Introduction to Machine Learning

Deep Learning Applications

Barnabás Póczos
Applications

- **Image Classification**
  - (Alexnet, VGG, Resnet) on Cifar 10, Cifar 100, Mnist, Imagenet

- **Art**
  - Neural style transfer on images and videos
  - Inception, deep dream

- **Visual Question Answering**

- **Image and Video Captioning**

- **Text generation from a style**
  - Shakespeare, Code, receipts, song lyrics, romantic novels, etc

- **Story based question answering**

- **Image generation, GAN**

- **Games, deep RL**
Deep Learning Software Packages

Collection: http://deeplearning.net/software_links/

- **Torch**: http://torch.ch/
- **Caffe**: http://caffe.berkeleyvision.org/
  - **Caffe Model Zoo**: https://github.com/BVLC/caffe/wiki/Model-Zoo
- **NVIDIA Digits**: https://developer.nvidia.com/digits
- **Tensorflow**: https://www.tensorflow.org/
- **Theano**: http://deeplearning.net/software/theano/
- **Lasagne**: http://lasagne.readthedocs.io/en/latest/
- **Keras**: https://keras.io/
- **MXNet**: http://mxnet.io/
- **Dynet**: https://github.com/clab/dynet
- **Microsoft Cognitive Toolkit (MCNTK)**
  
Torch

Torch is a scientific computing framework with wide support for machine learning algorithms that puts GPUs first.

It is easy to use and efficient, thanks to an easy and fast scripting language, LuaJIT, and an underlying C/CUDA implementation.

**Torch tutorials:**

- [https://github.com/bapoczos/TorchTutorial](https://github.com/bapoczos/TorchTutorial)

- Written in Lua
- Used by Facebook
- Often faster than Tensorflow, Theano
TensorFlow™ is an open source library for numerical computation using data flow graphs.

- Nodes in the graph represent mathematical operations,
- while the graph edges represent the multidimensional data arrays (tensors) communicated between them.

Tensorflow tutorials: [https://www.tensorflow.org/tutorials/](https://www.tensorflow.org/tutorials/)

- Developed by Google Brain and used by Google in many products
- Well-documented
- Probably the most popular
- Easy to use with Python
Image Classification
Image Classification with Keras

Keras for building and training a convolutional neural network and using the network for image classification:

Demonstration on MNIST:

https://github.com/bapoczos/keras-mnist-ipython/blob/master/Keras_mnist_tutorial_v1.ipynb
Number or parameters:

320 = 32*(3*3+1)
9248 = 32*(32*3*3+1)
4608 = 32*12*12
589952 = (4608+1)*128
1290 = 10*(128+1)
600810 = 320+9248+589952+1290

The shape of the weight matrices without the bias parameter:
Image Classification with Keras

The confusion matrix:

<table>
<thead>
<tr>
<th></th>
<th>0</th>
<th>1</th>
<th>2</th>
<th>3</th>
<th>4</th>
<th>5</th>
<th>6</th>
<th>7</th>
<th>8</th>
<th>9</th>
</tr>
</thead>
<tbody>
<tr>
<td>row_0</td>
<td>975</td>
<td>0</td>
<td>1</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>4</td>
<td>1</td>
<td>2</td>
<td>3</td>
</tr>
<tr>
<td>col_0</td>
<td>0</td>
<td>1128</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>2</td>
<td>1</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td></td>
<td>0</td>
<td>3</td>
<td>1026</td>
<td>2</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>11</td>
<td>1</td>
<td>0</td>
</tr>
<tr>
<td></td>
<td>1</td>
<td>2</td>
<td>0</td>
<td>1004</td>
<td>0</td>
<td>4</td>
<td>0</td>
<td>1</td>
<td>1</td>
<td>3</td>
</tr>
<tr>
<td></td>
<td>0</td>
<td>0</td>
<td>2</td>
<td>973</td>
<td>0</td>
<td>1</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>4</td>
</tr>
<tr>
<td></td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>3</td>
<td>0</td>
<td>887</td>
<td>4</td>
<td>0</td>
<td>0</td>
<td>4</td>
</tr>
<tr>
<td></td>
<td>3</td>
<td>1</td>
<td>0</td>
<td>4</td>
<td>1</td>
<td>946</td>
<td>0</td>
<td>1</td>
<td>0</td>
<td></td>
</tr>
<tr>
<td></td>
<td>0</td>
<td>0</td>
<td>3</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>1010</td>
<td>2</td>
<td>0</td>
<td></td>
</tr>
<tr>
<td></td>
<td>1</td>
<td>1</td>
<td>0</td>
<td>1</td>
<td>1</td>
<td>0</td>
<td>1</td>
<td>1</td>
<td>965</td>
<td>2</td>
</tr>
<tr>
<td></td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>4</td>
<td>0</td>
<td>0</td>
<td>3</td>
<td>2</td>
<td>993</td>
</tr>
</tbody>
</table>
Image Classification with Keras

Some misclassified images:

Red = Predicted label, Blue = True label.
Image Classification with Keras using VGG19

Vgg19 network test on Imagenet using keras:

https://github.com/bapoczos/keras-vgg19test-ipython/blob/master/keras_vggtest.ipynb
Image Classification using VGG

VERY DEEP CONVOLUTIONAL NETWORKS FOR LARGE-SCALE IMAGE RECOGNITION

Karen Simonyan & Andrew Zisserman

ICLR 2015

Visual Geometry Group, University of Oxford


- Networks of increasing depth using very small (3 × 3) convolution filters
- Shows that a significant improvement on the prior-art configurations can be achieved by pushing the depth to 16–19
- ImageNet Challenge 2014: first and the second places in the localization and classification tracks respectively.
VGG16

Image credit: https://www.cs.toronto.edu/~frossard/post/vgg16/
ConvNet configurations (columns). The depth increases from the left (A) to the right (E), as more layers are added (the added layers are shown in bold).

Convolutional layer parameters: “conv - receptive field size-number of channels”.

The ReLU activation function is not shown for brevity.

<table>
<thead>
<tr>
<th>ConvNet Configuration</th>
<th>A</th>
<th>A-LRN</th>
<th>B</th>
<th>C</th>
<th>D</th>
<th>E</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>11 weight layers</td>
<td>11 weight layers</td>
<td>13 weight layers</td>
<td>16 weight layers</td>
<td>16 weight layers</td>
<td>19 weight layers</td>
</tr>
<tr>
<td>input (224 x 224 RGB image)</td>
<td>conv3-64</td>
<td>conv3-64</td>
<td>conv3-64</td>
<td>conv3-64</td>
<td>conv3-64</td>
<td>conv3-64</td>
</tr>
<tr>
<td>maxpool</td>
<td>conv3-128</td>
<td>conv3-128</td>
<td>conv3-128</td>
<td>conv3-128</td>
<td>conv3-128</td>
<td>conv3-128</td>
</tr>
<tr>
<td>maxpool</td>
<td>conv3-256</td>
<td>conv3-256</td>
<td>conv3-256</td>
<td>conv3-256</td>
<td>conv3-256</td>
<td>conv3-256</td>
</tr>
<tr>
<td>maxpool</td>
<td>conv3-512</td>
<td>conv3-512</td>
<td>conv3-512</td>
<td>conv3-512</td>
<td>conv3-512</td>
<td>conv3-512</td>
</tr>
<tr>
<td>maxpool</td>
<td>conv3-512</td>
<td>conv3-512</td>
<td>conv3-512</td>
<td>conv3-512</td>
<td>conv3-512</td>
<td>conv3-512</td>
</tr>
<tr>
<td>maxpool</td>
<td>conv3-512</td>
<td>conv3-512</td>
<td>conv3-512</td>
<td>conv3-512</td>
<td>conv3-512</td>
<td>conv3-512</td>
</tr>
<tr>
<td>maxpool</td>
<td>FC-4096</td>
<td>FC-4096</td>
<td>FC-1000</td>
<td>soft-max</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

Table 2: Number of parameters (in millions).

<table>
<thead>
<tr>
<th>Network</th>
<th>A</th>
<th>A-LRN</th>
<th>B</th>
<th>C</th>
<th>D</th>
<th>E</th>
</tr>
</thead>
<tbody>
<tr>
<td>Number of parameters</td>
<td>133</td>
<td>133</td>
<td>134</td>
<td>138</td>
<td>144</td>
<td></td>
</tr>
</tbody>
</table>
model = Sequential()
model.add(ZeroPadding2D((1,1), input_shape=(3, 224, 224)))
model.add(Convolution2D(64, 3, 3, activation='relu'))
model.add(ZeroPadding2D((1,1)))
model.add(Convolution2D(64, 3, 3, activation='relu'))
model.add(MaxPooling2D((2,2), strides=(2,2)))
model.add(ZeroPadding2D((1,1)))
model.add(Convolution2D(128, 3, 3, activation='relu'))
model.add(ZeroPadding2D((1,1)))
model.add(Convolution2D(128, 3, 3, activation='relu'))
model.add(MaxPooling2D((2,2), strides=(2,2)))
model.add(ZeroPadding2D((1,1)))
model.add(Convolution2D(256, 3, 3, activation='relu'))
model.add(ZeroPadding2D((1,1)))
model.add(Convolution2D(256, 3, 3, activation='relu'))
model.add(MaxPooling2D((2,2), strides=(2,2)))
model.add(ZeroPadding2D((1,1)))
model.add(Convolution2D(512, 3, 3, activation='relu'))
model.add(ZeroPadding2D((1,1)))
model.add(Convolution2D(512, 3, 3, activation='relu'))
model.add(MaxPooling2D((2,2), strides=(2,2)))
model.add(ZeroPadding2D((1,1)))
model.add(Convolution2D(512, 3, 3, activation='relu'))
model.add(ZeroPadding2D((1,1)))
model.add(Convolution2D(512, 3, 3, activation='relu'))
model.add(MaxPooling2D((2,2), strides=(2,2)))
model.add(Flatten())
model.add(Dense(4096, activation='relu'))
model.add(Dropout(0.5))
model.add(Dense(4096, activation='relu'))
model.add(Dropout(0.5))
model.add(Dense(1000, activation='softmax'))
model.load_weights("./vgg19_weights.h5")
## VGG19 Parameters (Part 1)

<table>
<thead>
<tr>
<th>Layer (type)</th>
<th>Output Shape</th>
<th>Param #</th>
</tr>
</thead>
<tbody>
<tr>
<td>zeropadding2d_1 (ZeroPadding2D)</td>
<td>(None, 3, 226, 226)</td>
<td>0</td>
</tr>
<tr>
<td>convolution2d_1 (Convolution2D)</td>
<td>(None, 64, 224, 224)</td>
<td>1792</td>
</tr>
<tr>
<td>zeropadding2d_2 (ZeroPadding2D)</td>
<td>(None, 64, 226, 226)</td>
<td>0</td>
</tr>
<tr>
<td>convolution2d_2 (Convolution2D)</td>
<td>(None, 64, 224, 224)</td>
<td>36928</td>
</tr>
<tr>
<td>maxpooling2d_1 (MaxPooling2D)</td>
<td>(None, 64, 112, 112)</td>
<td>0</td>
</tr>
<tr>
<td>zeropadding2d_3 (ZeroPadding2D)</td>
<td>(None, 64, 114, 114)</td>
<td>0</td>
</tr>
<tr>
<td>convolution2d_3 (Convolution2D)</td>
<td>(None, 128, 112, 112)</td>
<td>73856</td>
</tr>
<tr>
<td>zeropadding2d_4 (ZeroPadding2D)</td>
<td>(None, 128, 114, 114)</td>
<td>0</td>
</tr>
<tr>
<td>convolution2d_4 (Convolution2D)</td>
<td>(None, 128, 112, 112)</td>
<td>147584</td>
</tr>
<tr>
<td>maxpooling2d_2 (MaxPooling2D)</td>
<td>(None, 128, 56, 56)</td>
<td>0</td>
</tr>
<tr>
<td>zeropadding2d_5 (ZeroPadding2D)</td>
<td>(None, 128, 58, 58)</td>
<td>0</td>
</tr>
<tr>
<td>convolution2d_5 (Convolution2D)</td>
<td>(None, 256, 56, 56)</td>
<td>295168</td>
</tr>
<tr>
<td>zeropadding2d_6 (ZeroPadding2D)</td>
<td>(None, 256, 58, 58)</td>
<td>0</td>
</tr>
<tr>
<td>convolution2d_6 (Convolution2D)</td>
<td>(None, 256, 56, 56)</td>
<td>590080</td>
</tr>
<tr>
<td>zeropadding2d_7 (ZeroPadding2D)</td>
<td>(None, 256, 58, 58)</td>
<td>0</td>
</tr>
<tr>
<td>convolution2d_7 (Convolution2D)</td>
<td>(None, 256, 56, 56)</td>
<td>590080</td>
</tr>
<tr>
<td>zeropadding2d_8 (ZeroPadding2D)</td>
<td>(None, 256, 58, 58)</td>
<td>0</td>
</tr>
<tr>
<td>convolution2d_8 (Convolution2D)</td>
<td>(None, 256, 56, 56)</td>
<td>590080</td>
</tr>
</tbody>
</table>

### Calculations:
- $1792 = (3 \times 3 \times 3 + 1) \times 64$
- $36928 = (64 \times 3 \times 3 + 1) \times 64$
- $73856 = (64 \times 3 \times 3 + 1) \times 128$
- $147584 = (128 \times 3 \times 3 + 1) \times 128$
- $295168 = (128 \times 3 \times 3 + 1) \times 256$
- $590080 = (256 \times 3 \times 3 + 1) \times 256$
## VGG19 (Part 2)

<table>
<thead>
<tr>
<th>Layer</th>
<th>Type</th>
<th>Output Shape</th>
<th>Output Size</th>
</tr>
</thead>
<tbody>
<tr>
<td>'pool3'</td>
<td>MaxPooling2D</td>
<td>(None, 256, 28, 28)</td>
<td>0</td>
</tr>
<tr>
<td>'conv4_1', 'relu4_1'</td>
<td>Convolution2D</td>
<td>(None, 512, 28, 28)</td>
<td>1180160</td>
</tr>
<tr>
<td>'conv4_2', 'relu4_2'</td>
<td>Convolution2D</td>
<td>(None, 512, 28, 28)</td>
<td>2359808</td>
</tr>
<tr>
<td>'conv4_3', 'relu4_3'</td>
<td>Convolution2D</td>
<td>(None, 512, 28, 28)</td>
<td>2359808</td>
</tr>
<tr>
<td>'conv4_4', 'relu4_4'</td>
<td>Convolution2D</td>
<td>(None, 512, 28, 28)</td>
<td>2359808</td>
</tr>
<tr>
<td>'pool4'</td>
<td>MaxPooling2D</td>
<td>(None, 512, 14, 14)</td>
<td>0</td>
</tr>
<tr>
<td>'conv5_1', 'relu5_1'</td>
<td>Convolution2D</td>
<td>(None, 512, 14, 14)</td>
<td>2359808</td>
</tr>
<tr>
<td>'conv5_2', 'relu5_2'</td>
<td>Convolution2D</td>
<td>(None, 512, 14, 14)</td>
<td>2359808</td>
</tr>
<tr>
<td>'conv5_3', 'relu5_3'</td>
<td>Convolution2D</td>
<td>(None, 512, 14, 14)</td>
<td>2359808</td>
</tr>
<tr>
<td>'conv5_4', 'relu5_4'</td>
<td>Convolution2D</td>
<td>(None, 512, 14, 14)</td>
<td>2359808</td>
</tr>
<tr>
<td>'pool5'</td>
<td>MaxPooling2D</td>
<td>(None, 512, 7, 7)</td>
<td>0</td>
</tr>
</tbody>
</table>

1180160 = (256*3*3+1) * 512
2359808 = (512*3*3+1) * 512
### VGG19 (Part 3)

<table>
<thead>
<tr>
<th>Layer</th>
<th>Input Shape</th>
<th>Output Shape</th>
<th>Parameters</th>
</tr>
</thead>
<tbody>
<tr>
<td>flatten_1</td>
<td>(None, 25088)</td>
<td>0</td>
<td>25088=512<em>7</em>7</td>
</tr>
<tr>
<td>dense_1</td>
<td>(None, 4096)</td>
<td>102764544</td>
<td>102764544=(25088+1)*4096</td>
</tr>
<tr>
<td>dropout_1</td>
<td>(None, 4096)</td>
<td>0</td>
<td></td>
</tr>
<tr>
<td>dense_2</td>
<td>(None, 4096)</td>
<td>16781312</td>
<td>16781312=(4096+1)*4096</td>
</tr>
<tr>
<td>dropout_2</td>
<td>(None, 4096)</td>
<td>0</td>
<td></td>
</tr>
<tr>
<td>dense_3</td>
<td>(None, 1000)</td>
<td>4097000</td>
<td>4097000=(4096+1)*1000</td>
</tr>
</tbody>
</table>

Total params: 143,667,240  
Trainable params: 143,667,240  
Non-trainable params: 0

### Softmax:

\[ K = 1000, \quad w_j, x \in \mathbb{R}^{(4096+1)}, \quad j \in 1, \ldots, K \]

\[
P(y = j | x) = \frac{e^{x^T w_j}}{\sum_{k=1}^{K} e^{x^T w_k}}
\]
VGG Results

ILSVRC-2012 dataset (which was used for ILSVRC 2012–2014 challenges). The dataset includes images of 1000 classes, and is split into three sets: training (1.3M images), validation (50K images), and testing (100K images with held-out class labels).

<table>
<thead>
<tr>
<th>ConvNet config. (Table 1)</th>
<th>smallest image side</th>
<th>top-1 val. error (%)</th>
<th>top-5 val. error (%)</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>train ($S$)</td>
<td>test ($Q$)</td>
<td></td>
</tr>
<tr>
<td>A</td>
<td>256</td>
<td>256</td>
<td>29.6</td>
</tr>
<tr>
<td>A-LRN</td>
<td>256</td>
<td>256</td>
<td>29.7</td>
</tr>
<tr>
<td>B</td>
<td>256</td>
<td>256</td>
<td>28.7</td>
</tr>
<tr>
<td>C</td>
<td>256</td>
<td>256</td>
<td>28.1</td>
</tr>
<tr>
<td></td>
<td>384</td>
<td>384</td>
<td>28.1</td>
</tr>
<tr>
<td>[256;512]</td>
<td>384</td>
<td>27.3</td>
<td>8.8</td>
</tr>
<tr>
<td>D</td>
<td>256</td>
<td>256</td>
<td>27.0</td>
</tr>
<tr>
<td></td>
<td>384</td>
<td>384</td>
<td>26.8</td>
</tr>
<tr>
<td>[256;512]</td>
<td>384</td>
<td>25.6</td>
<td>8.1</td>
</tr>
<tr>
<td>E</td>
<td>256</td>
<td>256</td>
<td>27.3</td>
</tr>
<tr>
<td></td>
<td>384</td>
<td>384</td>
<td>26.9</td>
</tr>
<tr>
<td>[256;512]</td>
<td>384</td>
<td><strong>25.5</strong></td>
<td><strong>8.0</strong></td>
</tr>
</tbody>
</table>
VGG Results

0.4170 - n01871265 tusker
0.2178 - n02504458 African elephant, Loxodonta africana
0.1055 - n01704323 triceratops
0.0496 - n02504013 Indian elephant, Elephas maximus
0.0374 - n01768244 trilobite
0.0187 - n01817953 African grey, African gray, Psittacus erithacus
0.0108 - n02398521 hippopotamus, hippo, river horse, Hippopotamus amphibius
0.0095 - n02056570 king penguin, Aptenodytes patagonica
0.0090 - n02071294 killer whale, killer, orca, grampus, sea wolf, Orcinus orca
0.0068 - n01855672 goose
VGG Results

0.7931 - n04335435 streetcar, tram, tramcar, trolley, trolley car
0.1298 - n04487081 trolleybus, trolley coach, trackless trolley
0.0321 - n03895866 passenger car, coach, carriage
0.0135 - n03769881 minibus
0.0103 - n03902125 pay-phone, pay-station
0.0054 - n03272562 electric locomotive
0.0012 - n03496892 harvester, reaper
0.0011 - n03126707 crane
0.0010 - n04465501 tractor
0.0010 - n03417042 garbage truck, dustcart
Video Classification

https://www.youtube.com/watch?v=qrzQ_AB1DZk
Andrej Karpathy, CVPR 2014
Style Transfer
Gatys, Ecker, Bethge: A Neural Algorithm of Artistic Style
Style Transfer, Relevant Papers

- Image Style Transfer Using Convolutional Neural Networks
  Leon A. Gatys, Alexander S. Ecker, Matthias Bethge

- Combining Markov Random Fields and Convolutional Neural Networks for Image Synthesis, Chuan Li, Michael Wand
The Shipwreck of the Minotaur by J.M.W. Turner, 1805.
Style Transfer

The Starry Night by Vincent van Gogh, 1889.
Style Transfer

Der Schrei by Edvard Munch, 1893
Style Transfer with Keras and Tensorflow

https://github.com/bapoczos/StyleTransfer/blob/master/style_transfer_keras_tensorflow.ipynb
Content image size: (1, 450, 845, 3)
Style Image

Style image size: (1, 507, 640, 3)
Style Transfer
Style Transform with VGG 19

depth=64
3x3 conv
conv1_1
conv1_2

depth=128
3x3 conv
conv2_1
conv2_2

depth=256
3x3 conv
conv3_1
conv3_2
conv3_3
conv3_4

depth=512
3x3 conv
conv4_1
conv4_2
conv4_3
conv4_4

depth=512
3x3 conv
conv5_1
conv5_2
conv5_3
conv5_4

size=4096
FC1
FC2

size=1000
softmax
Style Transfer

Algorithm:

1) Calculate content features (set of tensors which are neuron activities in the hidden layers)

2) Calculate style features (set of Gram matrices which are correlations between neuron activities in the hidden layers)

3) Create a new image that matches both the content activities and the style Gram matrices
Style Transform: Content features

We will use VGG19 without the final maxpool, Flat, Dense, Dropout, and Softmax Layers

Select CONTENT_LAYERS

For example:

\{'conv1_1', 'conv2_1', 'conv4_1', 'conv4_2'\}
or just simply \{'relu4_2'\}

Size of relu4_2', (1, 57, 106, 512)

[57 = 450 / 8, 106 = 845 / 8
8 = 2^3 Size decrease after 3 maxpool]

The elements of the (1, 57, 106, 512) tensor are the content features
Style Transform: Calculating Style Gram matrices

Select STYLE_LAYERS

For example:

{'conv3_1', 'conv5_1'}

Or

{'relu1_1', 'relu2_1', 'relu3_1', 'relu4_1', 'relu5_1'}

Style image size: (1, 507, 640, 3)

'relu1_1' shape: (1, 507, 640, 64)
reshaped: (324480, 64)
gram matrix shape: (64, 64)

'relu2_1' shape: (1, 254, 320, 128)
reshaped: (81280, 128)
gram matrix shape: (128, 128)

'relu3_1' shape: (1, 127, 160, 256)
reshaped: (20320, 256)
gram matrix shape: (256, 256)

'relu4_1' shape: (1, 64, 80, 512)
reshaped: (5120, 512)
gram matrix shape: (512, 512)

'relu5_1' shape: (1, 32, 40, 512)
reshaped: (1280, 512)
gram matrix shape: (512, 512)
Style Transform: Neural Doodle

#NeuralDoodle

Deep Convolutional Networks for Semantic Style Transfer

http://github.com/alexjc/neural-doodle

Music: Incoming Light Waves CC-BY-NC Zeropage.
Style Transfer for Videos

https://www.youtube.com/watch?v=Khuj4ASIdmU

Artistic style transfer for videos

Manuel Ruder
Alexey Dosovitskiy
Thomas Brox

University of Freiburg
Chair of Pattern Recognition and Image Processing
Inception / Deep Dream
Tune the Inputs

Instead of tuning the neural network weights, keep them fixed (e.g., VGG19 weights) and tune the input image of the network.

Starting from random noise, find the image that will maximize the probability of being classified as banana.

Image credit: https://research.googleblog.com/2015/06/inceptionism-going-deeper-into-neural.html
Tune the Inputs

Hartebeest  Measuring Cup  Ant  Starfish

Anemone Fish  Banana  Parachute  Screw
Deep Dream
Deep Dream

Goal: Find the image the maximizes the sum of the neuron activities on some selected channels of some selected layers
Deep Dream

layer = 'mixed4d_3x3_bottleneck_pre_relu' channel = 139
Deep Dream

After multiscale + smoothing

Blur the image a little every iteration by suppressing the higher frequencies, so that the lower frequencies can catch up
Deep Dream

Let's try to visualize another channel from the same layer
layer = 'mixed4d_3x3_bottleneck_pre_relu' channel = 65
Deep Dream

Lower layers produce features of lower complexity.

layer = 'mixed3b_1x1_pre_relu' channel = 121
Deep Dream

Optimizing a linear combination of features often gives a "mixture" pattern. (Channels 139 + 65)
https://github.com/bapoczos/deep-dream-tensorflow/blob/master/deepdream.ipynb
Deep Dream

Starting from an image instead of noise
Maximizing the sum of squared activities on the ‘mixed4c’ layer
Channel 139: Deep Dream
Caption Generation
Caption Generation

Implementations:

- Google’s tensorflow: Im2txt
  [https://github.com/tensorflow/models/tree/master/im2txt](https://github.com/tensorflow/models/tree/master/im2txt)


- Karpathy’s Neuraltalk2 Torch:
  [https://github.com/karpathy/neuraltalk2](https://github.com/karpathy/neuraltalk2)
Examples

A person on a beach flying a kite.

A black and white photo of a train on a train track.
Examples

A person skiing down a snow covered slope.

A group of giraffe standing next to each other.
Computer Vision + Natural Language Processing

Xu et al, 2015

1. Input Image
2. Convolutional Feature Extraction
3. RNN with attention over the image
4. Word by word generation

A bird flying over a body of water

A woman is throwing a **frisbee** in a park.

A little **girl** sitting on a bed with a teddy bear.
A stop sign is on a road with a mountain in the background.

A woman is sitting at a table with a large pizza.
Word Embedding, Word2Vec


The Skip-Gram model

The goal is to maximize

$$\frac{1}{T} \sum_{t=1}^{T} \sum_{-c \leq j \leq c, j \neq 0} \log p(w_{t+j} | w_t)$$

where

$$p(w_O | w_I) = \frac{\exp \left( v'_w^\top v_{w_I} \right)}{\sum_{w=1}^{W} \exp \left( v'_w^\top v_{w_I} \right)}$$

The Skip-gram model architecture. The training objective is to learn word vector representations that are good at predicting the nearby words.
The Continuous Bag-of-Words (CBOW) Model

The CBOW architecture predicts the current word based on the context.
Two-dimensional PCA projection of the 1000-dimensional Skip-gram vectors of countries and their capital cities. The figure illustrates ability of the model to automatically organize concepts and learn implicitly the relationships between them, as during the training we did not provide any supervised information about what a capital city means.
Recurrent Neural Networks

\[ h(t) = f_H \left( W_{IH} x(t) + W_{HH} h(t - 1) \right) \]
\[ y(t) = f_O \left( W_{HO} h(t) \right) \]
A LSTM block contains gates that determine when the input is significant enough to remember, when it should continue to remember or forget the value, and when it should output the value.

\[
\begin{align*}
    f_t &= \sigma_g(W_f x_t + U_f h_{t-1} + b_f) \\
    i_t &= \sigma_g(W_i x_t + U_i h_{t-1} + b_i) \\
    o_t &= \sigma_g(W_o x_t + U_o h_{t-1} + b_o) \\
    c_t &= f_t \circ c_{t-1} + i_t \circ \sigma_c(W_c x_t + U_c h_{t-1} + b_c) \\
    h_t &= o_t \circ \sigma_h(c_t)
\end{align*}
\]

Variables

- \(x_t\): input vector
- \(h_t\): output vector
- \(c_t\): cell state vector
- \(W, U\) and \(b\): parameter matrices and vector
- \(f_t, i_t\) and \(o_t\): gate vectors
- \(f_t\): Forget gate vector. Weight of remembering old information.
- \(i_t\): Input gate vector. Weight of acquiring new information.
- \(o_t\): Output gate vector.

http://colah.github.io/posts/2015-08-Understanding-LSTMs/
\{s_0, s_1, ..., s_{N-1}\} are the words of the caption and \{w_es_0, w_es_1, ..., w_es_{N-1}\} are their corresponding word embedding vectors.

The outputs \{p_1, p_2, ..., p_N\} of the LSTM are probability distributions generated by the model for the next word in the sentence. The terms \{\log p_1(s_1), \log p_2(s_2), ..., \log p_N(s_N)\} are the log-likelihoods of the correct word at each step.
Andrej Karpathy's "NeuralTalk2" code slightly modified to run from a webcam feed [github.com/karpathy/neuraltalk2 ]

NeuralTalk is trained on the MS COCO dataset [mscoco.org/dataset/#captions-challenge2015]

MS COCO contains 100k image-caption pairs

All processing is done on a 2013 MacBook Pro with the NVIDIA 750M and only 2GB of GPU memory.

Video recording: Walking around with the laptop open

The openFrameworks code for streaming the webcam and reading from disk is available at [gist.github.com/kylemcdonald/b02edbc33942a85856c8]

While the captions run at about four captions per second on the laptop, in this video one caption per second was generated to make it more reasonable.
Video Caption Generation

https://vimeo.com/146492001 from Kyle McDonald
Visual Question Answering
Visual Question Answering


“How many horses are in this image?”

Demo: https://cloudcv.org/vqa/
What is he doing?

<table>
<thead>
<tr>
<th>Answer</th>
<th>Confidence</th>
</tr>
</thead>
<tbody>
<tr>
<td>playing tennis</td>
<td>0.9071</td>
</tr>
<tr>
<td>skateboarding</td>
<td>0.0084</td>
</tr>
<tr>
<td>tennis</td>
<td>0.0070</td>
</tr>
<tr>
<td>yellow</td>
<td>0.0020</td>
</tr>
<tr>
<td>blue</td>
<td>0.0017</td>
</tr>
</tbody>
</table>

Demo: https://cloudcv.org/vqa/
What is the color of his shirt?

- white: 0.2547
- blue: 0.1810
- black: 0.1604
- yellow: 0.1192
- red: 0.0699

Demo: https://cloudcv.org/vqa/
Where was this picture taken?

Answer | Confidence
---|---
baseball field | 0.1447

Demo: https://cloudcv.org/vqa/
Thanks for your Attention! 😊