

## 《零基础学编程——Python》期末小组项目报告

项目名称	基于 TENSORFLOW 的图像快速风格转移		
组长	雷鑫	联系方式	18469143209
成员姓名	专业	学号	年级
赵赫	信息安全	20161120028	2016 级
林俊	信息安全	20171130126	2017 级
雷鑫	数字媒体技术	20171120117	2017 级
郭子尧	数字媒体技术	20171120078	2017 级
何俊洁	物理学	20171050157	2017 级
王雪松	应用物理学	20171050076	2017 级

# 基于 TENSORFLOW 的图像快速风格转移

## 一、成员介绍

1. 赵赫 2016 级 软件学院信息安全专业
2. 雷鑫 2017 级 软件学院数字技术媒体专业
3. 郭子尧 2017 级 软件学院数字技术媒体专业
4. 林俊 2017 级 软件学院信息安全专业
5. 何俊洁 2017 级 物理与天文学院物理学
6. 王雪松 2017 级 物理与天文学院应用物理学

## 二、作品简介

本项目基于 GitHub 上开源代码，将使用艺术风格的神经网络算法可以将图像的内容和图像的自然风格分离并保存，基于 TensorFlow 在高层次的机器学习计算中具备更好的灵活性和可延展性，并使用 NVIDIA CUDA 通用并行计算架构使 GPU 能够解决复杂的计算问题。本项目预期在实现基于风格迁移的基础上实现视频的整体风格转移，并且能够改进算法得到高质量的新图像，将大量众所周知的艺术作品和任意图像结合起来，从而得到较为有意义的艺术作品并加以实际运用。

## 三、总体设计

- a) 基本思路：
1. 完成基于 NVIDIA CUDA 加速的环境搭建并配置库
  2. 在完成目标一的基础上实现基于 CUDA 加速的图片是视频风格迁移
  3. 在完成以上两个目标的基础上实现较为友好的用户交互界面，并且对算法进行优化和改进

- b) 主要技术难点及解决方案：

主要技术难点在于环境的搭建以及自定义转移风格时参数的调整，在自定义风格化图片时，由于自身设备的限制，处理速度较慢。

实验的解决方案如下：

1. 环境基础

- 1) 硬件环境

设备一：

架构	x86 架构
CPU 主频	2.7HZ
内存	8GB
硬盘	128G SSD+1024G HDD 硬盘
显卡	GTX 1050M

设备二：

架构	x86 架构
CPU 主频	2.4HZ
内存	8GB
硬盘	512G SSD+1024G HDD 硬盘
显卡	GTX 960M

设备三：

架构	x86 架构
----	--------

CPU 主频	2.6HZ
内存	8GB
硬盘	128G SSD+1024G HDD 硬盘
显卡	GTX 1050M

设备四：

架构	x86 架构
CPU 主频	3.0HZ
内存	16GB
硬盘	1024G SSD 硬盘
显卡	Quadro M620

## 2) 操作系统

本次实验所使用系统环境为：Ubuntu 17.10（KDE 桌面环境版）

配置运行环境前所需要的系统环境包括：

CUDA9.0 + CUDNN7.0.5 + tensorflow1.8.0 + python2.7（Anaconda2）

预处理环境配置如下：

```
sudo apt-get update -y
```

```
sudo apt-get install -y gcc g++ gfortran git wget
```

```
sudo apt-get upgrade -y
```

## 3) 第三方库配置

TensorFlow 是本项目运行必需的 Python 第三方库，为了保持项目所需环境的独立，本项目使用 Anaconda 管理 Python 环境。环境配置过程如下：

### 1. 下载 anaconda

anaconda 官网（<https://www.anaconda.com/download/>）下载 anaconda，版本为 python2.7，操作系统为 Linux。

### 2. 安装 anaconda2

到下载目录下，执行 `bash Anaconda2-5.2.0-Linux-x86_64.sh`

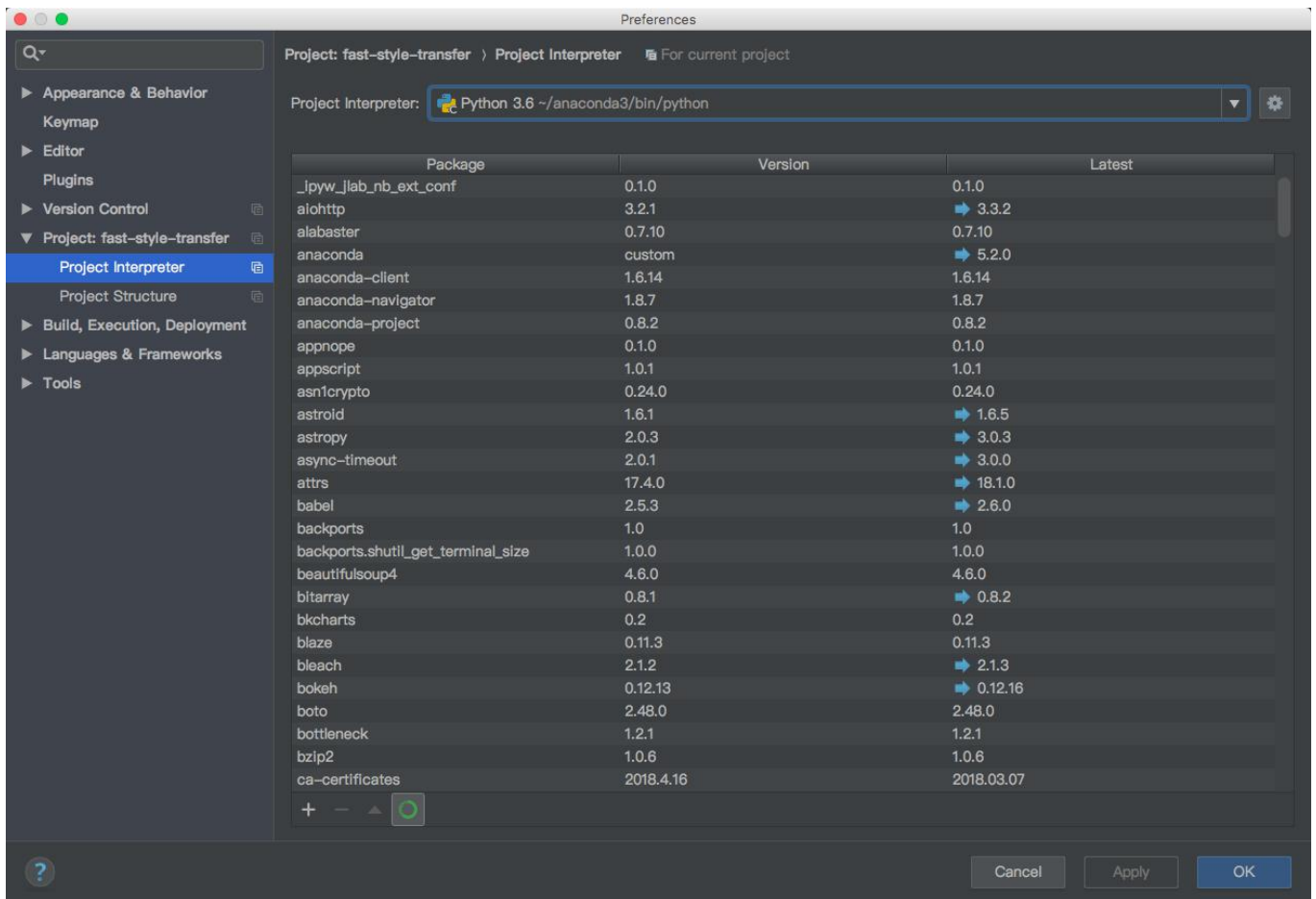
执行后选择默认安装，在“是否添加到环境配置”时，选择是添加 Anaconda 到系统环境变量。

### 3. 使用 anaconda+pycharm 安装项目需要的第三方包

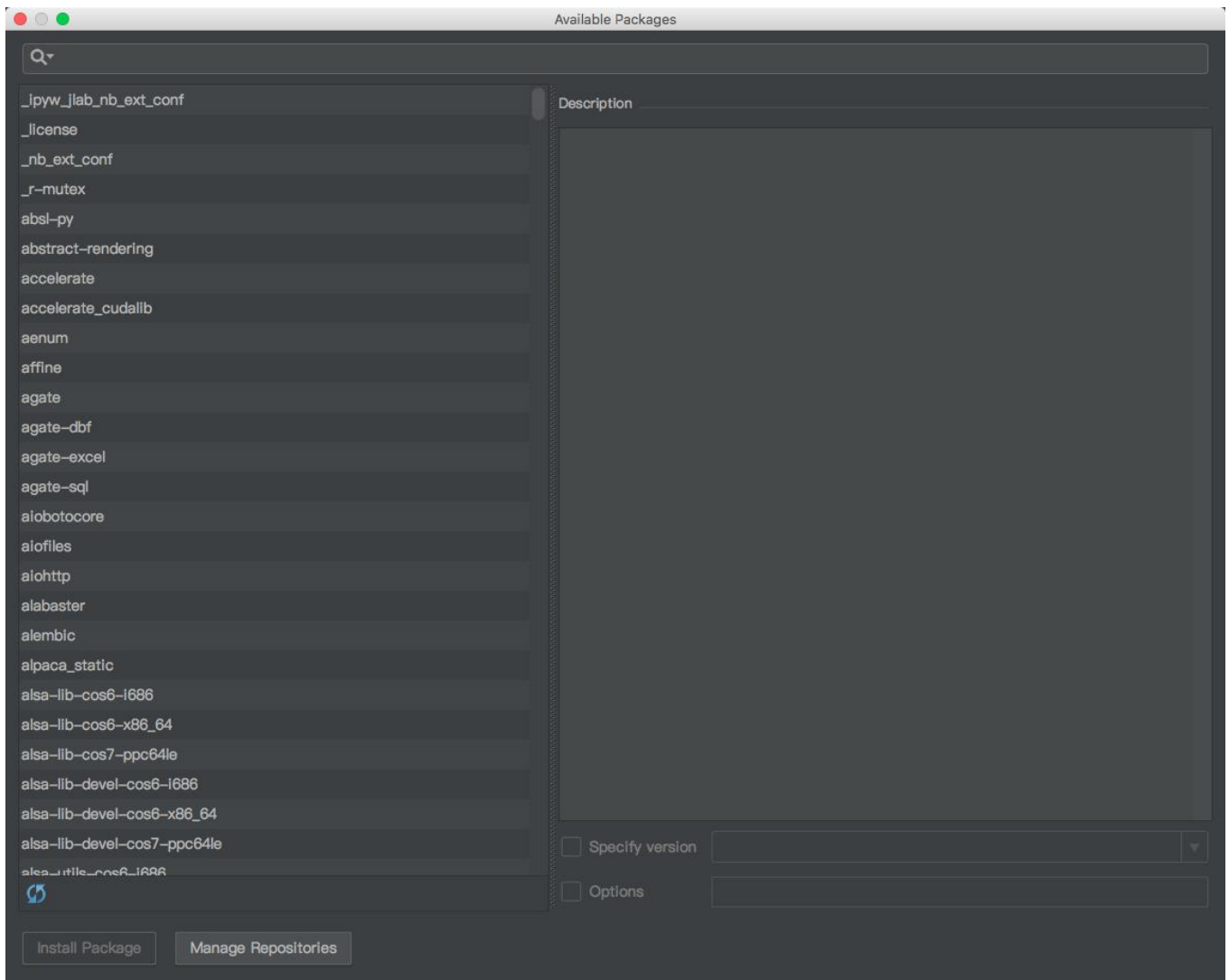
需要的依赖：pillow5.1.0、scipy1.1.0、numpy1.14.3、ffmpeg4.0、tensorflow1.8.0

### 4. 安装所需第三方库

进入 pycharm→file→setting→Project Interpreter



点击加号后在搜索栏输入需要的第三方库，选择安装即可。



## 2. Nvidia GPU 支持 (CUDA)

### 1) 驱动

在 ubuntu 系统下可以通过系统设置中的驱动管理器直接修改 ubuntu 使用的显卡。即  
系统设置->驱动管理器->选择相应显卡驱动

验证显卡安装情况

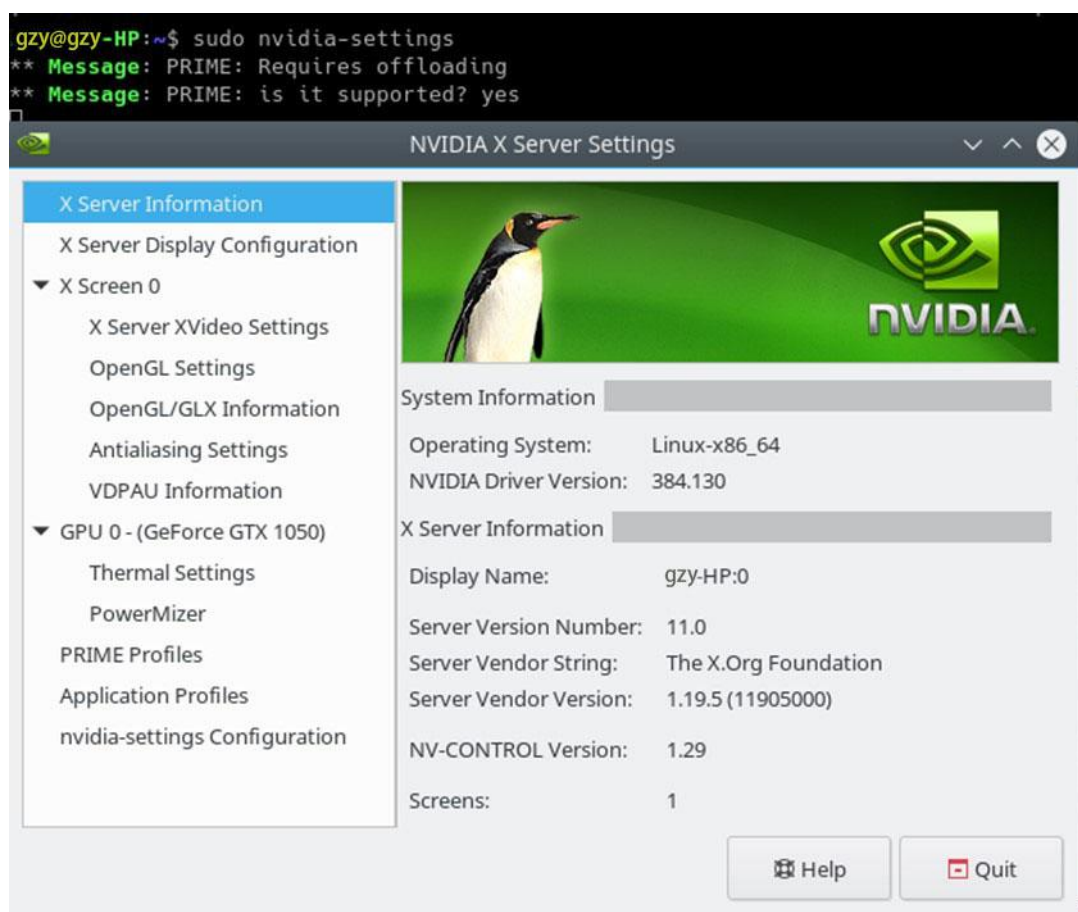
执行下面命令：



sudo nvidia-smi

sudo nvidia-settings

据图显示显卡已安装成功



2) CUDA





在 CUDA 资源站 ( <https://developer.nvidia.com/cuda-90-download-archive> ) 下载 CUDA9.0 版本:

### Select Target Platform ⓘ

Click on the green buttons that describe your target platform. Only supported platforms will be shown.

Operating System	Windows	Linux	Mac OSX			
Architecture ⓘ	x86_64	ppc64le				
Distribution	Fedora	OpenSUSE	RHEL	CentOS	SLES	Ubuntu
Version	17.04	16.04				
Installer Type ⓘ	runfile (local)	deb (local)	deb (network)			

在官方下载页面，给出了在下载完成后需要执行的相关操作：  
按下回车键，直到出现提示'是否为 NVIDIA 安装驱动'选择否，其他默认

<b>&gt; Base Installer</b>	<b>Download (1.6 GB)</b> 
Installation Instructions: 1. Run `sudo sh cuda_9.0.176_384.81_linux.run` 2. Follow the command-line prompts	
<b>&gt; Patch 1 (Released Jan 25, 2018)</b>	<b>Download (113.8 MB)</b> 
cuBLAS Patch Update: This update to CUDA 9.0 includes new GEMM kernels optimized for the Volta architecture and improved heuristics to select GEMM kernels for given input sizes.	
<b>&gt; Patch 2 (Released Mar 5, 2018)</b>	<b>Download (70.5 MB)</b> 
cuBLAS Patch Update: This update to CUDA 9 includes GEMM heuristics improvements to select the most optimized algorithms for input sizes commonly used in Deep Learning RNNs. The update also includes other bug-fixes and performance enhancements.	
<b>&gt; Patch 3 (Released Jun 7, 2018)</b>	<b>Download (74.7 MB)</b> 
cuBLAS 9.0 Update 4 [public patch] to address Convolutional Seq2Seq and RNN inference performance	

```
Do you accept the previously read EULA?
accept/decline/quit: accept

Install NVIDIA Accelerated Graphics Driver for Linux x86_64 384.81
(=====
= Summary =
I=====
(
(
Driver:      Not Selected
Toolkit:    Installed in /usr/local/cuda-9.1
E| Samples:  Installed in /home/arthuo, but missing recommended libraries

Please make sure that
D| - PATH includes /usr/local/cuda-9.1/bin
(  - LD_LIBRARY_PATH includes /usr/local/cuda-9.1/lib64, or, add /usr/local/cu

I| To uninstall the CUDA Toolkit, run the uninstall script in /usr/local/cuda-9.1/
(
(
Please see CUDA_Installation_Guide_Linux.pdf in /usr/local/cuda-9.1/doc/pdf for

E| ***WARNING: Incomplete installation! This installation did not install the CUDA
To install the driver using this installer, run the following command, replacing
sudo <CudaInstaller>.run -silent -driver

I|
Logfile is /tmp/cuda_install.1996.log
```

安装完成后发现出现警告，安装成功但缺少一些库。

执行下列命令安装缺失库

```
sudo apt-get install freeglut3-dev build-essential libx11-dev \
libxmu-dev libxi-dev libgl1-mesa-glx\
```

libglu1-mesa ibglu1-mesa-dev -y

验证 cuda 安装结果

重启电脑，如果电脑重启后能够登录成功，说明 CUDA 安装成功，检查 Device Node Verification

输入命令：ls /dev/nvidia\*，如下图所示则环境正常

```
gzy@gzy-HP:~$ ls /dev/nvidia*  
/dev/nvidia0 /dev/nvidiactl /dev/nvidia-modeset /dev/nvidia-uvm
```

### 3) 环境变量配置

输入命令：sudo kate ~/.bashrc 在该文件末尾添加 cuda 的路径，保存文件

export PATH=/usr/local/cuda-9.0/bin:\$PATH \$

export LD\_LIBRARY\_PATH=/usr/local/cuda-9.0/lib64:\$LD\_LIBRARY\_PATH

### 4) cuDNN

下载 cudnn7.0.5

cuDNN 的安装是建立在我们成功安装 cuda 的基础上的，cuDNN 同样需要我们去 NVIDIA 的官网下载适合 cuda 版本的 deb 文件或 tgz 文件。本次环境搭建使用的是.tgz 文件（下载 cudnn7.0.5 链接：  
<https://developer.nvidia.com/rdp/cudnn-download#a-collapse705-9>）

解压文件

tar -xzf cudnn-9.0-linux-x64-v7.tgz

复制文件到 cuda 下

sudo cp cuda/include/cudnn.h /usr/local/cuda/include

sudo cp cuda/lib64/libcudnn\* /usr/local/cuda/lib64

sudo chmod a+r /usr/local/cuda/include/cudnn.h /usr/local/cuda/lib64/libcudnn\*

算法分析与实现

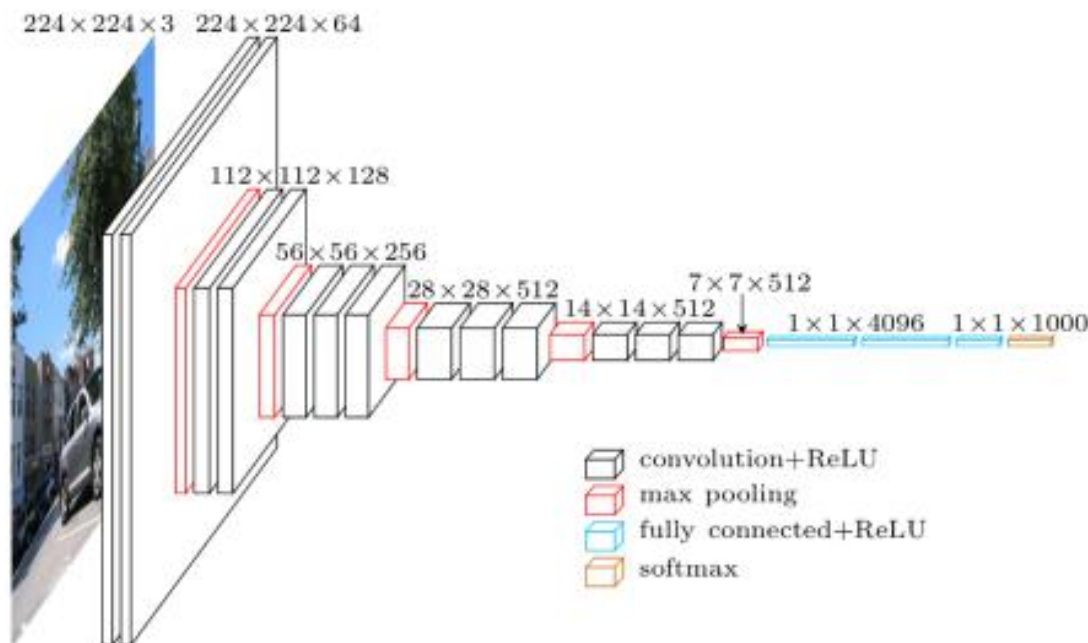


# 1. 算法原理分析

Simonyan 和 Zisserman 在其 2014 年的论文“用于大规模图像识别的非常深度卷积网络”中介绍了 VGG 网络架构。

ConvNet Configuration					
A	A-LRN	B	C	D	E
11 weight layers	11 weight layers	13 weight layers	16 weight layers	16 weight layers	19 weight layers
input (224 × 224 RGB image)					
conv3-64	conv3-64 LRN	conv3-64 <b>conv3-64</b>	conv3-64 conv3-64	conv3-64 conv3-64	conv3-64 conv3-64
maxpool					
conv3-128	conv3-128	conv3-128 <b>conv3-128</b>	conv3-128 conv3-128	conv3-128 conv3-128	conv3-128 conv3-128
maxpool					
conv3-256 conv3-256	conv3-256 conv3-256	conv3-256 conv3-256	conv3-256 conv3-256 <b>conv1-256</b>	conv3-256 conv3-256 <b>conv3-256</b>	conv3-256 conv3-256 conv3-256 <b>conv3-256</b>
maxpool					
conv3-512 conv3-512	conv3-512 conv3-512	conv3-512 conv3-512	conv3-512 conv3-512 <b>conv1-512</b>	conv3-512 conv3-512 <b>conv3-512</b>	conv3-512 conv3-512 conv3-512 <b>conv3-512</b>
maxpool					
conv3-512 conv3-512	conv3-512 conv3-512	conv3-512 conv3-512	conv3-512 conv3-512 <b>conv1-512</b>	conv3-512 conv3-512 <b>conv3-512</b>	conv3-512 conv3-512 conv3-512 <b>conv3-512</b>
maxpool					
FC-4096					
FC-4096					
FC-1000					
soft-max					

Network	A,A-LRN	B	C	D	E
Number of parameters	133	133	134	138	144



在本项目中，使用了完成了预训练的 VGG19 模型，通过源图片与一个超大图片训练集 Train2014 的对比，在不断的迭代中优化 VGG19 模型，使得在程序运行过程中，不断完成对 VGG19 模型参数的修正，提取出训练源图片的风格，减少目标图片的内容，并在程序中写入每经过 200 次迭代输出一次检查点文件，以便在训练过程中遇到不可抗因素导致训练非正常结束，在训练完成后，获取图像

## 2. 算法参数分析

1) style.py 训练可将样式中的样式转换为图像的网络。

--checkpoint-dir: 保存检查点的目录。必需。

--style: 风格图像的路径。需要。

--train-path: 训练图像文件夹的路径。默认值: data / train2014。

--test: 内容映像的路径，以在每个检查点迭代处测试网络。默认: 没有图像。

--test-dir: 保存测试图像的目录路径。如果--test 传递了一个值，则为必需。 - 信号: 培养的时代。默认值: 2。

--batch\_size: 训练的批量大小。默认值: 4。

--checkpoint-iterations: 在检查点之间进行的迭代次数。默认值: 2000。

--vgg-path: VGG19 网络的路径(默认)。如果您想尝试其他损失功能,可以通过 VGG16。默认值: data / imagenet-vgg-verydeep-19.mat。

--content-weight: 损失函数中的内容权重。默认值: 7.5e0。

--style-weight: 丢失函数中的样式权重。默认: 1e2。

--tv-weight: 损失函数中总变差项的权重。默认值: 2e2。

--learning\_rate: 优化器的学习率。默认: 1e-3。

--slow: 用于调试丢失功能。使用 Gatys 的方法直接对像素进行优化。将测试图像用作内容值, test\_dir 用于保存完全优化的图像。

2) evaluate.py 评估训练有素的网络给定一个检查点目录。如果从一个目录评估图像,目录中的每个图像必须具有相同的尺寸。

--checkpoint: 从中加载检查点的目录或 ckpt 文件。需要。

--in-path: 要转换的图像或图像目录的路径。需要。  
--out-path: 用于将转换后的图像从目录中放入 (如果 in\_path 是目录), 转换后的图像或出目录的出路径。需要。  
--device: 用于转换图像的设备。默认: / cpu: 0。  
--batchsize: 用于评估图像的批量大小。特别是用于目录转换。默认值: 4。  
--allow-different-dimensions: 允许不同的图像尺寸。默认值: 未启用  
3) transform\_video.py 将视频转换为风格化视频, 并赋予样式转换网。  
--checkpoint-dir: 从中加载检查点的目录或 ckpt 文件。需要。  
--in-path: 将视频传输到的视频路径。需要。  
--out-path: 输出视频的路径。需要。  
--tmp-dir: 放入临时处理文件的目录。如果不传递路径, 将生成一个 dir。之后会删除 tmpdir。默认值: 随机生成不可见的目录, 然后在执行完成后删除它。  
--device: 用于转换图像的设备。默认: / cpu: 0。  
--batchsize: 用于评估图像的批量大小。特别是用于目录转换。默认值: 4。

## 2. 算法实现

### 1) 训练风格转移网络

使用 style.py 来训练新的样式传输网络。运行 python style.py 查看所有可能的参数。训练需要 4-6 小时才能完成。在运行这个之前, 你应该运行 setup.sh。用法示例:

```
python style.py --style path/to/style/img.jpg \  
  --checkpoint-dir checkpoint/path \  
  --test path/to/test/img.jpg \  
  --test-dir path/to/test/dir \  
  --content-weight 1.5e1 \  
  --checkpoint-iterations 1000 \  
  --batch-size 20
```

### 2) 评估风格转移网络

使用 evaluate.py 来评估样式传输网络。运行 python evaluate.py 查看所有可能的参数。用法示例:

```
python evaluate.py --checkpoint path/to/style/model.ckpt \  
  --in-path dir/of/test/imgs/ \  
  --out-path dir/for/results/
```

### 3) 风格化视频

使用 transform\_video.py 将样式转换为视频。运行 python transform\_video.py 查看所有可能的参数。需要 ffmpeg。用法示例:

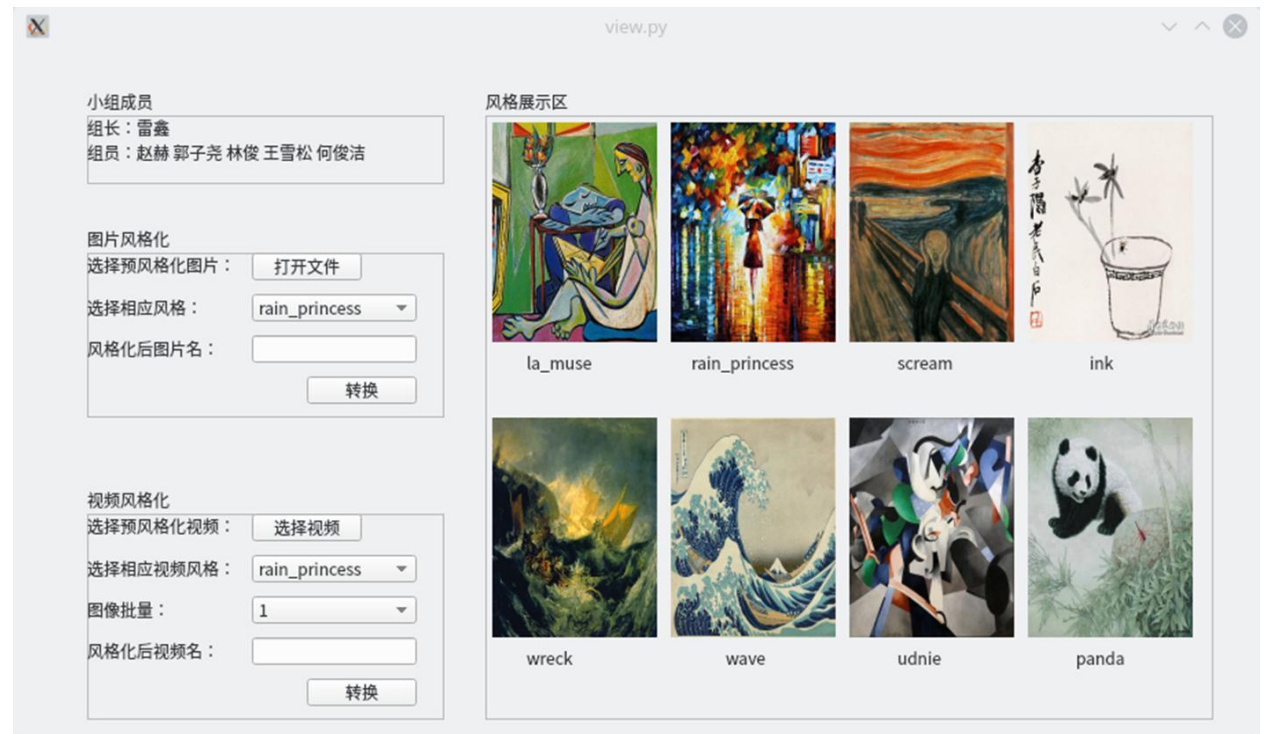
```
python transform_video.py --in-path path/to/input/vid.mp4 \  
  --checkpoint path/to/style/model.ckpt \  
  --out-path out/video.mp4 \  
  --device /gpu:0 \  
  --batch-size 4
```

## 四、创新及特色点

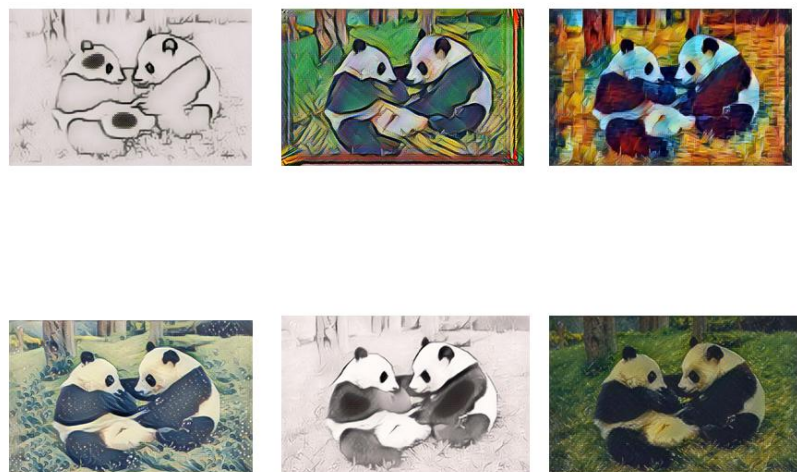
- a) 创新:
  - 1.可以自定义风格化内容,
  - 2.通过 Tkinter 和 py-qt 的结合, 可以选择想要转换的原图
  - 3.界面友好, 可以进行图片和视频风格转换

- b) 特点点：1.将视频按帧拆解，并最终还原为初始的效果  
 2.训练特点还原准确：通过 80000 多张图片作为基础训练集，准确定义图片特点，细节不丢失  
 3.基于卷积神经网络，使用卷积层的中间特征还原出对应这种特征的原始图像

## 五、运行截图



原图



## 六、项目组成员的工作心得：

成员一：赵赫，2016 级，项目负责人

学习能力方面：有一定的学习能力，通过网络的知识，了解到了 TensorFlow 的神经网络，有一定的了解

独立思考能力方面：缺乏一定的创新意识，独立思考方面有所欠缺

发现问题能力方面：本次实验中，在配置库和调参的过程中，刚开始遇见了许多问题，在封装 ckpt 检查点时，通过找资料，使问题得到解决

其它方面：在组织协调上存在一定能力欠缺

成员二：郭子尧，2017 级，项目参与人

学习能力方面：较为一般，接受全新知识比较慢

独立思考能力方面：对于问题能有自己的独立思考

发现问题能力方面：善于发现逻辑上的问题，对于不明显的小问题很难察觉

其它方面：愿意学习以前没接触过的知识

成员三：雷鑫，2017 级，项目参与人

学习能力方面：对代码程序设计方面有了更多的认识 and 了解，通过这个学期的 python 课程学习，逐渐走进了程序语言的世界，虽然了解不多，但是拓宽了自己的知识面。将理论学习和动手实践结合在一起，实践能力得到提高。

独立思考能力方面：python 是一个自己之前从未接触过的课程，学习的过程与以往不同，思维方式也有很大的转变，许多问题需要求解，同时自我思考能力得到锻炼。

发现问题能力方面：根据代码自身的特性，自己多读代码就能发现许多错误，尤其对于函数的使用，操作。

团队合作能力方面：与来自不同学院的同学一起完成项目，大家相互帮助，相互配合，学到了很多有用的东西。

其他方面：对比其他组员，更能发现自己的不足，了解到不同学院的教学模式和学习方法，要多学多问，多加反思。

成员四：何俊洁，2017 级，项目参与人

学习能力方面：通过与组员们的通力合作，我对 Python 语言有了更深层次的认识，在 Python 编程方面能力有了进一步的提升，并且在与小组成员合作的时候，对 Python 语言有了更进一步的认知，并且还对于 Python 语言以外的东西有了一定的了解。

独立思考方面：在与组员们的分工合作中，我对 Python 语言的一些内容也有了自己的见解和看法，独立思考的能力也有了进一步的提升，不过由于整个项目大部分内容是由个别对 Python 语言及其熟悉的组员来完成的，所以我认为我的独立思考能力提升的不够，还有待提高。

发现问题能力方面：由于是初学者所以在 Python 语句编写的时候遇到很多较为浅薄的问题，但是即使如此，这些问对我的在 Python 能力方面的提升依旧很有帮助。

其它方面：在整个团队的分工合作中，我意识到自己在 Python 语言方面知识的严重欠缺，并且意识到 Python 语言编程是一件需要认真对待和处理的学科，在以后的日子里我会去多了解这方面的知识来扩充丰富自己。

成员五：林俊，2017 级，项目参与人

学习能力方面：能够自主思考，也学会团队合作。

独立思考能力方面：能够独立的考虑问题，提出自己的意见和建议。

发现问题能力方面：不同的人有不同的看法，有意见分歧，有交流协作。

其它方面：懂得了站在别人的角度思考和分析问题，能够设身处地地为他人着想，勇于表现自己



成员六：王雪松，2017 级，项目参与人

学习能力方面：由于没有基础，对于见面课内容的掌握不是很好。因此花了较多时间自学。通过实践项目，自学能力有所提高。独立思考能力方面：学会自己思考问题，主动思考，会尝试自己解决问题，找到新思路。对事物对问题的思考，都应该有自己的思考，有自己的解决思路。

发现问题能力方面：课内容的掌握不是很好。因此花了较多时间自学。通过实践项目，自学能力有所提高。期间有许多问题是之前没有料想到的，虽然很困难，但是沉下心里反复地思考和查阅资料后，大部分问题都得以妥善解决。因此独力思考和解决问题的能力也得到锻炼。

团队合作能力方面：通过课堂作业、小组任务等，大家都不断熟悉磨合，有了更多的交流和了解，分工合作也进行得很愉快。清晰合理的分工以及必要的协调，是大家合作成功的必要条件。最终的成功，也离不开每位成员的尽心尽力。

其它方面：尤其是通过在网上查阅相关文献的过程，独立研究的能力得到训练。受益匪浅。

## 七、存在的问题、建议及其他需要说明的情况

本次项目基于开源代码，网址如下：<https://github.com/hzy46/Deep-Learning-21-Examples>。通过对开源项目的简单学习，完成了本次的 python 大作业项目。在实验的过程中，借助了外援的帮助，对于完成内容来说对其它小组并不公平

## 八、附件：代码

### 1.View.py

```
# -*- coding: utf-8 -*-
import os
# Form implementation generated from reading ui file 'self.ui'
#
# Created by: PyQt5 UI code generator 5.6
#
# WARNING! All changes made in this file will be lost!
import sys

import tkinter as tk
from tkinter import filedialog
from PyQt5 import QtCore, QtGui, QtWidgets
from PyQt5.QtGui import QPixmap
import PyQt5.QtWidgets
from PyQt5.QtWidgets import QFileDialog
import numpy as np
import subprocess

class Ui_MainWindow(object):
    def __init__(self):
        self.sourceimpath = "/home/zh/fast-style-transfer-master123/examples/in/1.jpg"
        self.saveimgname = "test.jpg"
```

```
self.sourcevideopath =  
'/home/zh/fast-style-transfer-master123/examples/in/fox.mp4'  
self.savevideoname = 'fox.mp4'
```

```
def setupUi(self, MainWindow):  
    MainWindow.setObjectName("MainWindow")  
    MainWindow.resize(900, 500)  
    self.centralwidget = QtWidgets.QWidget(MainWindow)  
    self.centralwidget.setObjectName("centralwidget")  
  
    self.frame = QtWidgets.QFrame(self.centralwidget)  
    self.frame.setGeometry(QtCore.QRect(50, 50, 260, 50))  
    self.frame.setFrameShape(QtWidgets.QFrame.StyledPanel)  
    self.frame.setFrameShadow(QtWidgets.QFrame.Raised)  
    self.frame.setObjectName("frame")
```

```
self.LaXiaozu = QtWidgets.QLabel(self.centralwidget)  
self.LaXiaozu.setObjectName("LaXiaozu")  
self.LaXiaozu.setGeometry(QtCore.QRect(50, 30, 80, 20))  
self.LaXiaozu.setText("小组成员")
```

```
self.label_name=QtWidgets.QLabel(self.frame)  
self.label_name.setObjectName("Disname")  
self.label_name.setText("组长：雷鑫\n组员：赵赫 郭子尧 林俊 王雪松 何俊洁")  
#self.horizontalLayout.addWidget(self.label)
```

```
self.LaTrimg = QtWidgets.QLabel(self.centralwidget)  
self.LaTrimg.setObjectName("LaTrimg")  
self.LaTrimg.setGeometry(QtCore.QRect(50, 130, 80, 20))  
self.LaTrimg.setText("图片风格化")
```

```
#图片风格化区  
self.frame2 = QtWidgets.QFrame(self.centralwidget)  
self.frame2.setGeometry(QtCore.QRect(50, 150, 260, 120))  
self.frame2.setFrameShape(QtWidgets.QFrame.StyledPanel)  
self.frame2.setFrameShadow(QtWidgets.QFrame.Raised)  
self.frame2.setObjectName("frame2")
```

```
self.La_img = QtWidgets.QLabel(self.frame2)  
self.La_img.setObjectName("Seimg")  
self.La_img.setText("选择预风格化图片： ")
```

```
self.But_img = QtWidgets.QPushButton(self.frame2)
```

```
self.But_img.setObjectName("Opimg")
self.But_img.setText("打开文件")
self.But_img.setGeometry(QRect(120,0,80,20))
self.But_img.clicked.connect(self.getfilename)
```

```
self.La_style = QtWidgets.QLabel(self.frame2)
self.La_style.setObjectName("Chostyle")
self.La_style.setText("选择相应风格： ")
self.La_style.setGeometry(QRect(0,30,90,20))
```

```
self.ImgStyle = QtWidgets.QComboBox(self.frame2)
self.ImgStyle.setObjectName("ImgStyle")
self.ImgStyle.setGeometry(QRect(120,30,120,20))
self.ImgStyle.addItem("rain_princess")
self.ImgStyle.addItem("la_muse")
self.ImgStyle.addItem("ink")
self.ImgStyle.addItem("scream")
self.ImgStyle.addItem("udnie")
self.ImgStyle.addItem("wave")
self.ImgStyle.addItem("wreck")
self.ImgStyle.addItem("panda")
```

```
#提示输入风格化后的图片名
self.La_ImagNa = QtWidgets.QLabel(self.frame2)
self.La_ImagNa.setObjectName("ImgName")
self.La_ImagNa.setGeometry(QRect(0,60,120,20))
self.La_ImagNa.setText("风格化后图片名： ")
```

```
#图片名称输入框
self.ImgName = QtWidgets.QLineEdit(self.frame2)
self.ImgName.setObjectName("SImgName")
self.ImgName.setGeometry(QRect(120,60,120,20))
self.ImgName.textChanged.connect(self.getname)
```

```
#风格化按钮
self.ButtrImag = QtWidgets.QPushButton(self.frame2)
self.ButtrImag.setGeometry(QRect(160,90,80,20))
self.ButtrImag.setText("转换")
self.ButtrImag.clicked.connect(self.transimg)
```

```
self.LaTrvideo = QtWidgets.QLabel(self.centralwidget)
self.LaTrvideo.setObjectName("LaTrvideo")
self.LaTrvideo.setGeometry(QRect(50, 320, 80, 20))
self.LaTrvideo.setText("视频风格化")
```



```
#视频风格化区
```

```
self.frame3 = QtWidgets.QFrame(self.centralwidget)
self.frame3.setGeometry(QtCore.QRect(50, 340, 260, 150))
self.frame3.setFrameShape(QtWidgets.QFrame.StyledPanel)
self.frame3.setFrameShadow(QtWidgets.QFrame.Raised)
self.frame3.setObjectName("frame2")
```

```
self.La_video = QtWidgets.QLabel(self.frame3)
self.La_video.setObjectName("Sevideo")
self.La_video.setText("选择预风格化视频： ")
```

```
self.But_video = QtWidgets.QPushButton(self.frame3)
self.But_video.setObjectName("Opvideo")
self.But_video.setText("选择视频")
self.But_video.setGeometry(QtCore.QRect(120, 0, 80, 20))
self.But_video.clicked.connect(self.getvideopath)
```

```
self.La_vistyle = QtWidgets.QLabel(self.frame3)
self.La_vistyle.setObjectName("ChoVistyle")
self.La_vistyle.setText("选择相应视频风格： ")
self.La_vistyle.setGeometry(QtCore.QRect(0, 30, 120, 20))
```

```
self.VideoStyle = QtWidgets.QComboBox(self.frame3)
self.VideoStyle.setObjectName("VideoStyle")
self.VideoStyle.setGeometry(QtCore.QRect(120, 30, 120, 20))
self.VideoStyle.addItem("rain_princess")
self.VideoStyle.addItem("la_muse")
self.VideoStyle.addItem("ink")
self.VideoStyle.addItem("scream")
self.VideoStyle.addItem("udnie")
self.VideoStyle.addItem("wave")
self.VideoStyle.addItem("wreck")
self.VideoStyle.addItem("panda")
```

```
self.La_batch = QtWidgets.QLabel(self.frame3)
self.La_batch.setObjectName("Labatch")
self.La_batch.setText("图像批量： ")
self.La_batch.setGeometry(QtCore.QRect(0, 60, 120, 20))
```

```
self.Batch = QtWidgets.QComboBox(self.frame3)
self.Batch.setObjectName("VideoStyle")
self.Batch.setGeometry(QtCore.QRect(120, 60, 120, 20))
self.Batch.addItem("1")
```

```
self.Batch.addItem("2")
self.Batch.addItem("3")
self.Batch.addItem("4")
```

```
# 提示输入风格化后的图片名
```

```
self.La_VideoNa = QtWidgets.QLabel(self.frame3)
self.La_VideoNa.setObjectName("VideoName")
self.La_VideoNa.setGeometry(QtCore.QRect(0, 90, 120, 20))
self.La_VideoNa.setText("风格化后视频名: ")
```

```
# 图片名称输入框
```

```
self.ImgName = QtWidgets.QLineEdit(self.frame3)
self.ImgName.setObjectName("SVideoName")
self.ImgName.setGeometry(QtCore.QRect(120, 90, 120, 20))
self.ImgName.textChanged.connect(self.getvideoname)
```

```
# 风格化按钮
```

```
self.ButtVideo = QtWidgets.QPushButton(self.frame3)
self.ButtVideo.setGeometry(QtCore.QRect(160, 120, 80, 20))
self.ButtVideo.setText("转换")
self.ButtVideo.clicked.connect(self.tranvideo)
```

```
self.LaStyleS = QtWidgets.QLabel(self.centralwidget)
self.LaStyleS.setObjectName("LaStyleS")
self.LaStyleS.setGeometry(QtCore.QRect(340, 30, 80, 20))
self.LaStyleS.setText("风格展示区")
```

```
#风格展示区
```

```
self.frame4 = QtWidgets.QFrame(self.centralwidget)
self.frame4.setGeometry(QtCore.QRect(340, 50, 530,440))
self.frame4.setFrameShape(QtWidgets.QFrame.StyledPanel)
self.frame4.setFrameShadow(QtWidgets.QFrame.Raised)
self.frame4.setObjectName("frame2")
```

```
#第一种风格显示
```

```
self.SIlabel1 = QtWidgets.QLabel(self.frame4)
self.SIlabel1.setObjectName("SIlabel1")
```

```
self.SIlabel1.setGeometry(QtCore.QRect(5,5, 120,160))#前两个点表示坐标，后面的点表示大小
```

```
pixmap1 =
```

```
QPixmap('/home/zh/fast-style-transfer-master123/examples/style/la_muse.jpg')
```

```
self.SIlabel1.setPixmap(pixmap1)
```

```
self.SIlabel1.setScaledContents(True)
```

```
self.NAlabel1 = QtWidgets.QLabel(self.frame4)
self.NAlabel1.setObjectName("NAlabel1")
self.NAlabel1.setGeometry(QtCore.QRect(30, 170, 80, 20))
self.NAlabel1.setText("la_muse")
```

#### #第二种风格显示

```
self.SIlabel2 = QtWidgets.QLabel(self.frame4)
self.SIlabel2.setObjectName("SIlabel2")
self.SIlabel2.setGeometry(QtCore.QRect(135, 5, 120, 160))
pixmap1 =
QPixmap('/home/zh/fast-style-transfer-master123/examples/style/rain_princess.jpg')
self.SIlabel2.setPixmap(pixmap1)
self.SIlabel2.setScaledContents(True)
```

```
self.NAlabel2 = QtWidgets.QLabel(self.frame4)
self.NAlabel2.setObjectName("NAlabel2")
self.NAlabel2.setGeometry(QtCore.QRect(150, 170, 80, 20))
self.NAlabel2.setText("rain_princess")
```

#### #第三种风格显示

```
self.SIlabel3 = QtWidgets.QLabel(self.frame4)
self.SIlabel3.setObjectName("SIlabel3")
self.SIlabel3.setGeometry(QtCore.QRect(265, 5, 120, 160))
pixmap1 =
QPixmap('/home/zh/fast-style-transfer-master123/examples/style/the_scream.jpg')
self.SIlabel3.setPixmap(pixmap1)
self.SIlabel3.setScaledContents(True)
```

```
self.NAlabel3 = QtWidgets.QLabel(self.frame4)
self.NAlabel3.setObjectName("NAlabel3")
self.NAlabel3.setGeometry(QtCore.QRect(300, 170, 80, 20))
self.NAlabel3.setText("scream")
```

#### #水墨画风格展示

```
self.SIlabel3 = QtWidgets.QLabel(self.frame4)
self.SIlabel3.setObjectName("SIlabelink")
self.SIlabel3.setGeometry(QtCore.QRect(395, 5, 120, 160))
pixmap1 =
QPixmap('/home/zh/fast-style-transfer-master123/examples/style/ink.jpg')
self.SIlabel3.setPixmap(pixmap1)
self.SIlabel3.setScaledContents(True)
```

```
self.NAlabel3 = QtWidgets.QLabel(self.frame4)
self.NAlabel3.setObjectName("NAlabelink")
```

```
self.NAlabl3.setGeometry(QRect(440, 170, 80, 20))
self.NAlabl3.setText("ink")
```

```
#熊猫水墨风格
```

```
self.Sllabel3 = QtWidgets.QLabel(self.frame4)
self.Sllabel3.setObjectName("Sllabelink")
self.Sllabel3.setGeometry(QRect(395, 220, 120, 160))
pixmap1 =
```

```
QPixmap('/home/zh/fast-style-transfer-master123/examples/style/panda.jpg')
```

```
self.Sllabel3.setPixmap(pixmap1)
self.Sllabel3.setScaledContents(True)
```

```
self.NAlabl3 = QtWidgets.QLabel(self.frame4)
self.NAlabl3.setObjectName("NAlablink")
self.NAlabl3.setGeometry(QRect(430, 385, 80, 20))
self.NAlabl3.setText("panda")
```

```
#第四种风格显示
```

```
self.Sllabel4 = QtWidgets.QLabel(self.frame4)
self.Sllabel4.setObjectName("Sllabel4")
self.Sllabel4.setGeometry(QRect(5, 220, 120, 160))
pixmap1 =
```

```
QPixmap('/home/zh/fast-style-transfer-master123/examples/style/wreck.jpg')
```

```
self.Sllabel4.setPixmap(pixmap1)
self.Sllabel4.setScaledContents(True)
```

```
self.NAlabl4 = QtWidgets.QLabel(self.frame4)
self.NAlabl4.setObjectName("NAlabl4")
self.NAlabl4.setGeometry(QRect(30, 385, 80, 20))
self.NAlabl4.setText("wreck")
```

```
#第五种风格显示
```

```
self.Sllabel5 = QtWidgets.QLabel(self.frame4)
self.Sllabel5.setObjectName("Sllabel5")
self.Sllabel5.setGeometry(QRect(135, 220, 120, 160))
pixmap1 =
```

```
QPixmap('/home/zh/fast-style-transfer-master123/examples/style/wave.jpg')
```

```
self.Sllabel5.setPixmap(pixmap1)
self.Sllabel5.setScaledContents(True)
```

```
self.NAlabl5 = QtWidgets.QLabel(self.frame4)
self.NAlabl5.setObjectName("NAlabl5")
self.NAlabl5.setGeometry(QRect(175, 385, 80, 20))
self.NAlabl5.setText("wave")
```

```

        #第六种风格显示
        self.Sllabel6 = QtWidgets.QLabel(self.frame4)
        self.Sllabel6.setObjectName("Sllabel6")
        self.Sllabel6.setGeometry(QtCore.QRect(265, 220, 120, 160))
        pixmap1 =
QPixmap('/home/zh/fast-style-transfer-master123/examples/style/udnie.jpg')
        self.Sllabel6.setPixmap(pixmap1)
        self.Sllabel6.setScaledContents(True)

        self.NAlabl6 = QtWidgets.QLabel(self.frame4)
        self.NAlabl6.setObjectName("NAlabl6")
        self.NAlabl6.setGeometry(QtCore.QRect(300, 385, 80, 20))
        self.NAlabl6.setText("udnie")

    def getname(self,text):
        self.saveimgname = text

    def getfilename(self):
        sourceimg = tk.filedialog.askopenfilename()
        self.sourceimgpath = sourceimg

    def getvideopath(self):
        sourcevide = tk.filedialog.askopenfilename()
        self.sourcevideopath = sourcevide

    def getvideoname(self,text):
        self.savevideoname = text

    def transimg(self):
        imagestyle = self.lmgStyle.currentText()
        cmd1 = "/home/zh/anaconda2/bin/python evaluate.py --checkpoint
models/"+imagestyle+".ckpt"+" --in-path "+self.sourceimgpath+" --out-path
examples/out/"+self.saveimgname
        cmd2 = "gwenview examples/out/"+self.saveimgname
        print(cmd1)
        cmd=cmd1+"&&"+ cmd2
        os.system(cmd)

    def tranvideo(self):
        videostyle = self.VideoStyle.currentText()
        batchsize = self.Batch.currentText()
        sourcevideo = self.sourcevideopath
        savevide = self.savevideoname

```

```

        #print(sourcevideo)
        cmd1 = "/home/zh/anaconda2/bin/python transform_video.py --in-path
"+sourcevideo+" --checkpoint models/"+videostyle+".ckpt"+" --out-path
examples/out/"+savevide+" --device /gpu:0 --batch-size "+batchsize
        cmd2="/usr/bin/vlc --started-from-file examples/out/"+savevide
        print(cmd2)
        cmd = cmd1 +"&&" +cmd2
        os.system(cmd)

```

```

if __name__ == '__main__':
    app=QtWidgets.QApplication(sys.argv)
    windows = QtWidgets.QWidget()
    ui=Ui_MainWindow()
    ui.setupUi(windows)
    windows.show()
    sys.exit(app.exec_())

```

## 2.Style.py

```

from __future__ import print_function
import sys, os, pdb
sys.path.insert(0, 'src')
import numpy as np, scipy.misc
from optimize import optimize
from argparse import ArgumentParser
from utils import save_img, get_img, exists, list_files
import evaluate

```

```

CONTENT_WEIGHT = 7.5e0
STYLE_WEIGHT = 1e2
TV_WEIGHT = 2e2

```

```

LEARNING_RATE = 1e-3
NUM_EPOCHS = 2
CHECKPOINT_DIR = 'checkpoints'
CHECKPOINT_ITERATIONS = 2000
VGG_PATH = '../data/imagenet-vgg-verydeep-19.mat'
TRAIN_PATH = '../data/train2014'
BATCH_SIZE = 4
DEVICE = '/gpu:0'
FRAC_GPU = 1

```

```

def build_parser():
    parser = ArgumentParser()
    parser.add_argument('--checkpoint-dir', type=str,

```

```
dest='checkpoint_dir', help='dir to save checkpoint in',  
metavar='CHECKPOINT_DIR', required=True)
```

```
parser.add_argument('--style', type=str,  
dest='style', help='style image path',  
metavar='STYLE', required=True)
```

```
parser.add_argument('--train-path', type=str,  
dest='train_path', help='path to training images folder',  
metavar='TRAIN_PATH', default=TRAIN_PATH)
```

```
parser.add_argument('--test', type=str,  
dest='test', help='test image path',  
metavar='TEST', default=False)
```

```
parser.add_argument('--test-dir', type=str,  
dest='test_dir', help='test image save dir',  
metavar='TEST_DIR', default=False)
```

```
parser.add_argument('--slow', dest='slow', action='store_true',  
help='gatys\' approach (for debugging, not supported)',  
default=False)
```

```
parser.add_argument('--epochs', type=int,  
dest='epochs', help='num epochs',  
metavar='EPOCHS', default=NUM_EPOCHS)
```

```
parser.add_argument('--batch-size', type=int,  
dest='batch_size', help='batch size',  
metavar='BATCH_SIZE', default=BATCH_SIZE)
```

```
parser.add_argument('--checkpoint-iterations', type=int,  
dest='checkpoint_iterations', help='checkpoint frequency',  
metavar='CHECKPOINT_ITERATIONS',  
default=CHECKPOINT_ITERATIONS)
```

```
parser.add_argument('--vgg-path', type=str,  
dest='vgg_path',  
help='path to VGG19 network (default %(default)s)',  
metavar='VGG_PATH', default=VGG_PATH)
```

```
parser.add_argument('--content-weight', type=float,  
dest='content_weight',  
help='content weight (default %(default)s)',
```

```

        metavar='CONTENT_WEIGHT', default=CONTENT_WEIGHT)

    parser.add_argument('--style-weight', type=float,
                        dest='style_weight',
                        help='style weight (default %(default)s)',
                        metavar='STYLE_WEIGHT', default=STYLE_WEIGHT)

    parser.add_argument('--tv-weight', type=float,
                        dest='tv_weight',
                        help='total variation regularization weight (default %(default)s)',
                        metavar='TV_WEIGHT', default=TV_WEIGHT)

    parser.add_argument('--learning-rate', type=float,
                        dest='learning_rate',
                        help='learning rate (default %(default)s)',
                        metavar='LEARNING_RATE', default=LEARNING_RATE)

    return parser

def check_opts(opts):
    exists(opts.checkpoint_dir, "checkpoint dir not found!")
    exists(opts.style, "style path not found!")
    exists(opts.train_path, "train path not found!")
    if opts.test or opts.test_dir:
        exists(opts.test, "test img not found!")
        exists(opts.test_dir, "test directory not found!")
    exists(opts.vgg_path, "vgg network data not found!")
    assert opts.epochs > 0
    assert opts.batch_size > 0
    assert opts.checkpoint_iterations > 0
    assert os.path.exists(opts.vgg_path)
    assert opts.content_weight >= 0
    assert opts.style_weight >= 0
    assert opts.tv_weight >= 0
    assert opts.learning_rate >= 0

def _get_files(img_dir):
    files = list_files(img_dir)
    return [os.path.join(img_dir,x) for x in files]

def main():
    parser = build_parser()
    options = parser.parse_args()

```



```
check_opts(options)
```

```
style_target = get_img(options.style)
```

```
if not options.slow:
```

```
    content_targets = _get_files(options.train_path)
```

```
elif options.test:
```

```
    content_targets = [options.test]
```

```
kwargs = {
```

```
    "slow":options.slow,
```

```
    "epochs":options.epochs,
```

```
    "print_iterations":options.checkpoint_iterations,
```

```
    "batch_size":options.batch_size,
```

```
    "save_path":os.path.join(options.checkpoint_dir,'fns.ckpt'),
```

```
    "learning_rate":options.learning_rate
```

```
}
```

```
if options.slow:
```

```
    if options.epochs < 10:
```

```
        kwargs['epochs'] = 1000
```

```
    if options.learning_rate < 1:
```

```
        kwargs['learning_rate'] = 1e1
```

```
args = [
```

```
    content_targets,
```

```
    style_target,
```

```
    options.content_weight,
```

```
    options.style_weight,
```

```
    options.tv_weight,
```

```
    options.vgg_path
```

```
]
```

```
for preds, losses, i, epoch in optimize(*args, **kwargs):
```

```
    style_loss, content_loss, tv_loss, loss = losses
```

```
    print('Epoch %d, Iteration: %d, Loss: %s' % (epoch, i, loss))
```

```
    to_print = (style_loss, content_loss, tv_loss)
```

```
    print('style: %s, content:%s, tv: %s' % to_print)
```

```
    if options.test:
```

```
        assert options.test_dir != False
```

```
        preds_path = '%s/%s_%s.png' % (options.test_dir,epoch,i)
```

```
        if not options.slow:
```

```
            ckpt_dir = os.path.dirname(options.checkpoint_dir)
```

```
            evaluate.ffwd_to_img(options.test,preds_path,
```

```

options.checkpoint_dir)
    else:
        save_img(preds_path, img)
    ckpt_dir = options.checkpoint_dir
    cmd_text = 'python evaluate.py --checkpoint %s ...' % ckpt_dir
    print("Training complete. For evaluation:\n    %s" % cmd_text)

if __name__ == '__main__':
    main()

```

### 3.evaluate.py

```

from __future__ import print_function
import sys
sys.path.insert(0, 'src')
import transform, numpy as np, vgg, pdb, os
import scipy.misc
import os
os.environ['TF_CPP_MIN_LOG_LEVEL']='2'
import tensorflow as tf
from utils import save_img, get_img, exists, list_files
from argparse import ArgumentParser
from collections import defaultdict
import time
import json
import subprocess
import numpy
from moviepy.video.io.VideoFileClip import VideoFileClip
import moviepy.video.io.ffmpeg_writer as ffmpeg_writer
os.environ['CUDA_VISIBLE_DEVICES'] = '0'

BATCH_SIZE = 4
DEVICE = '/gpu:0'

def fwd_video(path_in, path_out, checkpoint_dir, device_t='/gpu:0', batch_size=4):
    video_clip = VideoFileClip(path_in, audio=False)
    video_writer = ffmpeg_writer.FFMPEG_VideoWriter(path_out, video_clip.size,
video_clip.fps, codec="libx264",
preset="medium",
bitrate="2000k",
audiofile=path_in,
threads=None,
ffmpeg_params=None)

```

```

g = tf.Graph()
soft_config = tf.ConfigProto(allow_soft_placement=True)
soft_config.gpu_options.allow_growth = True
with g.as_default(), g.device(device_t), \
    tf.Session(config=soft_config) as sess:
    batch_shape = (batch_size, video_clip.size[1], video_clip.size[0], 3)
    img_placeholder = tf.placeholder(tf.float32, shape=batch_shape,
                                    name='img_placeholder')

    preds = transform.net(img_placeholder)
    saver = tf.train.Saver()
    if os.path.isdir(checkpoint_dir):
        ckpt = tf.train.get_checkpoint_state(checkpoint_dir)
        if ckpt and ckpt.model_checkpoint_path:
            saver.restore(sess, ckpt.model_checkpoint_path)
        else:
            raise Exception("No checkpoint found...")
    else:
        saver.restore(sess, checkpoint_dir)

    X = np.zeros(batch_shape, dtype=np.float32)

    def style_and_write(count):
        for i in range(count, batch_size):
            X[i] = X[count - 1] # Use last frame to fill X
            _preds = sess.run(preds, feed_dict={img_placeholder: X})
            for i in range(0, count):
                video_writer.write_frame(np.clip(_preds[i], 0, 255).astype(np.uint8))

    frame_count = 0 # The frame count that written to X
    for frame in video_clip.iter_frames():
        X[frame_count] = frame
        frame_count += 1
        if frame_count == batch_size:
            style_and_write(frame_count)
            frame_count = 0

    if frame_count != 0:
        style_and_write(frame_count)

    video_writer.close()

# get img_shape

```

```

def fwd(data_in, paths_out, checkpoint_dir, device_t='/gpu:0', batch_size=4):
    assert len(paths_out) > 0
    is_paths = type(data_in[0]) == str
    if is_paths:
        assert len(data_in) == len(paths_out)
        img_shape = get_img(data_in[0]).shape
    else:
        assert data_in.size[0] == len(paths_out)
        img_shape = X[0].shape

    g = tf.Graph()
    batch_size = min(len(paths_out), batch_size)
    curr_num = 0
    soft_config = tf.ConfigProto(allow_soft_placement=True)
    soft_config.gpu_options.allow_growth = True
    with g.as_default(), g.device(device_t), \
        tf.Session(config=soft_config) as sess:
        batch_shape = (batch_size,) + img_shape
        img_placeholder = tf.placeholder(tf.float32, shape=batch_shape,
                                         name='img_placeholder')

        preds = transform.net(img_placeholder)
        saver = tf.train.Saver()
        if os.path.isdir(checkpoint_dir):
            ckpt = tf.train.get_checkpoint_state(checkpoint_dir)
            if ckpt and ckpt.model_checkpoint_path:
                saver.restore(sess, ckpt.model_checkpoint_path)
            else:
                raise Exception("No checkpoint found...")
        else:
            saver.restore(sess, checkpoint_dir)

    num_iters = int(len(paths_out)/batch_size)
    for i in range(num_iters):
        pos = i * batch_size
        curr_batch_out = paths_out[pos:pos+batch_size]
        if is_paths:
            curr_batch_in = data_in[pos:pos+batch_size]
            X = np.zeros(batch_shape, dtype=np.float32)
            for j, path_in in enumerate(curr_batch_in):
                img = get_img(path_in)
                assert img.shape == img_shape, \
                    'Images have different dimensions. ' + \
                    'Resize images or use --allow-different-dimensions.'

```

```

        X[j] = img
    else:
        X = data_in[pos:pos+batch_size]

    _preds = sess.run(preds, feed_dict={img_placeholder:X})
    for j, path_out in enumerate(curr_batch_out):
        save_img(path_out, _preds[j])

    remaining_in = data_in[num_iters*batch_size:]
    remaining_out = paths_out[num_iters*batch_size:]
    if len(remaining_in) > 0:
        ffwd(remaining_in, remaining_out, checkpoint_dir,
            device_t=device_t, batch_size=1)

def ffwd_to_img(in_path, out_path, checkpoint_dir, device='/cpu:0'):
    paths_in, paths_out = [in_path], [out_path]
    ffwd(paths_in, paths_out, checkpoint_dir, batch_size=1, device_t=device)

def ffwd_different_dimensions(in_path, out_path, checkpoint_dir,
    device_t=DEVICE, batch_size=4):
    in_path_of_shape = defaultdict(list)
    out_path_of_shape = defaultdict(list)
    for i in range(len(in_path)):
        in_image = in_path[i]
        out_image = out_path[i]
        shape = "%dx%dx%d" % get_img(in_image).shape
        in_path_of_shape[shape].append(in_image)
        out_path_of_shape[shape].append(out_image)
    for shape in in_path_of_shape:
        print('Processing images of shape %s' % shape)
        ffwd(in_path_of_shape[shape], out_path_of_shape[shape],
            checkpoint_dir, device_t, batch_size)

def build_parser():
    parser = ArgumentParser()
    parser.add_argument('--checkpoint', type=str,
        dest='checkpoint_dir',
        help='dir or .ckpt file to load checkpoint from',
        metavar='CHECKPOINT', required=True)

    parser.add_argument('--in-path', type=str,
        dest='in_path', help='dir or file to transform',
        metavar='IN_PATH', required=True)

```

```

    help_out = 'destination (dir or file) of transformed file or files'
    parser.add_argument('--out-path', type=str,
                        dest='out_path', help=help_out, metavar='OUT_PATH',
                        required=True)

    parser.add_argument('--device', type=str,
                        dest='device', help='device to perform compute on',
                        metavar='DEVICE', default=DEVICE)

    parser.add_argument('--batch-size', type=int,
                        dest='batch_size', help='batch size for feedforwarding',
                        metavar='BATCH_SIZE', default=BATCH_SIZE)

    parser.add_argument('--allow-different-dimensions', action='store_true',
                        dest='allow_different_dimensions',
                        help='allow different image dimensions')

    return parser

def check_opts(opts):
    exists(opts.checkpoint_dir, 'Checkpoint not found!')
    exists(opts.in_path, 'In path not found!')
    if os.path.isdir(opts.out_path):
        exists(opts.out_path, 'out dir not found!')
    assert opts.batch_size > 0

def main():
    parser = build_parser()
    opts = parser.parse_args()
    check_opts(opts)

    if not os.path.isdir(opts.in_path):
        if os.path.exists(opts.out_path) and os.path.isdir(opts.out_path):
            out_path = \
                os.path.join(opts.out_path, os.path.basename(opts.in_path))
        else:
            out_path = opts.out_path

    fwd_to_img(opts.in_path, out_path, opts.checkpoint_dir,
               device=opts.device)
    else:
        files = list_files(opts.in_path)
        full_in = [os.path.join(opts.in_path, x) for x in files]
        full_out = [os.path.join(opts.out_path, x) for x in files]

```

```

        if opts.allow_different_dimensions:
            fwd_different_dimensions(full_in, full_out, opts.checkpoint_dir,
                                    device_t=opts.device, batch_size=opts.batch_size)
        else :
            fwd(full_in, full_out, opts.checkpoint_dir, device_t=opts.device,
                batch_size=opts.batch_size)

if __name__ == '__main__':
    main()

```

#### 4. optimize.py

```

from __future__ import print_function
import functools
import vgg, pdb, time
import tensorflow as tf, numpy as np, os
import transform
from utils import get_img

STYLE_LAYERS = ('relu1_1', 'relu2_1', 'relu3_1', 'relu4_1', 'relu5_1')
CONTENT_LAYER = 'relu4_2'
DEVICES = 'CUDA_VISIBLE_DEVICES'

# np arr, np arr
def optimize(content_targets, style_target, content_weight, style_weight,
            tv_weight, vgg_path, epochs=2, print_iterations=1000,
            batch_size=4, save_path='saver/fns.ckpt', slow=False,
            learning_rate=1e-3, debug=False):
    if slow:
        batch_size = 1
        mod = len(content_targets) % batch_size
        if mod > 0:
            print("Train set has been trimmed slightly..")
            content_targets = content_targets[:-mod]

    style_features = {}

    batch_shape = (batch_size, 256, 256, 3)
    style_shape = (1,) + style_target.shape
    print(style_shape)

    # precompute style features
    with tf.Graph().as_default(), tf.device('/cpu:0'), tf.Session() as sess:
        style_image = tf.placeholder(tf.float32, shape=style_shape, name='style_image')
        style_image_pre = vgg.preprocess(style_image)

```

```

net = vgg.net(vgg_path, style_image_pre)
style_pre = np.array([style_target])
for layer in STYLE_LAYERS:
    features = net[layer].eval(feed_dict={style_image:style_pre})
    features = np.reshape(features, (-1, features.shape[3]))
    gram = np.matmul(features.T, features) / features.size
    style_features[layer] = gram

```

```

with tf.Graph().as_default(), tf.Session() as sess:
    X_content = tf.placeholder(tf.float32, shape=batch_shape, name="X_content")
    X_pre = vgg.preprocess(X_content)

```

```

# precompute content features
content_features = {}
content_net = vgg.net(vgg_path, X_pre)
content_features[CONTENT_LAYER] = content_net[CONTENT_LAYER]

```

```

if slow:
    preds = tf.Variable(
        tf.random_normal(X_content.get_shape()) * 0.256
    )
    preds_pre = preds
else:
    preds = transform.net(X_content/255.0)
    preds_pre = vgg.preprocess(preds)

```

```

net = vgg.net(vgg_path, preds_pre)

```

```

content_size = _tensor_size(content_features[CONTENT_LAYER])*batch_size
assert _tensor_size(content_features[CONTENT_LAYER]) ==
_tensor_size(net[CONTENT_LAYER])
content_loss = content_weight * (2 * tf.nn.l2_loss(
    net[CONTENT_LAYER] - content_features[CONTENT_LAYER]) / content_size
)

```

```

style_losses = []
for style_layer in STYLE_LAYERS:
    layer = net[style_layer]
    bs, height, width, filters = map(lambda i:i.value,layer.get_shape())
    size = height * width * filters
    feats = tf.reshape(layer, (bs, height * width, filters))
    feats_T = tf.transpose(feats, perm=[0,2,1])
    grams = tf.matmul(feats_T, feats) / size
    style_gram = style_features[style_layer]

```



```
style_losses.append(2 * tf.nn.l2_loss(grams - style_gram)/style_gram.size)
```

```
style_loss = style_weight * functools.reduce(tf.add, style_losses) / batch_size
```

```
# total variation denoising
```

```
tv_y_size = _tensor_size(preds[:,1:,:,:])
```

```
tv_x_size = _tensor_size(preds[:, :,1:,:])
```

```
y_tv = tf.nn.l2_loss(preds[:,1:,:,:] - preds[:, :,batch_shape[1]-1,:,:])
```

```
x_tv = tf.nn.l2_loss(preds[:, :,1:,:] - preds[:, :, :,batch_shape[2]-1,:])
```

```
tv_loss = tv_weight*2*(x_tv/tv_x_size + y_tv/tv_y_size)/batch_size
```

```
loss = content_loss + style_loss + tv_loss
```

```
# overall loss
```

```
train_step = tf.train.AdamOptimizer(learning_rate).minimize(loss)
```

```
sess.run(tf.global_variables_initializer())
```

```
import random
```

```
uid = random.randint(1, 100)
```

```
print("UID: %s" % uid)
```

```
for epoch in range(epochs):
```

```
    num_examples = len(content_targets)
```

```
    iterations = 0
```

```
    while iterations * batch_size < num_examples:
```

```
        start_time = time.time()
```

```
        curr = iterations * batch_size
```

```
        step = curr + batch_size
```

```
        X_batch = np.zeros(batch_shape, dtype=np.float32)
```

```
        for j, img_p in enumerate(content_targets[curr:step]):
```

```
            X_batch[j] = get_img(img_p, (256,256,3)).astype(np.float32)
```

```
        iterations += 1
```

```
        assert X_batch.shape[0] == batch_size
```

```
        feed_dict = {
```

```
            X_content:X_batch
```

```
        }
```

```
        train_step.run(feed_dict=feed_dict)
```

```
        end_time = time.time()
```

```
        delta_time = end_time - start_time
```

```
        if debug:
```

```
            print("UID: %s, batch time: %s" % (uid, delta_time))
```

```
            is_print_iter = int(iterations) % print_iterations == 0
```

```
            if slow:
```

```

        is_print_iter = epoch % print_iterations == 0
        is_last = epoch == epochs - 1 and iterations * batch_size >=
num_examples
        should_print = is_print_iter or is_last
        if should_print:
            to_get = [style_loss, content_loss, tv_loss, loss, preds]
            test_feed_dict = {
                X_content:X_batch
            }

            tup = sess.run(to_get, feed_dict = test_feed_dict)
            _style_loss,_content_loss,_tv_loss,_loss,_preds = tup
            losses = (_style_loss, _content_loss, _tv_loss, _loss)
            if slow:
                _preds = vgg.unprocess(_preds)
            else:
                saver = tf.train.Saver()
                res = saver.save(sess, save_path)
            yield(_preds, losses, iterations, epoch)

```

```

def _tensor_size(tensor):
    from operator import mul
    return functools.reduce(mul, (d.value for d in tensor.get_shape()[1:]), 1)

```

## 5. transform.py

```
import tensorflow as tf, pdb
```

```
WEIGHTS_INIT_STDEV = .1
```

```

def net(image):
    conv1 = _conv_layer(image, 32, 9, 1)
    conv2 = _conv_layer(conv1, 64, 3, 2)
    conv3 = _conv_layer(conv2, 128, 3, 2)
    resid1 = _residual_block(conv3, 3)
    resid2 = _residual_block(resid1, 3)
    resid3 = _residual_block(resid2, 3)
    resid4 = _residual_block(resid3, 3)
    resid5 = _residual_block(resid4, 3)
    conv_t1 = _conv_tranpose_layer(resid5, 64, 3, 2)
    conv_t2 = _conv_tranpose_layer(conv_t1, 32, 3, 2)
    conv_t3 = _conv_layer(conv_t2, 3, 9, 1, relu=False)
    preds = tf.nn.tanh(conv_t3) * 150 + 255./2
    return preds

```

```

def _conv_layer(net, num_filters, filter_size, strides, relu=True):
    weights_init = _conv_init_vars(net, num_filters, filter_size)
    strides_shape = [1, strides, strides, 1]
    net = tf.nn.conv2d(net, weights_init, strides_shape, padding='SAME')
    net = _instance_norm(net)
    if relu:
        net = tf.nn.relu(net)

    return net

def _conv_tranpose_layer(net, num_filters, filter_size, strides):
    weights_init = _conv_init_vars(net, num_filters, filter_size, transpose=True)

    batch_size, rows, cols, in_channels = [i.value for i in net.get_shape()]
    new_rows, new_cols = int(rows * strides), int(cols * strides)
    # new_shape = #tf.pack([tf.shape(net)[0], new_rows, new_cols, num_filters])

    new_shape = [batch_size, new_rows, new_cols, num_filters]
    tf_shape = tf.stack(new_shape)
    strides_shape = [1, strides, strides, 1]

    net = tf.nn.conv2d_transpose(net, weights_init, tf_shape, strides_shape,
padding='SAME')
    net = _instance_norm(net)
    return tf.nn.relu(net)

def _residual_block(net, filter_size=3):
    tmp = _conv_layer(net, 128, filter_size, 1)
    return net + _conv_layer(tmp, 128, filter_size, 1, relu=False)

def _instance_norm(net, train=True):
    batch, rows, cols, channels = [i.value for i in net.get_shape()]
    var_shape = [channels]
    mu, sigma_sq = tf.nn.moments(net, [1,2], keep_dims=True)
    shift = tf.Variable(tf.zeros(var_shape))
    scale = tf.Variable(tf.ones(var_shape))
    epsilon = 1e-3
    normalized = (net-mu)/(sigma_sq + epsilon)**(.5)
    return scale * normalized + shift

def _conv_init_vars(net, out_channels, filter_size, transpose=False):
    _, rows, cols, in_channels = [i.value for i in net.get_shape()]
    if not transpose:
        weights_shape = [filter_size, filter_size, in_channels, out_channels]

```

```

    else:
        weights_shape = [filter_size, filter_size, out_channels, in_channels]

        weights_init = tf.Variable(tf.truncated_normal(weights_shape,
stddev=WEIGHTS_INIT_STDEV, seed=1), dtype=tf.float32)
        return weights_init

```

## 6. utils.py

```

import scipy.misc, numpy as np, os, sys

def save_img(out_path, img):
    img = np.clip(img, 0, 255).astype(np.uint8)
    scipy.misc.imsave(out_path, img)

def scale_img(style_path, style_scale):
    scale = float(style_scale)
    o0, o1, o2 = scipy.misc.imread(style_path, mode='RGB').shape
    scale = float(style_scale)
    new_shape = (int(o0 * scale), int(o1 * scale), o2)
    style_target = _get_img(style_path, img_size=new_shape)
    return style_target

def get_img(src, img_size=False):
    img = scipy.misc.imread(src, mode='RGB') # misc.imresize(, (256, 256, 3))
    if not (len(img.shape) == 3 and img.shape[2] == 3):
        img = np.dstack((img, img, img))
    if img_size != False:
        img = scipy.misc.imresize(img, img_size)
    return img

def exists(p, msg):
    assert os.path.exists(p), msg

def list_files(in_path):
    files = []
    for (dirpath, dirnames, filenames) in os.walk(in_path):
        files.extend(filenames)
        break

    return files

```

## 7. vgg.py

```

import tensorflow as tf
import numpy as np

```

```

import scipy.io
import pdb

MEAN_PIXEL = np.array([ 123.68 , 116.779, 103.939])

def net(data_path, input_image):
    layers = (
        'conv1_1', 'relu1_1', 'conv1_2', 'relu1_2', 'pool1',

        'conv2_1', 'relu2_1', 'conv2_2', 'relu2_2', 'pool2',

        'conv3_1', 'relu3_1', 'conv3_2', 'relu3_2', 'conv3_3',
        'relu3_3', 'conv3_4', 'relu3_4', 'pool3',

        'conv4_1', 'relu4_1', 'conv4_2', 'relu4_2', 'conv4_3',
        'relu4_3', 'conv4_4', 'relu4_4', 'pool4',

        'conv5_1', 'relu5_1', 'conv5_2', 'relu5_2', 'conv5_3',
        'relu5_3', 'conv5_4', 'relu5_4'
    )

    data = scipy.io.loadmat(data_path)
    mean = data['normalization'][0][0][0]
    mean_pixel = np.mean(mean, axis=(0, 1))
    weights = data['layers'][0]

    net = {}
    current = input_image
    for i, name in enumerate(layers):
        kind = name[:4]
        if kind == 'conv':
            kernels, bias = weights[i][0][0][0][0]
            # matconvnet: weights are [width, height, in_channels, out_channels]
            # tensorflow: weights are [height, width, in_channels, out_channels]
            kernels = np.transpose(kernels, (1, 0, 2, 3))
            bias = bias.reshape(-1)
            current = _conv_layer(current, kernels, bias)
        elif kind == 'relu':
            current = tf.nn.relu(current)
        elif kind == 'pool':
            current = _pool_layer(current)
        net[name] = current

    assert len(net) == len(layers)

```

```
return net
```

```
def _conv_layer(input, weights, bias):  
    conv = tf.nn.conv2d(input, tf.constant(weights), strides=(1, 1, 1, 1),  
        padding='SAME')  
    return tf.nn.bias_add(conv, bias)
```

```
def _pool_layer(input):  
    return tf.nn.max_pool(input, ksize=(1, 2, 2, 1), strides=(1, 2, 2, 1),  
        padding='SAME')
```

```
def preprocess(image):  
    return image - MEAN_PIXEL
```

```
def unprocess(image):  
    return image + MEAN_PIXEL
```