Dive into Deep Learning

Chapter 13. Computer Vision

Wayne L, Jan, 31

Recap

This image is CC0 public domain

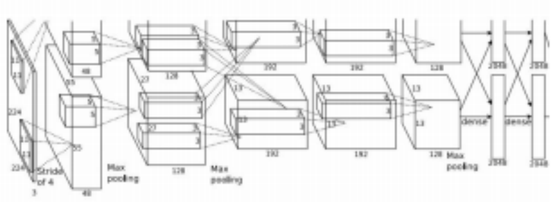


Figure copyright Alex Krizhevsky, Ilya Sutskever, and Geoffrey Hinton, 2012. Reproduced with permission.

Vector:
4096

Fully-Connected:
4096 to 1000

Class Scores
Cat: 0.9
Dog: 0.05
Car: 0.01
...

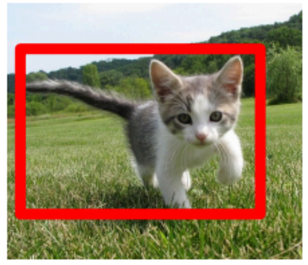
more specific more complex

Semantic Segmentation

GRASS, CAT, TREE, SKY

No objects, just pixels

Classification + Localization



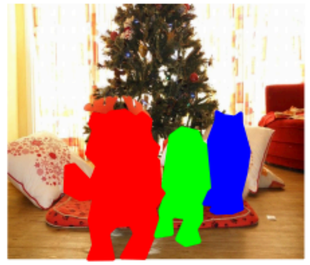
CAT

Single Object

Object Detection

DOG, DOG, CAT

Instance Segmentation

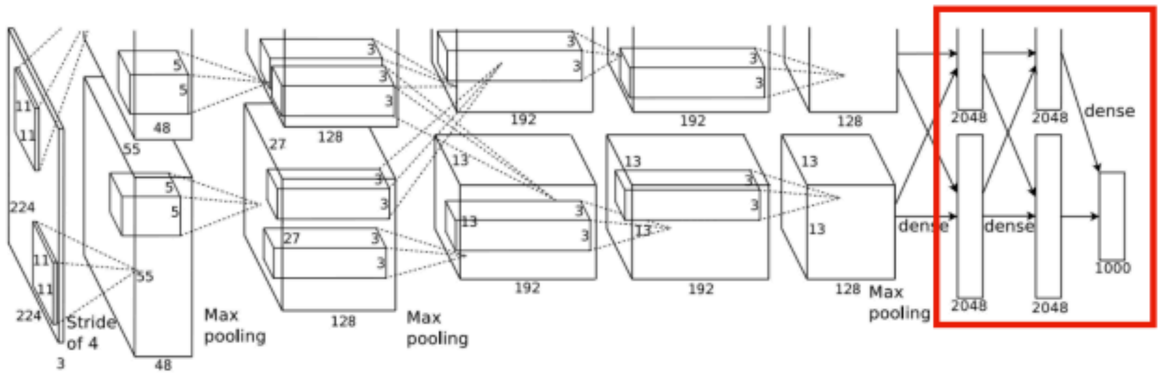


DOG, DOG, CAT

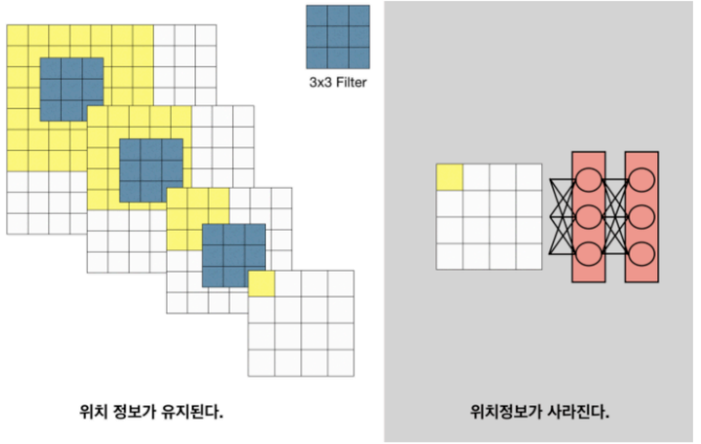
Multiple Object

This image is CC0 public domain

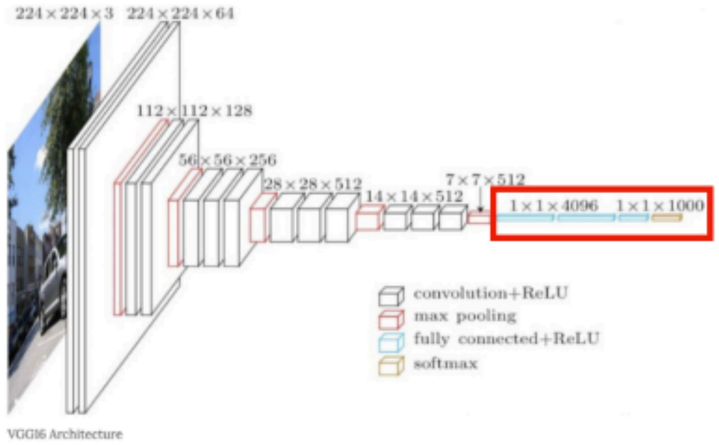
Recap



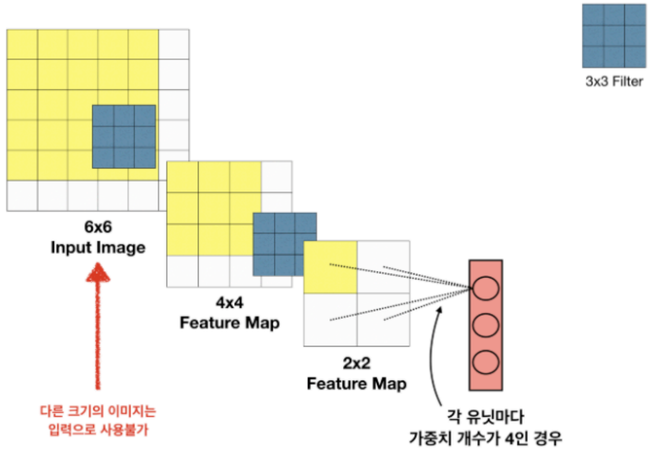
AlexNet



GoogLeNet



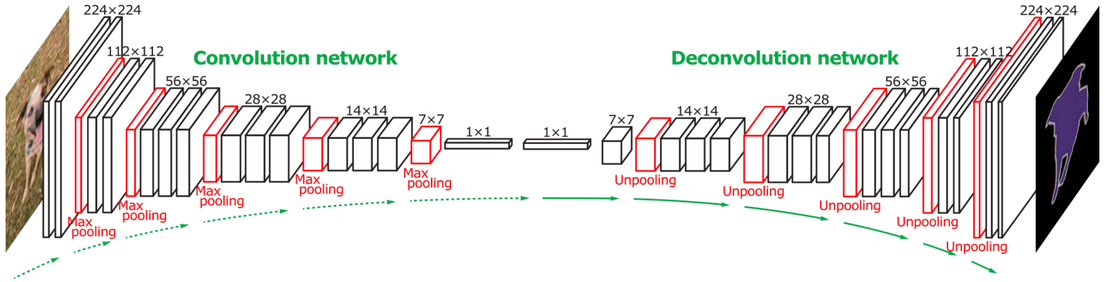
VGG16



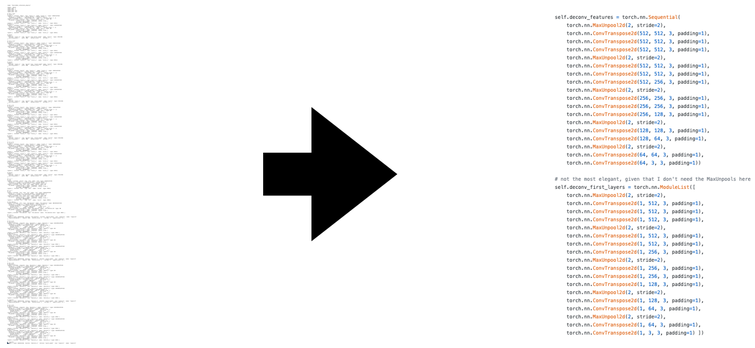
13.10 Transposed Convolution

Introduction

- The layers we introduced so far for CNNs, including convolutional layers and pooling layers, often **reduce** the input width and height, or keep them unchanged.
- Applications such as **semantic segmentation** and **generative adversarial networks**, however, require to predict values for each pixel and therefore needs to increase input width and height.
- Transposed convolution, also named ***fractionally-strided convolution*** or ***deconvolution***, serves this purpose



Noh et al. [ICCV 15]
http://cvlab.postech.ac.kr/research/deconvnet/

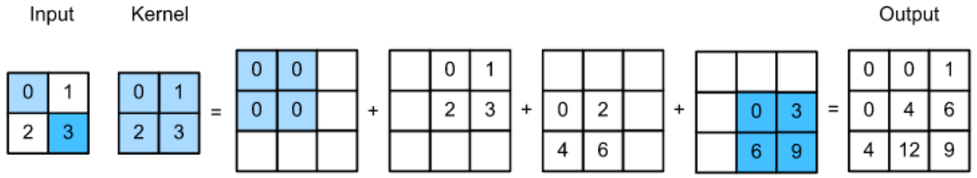


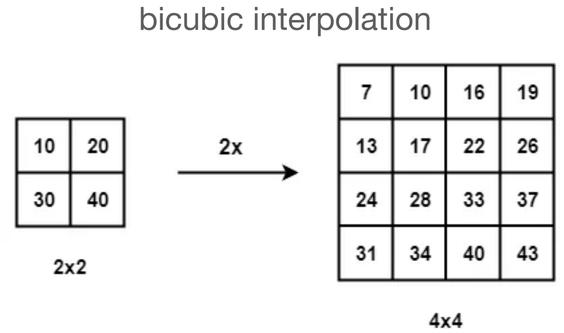
2015 Caffe

2017 Pytorch

13.10 Transposed Convolution

Basic 2D Transposed convolution





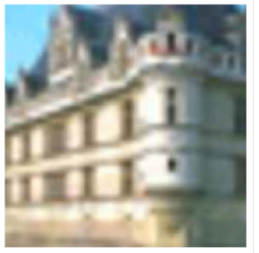
- Illustrates how transposed convolution with a 2 by 2 kernel is computed on the 2 by 2 input matrix

```
def trans_conv(X, K):
    h, w = K.shape
    Y = torch.zeros((X.shape[0] + h - 1, X.shape[1] + w - 1))
    for i in range(X.shape[0]):
        for j in range(X.shape[1]):
            Y[i: i + h, j: j + w] += X[i, j] * K
    return Y
```

```
X, K = X.reshape(1, 1, 2, 2), K.reshape(1, 1, 2, 2)
tconv = nn.ConvTranspose2d(1, 1, kernel_size=2, bias=False)
tconv.weight.data = K
tconv(X)
```

Ground Truth

¼ Sized Input

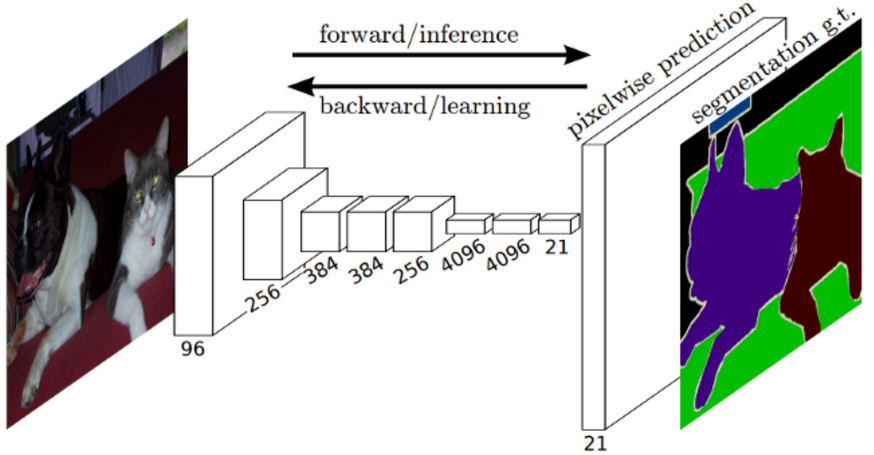


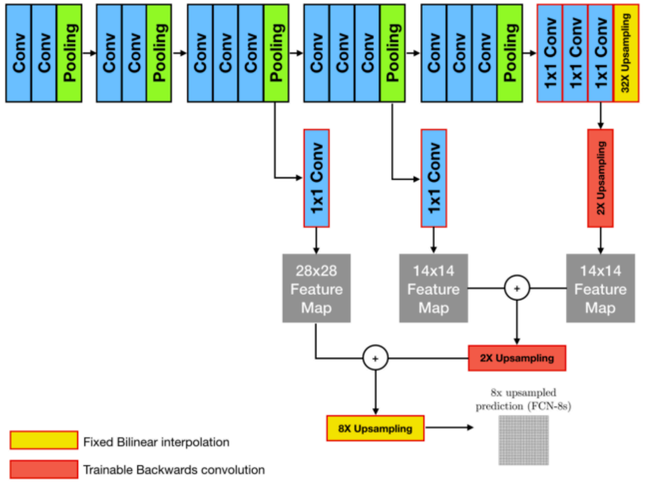
Bicubic

Super Resolution Network

13.11 Fully Convolutional Networks (FCN)

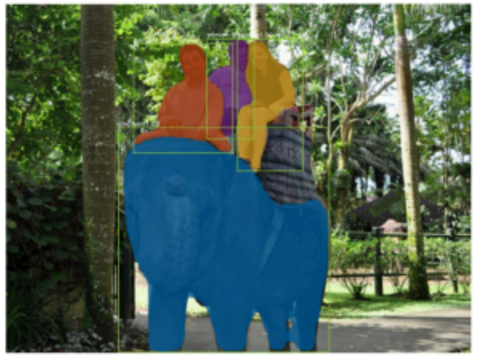
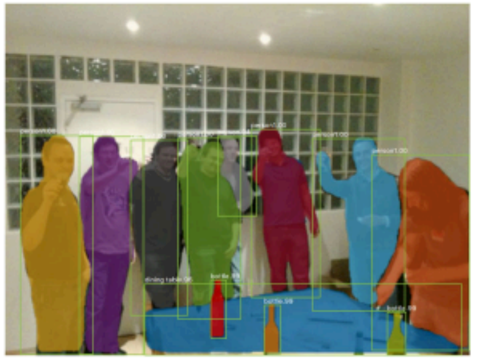
- We previously discussed **semantic segmentation** using each pixel in an image for category prediction
- FCN uses a CNN to transform image pixels to pixel categories
- Unlike the CNN previously introduced, an FCN transforms the height and width of the **intermediate layer feature map** back to the size of input image through the **transposed conv layer**, so that the predictions have a **one-to-one correspondence** with input image in spatial dimension (height and width). Given a position on the spatial dimension, the output of the channel dimension will be a category prediction of the pixel corresponding to the location.



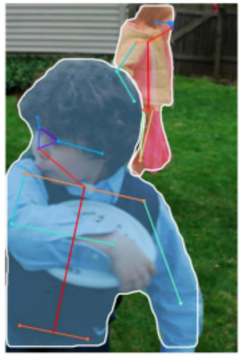


Long et al. [CVPR 15]
https://www.cv-foundation.org/openaccess/content_cvpr_2015/papers/Long_Fully_Convolutional_Networks_2015_CVPR_paper.pdf

Mask R-CNN: Very Good Results!

Mask R-CNN
Also does pose

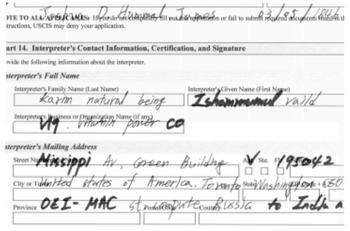
Open Source Frameworks

Lots of good implementations on GitHub!

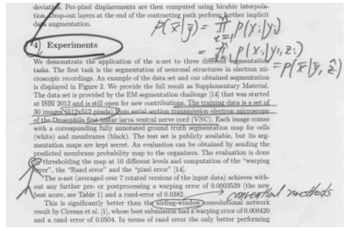
TensorFlow Detection API:
https://github.com/tensorflow/models/tree/master/research/object_detection
Faster RCNN, SSD, RFCN, Mask R-CNN, ...

Detectron2 (PyTorch)
https://github.com/facebookresearch/detectron2
Mask R-CNN, RetinaNet, Faster R-CNN, RPN, Fast R-CNN, R-FCN, ...

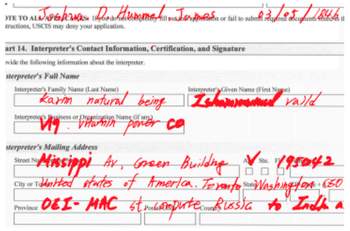
Finetune on your own dataset with pre-trained models


art 14. Interpreter's Contact Information, Certification, and Signature
Interpreter's Full Name
Interpreter's Family Name (Last Name)
Interpreter's Given Name (First Name)
Interpreter's Mailing Address
Street Number and Name
Apt. Ste. Flr.
City or Town
State
ZIP Code
Province
Postal Code
Country

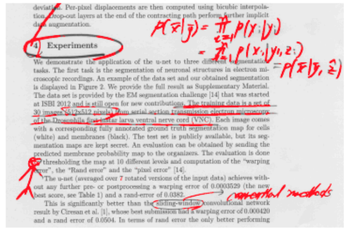
(a)


4 Experiments
We demonstrate the application of the u-net to three different segmentation tasks.
Thresholding the map at 10 different levels and computation of the "warping error", the "Rand error" and the "pixel error" [14].
This is significantly better than the sliding-window convolutional network

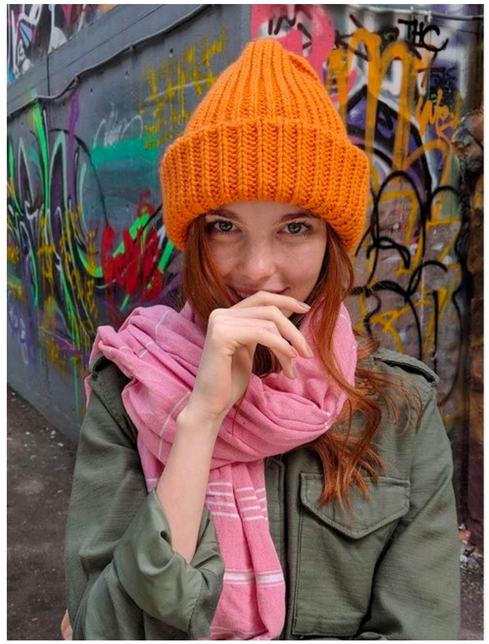
(b)

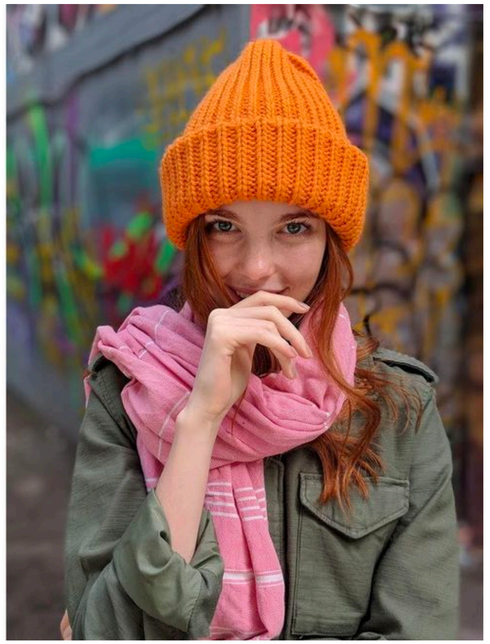

art 14. Interpreter's Contact Information, Certification, and Signature
Interpreter's Full Name
Interpreter's Family Name (Last Name)
Interpreter's Given Name (First Name)
Interpreter's Mailing Address
Street Number and Name
Apt. Ste. Flr.
City or Town
State
ZIP Code
Province
Postal Code
Country

(c)

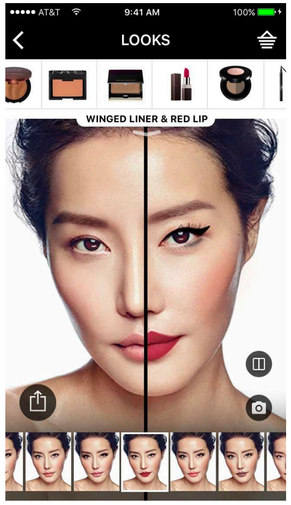

4 Experiments
We demonstrate the application of the u-net to three different segmentation tasks.
Thresholding the map at 10 different levels and computation of the "warping error", the "Rand error" and the "pixel error" [14].
This is significantly better than the sliding-window convolutional network

(d)



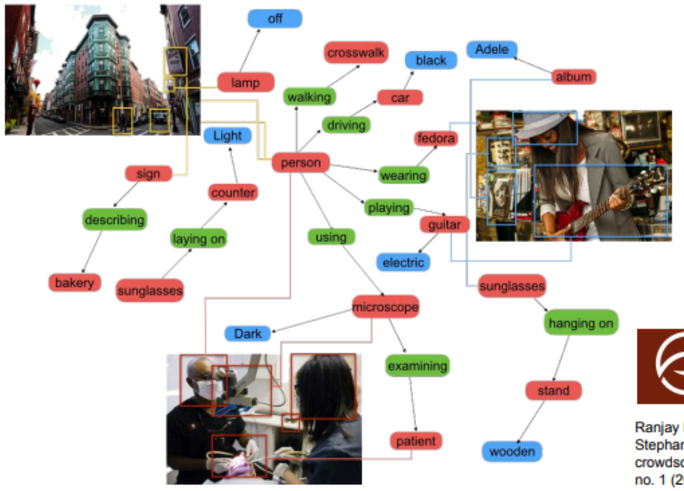



Normal
Portrait
Good morning
Good night


AT&T
9:41 AM
100%
LOOKS
WINGED LINER & RED LIP

Beyond 2D Object Detection…

Objects + Relationships = Scene Graphs

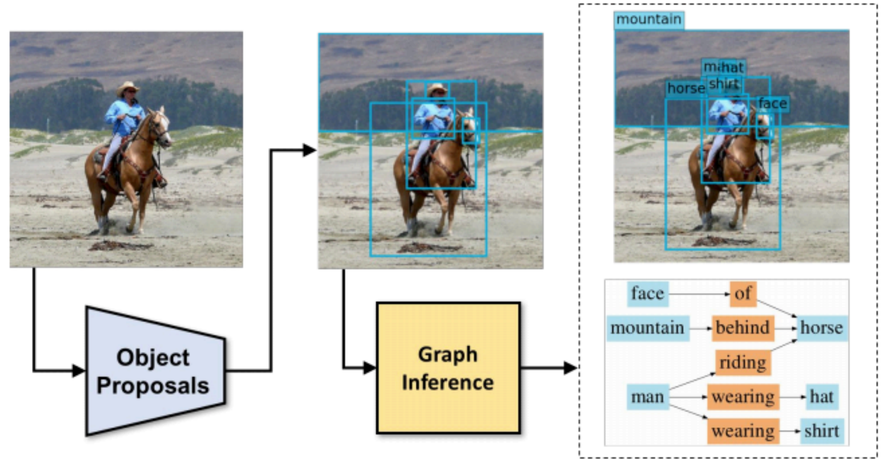


108,077 Images
5.4 Million Region Descriptions
1.7 Million Visual Question Answers
3.8 Million Object Instances
2.8 Million Attributes
2.3 Million Relationships
Everything Mapped to Wordnet Synsets

VISUALGENOME

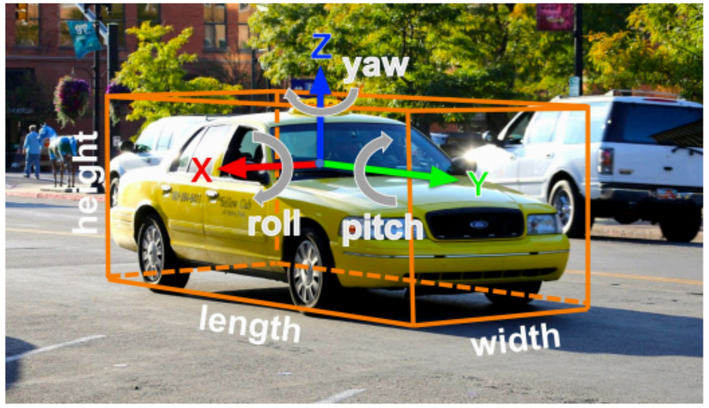
Ranjay Krishna, Yuke Zhu, Oliver Groth, Justin Johnson, Kenji Hata, Joshua Kravitz, Stephanie Chen et al. "Visual genome: Connecting language and vision using crowdsourced dense image annotations." International Journal of Computer Vision 123, no. 1 (2017): 32-73.

Scene Graph Prediction



Xu, Zhu, Choy, and Fei-Fei, "Scene Graph Generation by Iterative Message Passing", CVPR 2017
Figure copyright IEEE, 2018. Reproduced for educational purposes.

3D Object Detection



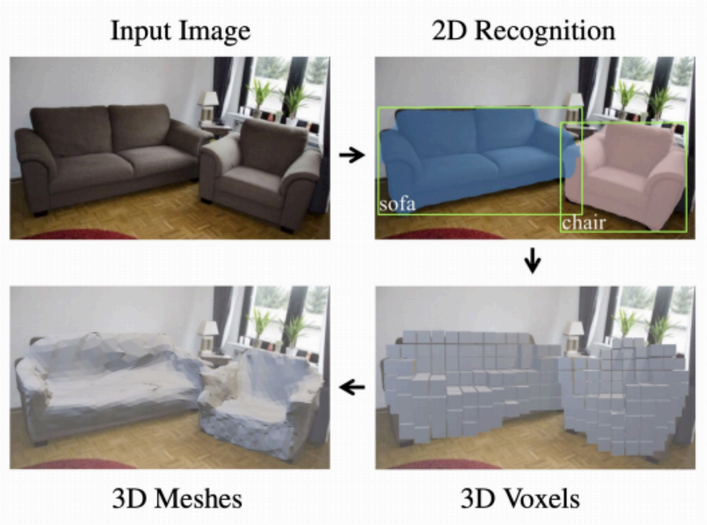
2D Object Detection:
2D bounding box
(x, y, w, h)

3D Object Detection:
3D oriented bounding box
(x, y, z, w, h, l, r, p, y)

Simplified bbox: no roll & pitch

Much harder problem than 2D object detection!

3D Shape Prediction: Mesh R-CNN



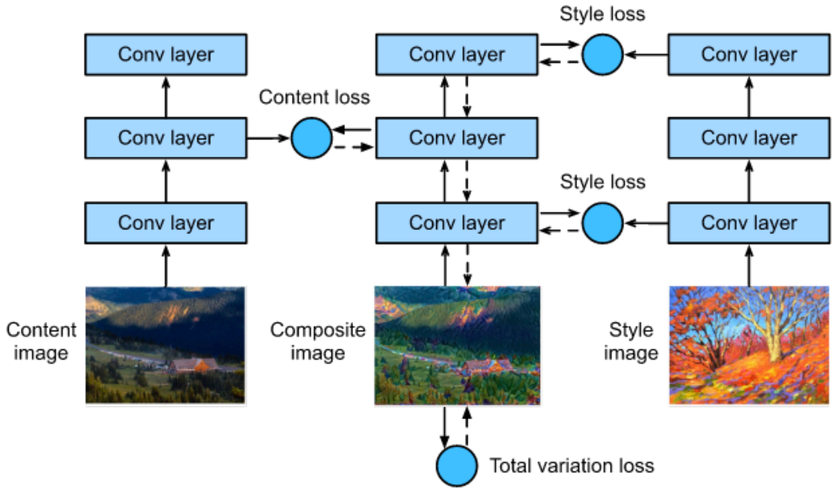
Gkioxari et al., Mesh RCNN, ICCV 2019

13.12 Neural Style Transfer

Leon et al. [2015] https://arxiv.org/abs/1508.06576

- https://www.tensorflow.org/tutorials/generative/style_transfer?hl=ko
- loss = distance(style(style_image) - style(composite_image))
  distance(content(content_image) - content(composite_image))

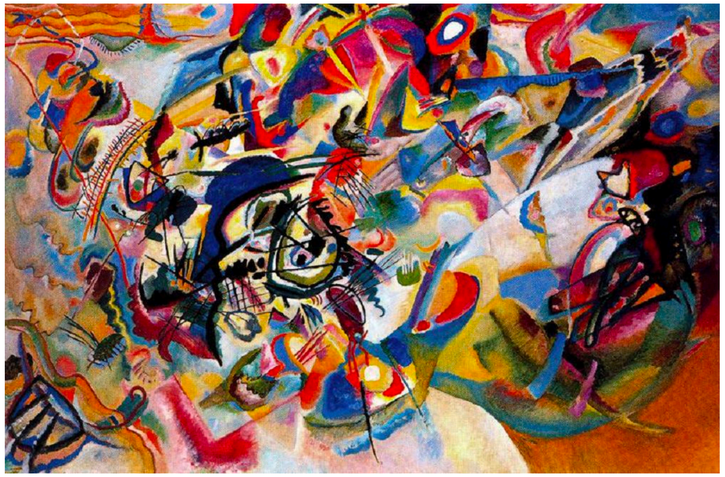
P: content image *A: style image* *X: synthesized image*



```
import tensorflow_hub as hub
hub_module = hub.load('https://tfhub.dev/google/magenta/arbitrary-image-stylization-v1-256/1')
stylized_image = hub_module(tf.constant(content_image), tf.constant(style_image))[0]
tensor_to_image(stylized_image)
```

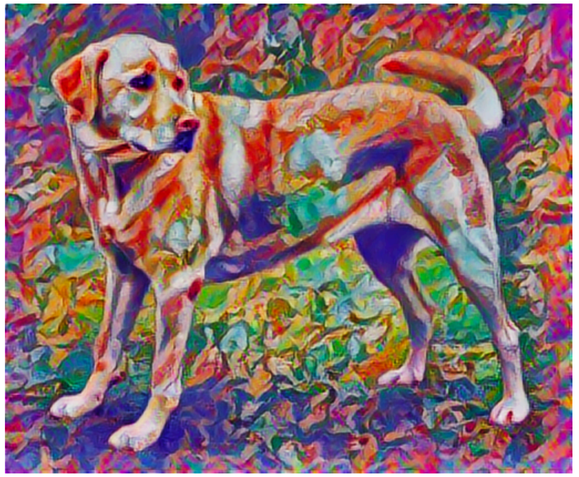
content image

+



reference image

=



generated image

```
Layer (type)                 Output Shape              Param #
=================================================================
input_1 (InputLayer)         (None, 300, 300, 3)       0
_________________________________________________________________
block1_conv1 (Conv2D)        (None, 300, 300, 64)      1792
_________________________________________________________________
block1_conv2 (Conv2D)        (None, 300, 300, 64)      36928
_________________________________________________________________
block1_pool (MaxPooling2D)   (None, 150, 150, 64)      0
_________________________________________________________________
block2_conv1 (Conv2D)        (None, 150, 150, 128)     73856
_________________________________________________________________
block2_conv2 (Conv2D)        (None, 150, 150, 128)     147584
_________________________________________________________________
block2_pool (MaxPooling2D)   (None, 75, 75, 128)       0
_________________________________________________________________
block3_conv1 (Conv2D)        (None, 75, 75, 256)       295168
_________________________________________________________________
block3_conv2 (Conv2D)        (None, 75, 75, 256)       590080
_________________________________________________________________
block3_conv3 (Conv2D)        (None, 75, 75, 256)       590080
_________________________________________________________________
block3_pool (MaxPooling2D)   (None, 37, 37, 256)       0
_________________________________________________________________
block4_conv1 (Conv2D)        (None, 37, 37, 512)       1180160
_________________________________________________________________
block4_conv2 (Conv2D)        (None, 37, 37, 512)       2359808
_________________________________________________________________
block4_conv3 (Conv2D)        (None, 37, 37, 512)       2359808
_________________________________________________________________
block4_pool (MaxPooling2D)   (None, 18, 18, 512)       0
_________________________________________________________________
block5_conv1 (Conv2D)        (None, 18, 18, 512)       2359808
_________________________________________________________________
block5_conv2 (Conv2D)        (None, 18, 18, 512)       2359808
_________________________________________________________________
block5_conv3 (Conv2D)        (None, 18, 18, 512)       2359808
_________________________________________________________________
block5_pool (MaxPooling2D)   (None, 9, 9, 512)         0
=================================================================
Total params: 14,714,688
Trainable params: 14,714,688
Non-trainable params: 0
```

```
content_layers = ['block5_conv2']

style_layers = ['block1_conv1',
                'block2_conv1',
                'block3_conv1',
                'block4_conv1',
                'block5_conv1']
```

```
def vgg_layers(layer_names):
  """ 중간층의 출력값을 배열로 반환하는 vgg 모델을 만듭니다."""
  # 이미지넷 데이터셋에 사전학습된 VGG 모델을 불러옵니다
  vgg = tf.keras.applications.VGG19(include_top=False, weights='imagenet')
  vgg.trainable = False

  outputs = [vgg.get_layer(name).output for name in layer_names]

  model = tf.keras.Model([vgg.input], outputs)
  return model
```

```
def style_content_loss(outputs):
    style_outputs = outputs['style']
    content_outputs = outputs['content']
    style_loss = tf.add_n([tf.reduce_mean((style_outputs[name]-style_targets[name])**2)
                           for name in style_outputs.keys()])
    style_loss *= style_weight / num_style_layers

    content_loss = tf.add_n([tf.reduce_mean((content_outputs[name]-content_targets[name])**2)
                             for name in content_outputs.keys()])
    content_loss *= content_weight / num_content_layers
    loss = style_loss + content_loss
    return loss
```