CAP 5516
Medical Image Computing
(Spring 2022)

Dr. Chen Chen
Center for Research in Computer Vision (CRCV)
University of Central Florida
Office: HEC 221
Address: 4328 Scorpius St., Orlando, FL 32816-2365
Email: chen.chen@crcv.ucf.edu
Web: https://www.crcv.ucf.edu/chenchen/
Lecture 7: Introduction to Deep Learning (3)
Regularization

L2 = weight decay
• Regularization term that penalizes big weights, added to the objective – reduce overfitting
• Weight decay value determines how dominant regularization is during gradient computation
• Big weight decay coefficient → big penalty for big weights

\[ J_{\text{reg}}(\theta) = J(\theta) + \lambda \sum_k \theta_k^2 \]

Dropout
• Randomly drop units (along with their connections) during training
• Each unit retained with fixed probability \( p \), independent of other units
• Hyper-parameter \( p \) to be chosen (tuned)


Early-stopping
• Use validation error to decide when to stop training
• Stop when monitored quantity has not improved after \( n \) subsequent epochs
Early Stopping

Error

Time to stop training

Validation

Training

Number of epochs

Credit: Stephen Marsland
Batch Normalization

Input: Values of $x$ over a mini-batch: $\mathcal{B} = \{x_1...m\}$; Parameters to be learned: $\gamma, \beta$

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

$\mu_{\mathcal{B}} \leftarrow \frac{1}{m} \sum_{i=1}^{m} x_i$  // mini-batch mean

$\sigma^2_{\mathcal{B}} \leftarrow \frac{1}{m} \sum_{i=1}^{m} (x_i - \mu_{\mathcal{B}})^2$  // mini-batch variance

$\hat{x}_i \leftarrow \frac{x_i - \mu_{\mathcal{B}}}{\sqrt{\sigma^2_{\mathcal{B}} + \epsilon}}$  // normalize

$y_i \leftarrow \gamma \hat{x}_i + \beta \equiv \text{BN}_{\gamma,\beta}(x_i)$  // scale and shift

Algorithm 1: Batch Normalizing Transform, applied to activation $x$ over a mini-batch.

Batch Normalization

```
# example of batch normalization for an cnn
from keras.layers import Dense
from keras.layers import Conv2D
from keras.layers import MaxPooling2D
from keras.layers import BatchNormalization
...
model.add(Conv2D(32, (3,3), activation='relu'))
model.add(Conv2D(32, (3,3), activation='relu'))
model.add(BatchNormalization())
model.add(MaxPooling2D())
model.add(Dense(1))
...
```

Resnet architecture (conv/bn/relu)

Dropout Training:

- Each time before updating the parameters
- Each neuron has p% to dropout

Dropout: A simple way to prevent neural networks from overfitting [Srivastava JMLR 2014]
Each time before updating the parameters:
- Each neuron has p% to dropout
- Using the new network for training

The structure of the network is changed.
Testing:

- No dropout
Dropout is a kind of ensemble.

**Ensemble**

Train a bunch of networks with different structures.
More about dropout

- Dropout works better with Maxout [Ian J. Goodfellow, ICML’13]
- Dropconnect [Li Wan, ICML’13]
  - Dropout delete neurons
  - Dropconnect deletes the connection between neurons
- Annealed dropout [S.J. Rennie, SLT’14]
  - Dropout rate decreases by epochs
- Standout [J. Ba, NISP’13]
  - Each neural has different dropout rate
Data Augmentation

- Why data augmentation -> increase training data

Credit: Andrew Ng
Data Augmentation

Color shifting

Credit: Andrew Ng
Data Augmentation

- Color space conversion

Comparison of results for different color spaces on CIFAR-10 with simple CNN

<table>
<thead>
<tr>
<th>Color Space</th>
<th>Accuracy</th>
<th>Time</th>
</tr>
</thead>
<tbody>
<tr>
<td>RGB</td>
<td>78.89</td>
<td>26 secs</td>
</tr>
<tr>
<td>HSV</td>
<td>78.57</td>
<td>26 secs</td>
</tr>
<tr>
<td>YUV</td>
<td>78.89</td>
<td>26 secs</td>
</tr>
<tr>
<td>LAB</td>
<td><strong>80.43</strong></td>
<td>26 secs</td>
</tr>
<tr>
<td>YIQ</td>
<td>78.79</td>
<td>26 secs</td>
</tr>
<tr>
<td>XYZ</td>
<td>78.72</td>
<td>26 secs</td>
</tr>
<tr>
<td>YPbPr</td>
<td>78.78</td>
<td>26 secs</td>
</tr>
<tr>
<td>YCbCr</td>
<td>78.81</td>
<td>26 secs</td>
</tr>
<tr>
<td>HED</td>
<td>78.98</td>
<td>26 secs</td>
</tr>
<tr>
<td>LCH</td>
<td>78.82</td>
<td>26 secs</td>
</tr>
</tbody>
</table>

Data Augmentation

- Random erasing

Reduces the risk of over-fitting and makes the model robust to occlusion
Improve the generalization ability of CNNs
Data Augmentation

- Learning Augmentation Policies from Data

Search space of operations: ShearX/Y, TranslateX/Y, Rotate, AutoContrast, Invert, Equalize, Solarize, Posterize, Contrast, Color, Brightness, Sharpness,

Data Augmentation

- **Cutmix**

<table>
<thead>
<tr>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>Label</td>
<td>Dog 1.0</td>
<td>Dog 0.5</td>
<td>Dog 1.0</td>
<td>Dog 0.6</td>
</tr>
<tr>
<td></td>
<td>76.3</td>
<td>77.4</td>
<td>77.1</td>
<td>78.6</td>
</tr>
<tr>
<td></td>
<td>(+0.0)</td>
<td>(+1.1)</td>
<td>(+0.8)</td>
<td>(+2.3)</td>
</tr>
<tr>
<td></td>
<td>ImageNet</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>46.3</td>
<td>45.8</td>
<td>46.7</td>
<td>47.3</td>
</tr>
<tr>
<td></td>
<td>(+0.0)</td>
<td>(-0.5)</td>
<td>(+0.4)</td>
<td>(+1.0)</td>
</tr>
<tr>
<td></td>
<td>ImageNet</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>75.6</td>
<td>73.9</td>
<td>75.1</td>
<td>76.7</td>
</tr>
<tr>
<td></td>
<td>(+0.0)</td>
<td>(-1.7)</td>
<td>(-0.5)</td>
<td>(+1.1)</td>
</tr>
</tbody>
</table>

• More on data augmentation

  - (Pytorch) torchvision.transforms:
    https://pytorch.org/vision/master/transforms.html

  - https://github.com/AgaMiko/data-augmentation-review
Transfer Learning

- Improvement of learning in a **new** task through the *transfer of knowledge* from a **related** task that has already been learned.
- Weight initialization for CNN

- Two major strategies
  - ConvNet as fixed feature extractor
  - Fine-tuning the ConvNet
Transfer Learning Overview

Task A

Input A

Layer n

Task B

Input B

AnB: Frozen Weights

AnB⁺: Fine-tuning

Back-propagation

(image: Aghamirzaie & Salomon)
When and how to fine-tune?

• Suppose we have model A, trained on dataset A
• Q: How do we apply transfer learning to dataset B to create model B?
When and how to fine-tune?

- New dataset is small and similar to original dataset.
  - train a linear classifier on the CNN codes
- New dataset is large and similar to the original dataset
  - fine-tune through the full network
- New dataset is small but very different from the original dataset
  - SVM classifier from activations somewhere earlier in the network
- New dataset is large and very different from the original dataset
  - fine-tune through the entire network

<table>
<thead>
<tr>
<th>Dataset size</th>
<th>Dataset similarity</th>
<th>Recommendation</th>
</tr>
</thead>
<tbody>
<tr>
<td>Large</td>
<td>Very different</td>
<td>Train model B from scratch, initialize weights from model A</td>
</tr>
<tr>
<td>Large</td>
<td>Similar</td>
<td>OK to fine-tune (less likely to overfit)</td>
</tr>
<tr>
<td>Small</td>
<td>Very different</td>
<td>Train classifier using the earlier layers (later layers won't help much)</td>
</tr>
<tr>
<td>Small</td>
<td>Similar</td>
<td>Don't fine-tune (overfitting). Train a linear classifier</td>
</tr>
</tbody>
</table>

https://cs231n.github.io/transfer-learning/
Examples

- https://pytorch.org/tutorials/beginner/transfer_learning_tutorial.html

- https://blog.keras.io/building-powerful-image-classification-models-using-very-little-data.html
Transfer Learning for Medical Imaging

Figure 1: Example images from the IMAGENET, the retinal fundus photographs, and the CHEXPERT datasets, respectively. The fundus photographs and chest x-rays have much higher resolution than the IMAGENET images, and are classified by looking for small local variations in tissue.

• Proposed Solution

- **Transfer the scale (range) of the weights instead of the weights themselves.** This offers feature-independent benefits that facilitate convergence. In particular, they initialized the weights from a normal distribution $N(\mu; \sigma)$. The mean and the variance of the weight matrix is calculated from the pretrained weights. This calculation was performed for each layer separately.

- **Use the pretrained weights only from the lowest two layers.** The rest of the network is randomly initialized and fine-tuned for the medical imaging task. This hybrid method has the biggest impact on convergence. To summarize, most of the most meaningful feature representations are learned in the lowest two layers.

Transfer Learning for Medical Imaging

- Med3D: Transfer Learning for 3D Medical Image Analysis

https://github.com/Tencent/MedicalNet
Common CNNs

- ImageNet challenge
Common CNNs

**AlexNet**
- Input
- Conv
- Conv
- Pool
- Conv
- Conv
- Pool
- FC
- FC
- Softmax

**VGGNet**
- Input
- Conv
- Conv
- Pool
- Conv
- Conv
- Pool
- Conv
- Conv
- Pool
- Conv
- Conv
- Pool
- FC
- FC
- FC
- Softmax
Common CNNs

GoogLeNet/Inception
Common CNNs

34-layer residual

34-layer plain

Residual Networks

VGG-19

UCF: CENTER FOR RESEARCH IN COMPUTER VISION
Residual Networks

• Deep networks performs worse
  – As we add more layers
• Problem
  – Vanishing gradients
• It models
  – $H(x) = F(x) + x$
• Skip connections
  – Help in backpropagation

He et. al. Deep Residual Learning for Image Recognition, 2015
Residual Networks

RESNET

By Pytorch Team

Deep residual networks pre-trained on ImageNet

View on Github  Open on Google Colab

import torch
model = torch.hub.load('pytorch/vision:v0.10.0', 'resnet18', pretrained=True)
# or any of these variants
# model = torch.hub.load('pytorch/vision:v0.10.0', 'resnet34', pretrained=True)
# model = torch.hub.load('pytorch/vision:v0.10.0', 'resnet50', pretrained=True)
# model = torch.hub.load('pytorch/vision:v0.10.0', 'resnet101', pretrained=True)
# model = torch.hub.load('pytorch/vision:v0.10.0', 'resnet152', pretrained=True)
model.eval()

https://pytorch.org/hub/pytorch_vision_resnet/
Common CNNs

More on MobileNet
Variants of Convolution Operation

- What is the computation of a convolution

Multiplication is an expensive operation relative to addition.
Variants of Convolution Operation

- Determine the number of multiplications

Source: https://www.youtube.com/watch?v=T7o3xvJLuHk
Variants of Convolution Operation

- Determine the number of multiplications

**Convolution**

Multiplications one position?

\[ D_K^2 \times M \]

Multiplications one kernel over the entire input?

\[ D_G^2 \times D_K^2 \times M \]

Multiplications N kernel?

\[ N \times D_G^2 \times D_K^2 \times M \]
Variants of Convolution Operation

Depthwise Separable Convolution

1. Depthwise Convolution: Filtering Stage

2. Pointwise Convolution: Combination Stage

Reduce computation (# of multiplications)
Variants of Convolution Operation

Depthwise Separable Convolution

1. Depthwise Convolution: Filtering Stage

Input feature map

One standard conv kernel

\[ D_K \times D_K \times M \]

M different kernels
Variants of Convolution Operation

Depthwise Separable Convolution

1. Depthwise Convolution: Filtering Stage

Example:

https://towardsdatascience.com/a-basic-introduction-to-separable-convolutions-b99ec3102728
Variants of Convolution Operation

Depthwise Separable Convolution

1. Depthwise Convolution: Filtering Stage

Source: https://www.youtube.com/watch?v=T7o3xvJLuHk
Variants of Convolution Operation

Depthwise Separable Convolution

2. Pointwise Convolution: Filtering Stage

Source: https://www.youtube.com/watch?v=T7o3xvJLuHk
Variants of Convolution Operation

Depthwise Separable Convolution

2. Pointwise Convolution: Filtering Stage

Source: https://www.youtube.com/watch?v=T7o3xvJLuHk
**Variants of Convolution Operation**

- Mults once = $D_K^2$
- Mults 1 Channel = $D_G^2 \times D_K^2$
- DC Mults = $M \times D_G^2 \times D_K^2$
- Mults once = $M$
- Mults 1 Kernel = $D_G \times D_G \times M$
- PC Mults = $N \times D_G \times D_G \times M$

**Total** = DC Mults + PC Mults

$M \times D_G^2 \times D_K^2 + N \times D_G^2 \times M$

$M \times D_G^2 \left( D_K^2 + N \right)$
Variants of Convolution Operation

Comparison Standard Vs. Depthwise

\[
\frac{\text{No. Mults in Depthwise Separable Conv}}{\text{No. Mults in Standard Conv}} = \frac{M \times D_G^2 (D_K^2 + N)}{N \times D_G \times D_G \times D_K \times D_K \times M}
\]

\[
\frac{\text{No. Mults in Depthwise Separable Conv}}{\text{No. Mults in Standard Conv}} = \frac{D_K^2 + N}{(D_K^2 \times N)} = \frac{1}{N} + \frac{1}{D_K^2}
\]

E.g. \(N = 1,024\) \(D_K = 3\)

\[
\frac{\text{No. Mults in Depthwise Separable Conv}}{\text{No. Mults in Standard Conv}} = \frac{1}{1024} + \frac{1}{3^2} = 0.112
\]
Variants of Convolution Operation

Comparison Standard Vs. Depthwise

\[
\text{Param 1 Kernel} = D_K^2 \times M \\
\text{Param } N \text{ Kernels} = N \times M \times D_K^2
\]

\[
\text{Param 1 Kernel} = D_K^2 \\
\text{Param } M \text{ Kernels} = M \times D_K^2 \\
\text{Param 1 Kernel} = M \\
\text{Param } N \text{ Kernels} = N \times M
\]

\[
\frac{\text{No. params in Depthwise Separable Conv}}{\text{No. params in Standard Conv}} = \frac{M \times (D_K^2 + N)}{N \times D_K^2 \times M} = \frac{1}{N} + \frac{1}{D_K^2}
\]
One Note on 1x1 conv

- Often used for channel reduction
MobileNet

## Depthwise Separable Convolution

<table>
<thead>
<tr>
<th>Model</th>
<th>ImageNet Accuracy</th>
<th>Million Mult-Adds</th>
<th>Million Parameters</th>
</tr>
</thead>
<tbody>
<tr>
<td>1.0 MobileNet-224</td>
<td>70.6%</td>
<td>569</td>
<td>4.2</td>
</tr>
<tr>
<td>GoogleNet</td>
<td>69.8%</td>
<td>1550</td>
<td>6.8</td>
</tr>
<tr>
<td>VGG 16</td>
<td>71.5%</td>
<td>15300</td>
<td>138</td>
</tr>
</tbody>
</table>

MobileNet Family

- MobileNet V1

- MobileNet V2

- MobileNet V3
Common network models

- Pytorch

```python
import torchvision.models as models
resnet18 = models.resnet18()
alexnet = models.alexnet()
vgg16 = models.vgg16()
squeezenet = models.squeezenet1_0()
densenet = models.densenet161()
inception = models.inception_v3()
googlenet = models.googlenet()
shufflenet = models.shufflenet_v2_x1_0()
mobilenet = models.mobilenet_v2()
resnext50_32x4d = models.resnext50_32x4d()
wide_resnet50_2 = models.wide_resnet50_2()
mnasnet = models.mnasnet1_0()
```

TORCHVISION.MODELS

The models subpackage contains definitions of models for addressing different semantic segmentation, object detection, instance segmentation, person keypoint detection, and video classification.

Classification

The models subpackage contains definitions for the following model architectures:

- AlexNet
- VGG
- ResNet
- SqueezeNet
- DenseNet
- Inception v3
- GoogLeNet
- ShuffleNet v2
- MobileNet v2
- ResNeXt
- Wide ResNet
- MNASNet

Common network models

- Keras

Available models

<table>
<thead>
<tr>
<th>Model</th>
<th>Size</th>
<th>Top-1 Accuracy</th>
<th>Top-5 Accuracy</th>
</tr>
</thead>
<tbody>
<tr>
<td>Xception</td>
<td>88 MB</td>
<td>0.790</td>
<td>0.945</td>
</tr>
<tr>
<td>VGG16</td>
<td>528 MB</td>
<td>0.713</td>
<td>0.901</td>
</tr>
<tr>
<td>VGG19</td>
<td>549 MB</td>
<td>0.713</td>
<td>0.900</td>
</tr>
<tr>
<td>ResNet50</td>
<td>98 MB</td>
<td>0.749</td>
<td>0.921</td>
</tr>
<tr>
<td>ResNet101</td>
<td>171 MB</td>
<td>0.764</td>
<td>0.928</td>
</tr>
<tr>
<td>ResNet152</td>
<td>232 MB</td>
<td>0.766</td>
<td>0.931</td>
</tr>
<tr>
<td>ResNet50V2</td>
<td>98 MB</td>
<td>0.760</td>
<td>0.930</td>
</tr>
<tr>
<td>ResNet101V2</td>
<td>171 MB</td>
<td>0.772</td>
<td>0.938</td>
</tr>
<tr>
<td>ResNet152V2</td>
<td>232 MB</td>
<td>0.780</td>
<td>0.942</td>
</tr>
<tr>
<td>InceptionV3</td>
<td>92 MB</td>
<td>0.779</td>
<td>0.937</td>
</tr>
<tr>
<td>InceptionResNetV2</td>
<td>215 MB</td>
<td>0.803</td>
<td>0.953</td>
</tr>
<tr>
<td>MobileNet</td>
<td>16 MB</td>
<td>0.704</td>
<td>0.895</td>
</tr>
<tr>
<td>MobileNetV2</td>
<td>14 MB</td>
<td>0.713</td>
<td>0.901</td>
</tr>
<tr>
<td>DenseNet121</td>
<td>33 MB</td>
<td>0.750</td>
<td>0.923</td>
</tr>
<tr>
<td>DenseNet169</td>
<td>57 MB</td>
<td>0.762</td>
<td>0.932</td>
</tr>
<tr>
<td>DenseNet201</td>
<td>80 MB</td>
<td>0.773</td>
<td>0.936</td>
</tr>
<tr>
<td>NASNetMobile</td>
<td>23 MB</td>
<td>0.744</td>
<td>0.919</td>
</tr>
</tbody>
</table>

https://keras.io/api/applications/
Reading Material

- Training a Classifier — PyTorch Tutorials: [https://pytorch.org/tutorials/beginner/blitz/cifar10_tutorial.html](https://pytorch.org/tutorials/beginner/blitz/cifar10_tutorial.html)

- CNNs, Part 1: An Introduction to Convolutional Neural Networks (Keras implementation)
  - [https://victorzhou.com/blog/intro-to-cnns-part-1/](https://victorzhou.com/blog/intro-to-cnns-part-1/)

- CNNs, Part 2: Training a Convolutional Neural Network (Keras implementation)
  - [https://victorzhou.com/blog/intro-to-cnns-part-2/](https://victorzhou.com/blog/intro-to-cnns-part-2/)

- **MNIST digit classification (Keras implementation)**
  - [https://victorzhou.com/blog/keras-cnn-tutorial/#the-full-code](https://victorzhou.com/blog/keras-cnn-tutorial/#the-full-code)

- Mnist classification (CNN Keras)

- Bag of Tricks for Image Classification with Convolutional Neural Networks
Thank you!

Question?
References and Slide Credits

• Many slides are adapted from the existing teaching or tutorial slides by Hung-yi Lee, Andrew Ng, Alexander Amini, Lex Fridman, Stanford course - CS231n: Convolutional Neural Networks for Visual Recognition, and many others

• Special thanks to Dr. Hung-yi Lee for making his machine learning course slides and materials available

  – Youtube videos: https://www.youtube.com/watch?v=5tvMx8r_OM&list=PLtBw6njQRU-rwp5__7C0olVt26ZgjG9NI&index=1

• Lex Fridman, MIT Deep Learning and Artificial Intelligence Lectures: https://deeplearning.mit.edu/
  https://www.youtube.com/watch?v=O5xeyoRL95U