Lecture 12:

Imposing Task-Specific Structure on DNNs

Visual Computing Systems
CMU 15-769, Fall 2016
Today

- Four examples of DNN authors imposing structure on networks to better perform a desired task
- For each example, consider:
  - What knowledge does the human inject?
  - What does the computer learn?

- Image compression networks [Toderici ICLR 16]
- Cross-stitch networks: sharing lower DNN layers [Misra CVPR 2016]
- Spatial transformers [Jaderberg NIPS 2015]
- Convolutional pose machines for pose estimation [Wei CVPR16]
Image compression using DNNs
Review: JPG image compression

- Lossy compression designed to retain information that is most important to human perception
- Human-designed compact representation

![Diagram of image compression process]

- Coefficient reordering
- RLE compression of zeros
- Entropy compression of non-zeros
- Compressed bits

**Steps of JPG image compression**

1. **Coefficient reordering**
2. **DCT**
3. **Quantization Matrix**
4. **Quantized DCT**
5. **RLE compression of zeros**
6. **Entropy compression of non-zeros**
7. **Compressed bits**
Deep learning learns useful representations

- Can we apply deep learning techniques to obtain compact image representations for efficient storage and transmission?

- Class discussion: Why?
Example use case

Compressing 32x32 8-bit RGB thumbnails (24 bpp)

\[ x' = D(B(E(x))) \]

Auto-encoder: learn to compress (encode) and reconstruct (decode) the input signal

- Jointly train \( D, B, \) and \( E \) using supervision from \( \text{Loss}(x, x') \)
Progressive encoding: chain copies of autoencoder (each iteration contributes bits)

Input: 32x32x3 residual \( r_{t-1} \)

Encoder (E) \( \rightarrow \) Binarization Function (B) \( \rightarrow \) Decoder (D) \( \rightarrow \) Output: 32x32x3

\[
F_t(r_{t-1}) = D_t(B(E_t(r_{t-1})))
\]

\( r_0 = \) input image to compress

Version 1: each iteration predicts the residual

\[
r_t = F_t(r_{t-1}) - r_{t-1}
\]

\[
x' = \sum_{t=1}^{N} F_t(r_{t-1})
\]

Version 2: (stateful E() and D() units) each iteration predicts input image

\[
r_t = F_t(r_{t-1}) - r_0
\]

\[
x' = F_N(r_{N-1})
\]

In both cases, loss given by  \( \|r_t\|_2^2 \) for all \( t \)

[Toderici ICLR 16]
Binarization

- Step 1: output of encoder passes through fully-connected layer with \( m \) outputs (to “squeeze” to desired number of outputs)
- Step 2: quantize each output to a bit

\[
B(x) = f(tanh(Wx + b))
\]

\[
f(x) = \begin{cases} 
-1 & x < 0 \\
+1 & x \geq 0
\end{cases}
\]

Add random perturbation during training (regularization):

\[
f(x) = x + \epsilon
\]

\[
\epsilon \sim \begin{cases} 
1 - x & \text{with probability } \frac{1+x}{2}, \\
-x - 1 & \text{with probability } \frac{1-x}{2},
\end{cases}
\]
Version 1 autoencoder

![Autoencoder Diagram]

Fully-connected version:

Input is 8x8 block

Each fully connected layer has 512 outputs and tanh non-linearity

Each iteration through auto encoder yields 4 bits (two iterations shown)
Version 1 autoencoder (convolutional)

Convolutional version:
2 bits per spatial location of output per iteration
32x32 input → 8x8 spatial outputs (128 bits per iteration)
Version 2 autoencoder (LSTM-based)

(Convolutional form also exists)

LSTM version: predicts source image each iteration (not a residual)

LSTM units:
- Recurrent: output from iteration \( t-1 \) fed into unit in iteration \( t \)
- Stateful: each unit maintains its own hidden state
We describe various methods for variable-length encoding of image patches using neural networks, with JPEG, while the convolutional/deconvolutional LSTM model is able to significantly outperform JPEG on the SSIM perceptual metric. We demonstrate that for the given benchmark, the fully-connected LSTM model can perform on par and produces SSIM values that are equivalent to the autoencoder, even though the resulting model is more flexible.

In terms of coding efficiency, we took an autoencoder architecture (one iteration of the model presented in Section 3.5) with a given bit budget of either 64 or 128 bytes, and compared its SSIM against the (de)convolutional LSTM encoder at these targets. In both cases, the LSTM model reduce JPEG encoding quality in order to produce 4:4:4 JPEGs at a comparable bitrate to the LSTM architecture.

**Compression results**

![Original (32×32)](image)

<table>
<thead>
<tr>
<th>Average bpp:</th>
<th>JPEG</th>
<th>0.641</th>
<th>0.875</th>
<th>1.117</th>
<th>1.375</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>LSTM</td>
<td>0.625</td>
<td>0.875</td>
<td>1.125</td>
<td>1.375</td>
</tr>
<tr>
<td></td>
<td>(De)Convolutional LSTM</td>
<td>0.625</td>
<td>0.875</td>
<td>1.125</td>
<td>1.375</td>
</tr>
</tbody>
</table>

From left to right:

- **JPEG compressed images**
- **Compressed images with LSTM architecture**
- **Compressed images with conv/deconv LSTM architecture**
# Compression results

<table>
<thead>
<tr>
<th>Patch Size</th>
<th>SSIM / 64B Target (Header-less Size)</th>
<th>SSIM / 128B Target (Header-less Size)</th>
</tr>
</thead>
<tbody>
<tr>
<td>Header-less JPEG 8×8</td>
<td>0.70 (72.5 bytes avg.)</td>
<td>0.80 (133 bytes avg.)</td>
</tr>
<tr>
<td>Header-less JPEG 2000 8×8</td>
<td>0.66 (73 bytes avg.)</td>
<td>0.77 (156 bytes avg.)</td>
</tr>
<tr>
<td>Header-less WebP 8×8</td>
<td>0.62 (80.7 bytes avg.)</td>
<td>0.73 (128.2 bytes avg.)</td>
</tr>
<tr>
<td>Fully Connected Residual Encoder (Shared Weights) 8×8</td>
<td>0.46</td>
<td>0.48</td>
</tr>
<tr>
<td>Fully Connected Residual Encoder (Distinct Weights) 8×8</td>
<td>0.65</td>
<td>0.75</td>
</tr>
<tr>
<td><strong>LSTM Compressor</strong> 8×8</td>
<td><strong>0.69</strong></td>
<td><strong>0.81</strong></td>
</tr>
<tr>
<td>Conv/Deconv Residual Encoder (Shared Weights) 32×32</td>
<td>0.45</td>
<td>0.46</td>
</tr>
<tr>
<td>Conv/Deconv Residual Encoder (Distinct Weights) 32×32</td>
<td>0.65</td>
<td>0.75</td>
</tr>
<tr>
<td>Convolutional/Deconvolutional Autoencoder 32×32</td>
<td>0.76</td>
<td>0.86</td>
</tr>
<tr>
<td>Conv/Deconv LSTM Compressor 32×32</td>
<td><strong>0.77</strong></td>
<td><strong>0.87</strong></td>
</tr>
</tbody>
</table>

SSIM: structural similarity index

Table 1 summarizes the results on the 32×32 benchmark, comparing our two LSTM approaches to two JPEG codecs and to WebP. To avoid unfairly penalizing the codecs due to the unavoidable cost of their file headers, we exclude the header size from all metrics. Note also that since these standard codecs cannot be tuned to an exact byte budget (e.g., 64 bytes excluding the file header), we search for the encoder quality setting that leads to a file whose size is as close as possible, but never less than, the target size. On average, this leads to each JPEG and WebP image consuming slightly more space than we allow for the LSTM models.

These 32×32 images contain considerable detail that is perceptually relevant. As can be seen in Figure 4, compressing these images without destroying salient visual information or hallucinating false details is challenging. At these very low bitrates and spatial resolution, JPEG block artifacts become extremely prominent, and WebP either introduces blocking or overly blurs the image depending on the strength of the internal filter. Color smearing artifacts due to the codecs' default (4:2:0) chroma subsampling are also clearly visible.

Compared to JPEG, the non-convolutional LSTM model slightly reduces inter-block boundaries on some images but can also lead to increased color bleeding (e.g., on mandrill as shown in Figure 4). Furthermore, the visual quality never exceeds JPEG on average as measured by SSIM and shown in Figure 5. This motivates the (de)convolutional LSTM model, which eliminates block artifacts while avoiding excessive smoothing. It strikes the best balance between preserving real detail and avoiding color smearing, false gradients, and hallucinated detail not present in the original image.

Note that the (de)convolutional LSTM model exhibits perceptual quality levels that are equal to or better than both JPEG and WebP at 4%–12% lower average bitrate. We see this improvement despite the fact that, unlike JPEG and WebP, the LSTMs do not perform chroma subsampling as a preprocess. However, at the JPEG quality levels used in Figure 4, disabling subsampling (i.e., using 4:4:4 encoding) leads to a costly increase in JPEG’s bitrate: 1.32–1.77 bpp instead of 1.05–1.406 bpp, or 26% greater. This means that if we desired to preserve chroma fidelity, we would need to drastically increase the bitrate.
**Summary / thoughts**

- **Idea:** learn how to compress thumbnail-sized images by trying to compress large database of tiny images
  - Loss is not perceptually motivated (if there was a differentiable perceptual loss metric, they would have used it instead of L2 on pixel residual)

- **Improvement on JPG for small images, future work extends to large images by exploiting global redundancy [Toderici 2016]**

- **Why use learning for this problem?**
  - Potential for higher quality encode (learn better representations than humans can manually craft)
  - General mechanism to specialize representations for task
    - [Toderici 2016]: specific to thumbnail images
    - What about camera-viewpoint specific compression?
    - Task-based definition of loss rather than pixels (compress subject to still being able to recognize objects)
Camera-specific compression?

Security cameras (stationary)

Head mounted cameras

On-vehicle cameras
Cross-Stitch Networks
Recall object classification networks

Lower levels of network “shared” across all categories
(Lower level convolutions produce useful features)

Input:
fixed size image

Output:
probability of label
(for 1000 class labels)

VGG-16
Recall Faster R-CNN

Lower conv layers shared between two tasks:
(1) object bounding box prediction and (2) object detection

Input image:
(of any size)

DNN (conv layers)

Response maps $W \times H \times 512$

512 3x3 conv filters
(3x3x512x512 weights)

1x1 conv
(2-way softmax)
512 x (9*2) weights

Objectness score
(for 9 boxes)

1x1 conv
(bbox regressor)
512 x (9x4) weights

Bbox offset
(for 9 boxes)

List of proposed regions

ROI pooling layer

Pixel region
(of canonical size)

Fully-connected layers

Class-label softmax

bbox regression softmax

bbox

Object label

List of proposed regions

CMU 15-769, Fall 2016
Multi-task learning

- For better accuracy
  - Representations learned to successfully perform multiple tasks A, B, C may be more general, less prone to overfitting, etc.
  - One task serves as a form of supervision for another (as there are effectively more examples available to train the shared lower layers of the network)

- For increased computational efficiency
  - Share early layer computations across many tasks (consider performing N related tasks on an input video stream)
Example: various topologies for multi-class learning

[Image from: Misra et al. CVPR 2016]
Idea: learn how much to share weights

**Setting** $\alpha_{AB}$ or $\alpha_{BA}$ to 0 implies no input sharing.
Setting row's coefficients to same value implies full sharing.

$$
\begin{bmatrix}
\tilde{x}_{ij} \\
\tilde{x}_A \\
\tilde{x}_B
\end{bmatrix} =
\begin{bmatrix}
\alpha_{AA} & \alpha_{AB} \\
\alpha_{BA} & \alpha_{BB}
\end{bmatrix}
\begin{bmatrix}
x_{ij} \\
x_A \\
x_B
\end{bmatrix}
$$

[Image from: Misra et al. CVPR 2016]
Network initialization:

Cross-stitch units initialization and learning rates:

By varying values, the unit can freely move between shared and task-specific representations, and choose a middle ground if needed.

Next, we enumerate the design decisions when using cross-stitch units with networks, and in later sections per-

“data-starved” tasks.

We then finetune the network (referred to as based on CaffeNet \[\text{29}\] which have their angular distance under a threshold (de-

Networks:

The sub-network in Figure \[\text{4}\] shows this architecture

Since \[\text{task}\] have less labels than the other, such regularization helps the

Networks:

We denote \[\text{Network } A\]

Network B

Datasets and Tasks:

We use the cross-stitch unit for multi-task learning in

ConvNets. For the sake of simplicity, we assume multi-task

A

\[\text{Task A}\]

B

\[\text{Task B}\]

The Manhattan-World post-processing following the method

cast the problem of surface normal prediction as classifi-

We call the sub-network that gets direct supervision from task

Task A

Task B

With the model predictions to 3D surface normals and apply

A

\[\text{Task A}\]

B

\[\text{Task B}\]

Figure \[\text{4}\]: Using cross-stitch units to stitch two AlexNet \[\text{29}\]

\[\text{Image from: Misra et al. CVPR 2016}\]

\[\text{Image from: Misra et al. CVPR 2016}\]
Example result: segmentation + normal estimation task

<table>
<thead>
<tr>
<th>Method</th>
<th>Surface Normal</th>
<th>Segmentation (Higher Better)</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>Angle Distance</td>
<td>Within $t^\circ$</td>
</tr>
<tr>
<td></td>
<td>(Lower Better)</td>
<td>(Higher Better)</td>
</tr>
<tr>
<td></td>
<td>Mean</td>
<td>Med.</td>
</tr>
<tr>
<td>One-task</td>
<td>34.8</td>
<td>19.0</td>
</tr>
<tr>
<td>Ensemble</td>
<td>34.4</td>
<td>18.5</td>
</tr>
<tr>
<td>Split conv4</td>
<td>34.7</td>
<td>19.1</td>
</tr>
<tr>
<td>MTL-shared</td>
<td>34.7</td>
<td>18.9</td>
</tr>
<tr>
<td>Cross-stitch [ours]</td>
<td>34.1</td>
<td>18.2</td>
</tr>
</tbody>
</table>
Spatial Transformer Networks
Common pattern

- Train DNN for perform task on canonical image form
- Example, R-CNN expects contents of boxes to be scaled to fixed-size input
  - This is a crop of original image, followed by resample

    for all candidate boxes \((x,y,w,h)\):
    cropped = image_crop(image, bbox(x,y,w,h))
    resized = image_resize(227,227)
    label = detect_object(resized)
    if (label != background)
      // region defined by bbox(x,y,w,h) contains object
      // of class ‘label’

- Recall Faster R-CNN: given pixel pattern... predict box, then crop/resample to canonicalize box’s contents
Generalization: learning to canonicalize

- Modular thinking:
  - Step 1: canonicalize
  - Step 2: perform task (e.g., detect)

- Why not jointly learn network to perform task, and network to canonicalize the input?
  - (In other words, learn to be spatial invariant)
Spatial transformer network

Given input activation predict appropriate transform parameters

Resample to perform affine transformation of input (differentiable resampling operation)

\[
\begin{pmatrix}
x_i^s \\
y_i^s
\end{pmatrix} = \mathcal{T}_\theta(G_i) = A_\theta \begin{pmatrix}
x_i^t \\
y_i^t \\
1
\end{pmatrix} = \begin{bmatrix}
\theta_{11} & \theta_{12} & \theta_{13} \\
\theta_{21} & \theta_{22} & \theta_{23}
\end{bmatrix} \begin{pmatrix}
x_i^t \\
y_i^t \\
1
\end{pmatrix}
\]

\[\mathcal{T}_\theta(G')\]
Convolutional Pose Machines
Problem statement

- Given image containing a human, output the \((x,y)\) position of each of the human’s \(P\) parts (parts are joints)
Knowledge

- There are P parts
  - No connectivity or constraints are given

- Context is useful: for any part $p_i$, knowing the position of other parts may be useful to localizing $p_i$
  - If I know where the knee is, that should provide information about where the foot is likely to be

- Iteratively update belief about distributions until convergence
Convolutional pose machine (CPM)

Stage 1: estimate part positions from local information (160x160 receptive field)

Stage N: accepts original image (locally computed parts) and previously predicted locations of all parts from stage N-1

[Wei CVPR16]
Example contextual refinement

Localization of shoulder, head, neck (easy to detect in stage 1) help eliminate incorrect distribution (from stage 1) for position of elbow
Takeaway

- Designers of modern networks impose structure on topology based on human knowledge of how a solution to the task at hand should proceed (topology suggests basic structure of solution)
  - e.g., share these layers
  - e.g., canonicalize and then detect
  - e.g., the number of joints in a human

- Use end-to-end learning to learn the “details” that would be hard (or tedious) for a human to craft
  - e.g., how much to share for each layer
  - e.g., how to canonicalize
  - e.g., how does context help localize human joints in an image