Hollow Man

基于敦煌文化的快速风格迁移
“文艺复兴”——AI+敦煌文化

摘要

敦煌文化开始于十六国时期的前秦时期，距离今天已经有 1600 多年的历史了，先后经历了十个朝代。每个朝代都在敦煌莫高窟中留下了属于他们自己的那个时代的风格，古人的高超的艺术手法在敦煌莫高窟中体现的淋漓尽至，现代的很多好看的花纹也都是从莫高窟的壁画中选取的，所以莫高窟中的艺术风格在现在来看，也是非常有价值的。

随着在 2017 年，谷歌研发的 AlphaGo 战胜了世界上所有的围棋高手之后，机器学习和深度学习重新变得火热起来，也真正让除了 IT 界的其他人都知道了这些名词。与机器学习和深度学习相关的领域包括自然语言处理、图像识别、风格迁移等。

而我们这个团队，充分利用我们的地理位置的优势——与敦煌在同一个省份，结合我们的专业知识——团队四个人都是计算机专业，以及在研究深度学习的周庆国教授的帮助下，于今年开展了我们的项目——通过风格迁移，提取敦煌壁画各个朝代的艺术风格，并将这些提取的风格与现代服饰相结合，生成一种新的风格的服饰。

关键词：深度学习；快速风格迁移；敦煌文化；服装艺术

正文

引言

在这个深度学习非常火热的时代，我们团队充分响应时代潮流将风格迁移技术与敦煌文化相结合，通过不断地训练，将敦煌壁画中众多艺术风格提取出来，并且利用机器将这些风格结合到现代服饰中。在训练过程中，由于敦煌壁画经过长年累月的侵蚀，普遍呈现的都是深褐色的、纹理不清晰的图画，而且又需要不断地修改各类参数，导致训练起来极为艰难，并且，即使训练完成，在将训练成果与现代服饰的图片结合时，也需要大量的时间才能最终生成一张图片，因此在考虑到效率的前提下，我们团队内部经过讨论，决定采用快速风格迁移的技术，并将其最终图形界面放置在网页客户端上，由于训练已经完成，所以只要客户输入指定的 IP 地址，挑选一种风格并上传任意一张服饰图片，即可快速生成一张具有敦煌风格的服饰图片。

正文

今年 4 月份，我们团队在讨论校创立项的项目方向的时候，向指导老师周庆国教授咨询了意见，老师给我们的想法就是做深度学习这方面，考虑到我们兰州大学地处西部，与敦煌在一个省份，并且敦煌莫高窟在中国、在全世界上有着非常重要的地位，而中国的很多人，对敦煌的那些璀璨文化却并不是很了

解，因此，为了做深度学习，并且弘扬敦煌文化，让更多的人对敦煌的文化有所了解并感兴趣，我们团队与老师协商，决定做风格迁移，将敦煌壁画艺术风格提取出来，与现代服饰相结合，从而形成一种新的风格的服饰。由于我们一开始对风格迁移并不熟悉，所以我们立项成功之后，做的第一步工作，就是团队中每个人都去学习深度学习、风格迁移的知识，我们用的学习资料就是，吴恩达的深度学习视频，从他的教学视频中，我们对深度学习有了更深层的了解。

同时，为了加深我们对深度学习的了解，以及敦煌莫高窟博大精深的文化的了解。在这个暑假，团队中的 2 人参加了周庆国教授的一个暑期夏令营，在周庆国教授的研究生帮助下，学习了如何借助谷歌的 Colab 云服务器对大量图片进行训练，并借助 Colab 云服务器生成一张结合后的图片，一开始，我们从老师那里学到的思想，是将客户选择的风格以及上传的图片进行临时的训练，最终生成结果，但是，由于即使是在有专门的 GPU 和 CPU 下，将风格与图片进行临时的训练也需要 1 至 2 分钟，较为耗时间，而如果是在我们自己的电脑上，训练时间更长，很明显，作为一个客户，绝对是会受不了这种等待的。同时，暑假期间，1 人借助暑期社会实践的机会，去敦煌莫高窟亲身接触和感受了一下敦煌壁画，这些敦煌壁画中，不同朝代有着不同的艺术风格，并且由于我们得到了老师敦煌壁画数据集的支持，可以训练出很多的风格，但是，由于这些壁画经过长年累月的风沙流水的侵蚀，原本的颜色已经都黯淡下来了，而且线条纹理也不是很清晰。于是，在这个学期开学，我们团队经过多次讨论和试验，得出了一个结论，那就是如果按照夏令营中老师教的方法去做我们的这个项目的话，有几个比较麻烦的问题。首先，就是由于敦煌壁画纹理和色彩都不清晰，导致不同朝代的很多图片几乎差不多，而夏令营中训练的图片，则纹理和色彩都是非常鲜明简单的现代图片，就这样的图片，在谷歌的 Colab 云服务器上训练也需要接近两个小时的训练时间，但是我们自己训练敦煌壁画的时候却只能使用自己的电脑，在性能不好的情况下，又要训练更为复杂的图片，所需要花费的时间和调整的参数非常之多；其次，即使训练完成之后，如果我们使用夏令营中的临时将训练出来的风格图片与客户提供的服饰图片进行融合的话，使用客户的电脑生成图片会花费大量的时间，而如果是直接在 Colab 上运行的话，确实能够节省很多时间，但却会将我们的源代码暴露给客户，这样就会存在极大的风险，因为客户就有权力去直接对我们的代码进行修改；最后，我们最初的一个设想是，使用一个桌面应用程序来生成图片显然也是有些困难的，因为这就需要看客户的电脑配置了，很容易让客户因为生成时间过长，占用电脑 GPU、CPU 资源过多而产生不好的映像。下面我们将就以上列举出来的问题一一探讨我们团队的解决过程和方案。

1. 训练风格图片的困难

上面提到过，如果使用我们自己的电脑来对图片进行训练的话，出现的结果就是，我们将要花费大量的时间在这训练上，因此，我们讨论之后，选择租用服务器，在服务器上专门训练这些图片风格，那么我们选择的服务器则是夏令营中用过的谷歌云服务器 Colab，因为在这个服务器上运行过训练图片的代码，我们对这个服务器的整体还是很满意的，而且也知道该怎么使用。在训练的过程中，由于深度学习的参数非常之多，为了训练出一种风格的图片，我们需要不断地对其进行调参，我们在训练图片风格的时候，使用的是 Python 编程语言和 CNN 的方法，因为 CNN 最适合用在图片识别这方面，而 Python 在深度学习方面是最方便的语言。训练图片风格的过程主要是 1 人在负责，以下我们将展示一部分我们训练模型时使用的代码：

```
def parse_args():
    parser = argparse.ArgumentParser()
    parser.add_argument( '-c', '--conf', default = 'conf/mosaic.yml', help = 'the path to the conf file' )
    return parser.parse_args()


def main( FLAGS ):
    style_features_t = losses.get_style_features( FLAGS )

    # Make sure the training path exists.
    training_path = os.path.join( FLAGS.model_path, FLAGS.naming )
    if not( os.path.exists( training_path ) ):
        os.makedirs( training_path )

    with tf.Graph().as_default():
        with tf.Session() as sess:
            """Build Network"""
            network_fn = nets_factory.get_network_fn(
                FLAGS.loss_model,
                num_classes = 1,
                is_training = False )

            image_preprocessing_fn, image_unprocessing_fn = preprocessing_factory.get_preprocessing(
                FLAGS.loss_model,
                is_training = False )
            processed_images = reader.image( FLAGS.batch_size, FLAGS.image_size,
FLAGS.image_size,
                                             'train2014/', image_preprocessing_fn, epochs = FLAGS.epoch )
            generated = model.net( processed_images, training = True )
            processed_generated = [ image_preprocessing_fn( image, FLAGS.image_size,
FLAGS.image_size )
                                    for image in tf.unstack( generated, axis=0, num = FLAGS.batch_size )
                                    ]
            processed_generated = tf.stack( processed_generated )
            _, endpoints_dict = network_fn( tf.concat( [ processed_generated, processed_images ], 0 ),
spatial_squeeze = False )

            # Log the structure of loss network
            tf.logging.info( 'Loss network layers( You can define them in "content_layers" and "style_layers"
) :' )
            for key in endpoints_dict:
```

```
            tf.logging.info( key )

        """Build Losses"""
        content_loss = losses.content_loss( endpoints_dict, FLAGS.content_layers )
        style_loss, style_loss_summary = losses.style_loss( endpoints_dict, style_features_t,
FLAGS.style_layers )
        tv_loss = losses.total_variation_loss( generated )  # use the unprocessed image

        loss = FLAGS.style_weight * style_loss + FLAGS.content_weight * content_loss +
FLAGS.tv_weight * tv_loss

        # Add Summary for visualization in tensorboard.
        """Add Summary"""
        tf.summary.scalar( 'losses/content_loss', content_loss )
        tf.summary.scalar( 'losses/style_loss', style_loss )
        tf.summary.scalar( 'losses/regularizer_loss', tv_loss )

        tf.summary.scalar( 'weighted_losses/weighted_content_loss', content_loss *
FLAGS.content_weight )
        tf.summary.scalar( 'weighted_losses/weighted_style_loss', style_loss * FLAGS.style_weight )
        tf.summary.scalar( 'weighted_losses/weighted_regularizer_loss', tv_loss * FLAGS.tv_weight )
        tf.summary.scalar( 'total_loss', loss )

        for layer in FLAGS.style_layers:
            tf.summary.scalar( 'style_losses/' + layer, style_loss_summary[ layer ] )
        tf.summary.image( 'generated', generated )
        # tf.image_summary( 'processed_generated', processed_generated )  # May be better?
        tf.summary.image( 'origin', tf.stack( [
            image_unprocessing_fn( image ) for image in tf.unstack( processed_images, axis = 0, num =
FLAGS.batch_size )
        ] ) )
        summary = tf.summary.merge_all()
        writer = tf.summary.FileWriter( training_path )

        """Prepare to Train"""
        global_step = tf.Variable( 0, name = "global_step", trainable = False )

        variable_to_train = []
        for variable in tf.trainable_variables():
            if not( variable.name.startswith( FLAGS.loss_model ) ):
                variable_to_train.append( variable )
        train_op = tf.train.AdamOptimizer( 1e-3 ).minimize( loss, global_step = global_step, var_list =
variable_to_train )

        variables_to_restore = []
        for v in tf.global_variables():
            if not(v.name.startswith( FLAGS.loss_model ) ):
                variables_to_restore.append( v )
        saver = tf.train.Saver( variables_to_restore, write_version = tf.train.SaverDef.V1 )

        sess.run( [ tf.global_variables_initializer(), tf.local_variables_initializer() ] )

        # Restore variables for loss network.
        init_func = utils._get_init_fn( FLAGS )
        init_func( sess )
```

```
# Restore variables for training model if the checkpoint file exists.
last_file = tf.train.latest_checkpoint( training_path )
if last_file:
    tf.logging.info( 'Restoring model from {}'.format( last_file ) )
    saver.restore( sess, last_file )

"""Start Training"""
coord = tf.train.Coordinator()
threads = tf.train.start_queue_runners( coord=coord )
start_time = time.time()
try:
    while not coord.should_stop():
        _, loss_t, step = sess.run( [ train_op, loss, global_step ] )
        elapsed_time = time.time() - start_time
        start_time = time.time()
        """logging"""
        # print( step )
        if step % 10 == 0:
            tf.logging.info ( 'step: %d,  total Loss %f, secs/step: %f' % ( step, loss_t, elapsed_time ) )
        """summary"""
        if step % 25 == 0:
            tf.logging.info ( 'adding summary...' )
            summary_str = sess.run( summary )
            writer.add_summary( summary_str, step )
            writer.flush()
        """checkpoint"""
        if step % 1000 == 0:
            saver.save( sess, os.path.join( training_path, 'fast-style-model.ckpt' ), global_step=step )
except tf.errors.OutOfRangeError:
    saver.save( sess, os.path.join( training_path, 'fast-style-model.ckpt-done' ) )
    tf.logging.info( 'Done training -- epoch limit reached' )
finally:
    coord.request_stop()
coord.join( threads )
```

以上为我们训练一种风格图片的一部分代码，经过多次测试之后，我们找到了较为理想的参数设置，虽然训练出来的风格模型的效果仍旧不是很好，但是在资金不是非常足够的情况下，我们一致认为，该效果已经足够好了。也因此，我们在训练图片风格模型上的困难，也基本算是解决了。

2. 生成最终效果图片时间长的困难

首先，我们从夏令营中学到的生成方法是将训练好的模型与客户提供的图片进行再训练，从而生成最终的结果。这种方式如果是在 Colab 云服务器上运行，并没有什么问题，也就只需要 1 至 2 分钟的时间，但是这样的话，我们就无法将我们的成果向客户开放了，因为运行在 Colab 云服务器上的代码可以被所有人看到，而且可以任意更改，我们原本的目的确实是想要做一个开源程序，但这样的程序却完完全全不是一个开源程序了，同时，这样的程序非常的危险。所以我们一开始考虑的是能否将其与一个网页或者一个桌面应用程序形成一个链接，让这样一个网页或者桌面程序调用云服务器上的程序，但是，由于 Colab 自身的局限性，只能我们人工去开启运行这些程序，因此，我们没办法

将图片生成的过程放在服务器上进行，那么留给我们自己的路就只剩下要么选择将服务器开放，要么选择将图片生成的过程放在客户的电脑上进行，或者就是将我们之前学到的东西全部推倒重来，另辟蹊径。经过反复的讨论，我们最终决定，还是只能推倒重来了，因为前面两种方法，不是风险大，就是严重消耗客户时间和资源，给客户非常不好的映像。那么既然决定推倒重来，另辟蹊径，我们该何去何从呢？

在不断地搜索中，我们偶然间了解到了另外一个知识——快速风格迁移，这项技术可以说是风格迁移的一种升级版，能够解决最后生成图片时候的生成时间过长，占用 GPU、CPU 资源过多的问题，而我们在网上看到别的程序员编写的快速风格迁移之后的成果，也是深深感受到一点，快速风格迁移在有了已经训练好的模型之后，将模型与图片结合生成一张新的成果图片会非常快，哪怕是在我们自己的电脑上也是一样的，因此，我们决定往这个方向发展。但是，在一开始的一段时间里，我们都没有什么进展，因为关于快速风格迁移的 Python 项目比较少，而且基本都不是用的 tensorflow，我们只能先在网上寻找其他解决方案，在网上不断地寻找的过程中，我们偶然间发现 github 上有一位程序员，他使用的是 JavaScript 编程语言，通过使用 tensorflow 中的 tensorflow.js 库，可以实现快速风格迁移，因此我们决定学习他的这种方法，使用 tensorflow.js 库。而在这期间，由于我们训练图片风格模型的方法还是采用的老师之前教的那种使用 Python 实现的，而我们也没有料想到会从 Python 转变成 JavaScript 来实现风格迁移，所以我们对 JavaScript 都没有什么了解，也就更谈不上通过它来实现深度学习了，进度又陷入了迟滞的状态。后来，我们在查阅 tensorflow 的 API 文档时发现 tensorflow 中有应对这种问题的解决方案，那就是通过 tensorflow 中的 tensorflowjs_converter 可以将我们用 Python 训练得到的风格模型转换成兼容 JavaScript 的模型，于是我们开始尝试着将我们训练的模型通过这种方式进行转换，在经过不断地学习和尝试之后，我们较为成功地将这个模型转换成了 JavaScript 对应的模型，在此之后，我们也算是基本解决了图片生成方面，消耗时间过多的问题。

3. 图形界面的困难

我们前面多次提到过，在图形界面上面，我们碰到了几个问题，就是如果在 Colab 云服务器上直接运行的话，我们将要面临的是代码可能被恶意修改的风险，而如果搭建桌面应用程序的话，就可能面临占用客户电脑资源过多，耗费时间过多的问题。而当我们改为使用 JavaScript 来建立模型之后，由于我们对 JavaScript 的一些相关知识并不是很了解，而时间又有些紧迫，没有多少时间慢慢学习 JavaScript 的知识了，所以搭建桌面应用程序显然有些不现实，而众所周知，JavaScript 被广泛用于网页端，我们在 github 上找到的那位程序员

写的 JavaScript 快速风格迁移也是通过网页端呈现的，因此，我们决定借用这种方式，

```
<head>

    <title>文艺复兴--AI+敦煌 设计端</title>

    <meta http-equiv = "content-type" content = "text/html; charset = utf-8" />
    <meta http-equiv = "X-UA-Compatible" content = "IE=edge,chrome = 1">
    <meta name = "viewport" content = "width = device-width, initial-scale = 1, maximum-scale = 1" />

    <!-- CSS | bootstrap -->
    <!-- Credits: http://getbootstrap.com/ -->
    <link rel = "stylesheet" type = "text/css" href = "css/bootstrap.min.css" />

    <!-- CSS | font-awesome -->
    <!-- Credits: http://fortawesome.github.io/Font-Awesome/icons/ -->
    <link rel = "stylesheet" type = "text/css" href = "css/font-awesome.min.css" />

    <!-- CSS | animate -->
    <!-- Credits: http://daneden.github.io/animate.css/ -->
    <link rel = "stylesheet" type = "text/css" href = "css/animate.min.css" />

    <!-- CSS | Normalize -->
    <!-- Credits: http://manos.malihu.gr/jquery-custom-content-scroller -->
    <link rel = "stylesheet" type = "text/css" href = "css/jquery.mCustomScrollbar.css" />

    <!-- CSS | Colors -->
    <link rel = "stylesheet" type = "text/css" href = "css/colors/DarkBlue.css" id = "colors-style" />
    <link rel = "stylesheet" type = "text/css" href = "css/switcher.css" />

    <!-- CSS | Style -->
    <!-- Credits: http://themeforest.net/user/FlexyCodes -->
    <link rel = "stylesheet" type = "text/css" href = "css/main.css" />

    <!-- CSS | prettyPhoto -->
    <!-- Credits: http://www.no-margin-for-errors.com/ -->
    <link rel = "stylesheet" type = "text/css" href = "css/prettyPhoto.css" />

    <!-- Favicon -->
    <link rel = "shortcut icon" type = "image/x-icon" href = "images/icon.ico">

    <!--[if IE 7]>
            <link rel = "stylesheet" type = "text/css" href = "css/icons/font-awesome-ie7.min.css"/>
      <![endif]-->

    <style>
      @media only screen and (max-width : 991px) {
        .resp-vtabs .resp-tabs-container {
          margin-left: 13px;
        }
      }

      @media only screen and (min-width : 800px) and (max-width : 991px) {
        .resp-vtabs .resp-tabs-container {
          margin-left: 13px;
```

```
            width: 89%;
        }
    }
</style>

</head>

<body>

<!--[if lt IE 7]>
        <p class = "browsehappy">你正在使用一个 <strong> 旧版本 </strong> 的浏览器.请 <a href =
"http://browsehappy.com/">更新你的浏览器</a>以获得最佳体验</p>
    <![endif]-->

<!-- Laoding page -->
<div id = "preloader">
    <div id = "spinner"></div>
</div>

<!-- .wrapper -->
<div class = "wrapper">

    <!--- .Content -->
    <section class = "tab-content">
        <div class = "container">

            <div class = "row">

                <div class = "col-md-12">

                    <div class = "row">

                        <div class = "col-md-3 widget-profil">
                            <div class = "row">
                                <!-- Profile Image -->
                                <div class = "col-lg-12 col-md-12 col-sm-3 col-xs-12 ">



                                    <div class = "image-holder one" id = "pic_prof_1" style = "display:block">

                                        <img class = "head-image front circle" src = "images/icon.png" width = "150"
                                            height = "150" alt = "" />

                                    </div>

                                    <!-- style for simple image profile -->
                                    <div class = "circle-img" id = "pic_prof_2" style = "display:none"></div>


                                </div>
                                <!-- End Profile Image -->
                                <div class = "col-lg-12 col-md-12 col-sm-9 col-xs-12">
                                    <!-- Profile info -->
                                    <div id = "profile_info">
                                        <h1 id = "name" class = "transition-02">文艺复兴--AI+敦煌</h1>
```

```
                    <h4 class = "line">大学生创新创业训练计划项目</h4>
                    <h6><span class = "fa fa-map-marker"></span>兰州大学</h6>
                </div>
                <!-- End Profile info -->


                <!-- Profile Description -->
                <div id = "profile_desc">

                    <p>
                        欢迎！这是我们的兰州大学校级大学生创新创业训练计划项目：文艺复兴——AI+敦煌的服装设计工作端，请尽情享受服装设计的乐趣，祝您使用愉快！
                    </p>

                </div>
                <!-- End Profile Description -->


            </div>

        </div>
    </div>
```

以上呈现的，则是我们通过 HTML 编写的网页端的一部分代码，通过与服务器进行连接，然后将生成最终图片的过程，考虑到我们已经使用的是快速风格迁移的技术，生成最终的图片对电脑的配置要求不是特别高，而如果我们是租用服务器，由于训练已经完成，后续工作对服务器的要求不高，并且资金紧缺，所以我们决定采用的是，在自己的电脑上搭建一个简易的服务器，我们使用的是 xampp 软件，这款软件可以在个人电脑上搭建一个服务器，只需要将文件放入到该软件特定文件夹下，就可以访问这些文件，唯一的缺陷就是，我们只能将该服务器在 ilzu 的局域网下启动，之后就可以通过 lzu 或者 ilzu 实现访问。这个网页的功能是，首先呈现在眼前的是几种风格图片，客户可以挑选一种风格，然后网页端会跳转到另一个页面，在不到 1 分钟的时间内，就可以将最终的图片生成，而这些步骤，都不会占用客户电脑的资源，全部都是在服务器，也就是我们自己的电脑上生成的。当然，这里面还有一个小缺陷就是，由于我们团队临时改变了方案，从使用 Python 变成了使用 JavaScript，这就导致了我们后续没有太多时间去学习 JavaScript 的更多的知识，有些方面很难做出调整，只能勉强达到目标，暂时还不能实现让客户自行上传图片，然后对客户的图片进行处理。

这次校创，虽然或许是因为我们进入计算机领域还不是很久，对计算机深度学习的一些算法不是很了解，导致研究过程中困难重重，最终也只是勉强完成。但是我们还是收获重大，对深度学习有了更深层的了解，对 CNN、Python 等也有了很多的了解。