



# Style Transfer

CS 20: TensorFlow for Deep Learning Research

Lecture 9

2/9/2017

# Announcements

Assignment 2 is out. It's fun, but tricky. Start early.

Sign up for check-ins/IGs with the course staff!  
[cs20-win1718-staff@lists.stanford.edu](mailto:cs20-win1718-staff@lists.stanford.edu)

# Guest lectures next week



Alec Radford  
OpenAI  
Topic: GANs  
2/9



Danijar Hafner  
Google Brain  
Topic: Variational Autoencoder  
2/14

# Agenda

TFRecord

Getting to know each other!

Style Transfer





TFRecord

# What's TFRecord

1. The recommended format for TensorFlow
2. Binary file format

# What's TFRecord

1. The recommended format for TensorFlow
2. Binary file format  
a serialized `tf.train.Example` protobuf object

# Why binary

- make better use of disk cache



# Why binary

- make better use of disk cache
- faster to move around

# Why binary

- make better use of disk cache
- faster to move around
- can handle data of different types  
e.g. you can put both images and labels in one place

# Convert to TFRecord format

- Feature: an image
- Label: a number

# Convert to TFRecord format

```
# Step 1: create a writer to write tfrecord to that file
writer = tf.python_io.TFRecordWriter(out_file)

# Step 2: get serialized shape and values of the image
shape, binary_image = get_image_binary(image_file)

# Step 3: create a tf.train.Features object
features = tf.train.Features(feature={'label': _int64_feature(label),
                                     'shape': _bytes_feature(shape),
                                     'image': _bytes_feature(binary_image)})

# Step 4: create a sample containing of features defined above
sample = tf.train.Example(features=features)

# Step 5: write the sample to the tfrecord file
writer.write(sample.SerializeToString())
writer.close()
```

# Convert to TFRecord format

# Step 1: create a writer to write tfrecord to that file

```
writer = tf.python_io.TFRecordWriter(out_file)
```

# Step 2: get serialized shape and values of the image

```
shape, binary_image = get_image_binary(image_file)
```

# Step 3: create a tf.train.Features object

```
features = tf.train.Features(feature={'label': _int64_feature(label),  
                                     'shape': _bytes_feature(shape),  
                                     'image': _bytes_feature(binary_image)})
```

# Step 4: create a sample containing of features defined above

```
sample = tf.train.Example(features=features)
```

# Step 5: write the sample to the tfrecord file

```
writer.write(sample.SerializeToString())
```

```
writer.close()
```

Serialize different data type  
into byte strings

# Convert to TFRecord format

```
def _int64_feature(value):  
    return tf.train.Feature(int64_list=tf.train.Int64List(value=[value]))  
  
def _bytes_feature(value):  
    return tf.train.Feature(bytes_list=tf.train.BytesList(value=[value]))
```

# Read TFRecord

Using TFRecordDataset

# Read TFRecord

```
dataset = tf.data.TFRecordDataset(tfreCORD_files)  
dataset = dataset.map(_parse_function)
```

Parse each tfrecord\_file into  
different features that we want

In this case, a tuple of  
(label, shape, image)



# Read TFRecord

```
dataset = tf.data.TFRecordDataset(tfreCORD_files)
dataset = dataset.map(_parse_function)

def _parse_function(tfreCORD_serialized):
    features={'label': tf.FixedLenFeature([], tf.int64),
              'shape': tf.FixedLenFeature([], tf.string),
              'image': tf.FixedLenFeature([], tf.string)}

    parsed_features = tf.parse_single_example(tfreCORD_serialized, features)

    return parsed_features['label'], parsed_features['shape'], parsed_features['image']
```

**See**  
**o8\_tfrecord\_example.py**



## Assignment 2: Style Transfer

# Bringing Impressionism to Life with Neural Style Transfer in *Come Swim*

Bhautik J Joshi\*  
Research Engineer, Adobe

Kristen Stewart  
Director, *Come Swim*

David Shapiro  
Producer, Starlight Studios



**Figure 1:** Usage of Neural Style Transfer in *Come Swim*; left: content image, middle: style image, right: upsampled result. Images used with permission, (c) 2017 Starlight Studios LLC & Kristen Stewart.

**Yes, that Kristen Stewart!**



# Deadpool





# Guernica



# Deadpool and Guernica









# Style Transfer

The math is aight  
but the implementation is tricky

# Mathy stuff

Find a new image:

- whose content is closest to the content image and
- whose style is closest to the style image

# It's all about the loss functions

- **Content loss**

Measure the content loss between the content of the generated image and the content of the content image

- **Style loss**

Measure the style loss between the style of the generated image and the style of the style image

**WHAT'S CONTENT?**



**WHAT'S STYLE?**

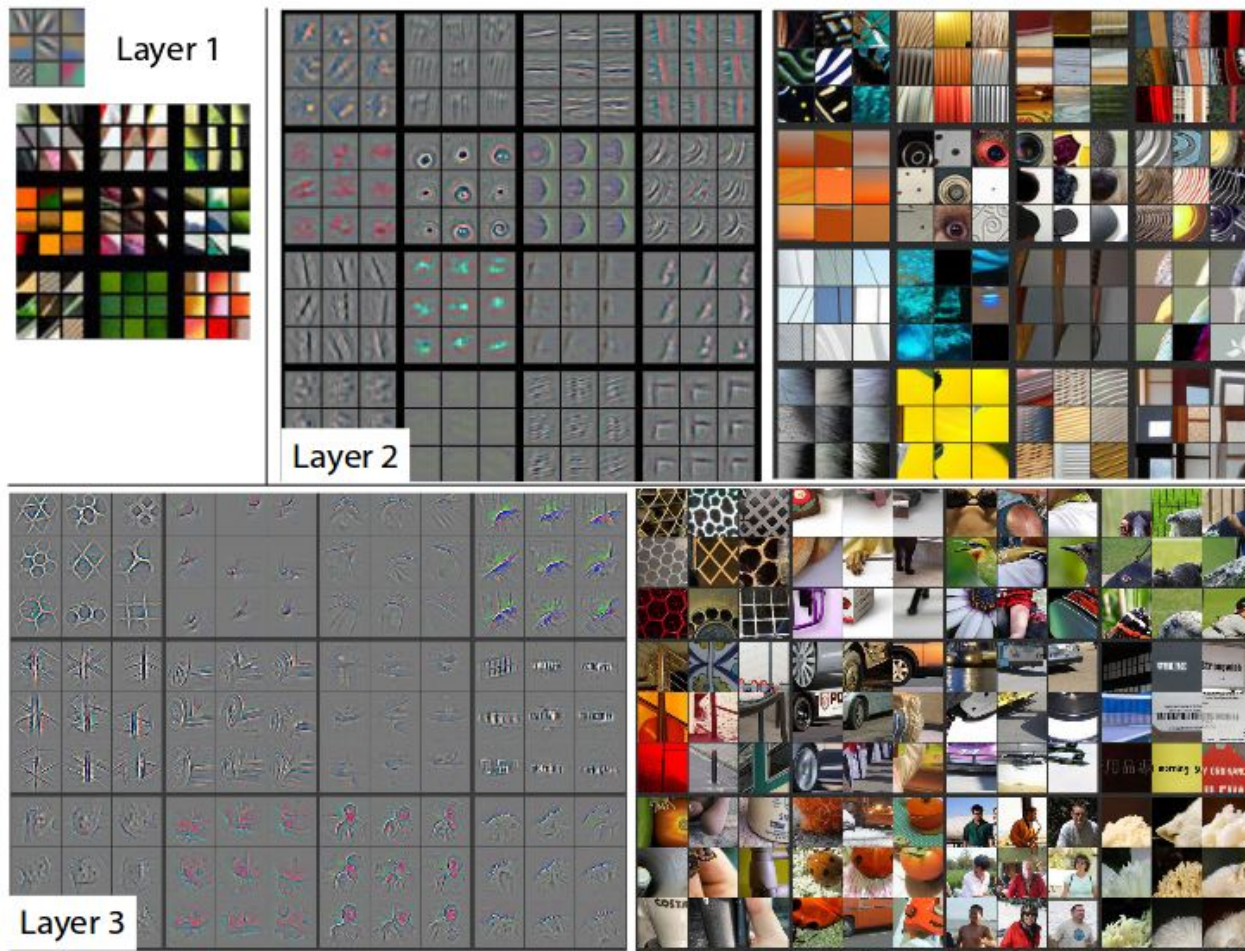




# Feature maps

A convolutional network has many layers, each layer is a function that extracts certain features





# Content/style of an image

Feature visualization have shown that:

- lower layers extract features related to content
- higher layers extract features related to style

# Loss functions revisited

- Content loss

Measure the loss between the feature maps in the content layer of the generated image and the content image

- Style loss

Measure the loss between the feature maps in the style layers of the generated image and the style image

# Loss functions revisited

- Content loss

To measure the content loss between **the feature map in the content layer** of the generated image and the content image

Paper: 'conv4\_4'

- Style loss

To measure the style loss between **the gram matrices of feature maps in the style layers** of the generated image and the style image

Paper: ['conv1\_1', 'conv2\_1', 'conv3\_1', 'conv4\_1' and 'conv5\_1']

# Loss functions revisited

- Content loss

To measure the content loss between **the feature map in the content layer** of the generated image and the content image

Paper: 'conv4\_4'

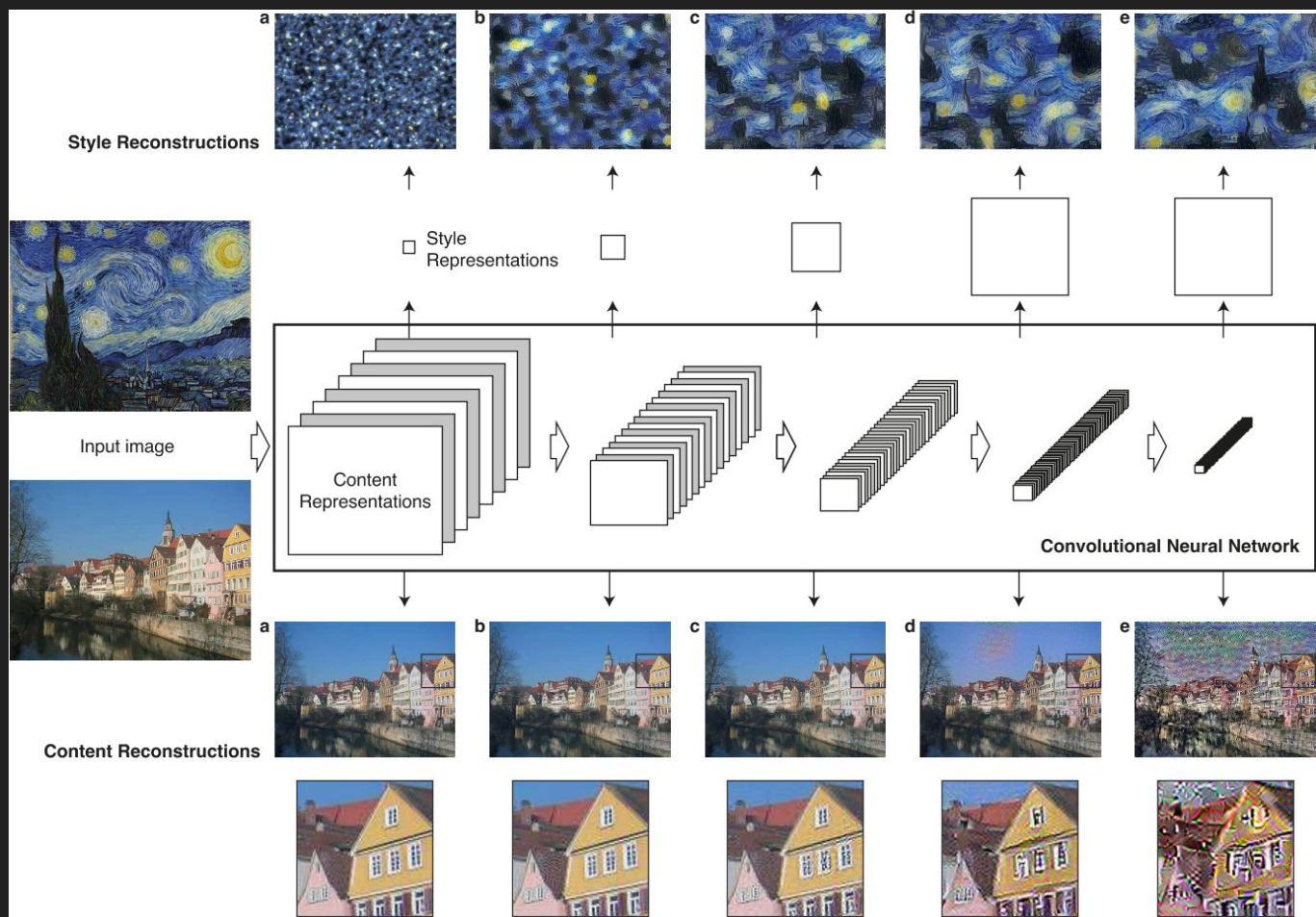
- Style loss

To measure the style loss between **the gram matrices of feature maps in the style layers** of the generated image and the style image

Paper: ['conv1\_1', 'conv2\_1', 'conv3\_1', 'conv4\_1' and 'conv5\_1']

Weighted sum. Give more weight to deeper layers

E.g. 1.0 for 'conv1\_1', 2.0 for 'conv2\_1', ...



**How to find these magic feature maps?**

**Use pretrained weights (functions) such  
as VGG, AlexNet, GoogleNet**



# Loss functions revisited

- Content loss

$$\mathcal{L}_{content}(\vec{p}, \vec{x}, l) = \frac{1}{2} \sum_{i,j} (F_{ij}^l - P_{ij}^l)^2$$

- Style loss

$$E_l = \frac{1}{4N_l^2 M_l^2} \sum_{i,j} (G_{ij}^l - A_{ij}^l)^2$$

$$\mathcal{L}_{style}(\vec{a}, \vec{x}) = \sum_{l=0}^L w_l E_l$$

# Optimizer

Optimizes the initial image to minimize the combination of the two losses

$$\mathcal{L}_{total}(\vec{p}, \vec{a}, \vec{x}) = \alpha \mathcal{L}_{content}(\vec{p}, \vec{x}) + \beta \mathcal{L}_{style}(\vec{a}, \vec{x})$$

Do not optimize the weights!

# Tricky implementation details

1. Train input instead of weights

# Tricky implementation details

1. Train input instead of weights
2. Multiple tensors share the same variable to avoid assembling identical subgraphs

# Tricky implementation details

1. Train input instead of weights
2. Multiple tensors share the same variable to avoid assembling identical subgraphs
3. Use pre-trained weights (from VGG-19)
  - a. Weights and biases already loaded for you
  - b. They are numpy, so need to be converted to tensors
  - c. Must not be trainable!!

# Progress



**Cool story, bro. So what?**

# Fun applications

- Snapchat filters
- Google photos
- Movies!!!



**Is art exclusively a human domain?**

# Next class

GANs by Alec Radford!

Feedback: [chiphuyen@cs.stanford.edu](mailto:chiphuyen@cs.stanford.edu)

Thanks!