# TensorFlow框架剖析与应用


王琛 @ 百纳新闻

# AlphaGo：人工智能战胜职业棋手

# Prisma

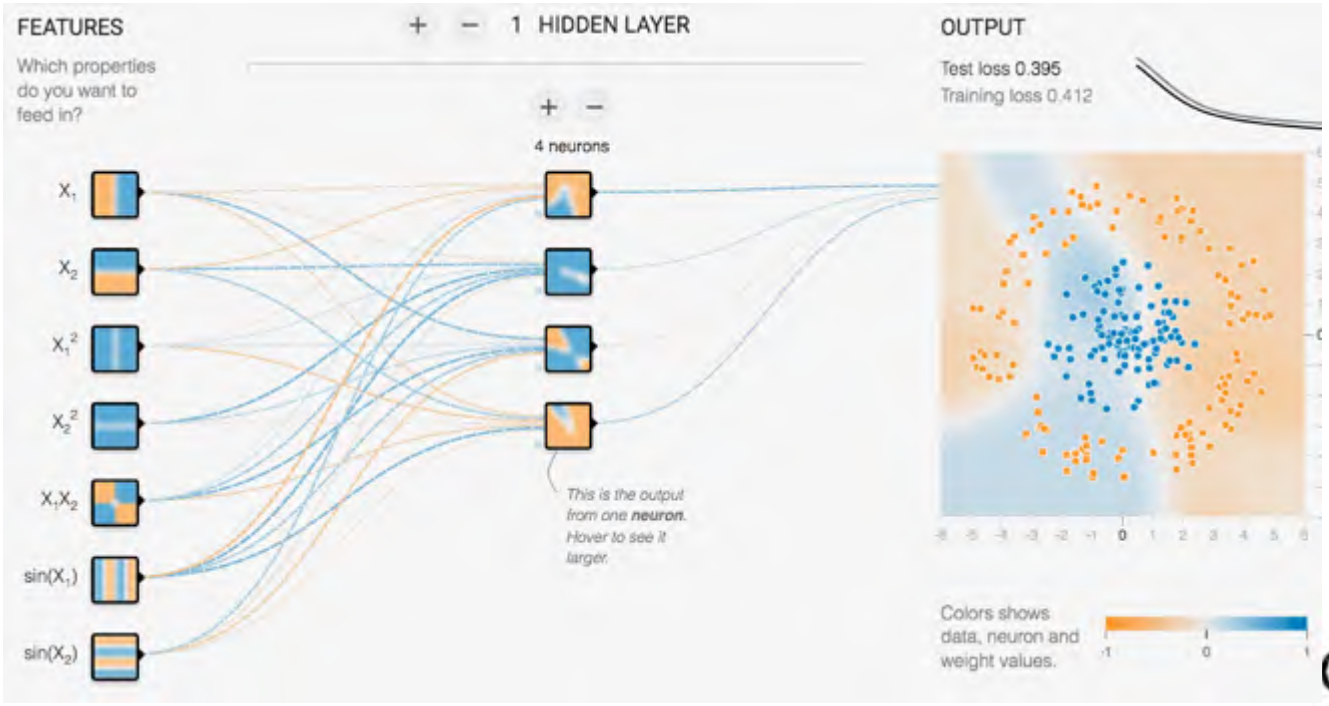
基于深度学习技术的修图APP，刷爆全球社交网络

# 更多人工智能应用

- 自动驾驶
- 人脸识别
- …

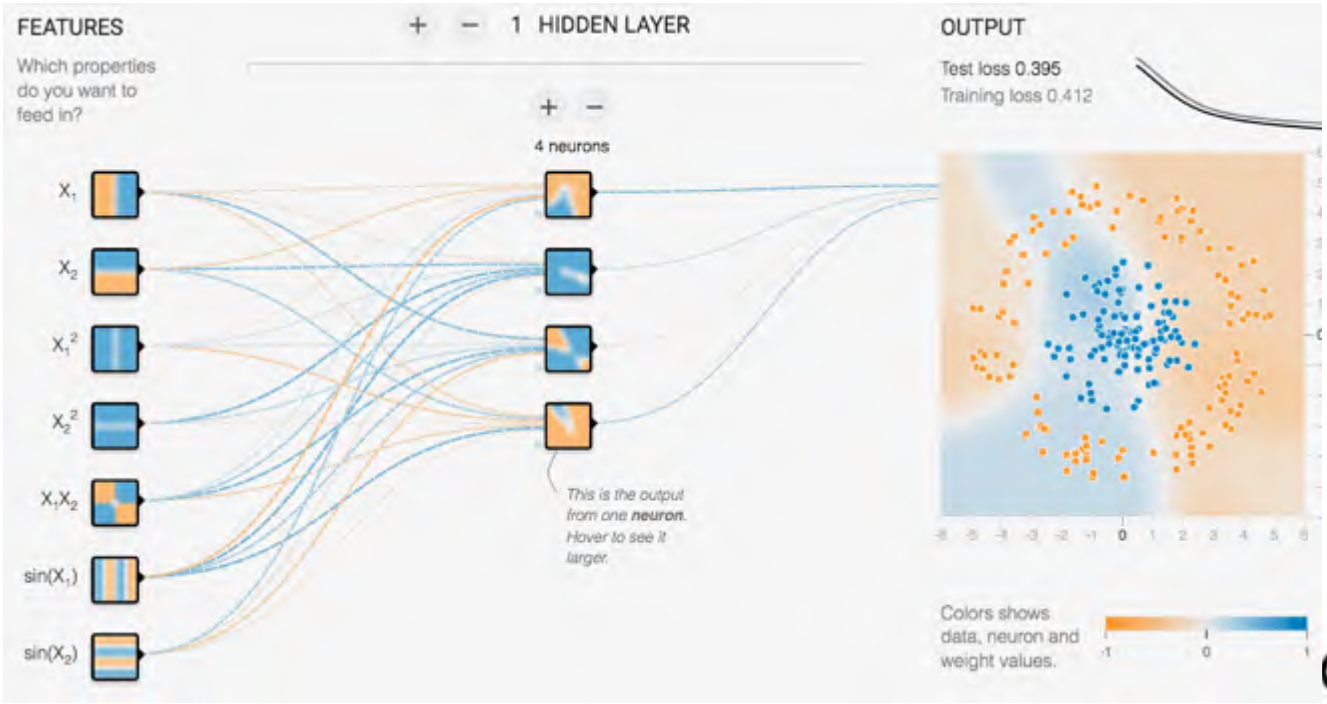
# 机器学习之神经网络

- 通过简单非线性函数的组合，可以拟合任意复杂的函数
- 边的权重决定节点的组合方式，不同的组合方式得到不同的模型逻辑



# 机器学习之神经网络

- 模型训练，BackPropagation算法
  - 通过事实数据不断矫正函数逻辑，使模型的输出符合数据分布
  - 代价函数：衡量函数输出于预期之间的差距



# 机器学习 in Python

- NumPy & SciPy
  - 为Python提供了数值计算与科学计算能力
- Caffe
  - 第一个主流的工业级深度学习框架
  - 灵活性较差，久无更新
- Theano
  - 灵活，支持递归网络
  - 全python实现，速度较慢，执行效率低

# 研究趋势：更复杂的模型 + 更多数据

- 算法模型越来越复杂
  - 2012年第一代深度学习图像识别模型 Al… …层神经元
  - 2014年ImageNet竞赛冠军模型 VGG19：19层神经元
  - 2015年ImageNet竞赛冠军模型 Deep Residual Networks：150 ～ 200层神经元
- 数据集越来越大
  - 1998年MNIST手写数字图片数据集：11MB，… …像素黑白图片
  - CIFAR-10图像识别数据集：160MB，6w张32x32像素彩色图片
  - 2016年ImageNet图像识别竞赛数据集：> 155GB，1400w+张大尺寸彩色图片

代码实现代价大

计算量大

# TensorFlow

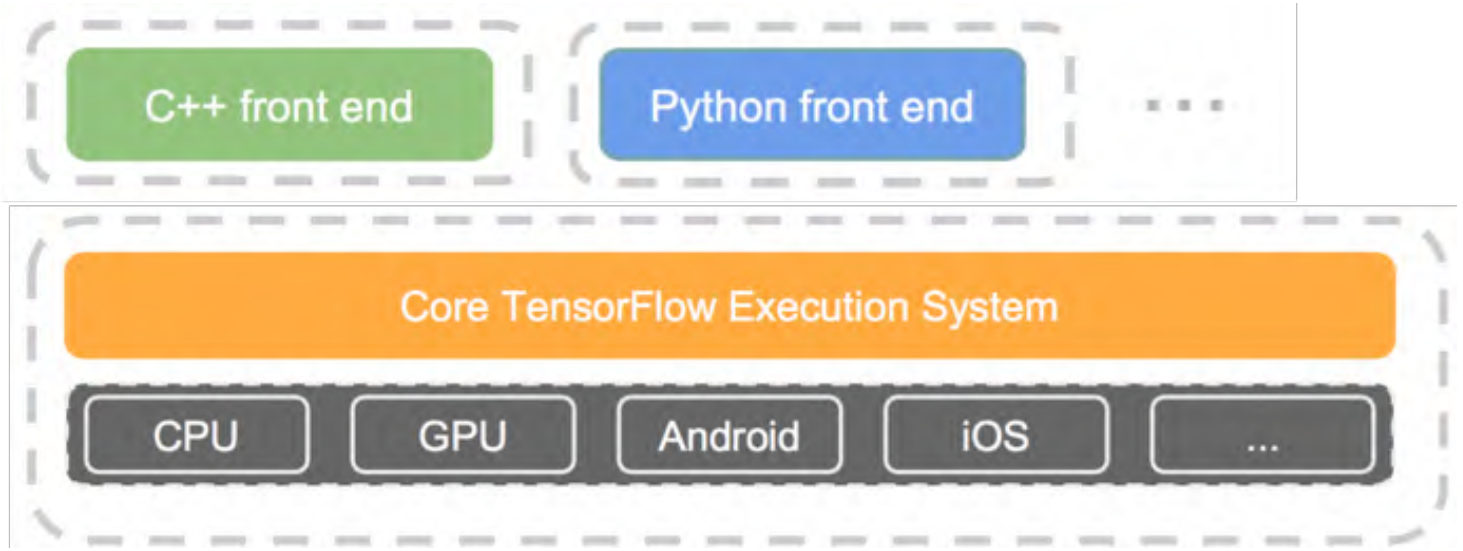
- Google Brain计划产物
- Gmail，Google Maps等1000多个应用
- 于2015年11月开源
  - Github 3w+ stars
- 架构师 Jeff Dean
  - 领导实现MapReduce、BigTable、Spanner

# TensorFlow的核心需求

- 表达能力，能用简洁的代码实现各种复杂算法
- 高性能、高扩展，通过分布式技术提高执行效率
- 可移植、可复现，能共享研究的结果和过程
- 产品化，能快速将研究结果应用到真实生产场景

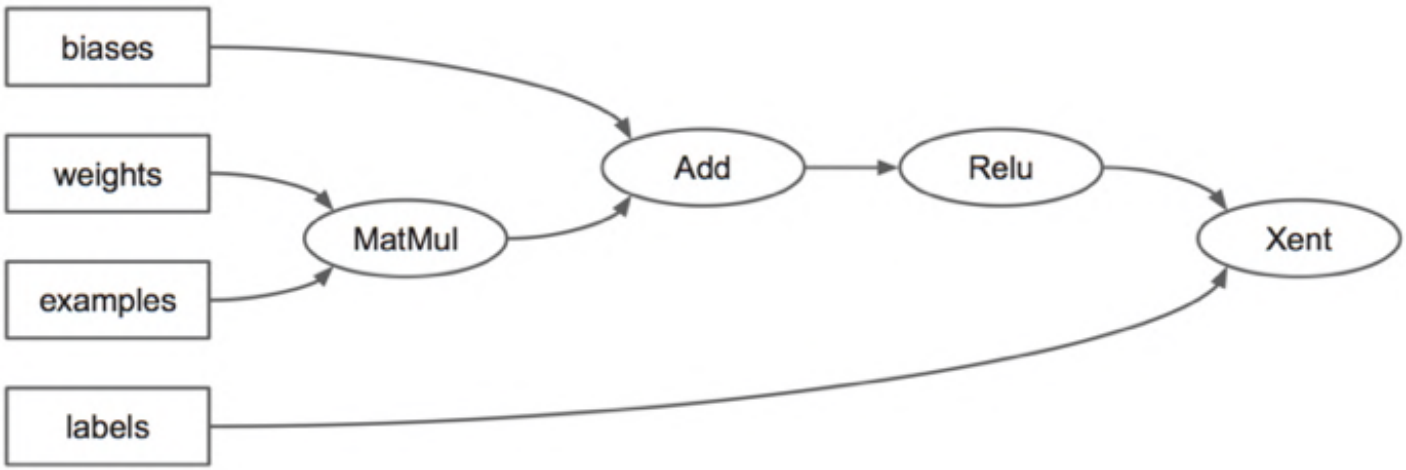
# 架构设计

- 核心执行系统，由C++编写 （相当于编译器 + 执行调度器）
- 嵌入在多种主语言中的前端编程语言 （相当于高级语言）
  - 前后端使用protobuf格式的中间语言交互
- 面向硬件设备的算法实现，由C++编写 （相当于机器指令集）



# 计算表示模型：Dataflow Graph

- 数据流图 or 计算图
  - 使用有向图来描述计算过程
  - 类似于表达式树，区别为计算图节点可以复用、可以有多个输出

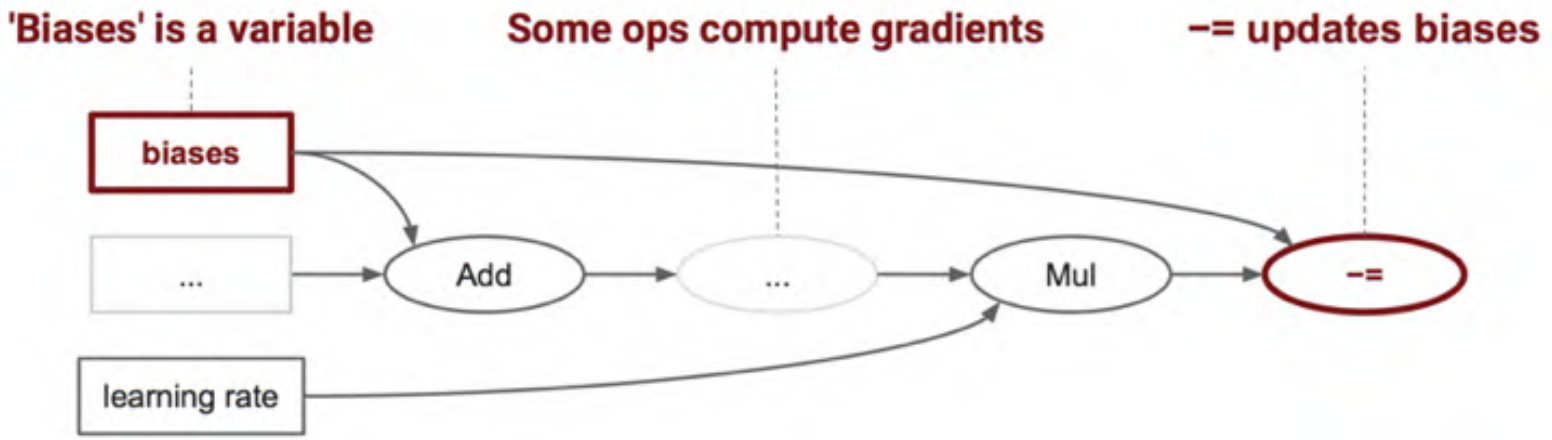


# 计算表示模型：Dataflow Graph

- **Tensor（数据）**：N维数组
  - 1维Tensor为Vector，2维Tensor即是Matrix
  - 图像一般由3维Tensor表示：行、列、RGB颜色
- **Operations（节点）**：算子，施加在Tensor上的计算操作
  - 数组操作算子：Split，Shuffle，Slice，Concat，…
  - 聚合计算算子：Reduce Sum，Reduce Max，Reduce Min，…
  - 矩阵计算算子：MatMul，Transpose，…
  - 数据流控制算子：Merge，Switch，…
  - 复杂算子：Conv2D，Pooling，…

# 计算表示模型：Dataflow Graph

- **Variable**：用于存储模型的权值
  - 每次正向计算完成之后，根据BP算法调整权值，代入下一轮计算



# 前端编程接口

普通函数

```
def add(a, b):
    return a + b

a = 1
b = 2
c = apply(add, (a, b))
```

=>

TensorFlow

```
mat_a = tf.placeholder(tf.float32, shape=[2, 2])
mat_b = tf.placeholder(tf.float32, shape=[2, 2])

add = tf.add(mat_a, mat_b)
```

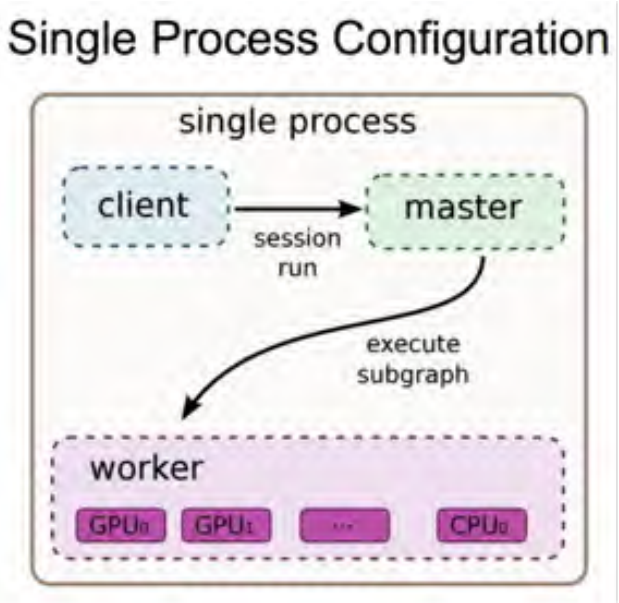
定义计算图

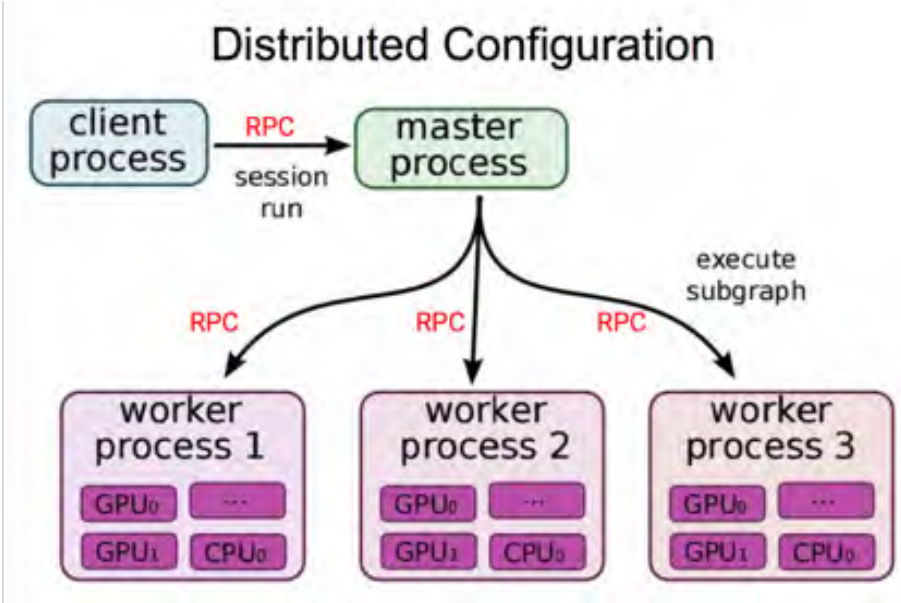
```
with tf.Session() as sess:
    feed_dict = {
        mat_a: [[1., 1., ], [1., 1., ]],
        mat_b: [[2., 2., ], [2., 2., ]],
    }
    mat_c = sess.run(add, feed_dict=feed_dict)
```

Feed数据并执行

# 分布式架构

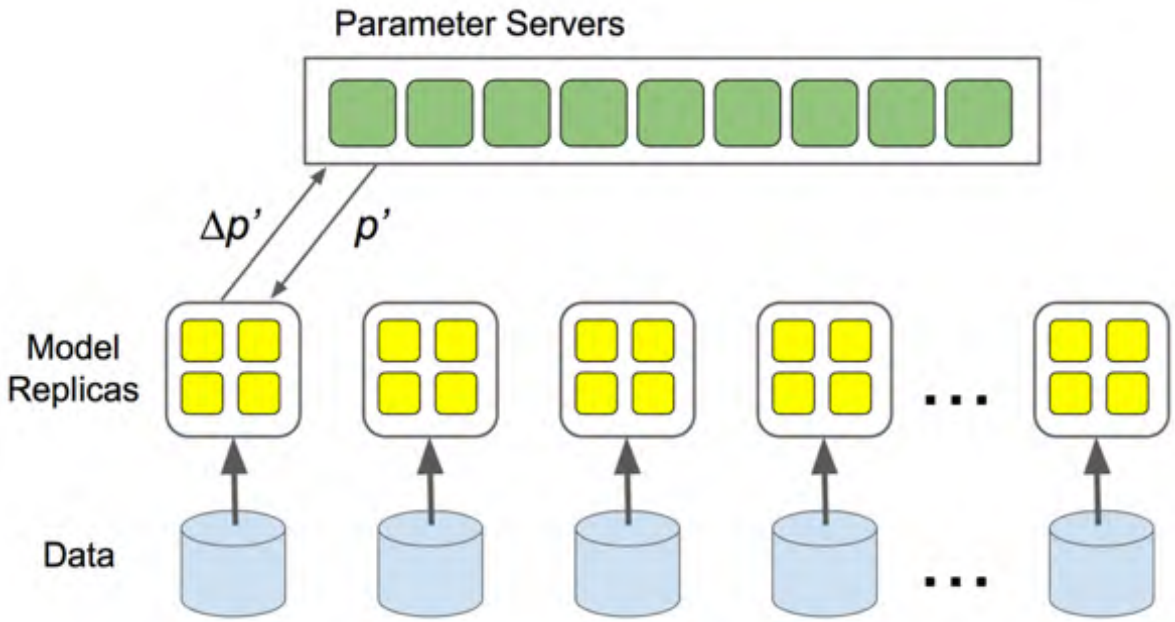
- 分布式是提升性能最重要的手段
  - 50 replica 带来 30~40x 性能提升
- Client -> Master -> Worker 结构，之间通过GRPC通讯



# 数据并行

- 数据并行是最主要的分布式方式
  - 将训练数据分成多个partition，每个replica只负责一个partition
  - 权值更新被上传到中心化的Parameter Server上，由Parameter Server统一更新并分发



# 社区第三方贡献

- 更高层API封装
  - TFLearn，SKFlow，TF-Slim更易用、更易懂的API
  - 与原生TensorFlow无缝兼容
- 更多算法模型，更多算子
- 丰富强大的辅助函数库
  - 用于辅助构建数据集

```
import tflearn

# Logical NOT operator
X = [[0.], [1.]]
Y = [[1.], [0.]]

# Define dataflow graph
graph = tflearn.input_data(shape=[None, 1])
graph = tflearn.fully_connected(graph, 128,
                                activation='linear')
graph = tflearn.fully_connected(graph, 1,
                                activation='linear')
graph = tflearn.regression(graph, optimizer='sgd',
                           learning_rate=0.1,
                           loss='mean_square')

# Train model
model = tflearn.DNN(graph)
model.fit(X, Y, n_epoch=100, snapshot_epoch=False)

# Test trained model
print("Testing NOT operator")
print("NOT 0:", model.predict([[0.]]))
print("NOT 1:", model.predict([[1.]]))
```

```
Training Step: 100  | total loss: 0.00085
| SGD | epoch: 100 | loss: 0.00085 -- iter: 2/2
Testing NOT operator
('NOT 0:', [[0.9951861500740051]])
('NOT 1:', [[0.00324314832687377 93]])
```

10行代码完成三层神经网络

# 图片风格化应用


+

Content Image

Style Image

# 图片风格化应用

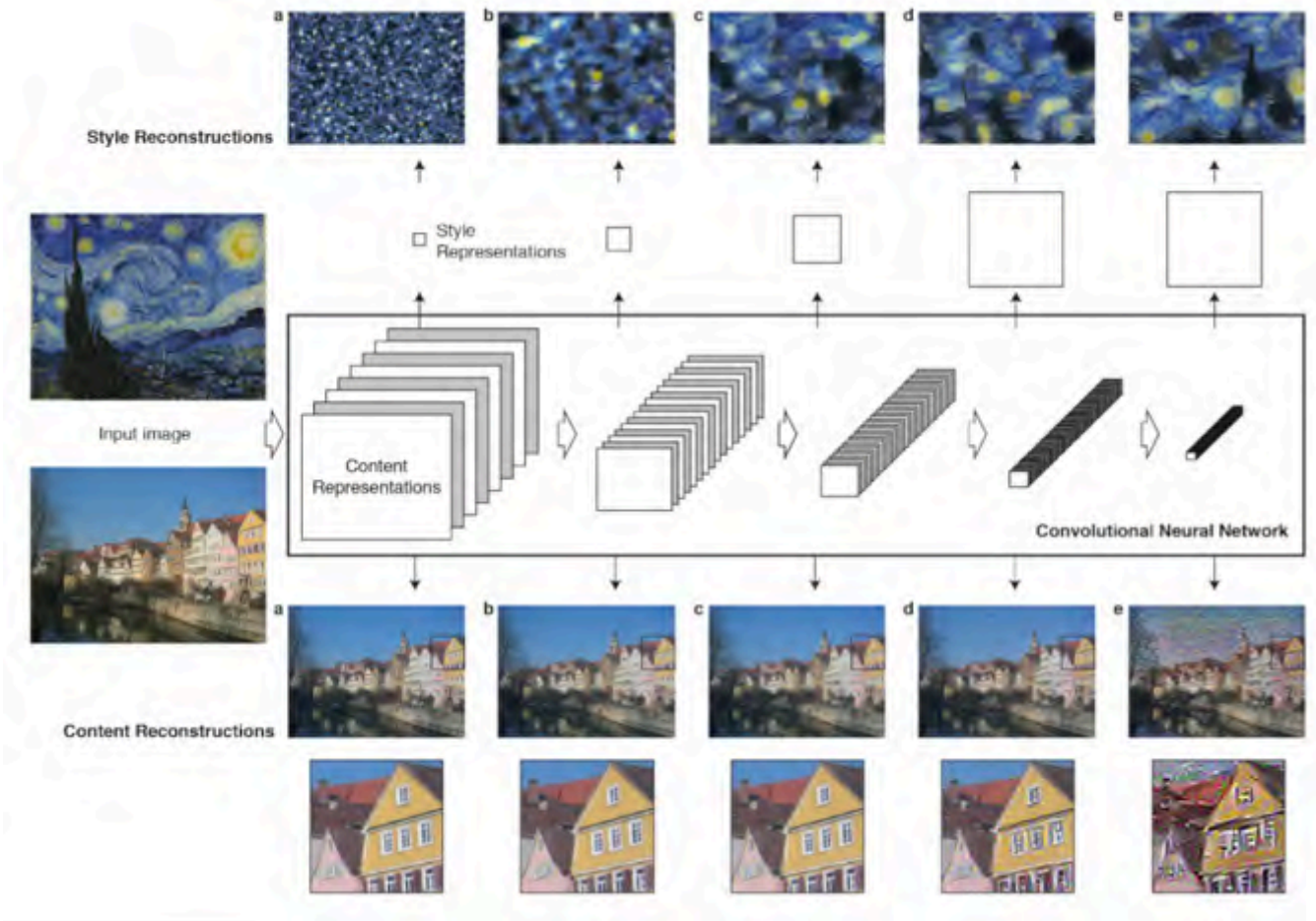

Our Work


Prisma

# 图片风格化应用

- 理论基础：A Neural Algorithm of Artistic Style
  - 利用深度卷积神经网络VGG19，提取图片中的线条和颜色特征，部分特征代表画风，部分特征代表内容，将图片表示为特征域上的向量
  - 通过训练，将一张噪点图调整为：画风维度的向量值趋近于风格图片，内容维度的向量值趋近于内容图片

# 图片风格化应用

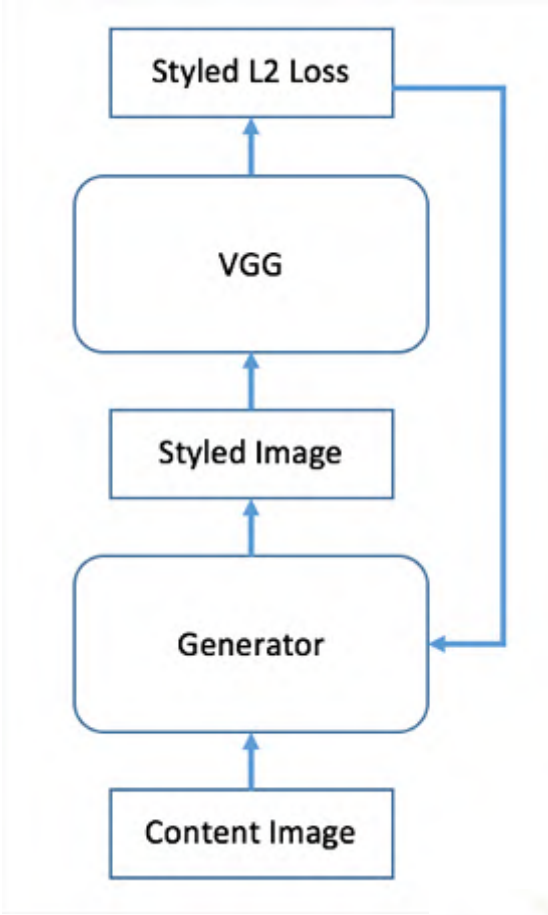

a
b
c
d
e
Style Reconstructions
Style Representations
Input image
Content Representations
Convolutional Neural Network
Content Reconstructions

# 图片风格化应用

- 理论基础：A Neural Algorithm of Artistic Style
  - TensorFlow版开源实现（仅350行代码）：https://github.com/anishathalye/neural-style
  - 问题：处理速度慢，即使使用GPU(K40)渲染一张图片需要 >1min 的时间，无法大规模应用

# 图片风格化应用

- 速度的关键：Generative Model
  - 通过训练得到每一张风格图片相应的生成器
  - 将生成器作为滤镜，输入为内容图输出为带有风格的新图
    - 可理解为对原图的每个像素进行函数变换



# 图片风格化应用

```
def net(input_image):
    ratios = [32, 16, 8, 4, 2, 1]
    conv_num = 8
    network = []
    for i in range(len(ratios)):
        network.append(avg_pool_2d(input_image, ratios[i]))
        for block in range(3):
            with tf.name_scope('block_%d_%d' % (i, block)):
                filter_size = 1 if (block + 1) % 3 == 0 else 3
                network[i] = tflearn.conv_2d(network[i], nb_filter=conv_num, filter_size=filter_size,
                                             weights_init='xavier', name='Conv_%d_%d' % (i, block))
                network[i] = tf.nn.batch_normalization(network[i], 0, 1.0, 0.0, 1.0, 1e-5, 'BatchNorm')
                network[i] = tflearn.activations.leaky_relu(network[i])

        if i == 0:
            network[i] = tflearn.upsample_2d(network[i], 2)
        else:
            upnet = tf.nn.batch_normalization(network[i - 1], 0, 1.0, 0.0, 1.0, 1e-5, 'BatchNorm')
            downnet = tf.nn.batch_normalization(network[i], 0, 1.0, 0.0, 1.0, 1e-5, 'BatchNorm')
            network[i] = tflearn.merge([upnet, downnet], 'concat', axis=3)

            for block in range(3, 6):
                with tf.name_scope('block_%d_%d' % (i, block)):
                    filter_size = 1 if (block + 1) % 3 == 0 else 3
                    network[i] = tflearn.conv_2d(network[i], nb_filter=conv_num * i, filter_size=filter_size,
                                                 weights_init='xavier', name='Conv_%d_%d' % (i, block))
                    network[i] = tf.nn.batch_normalization(network[i], 0, 1.0, 0.0, 1.0, 1e-5, 'BatchNorm')
                    network[i] = tflearn.activations.leaky_relu(network[i])

            if i != len(ratios) - 1:
                network[i] = tflearn.upsample_2d(network[i], 2)

    network[len(ratios) - 1] = tflearn.conv_2d(network[len(ratios) - 1], nb_filter=3, filter_size=1,
                                               weights_init='xavier', name='Conv2d_out')
    return network[len(ratios) - 1]
```

# 图片风格化应用

- 实践经验
  - 训练数据集：2000张随机图片（人像、动物、风景）
  - 单图训练时间：2～4 Hours，on Nvidia Geforce Titan X
  - 滤镜渲染时间：100ms / pic，on i7 CPU
  - 问题：
    - 生成器仅能表现一部分风格，大部分风格学习不到或效果不好
    - 不同特征维度对生成图像会产生影响，但影响趋势和原因尚不了解

# Thanks!

Questions?