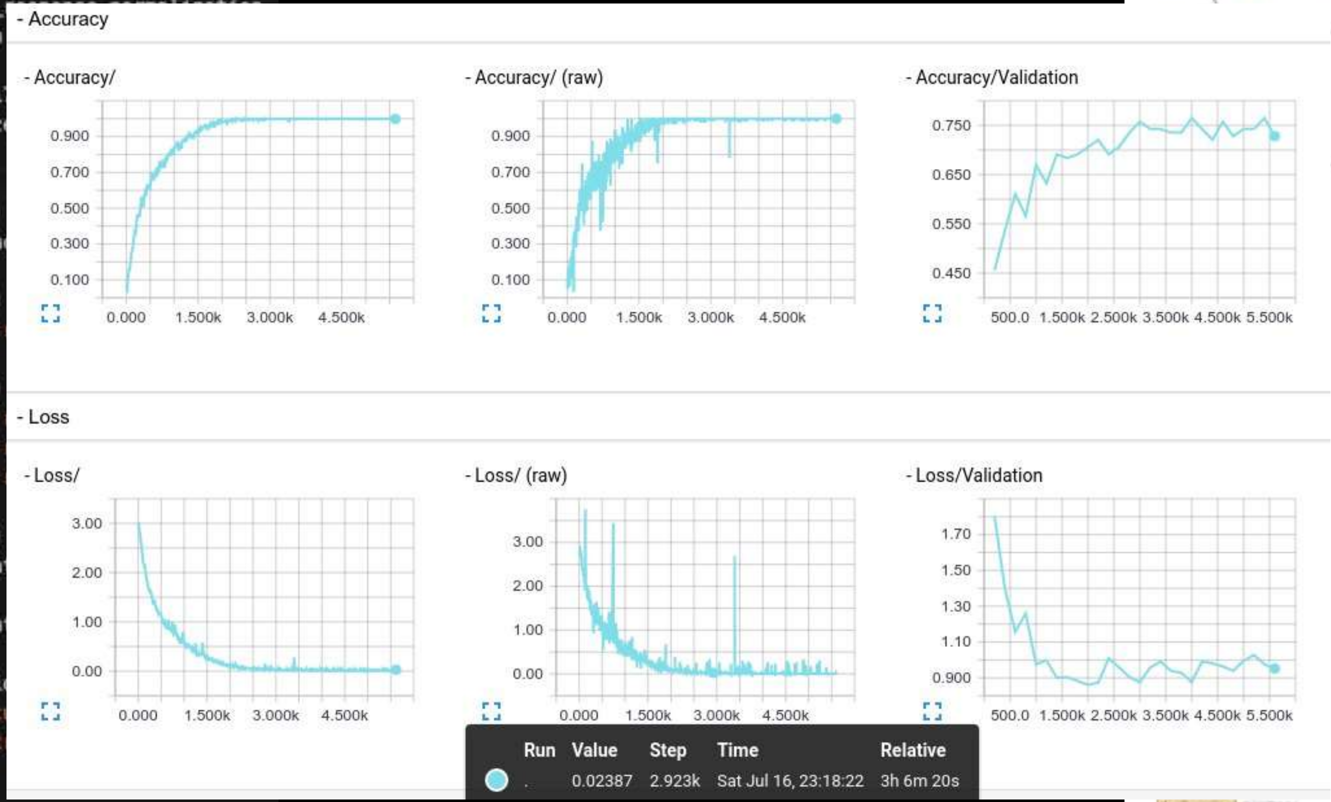
```
from __future__ import division, print_function, absolute_import

import tflearn
from tflearn.layers.core import input_data, dropout, fully_connected
from tflearn.layers.conv import conv_2d, max_pool_2d
from tflearn.layers.normalization import local_
from tflearn.layers.estimator import regression

import tflearn.datasets.oxflower17 as oxflower17
X, Y = oxflower17.load_data(one_hot=True, resiz

# Building 'AlexNet'
network = input_data(shape=[None, 227, 227, 3])
network = conv_2d(network, 96, 11, strides=4, a
network = max_pool_2d(network, 3, strides=2)
network = local_response_normalization(network)
network = conv_2d(network, 256, 5, activation='
network = max_pool_2d(network, 3, strides=2)
network = local_response_normalization(network)
network = conv_2d(network, 384, 3, activation='
network = conv_2d(network, 384, 3, activation='
network = conv_2d(network, 256, 3, activation='
network = max_pool_2d(network, 3, strides=2)
network = local_response_normalization(network)
network = fully_connected(network, 4096, activa
network = dropout(network, 0.5)
network = fully_connected(network, 4096, activa
network = dropout(network, 0.5)
network = fully_connected(network, 17, activati
network = regression(network, optimizer='moment
                     loss='categorical_crossent
                     learning_rate=0.001)

# Training
model = tflearn.DNN(network, checkpoint_path='model_alexnet',
                    max_checkpoints=1, tensorboard_verbose=2)
model.fit(X, Y, n_epoch=1000, validation_set=0.1, shuffle=True,
          show_metric=True, batch_size=64, snapshot_step=200,
          snapshot_epoch=False, run_id='alexnet_oxflowers17')
```



THANKS
&
QA

专业云计算服务商