CNNs and Computer Vision

Applied Machine Learning
Derek Hoiem
<table>
<thead>
<tr>
<th>Week</th>
<th>Date</th>
<th>Topic</th>
<th>Link</th>
<th>Reading/Notes</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>Jan 17 (Tues)</td>
<td>Introduction</td>
<td><code>pdf</code>: <code>pdf</code></td>
<td>Jupyter notebook tutorial vid <code>synth cc</code> Numpy tutorial vid <code>cc</code> Linear algebra tutorial vid <code>cc</code></td>
</tr>
<tr>
<td></td>
<td></td>
<td><strong>Supervised Learning Fundamentals</strong></td>
<td></td>
<td></td>
</tr>
<tr>
<td>1</td>
<td>Jan 19 (Thurs)</td>
<td>KNN, key concepts in ML</td>
<td><code>pdf</code>: <code>pdf</code></td>
<td>AML Ch 1</td>
</tr>
<tr>
<td>2</td>
<td>Jan 24 (Tues)</td>
<td>Probability and Naive Bayes</td>
<td><code>pdf</code>: <code>pdf</code></td>
<td>AML Ch 1</td>
</tr>
<tr>
<td>2</td>
<td>Jan 26 (Thurs)</td>
<td>Linear Least Squares and Logistic Regression</td>
<td><code>pdf</code>: <code>pdf</code></td>
<td>AML 10.1-10.2, 11</td>
</tr>
<tr>
<td>3</td>
<td>Jan 31 (Thurs)</td>
<td>Decision Trees</td>
<td><code>pdf</code>: <code>pdf</code></td>
<td>AML Ch 2</td>
</tr>
<tr>
<td>3</td>
<td>Feb 2 (Thurs)</td>
<td>Consolidation and Review</td>
<td><code>pdf</code>: <code>pdf</code></td>
<td>AML Ch 2</td>
</tr>
<tr>
<td>3</td>
<td>Feb 6 (Mon)</td>
<td>HW 1 (Classification &amp; Regression) due</td>
<td><code>pdf</code>: <code>pdf</code></td>
<td>AML Ch 2</td>
</tr>
<tr>
<td>4</td>
<td>Feb 7 (Thurs)</td>
<td>Ensembles and Random Forests</td>
<td><code>pdf</code>: <code>pdf</code></td>
<td>AML Ch 2, Ch 12</td>
</tr>
<tr>
<td>4</td>
<td>Feb 9 (Thurs)</td>
<td>SVMs and SGD</td>
<td><code>pdf</code>: <code>pdf</code></td>
<td>AML Ch 2</td>
</tr>
<tr>
<td>5</td>
<td>Feb 14 (Tues)</td>
<td>MLPs and Backprop</td>
<td><code>pdf</code>: <code>pdf</code></td>
<td>AML Ch 16</td>
</tr>
<tr>
<td>5</td>
<td>Feb 16 (Thurs)</td>
<td>Deep Learning</td>
<td><code>pdf</code>: <code>pdf</code></td>
<td>AML Ch 16 / RedNet (He et al, 2016)</td>
</tr>
<tr>
<td>6</td>
<td>Feb 21 (Tues)</td>
<td>Consolidation and Review</td>
<td><code>pdf</code>: <code>pdf</code></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td><strong>Vision, Language, and Applications</strong></td>
<td></td>
<td></td>
</tr>
<tr>
<td>6</td>
<td>Feb 23 (Thurs)</td>
<td>CNNs in Computer Vision</td>
<td><code>pdf</code>: <code>pdf</code></td>
<td>AML Ch 17-18</td>
</tr>
<tr>
<td>7</td>
<td>Feb 27 (Mon)</td>
<td>HW 2 (Trees &amp; MLPs) due</td>
<td><code>pdf</code>: <code>pdf</code></td>
<td></td>
</tr>
<tr>
<td>7</td>
<td>Feb 28 (Tues)</td>
<td>Language Models</td>
<td><code>pdf</code>: <code>pdf</code></td>
<td></td>
</tr>
<tr>
<td>7</td>
<td>Mar 2 (Thurs)</td>
<td>Transformers in Language and Vision</td>
<td><code>pdf</code>: <code>pdf</code></td>
<td></td>
</tr>
<tr>
<td>8</td>
<td>Mar 7 (Tues)</td>
<td>Foundation Models: CLIP and GPT-3</td>
<td><code>pdf</code>: <code>pdf</code></td>
<td></td>
</tr>
<tr>
<td>8</td>
<td>Mar 9 (Thurs)</td>
<td>Exam 1 (on PrairieLearn)</td>
<td><code>pdf</code>: <code>pdf</code></td>
<td></td>
</tr>
<tr>
<td>9</td>
<td>Mar 11-19</td>
<td>Spring Break (no classes)</td>
<td><code>pdf</code>: <code>pdf</code></td>
<td></td>
</tr>
<tr>
<td>10</td>
<td>Mar 21 (Tues)</td>
<td>Task Adaptation</td>
<td><code>pdf</code>: <code>pdf</code></td>
<td></td>
</tr>
<tr>
<td>10</td>
<td>Mar 23 (Thurs)</td>
<td>Fairness and Impact on society</td>
<td><code>pdf</code>: <code>pdf</code></td>
<td></td>
</tr>
<tr>
<td>11</td>
<td>Mar 27 (Mon)</td>
<td>HW 3 (Application Domains) due</td>
<td><code>pdf</code>: <code>pdf</code></td>
<td></td>
</tr>
<tr>
<td>11</td>
<td>Mar 28 (Tues)</td>
<td>Big Data and Dataset Bias</td>
<td><code>pdf</code>: <code>pdf</code></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td><strong>Pattern Discovery</strong></td>
<td></td>
<td></td>
</tr>
<tr>
<td>11</td>
<td>Mar 30 (Thurs)</td>
<td>K-Means, KD-tree, LSH</td>
<td><code>pdf</code>: <code>pdf</code></td>
<td>AML Ch 8</td>
</tr>
<tr>
<td>12</td>
<td>Apr 4 (Tues)</td>
<td>Missing Data and EM</td>
<td><code>pdf</code>: <code>pdf</code></td>
<td>AML Ch 9</td>
</tr>
<tr>
<td>12</td>
<td>Apr 6 (Thurs)</td>
<td>Density estimations: MoG, Kernels, Hists</td>
<td><code>pdf</code>: <code>pdf</code></td>
<td>AML Ch 9</td>
</tr>
<tr>
<td>13</td>
<td>Apr 11 (Tues)</td>
<td>Data visualization: PCA and t-SNE</td>
<td><code>pdf</code>: <code>pdf</code></td>
<td>AML Ch 11</td>
</tr>
<tr>
<td>13</td>
<td>Apr 13 (Thurs)</td>
<td>Topic Modeling</td>
<td><code>pdf</code>: <code>pdf</code></td>
<td></td>
</tr>
<tr>
<td>14</td>
<td>Apr 17 (Mon)</td>
<td>HW 4 (Pattern Discovery) due</td>
<td><code>pdf</code>: <code>pdf</code></td>
<td></td>
</tr>
<tr>
<td>14</td>
<td>Apr 18 (Tues)</td>
<td>CCA</td>
<td><code>pdf</code>: <code>pdf</code></td>
<td>AML Ch 6, Ch 19</td>
</tr>
<tr>
<td></td>
<td></td>
<td><strong>More Applications and Topics</strong></td>
<td></td>
<td></td>
</tr>
<tr>
<td>14</td>
<td>Apr 20 (Thurs)</td>
<td>Audio</td>
<td><code>pdf</code>: <code>pdf</code></td>
<td>Audio Deep Learning</td>
</tr>
<tr>
<td>15</td>
<td>Apr 25 (Tues)</td>
<td>TBD</td>
<td><code>pdf</code>: <code>pdf</code></td>
<td></td>
</tr>
<tr>
<td>16</td>
<td>May 2 (Tues)</td>
<td>Looking Forward &amp; Requested Topics</td>
<td><code>pdf</code>: <code>pdf</code></td>
<td></td>
</tr>
</tbody>
</table>
Today’s Lecture

• ImageNet Challenge Overview
• ResNet model in more detail
• Adapting a pre-trained network to new tasks
• Mask-RCNN line of detection/segmentation
• U-Net Architecture
IMAGENET

22K categories and 15M images

- Animals
  - Bird
  - Fish
  - Mammal
  - Invertebrate
- Plants
  - Tree
  - Flower
  - Food
  - Materials
- Structures
  - Artifact
  - Tools
  - Appliances
  - Structures
- Person
- Scenes
  - Indoor
  - Geological Formations
  - Sport Activity

www.image-net.org

Deng et al. 2009,
Russakovsky et al. 2015
Step 1: Ontological structure based on WordNet
Step 2: Populate categories with thousands of images from the Internet
Dog

German Shepherd

Step 3: Clean results by hand
Crowdsourcing

ImageNet
PhD
Students

Crowdsourced Labor

Amazon Mechanical Turk™
Artificial Intelligence

49k Workers from 167 Countries
2007-2010

Slide credit: Fei-Fei Li and Jia Deng
ILSVRC image classification task

Steel drum

Output: Scale T-shirt Steel drum Drumstick Mud turtle

Output: Scale T-shirt Giant panda Drumstick Mud turtle

Error = \frac{1}{100,000} \sum_{i} 1[\text{incorrect on image } i]
2012 ImageNet 1K
(Fall 2012)

Error

LEAR-XRCE
U. of Amsterdam
XRCE/INRIA
Oxford
ISI
SuperVision

AlexNet
AlexNet: ILSVRC 2012 winner

- Similar framework to LeNet but:
  - Max pooling, **ReLU nonlinearity**
  - **More data** and **bigger model** (7 hidden layers, 650K units, 60M params)
  - GPU implementation (**50x speedup** over CPU)
    - Trained on two GPUs for a week
  - Dropout regularization

ResNet: the residual module

- Use *skip* or *shortcut* connections around 2-3 layer MLPs
- Gradients can flow quickly back through skip connections
- Each module needs only add information to the previous layers

Kaiming He, Xiangyu Zhang, Shaoqing Ren, and Jian Sun, *Deep Residual Learning for Image Recognition*, CVPR 2016 (Best Paper), 150K+ citations
ResNet: Residual Bottleneck Module

- Directly performing 3x3 convolutions with 256 feature maps at input and output: 
  \[256 \times 256 \times 3 \times 3 \approx 600K \text{ operations}\]

- Using 1x1 convolutions to reduce 256 to 64 feature maps, followed by 3x3 convolutions, followed by 1x1 convolutions to expand back to 256 maps:
  \[256 \times 64 \times 1 \times 1 \approx 16K \]
  \[64 \times 64 \times 3 \times 3 \approx 36K \]
  \[64 \times 256 \times 1 \times 1 \approx 16K \]
  Total: \(\approx 70K\)

Kaiming He, Xiangyu Zhang, Shaoqing Ren, and Jian Sun, Deep Residual Learning for Image Recognition, CVPR 2016 (Best Paper)
ResNet: going real deep

Revolution of Depth

AlexNet, 8 layers (ILSVRC 2012)  VGG, 19 layers (ILSVRC 2014)  ResNet, 152 layers (ILSVRC 2015)

Despite depth, the residual connections enable error gradients to “skip” all the way back to the beginning

Kaiming He, Xiangyu Zhang, Shaoqing Ren, and Jian Sun, Deep Residual Learning for Image Recognition, CVPR 2016
ResNet CNN Components

• Conv2d: learned 2D convolutional filters (same linear weights applied to a patch surrounding each pixel)
• BatchNorm2D: Convolutional batch normalization (see next slide)
• ReLU: non-linearity with gradient={0,1}
• Linear layer: feature projection / final linear classifier
Batch Normalization

- During training, the feature distribution at intermediate layers keep changing as the network learns.
- This destabilizes training.
- BatchNorm normalizes features of each mini-batch according to its mean and variance and learned parameters γ, β.
- Using BatchNorm often improves speed and effectiveness of training.

---

**Input:** Values of $x$ over a mini-batch: $B = \{x_1, \ldots, x_m\}$

**Parameters to be learned:** $\gamma, \beta$

**Output:** $\{y_i = \text{BN}_{\gamma,\beta}(x_i)\}$

\[
\begin{align*}
\mu_B & \leftarrow \frac{1}{m} \sum_{i=1}^{m} x_i \quad \text{// mini-batch mean} \\
\sigma_B^2 & \leftarrow \frac{1}{m} \sum_{i=1}^{m} (x_i - \mu_B)^2 \quad \text{// mini-batch variance} \\
\hat{x}_i & \leftarrow \frac{x_i - \mu_B}{\sqrt{\sigma_B^2 + \epsilon}} \quad \text{// normalize} \\
y_i & \leftarrow \gamma \hat{x}_i + \beta \equiv \text{BN}_{\gamma,\beta}(x_i) \quad \text{// scale and shift}
\end{align*}
\]

---

Batch Normalization: Accelerating Deep Network Training by Reducing Internal Covariate Shift [Ioffe and Szegedy 2015]
Example code: ResBlock

```python
class ResBlock(nn.Module):
    def __init__(self, in_channels, out_channels, downsample):
        super().__init__()
        if downsample:
            self.conv1 = nn.Conv2d(in_channels, out_channels, kernel_size=3, stride=2, padding=1)
            self.shortcut = nn.Sequential(
                nn.Conv2d(in_channels, out_channels, kernel_size=1, stride=2),
                nn.BatchNorm2d(out_channels)
            )
        else:
            self.conv1 = nn.Conv2d(in_channels, out_channels, kernel_size=3, stride=1, padding=1)
            self.shortcut = nn.Sequential()

        self.conv2 = nn.Conv2d(out_channels, out_channels, kernel_size=3, stride=1, padding=1)
        self.bn1 = nn.BatchNorm2d(out_channels)
        self.bn2 = nn.BatchNorm2d(out_channels)

    def forward(self, input):
        shortcut = selfshortcut(input)
        input = nn.ReLU()(self.bn1(self.conv1(input)))
        input = nn.ReLU()(self.bn2(self.conv2(input)))
        input = input + shortcut
        return nn.ReLU()(input)
```

"channels" = # feature maps
kernel_size = filter size, e.g. 3x3
stride = # pixels to skip when evaluating convolution
padding: to calculate filter values near edge of image/map

If downsampling, do it here too so dimensions match

This ‘+’ is the skip connection!
class Network(nn.Module):
    def __init__(self, num_classes=1000):
        super().__init__()
        resblock = ResBlock
        self.layer0 = nn.Sequential(
            nn.Conv2d(3, 64, kernel_size=7, stride=2, padding=3),
            nn.MaxPool2d(kernel_size=3, stride=2, padding=1),
            nn.BatchNorm2d(64),
            nn.ReLU()
        )
        self.layer1 = nn.Sequential(
            resblock(64, 64, downsample=False),
            resblock(64, 64, downsample=False)
        )
        self.layer2 = nn.Sequential(
            resblock(64, 128, downsample=True),
            resblock(128, 128, downsample=False)
        )
        self.layer3 = nn.Sequential(
            resblock(128, 256, downsample=True),
            resblock(256, 256, downsample=False)
        )
        self.layer4 = nn.Sequential(
            resblock(256, 512, downsample=True),
            resblock(512, 512, downsample=False)
        )
        self.gap = torch.nn.AdaptiveAvgPool2d(1)
        self.fc = torch.nn.Linear(512, num_classes)

    def forward(self, input):
        input = self.layer0(input)
        input = self.layer1(input)
        input = self.layer2(input)
        input = self.layer3(input)
        input = self.layer4(input)
        input = self.gap(input)
        input = torch.flatten(input, 1)
        input = self.fc(input)
        return input

Pretrained Torch models
### ResNet Architectures and Results

<table>
<thead>
<tr>
<th>layer name</th>
<th>output size</th>
<th>18-layer</th>
<th>34-layer</th>
<th>50-layer</th>
<th>101-layer</th>
<th>152-layer</th>
</tr>
</thead>
<tbody>
<tr>
<td>conv 1</td>
<td>112 x 112</td>
<td>7x7, 64, stride 2</td>
<td>3x3 max pool, stride 2</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>conv 2, x</td>
<td>56 x 56</td>
<td>[3x3, 64] x2</td>
<td>[3x3, 64] x3</td>
<td>[1x1, 64] x3</td>
<td>[1x1, 64] x3</td>
<td>[1x1, 64] x3</td>
</tr>
<tr>
<td>conv 3, x</td>
<td>28 x 28</td>
<td>[3x3, 128] x2</td>
<td>[3x3, 128] x4</td>
<td>[1x1, 128] x4</td>
<td>[1x1, 128] x4</td>
<td>[1x1, 128] x4</td>
</tr>
<tr>
<td>conv 4, x</td>
<td>14 x 14</td>
<td>[3x3, 256] x2</td>
<td>[3x3, 256] x6</td>
<td>[1x1, 256] x6</td>
<td>[1x1, 256] x23</td>
<td>[1x1, 256] x36</td>
</tr>
<tr>
<td>conv 5, x</td>
<td>7 x 7</td>
<td>[3x3, 512] x2</td>
<td>[3x3, 512] x3</td>
<td>[1x1, 512] x3</td>
<td>[1x1, 512] x3</td>
<td>[1x1, 512] x3</td>
</tr>
<tr>
<td>1 x 1</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

Average pool, 1000-d fc, softmax

<table>
<thead>
<tr>
<th>method</th>
<th>top-1 err.</th>
<th>top-5 err.</th>
</tr>
</thead>
<tbody>
<tr>
<td>VGG [41] (ILSVRC'14)</td>
<td>-</td>
<td>8.43†</td>
</tr>
<tr>
<td>GoogLeNet [44] (ILSVRC'14)</td>
<td>-</td>
<td>7.89</td>
</tr>
<tr>
<td>VGG [41] (v5)</td>
<td>24.4</td>
<td>7.1</td>
</tr>
<tr>
<td>BN-inception [16]</td>
<td>21.99</td>
<td>5.81</td>
</tr>
<tr>
<td>ResNet-34 B</td>
<td>21.84</td>
<td>5.71</td>
</tr>
<tr>
<td>ResNet-34 C</td>
<td>21.53</td>
<td>5.60</td>
</tr>
<tr>
<td>ResNet-50</td>
<td>20.74</td>
<td>5.25</td>
</tr>
<tr>
<td>ResNet-101</td>
<td>19.87</td>
<td>4.60</td>
</tr>
<tr>
<td>ResNet-152</td>
<td>19.38</td>
<td>4.49</td>
</tr>
</tbody>
</table>

Table 4. Error rates (%) of single-model results on the ImageNet validation set (except † reported on the test set).

(vs. 15% top-5 err for AlexNet)
Another common trick: “data augmentation”

- Randomly translate, crop, rotate, mirror, shift colors, or overlay images to create more variations
  - Apply random transformation to each sample in each batch as it is processed in training
  - Simulates a larger training set, and makes it so that the network will learn from variations of the original example in each epoch

- Can improve performance, even with fairly large datasets
Data Augmentation (Jittering)

- Create *virtual* training samples
  - Horizontal flip
  - Random crop
  - Color casting
  - Geometric distortion

- Idea goes back to Pomerleau 1995 at least (neural net for car driving)

Deep Image [Wu et al. 2015]
Applying Data Augmentation

1. Define transformation sequence
2. Input transform specification to data loader

```python
import torch
from torchvision import datasets, transforms

batch_size=200

train_loader = torch.utils.data.DataLoader(
    dataset=MNIST('../data', train=True, download=True,
    transform=transforms.Compose([
        transforms.RandomHorizontalFlip(),
        transforms.RandomVerticalFlip(),
        transforms.RandomRotation(15),
        transforms.RandomRotation([90, 180, 270]),
        transforms.Resize([32, 32]),
        transforms.RandomCrop([28, 28]),
        transforms.ToTensor()
    ]),
    batch_size=batch_size, shuffle=True)
```

References:
https://pytorch.org/vision/main/transforms.html
What if we want to do some new task?

• Suppose we’ve trained ImageNet model
• But we want to do something else, e.g. classify flowers or dog breeds
• We don’t have a huge dataset for that task
New Task Solution 1: “Linear probe” / “Feature extraction”

- **Pre-trained Model**
  - ImageNet Trained Model
  - Encoder
    - E.g. weights of convolutional layers, trained on ImageNet
  - Decoder
    - E.g. final 1000 class linear layer weights
  - Output 1000 Class Logits

- **Target Model**
  - Input Image
  - ImageNet Trained Encoder
  - Encoder (Frozen)
  - New Task Decoder
    - Decoder (Tuned)
  - Output $N_c$ Class Logits

Keep original encoder weights. Replace decoder linear layer and train its weights on new task without changing encoder.

Equivalently, extract features from encoder and train linear model on those features.
How to apply linear probe

Pre-compute features method
1. Load pretrained model (many available)
2. Remove prediction final layer
3. Apply model to each image to get features; save them
4. Train new linear model (e.g. logistic regression or SVM) on the features

Freeze encoder method
1. Load pretrained model (many available)
2. Set network to not update weights
3. Replace last layer
4. Retrain network with new dataset
   - Slower than method on left but does not require storing features, and can apply data augmentation

```python
import torch
import torch.nn as nn
from torchvision import models

model = models.alexnet(pretrained=True)

# remove last fully-connected layer
new_classifier = nn.Sequential(*list(model.classifier.children())[:-1])
model.classifier = new_classifier
```
New Task Solution 2: “Fine-tuning”

1. Initialize with original encoder weights.
2. Replace decoder linear layer.
3. Use 10x smaller learning rate than normal and train.
How to apply fine-tuning

1. Load pre-trained model
2. Replace last layer
3. Set a low learning rate (e.g. lr=e-4)
   – Learning rate is often at least 10x lower than for “scratch” training
   – Can “warm start” by freezing earlier layers initially and then unfreezing after a few epochs when the linear layer is mostly trained (avoids messing up encoder while classifier is adjusting)
   – Can set lower learning rate for earlier layers

```python
target_class = 37
model = torch.hub.load('pytorch/vision:v0.10.0', 'resnet34', pretrained=True)
model.fc = nn.Linear(512, target_class)
```

Assumes last layer has 512 features and is called “fc”

Other examples of layer customization (from Weijie)
Task transfer vs. # target task examples

(a) Transfer: ImageNet to Cifar100
(b) Transfer: ImageNet to Places365

Green: Train from scratch
Blue: Linear Probe from ImageNet
Purple: Fine-tune from ImageNet

ResNet18, Err vs # examples / class (in paren)
2 Minute Break

• Comparing linear probe, fine-tuning, and training from scratch, when does each have an advantage and why?
Statistical template approach to object detection

**Propose Window**
- Sliding window: scan image pyramid
- Region proposals: edge/region-based, resize to fixed window

**Extract Features**
- HOG
- Fast randomized features
- CNN features

**Classify**
- SVM
- Boosted stubs
- Neural network

**Post-process**
- Non-max suppression
- Segment or refine localization
R-CNN (Girshick et al. CVPR 2014)

1. Input image
2. Extract region proposals (~2k)
3. Compute CNN features
4. Classify regions

- Extract regions using Selective Search method (Uijlings et al. IJCV 2013)
- Extract rectangles around regions and resize to 227x227
- Extract features with fine-tuned CNN (that was initialized with network trained on ImageNet before training)
- Classify last layer of network features with SVM

Fast R-CNN – Girshick 2015

- Compute CNN features for image once
- ROI Pooling: Pool into 7x7 spatial bins for each region proposal, output class scores and regressed bboxes
- Other refinements: compress classification layer, use network for final classification, end-to-end training
- 100x speed up of R-CNN (0.02 – 0.1 FPS → 0.5-20 FPS) with similar accuracy

- Convolutional features used for generating proposals and scoring
  - Generate proposals with “objectness” scores and refined bboxes for each of k “anchors”
  - Score proposals in same way as Fast R-CNN
- Similar accuracy to Fast R-CNN with 10x speedup
Mask R-CNN – He Gxioxari Dollar Girshick (2017)

• Same network as Faster R-CNN, except
  – Bilinearly interpolate when extracting 7x7 cells of ROI features for better alignment of features to image
  – Instance segmentation: produce a 28x28 mask for each object category
  – Keypoint prediction: produce a 56x56 mask for each keypoint (aim is to label single pixel as correct keypoint)
Top performing object detector, keypoint segmenter, instance segmenter (at time of release and for a bit after)

<table>
<thead>
<tr>
<th>backbone</th>
<th>( \text{AP}^{bb} )</th>
<th>( \text{AP}_{50}^{bb} )</th>
<th>( \text{AP}_{75}^{bb} )</th>
<th>( \text{AP}^{bb} )</th>
<th>( \text{AP}_{50}^{bb} )</th>
<th>( \text{AP}_{75}^{bb} )</th>
</tr>
</thead>
<tbody>
<tr>
<td>Faster R-CNN+++ [19]</td>
<td>ResNet-101-C4</td>
<td>34.9</td>
<td>55.7</td>
<td>37.4</td>
<td>15.6</td>
<td>38.7</td>
</tr>
<tr>
<td>Faster R-CNN w FPN [27]</td>
<td>ResNet-101-FPN</td>
<td>36.2</td>
<td>59.1</td>
<td>39.0</td>
<td>18.2</td>
<td>39.0</td>
</tr>
<tr>
<td>Faster R-CNN by G-RMI[21]</td>
<td>Inception-ResNet-v2 [37]</td>
<td>34.7</td>
<td>55.5</td>
<td>36.7</td>
<td>13.5</td>
<td>38.1</td>
</tr>
<tr>
<td>Faster R-CNN w TDM [36]</td>
<td>Inception-ResNet-v2-TDM</td>
<td>36.8</td>
<td>57.7</td>
<td>39.2</td>
<td>16.2</td>
<td>39.8</td>
</tr>
<tr>
<td>Faster R-CNN, RoIAlign</td>
<td>ResNet-101-FPN</td>
<td>37.3</td>
<td>59.6</td>
<td>40.3</td>
<td>19.8</td>
<td>40.2</td>
</tr>
<tr>
<td>Mask R-CNN</td>
<td>ResNet-101-FPN</td>
<td>38.2</td>
<td>60.3</td>
<td>41.7</td>
<td>20.1</td>
<td>41.1</td>
</tr>
<tr>
<td>Mask R-CNN</td>
<td>ResNetXt-101-FPN</td>
<td>39.8</td>
<td>62.3</td>
<td>43.4</td>
<td>22.1</td>
<td>43.2</td>
</tr>
</tbody>
</table>

Table 3. Object detection single-model results (bounding box AP), vs. state-of-the-art on test-dev. Mask R-CNN usii

<table>
<thead>
<tr>
<th>backbone</th>
<th>( \text{AP} )</th>
<th>( \text{AP}_{50} )</th>
<th>( \text{AP}_{75} )</th>
<th>( \text{AP} )</th>
<th>( \text{AP}_{50} )</th>
<th>( \text{AP}_{75} )</th>
</tr>
</thead>
<tbody>
<tr>
<td>MNC [10]</td>
<td>ResNet-101-C4</td>
<td>24.6</td>
<td>44.3</td>
<td>24.8</td>
<td>4.7</td>
<td>25.9</td>
</tr>
<tr>
<td>FCIS [26] +OHEM</td>
<td>ResNet-101-C5-dilated</td>
<td>29.2</td>
<td>49.5</td>
<td>25.6</td>
<td>7.1</td>
<td>31.3</td>
</tr>
<tr>
<td>FCIS++ [26] +OHEM</td>
<td>ResNet-101-C5-dilated</td>
<td>33.6</td>
<td>54.5</td>
<td>-</td>
<td>-</td>
<td>-</td>
</tr>
<tr>
<td>Mask R-CNN</td>
<td>ResNet-101-C4</td>
<td>33.1</td>
<td>54.9</td>
<td>34.8</td>
<td>12.1</td>
<td>35.6</td>
</tr>
<tr>
<td>Mask R-CNN</td>
<td>ResNet-101-FPN</td>
<td>35.7</td>
<td>58.0</td>
<td>37.8</td>
<td>15.5</td>
<td>38.1</td>
</tr>
<tr>
<td>Mask R-CNN</td>
<td>ResNetXt-101-FPN</td>
<td>37.1</td>
<td>60.0</td>
<td>39.4</td>
<td>16.9</td>
<td>39.9</td>
</tr>
</tbody>
</table>

Table 1. Instance segmentation mask AP on COCO test-dev. MNC [10] and FCIS [26] are the winners of the COCO 2015 and 2016

<table>
<thead>
<tr>
<th></th>
<th>( \text{AP}^{kp} )</th>
<th>( \text{AP}_{50}^{kp} )</th>
<th>( \text{AP}_{75}^{kp} )</th>
<th>( \text{AP}^{kp} )</th>
<th>( \text{AP}_{50}^{kp} )</th>
<th>( \text{AP}_{75}^{kp} )</th>
</tr>
</thead>
<tbody>
<tr>
<td>CMU-Pose+++ [6]</td>
<td>61.8</td>
<td>84.9</td>
<td>67.5</td>
<td>57.1</td>
<td>68.2</td>
<td>64.2</td>
</tr>
<tr>
<td>G-RMI [31]†</td>
<td>62.4</td>
<td>84.0</td>
<td>68.5</td>
<td>59.1</td>
<td>68.1</td>
<td>65.4</td>
</tr>
<tr>
<td>Mask R-CNN, keypoint-only</td>
<td>62.7</td>
<td>87.0</td>
<td>68.4</td>
<td>57.4</td>
<td>71.1</td>
<td>65.8</td>
</tr>
<tr>
<td>Mask R-CNN, keypoint &amp; mask</td>
<td><strong>63.1</strong></td>
<td><strong>87.3</strong></td>
<td><strong>68.7</strong></td>
<td><strong>57.8</strong></td>
<td><strong>71.4</strong></td>
<td><strong>66.0</strong></td>
</tr>
</tbody>
</table>

Table 4. Keypoint detection AP on COCO test-dev. Ours
Example detections and instance segmentations
Example keypoint detections
What does the CNN learn?
Map activation back to the input pixel space

- What input pattern originally caused a given activation in the feature maps?

Visualizing and Understanding Convolutional Networks [Zeiler and Fergus, ECCV 2014]
Layer 1 (visualization of randomly sampled features)

Activations (which pixels caused the feature to have a high magnitude)

Image patches that had high activations

Visualizing and Understanding Convolutional Networks [Zeiler and Fergus, ECCV 2014]
Visualizing and Understanding Convolutional Networks [Zeiler and Fergus, ECCV 2014]
Visualizing and Understanding Convolutional Networks [Zeiler and Fergus, ECCV 2014]
Layer 4 and 5

Visualizing and Understanding Convolutional Networks [Zeiler and Fergus, ECCV 2014]
The “U-Net” is an encoder-decoder with skip connections between mirrored layers in the encoder and decoder stacks.

U-Net style architectures are used to generate pixel maps (e.g., RGB images or per-pixel labels)
Things to remember

- Massive ImageNet dataset was an ingredient to deep learning breakthrough

- Skip connections, data augmentation, and batch normalization are commonly used techniques

- Models trained on ImageNet are used as pretrained “backbones” for other vision tasks

- Mask-RCNN samples patches in feature maps and predicts boxes, object region, and keypoints

- Many image generation and segmentation methods are based on U-Net downsamples while deepening features, then upsamples with skip connections