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 6: Introduction to Deep Learning (2)
Convolutional Neural Network (CNN)

Widely used in image processing and computer vision
The whole CNN

**Property 1**
- Some patterns are much smaller than the whole image

**Property 2**
- The same patterns appear in different regions.

**Property 3**
- Subsampling the pixels will not change the object

Can repeat many times
Convolution v.s. Fully Connected

**Image:**

```
1 0 0 0 0 1
0 1 0 0 1 0
0 0 1 1 0 0
1 0 0 0 1 0
0 1 0 0 1 0
0 0 1 0 1 0
```

**Convolution:**

```
1 -1 -1
-1 1 -1
-1 -1 1
-1 1 -1
```

**Fully-connected:**

```
1 0 0 0 0 1
0 1 0 0 1 0
0 0 1 1 0 0
1 0 0 0 1 0
0 1 0 0 1 0
0 0 1 0 1 0
```

```
x_1
x_2
x_36
```
Filter 1

6 x 6 image

Less parameters!

Only connect to 9 input, not fully connected

Less parameters!
Less parameters!

Even less parameters!
Question

• What kernel size to use? 3x3, 5x5, …

• How many kernels to use in each layer?
Variants of Convolution Operation

- Dilated convolution [1]
- Depth-wise separable convolution [2]
- Grouped convolution [3]

Variants of Convolution Operation

- **Dilated/Atrous Convolution**
  
  Dilation rate = 2  
  Receptive field = 5x5  
  With only 9 parameters

  Receptive filed = 3x3

- Normal convolution

  Effective in segmentation task

Variants of Convolution Operation

• Dilated Convolution


• Large value of atrous rate enlarges the model's field-of-view

• Enabling object encoding at multiple scales
Variants of Convolution Operation

The whole CNN

cat dog ……

Fully Connected Feedforward network

Convolution

Max Pooling

Convolution

Max Pooling

Flatten

Can repeat many times
CNN – Max Pooling

Filter 1

Filter 2
CNN – Max Pooling

6 x 6 image

Conv

Max Pooling

New image but smaller

2 x 2 image
The Whole CNN

A new image

Smaller than the original image

The number of the channel is the number of filters

Can repeat many times

Convolution

Max Pooling

Convolution

Max Pooling

-1  1
0  3
Why Pooling

• Pooling is to progressively reduce the spatial size of the representation to reduce the number of parameters and computation in the network.

• Another way to reduce feature map size? (convolution with stride, e.g., 2)

• If instead of taking the maximum, using the average will be the average pooling.

Other benefit?
The Whole CNN

cat dog ……

Fully Connected Feedforward network

Convolution

Max Pooling

Convolution

Max Pooling

Flatten

Can repeat many times
Flatten

Fully Connected Feedforward network
CNN in Keras

```python
model2.add(Convolution2D(25, 3, 3, input_shape=(1, 28, 28)))
```

There are 25 3x3 filters.

Input_shape = (1, 28, 28)

1: grayscale, 3: RGB

28 x 28 pixels

```python
model2.add(MaxPooling2D((2, 2)))
```
CNN in Keras

How many parameters for each filter?

1. \( \text{model2.add(\ Convolution2D( 25,3,3, \)} \)
\( \text{input\_shape=(1,28,28) ) } \)
- Parameters: 9

2. \( \text{model2.add(MaxPooling2D((2,2)))} \)
- Output: 25 x 26 x 26

3. \( \text{model2.add(Convolution2D(50,3,3))} \)
- Parameters: 225

4. \( \text{model2.add(MaxPooling2D((2,2)))} \)
- Output: 50 x 11 x 11

5. \( \text{model2.add(Convolution2D(50,3,3))} \)
- Parameters: 225

6. \( \text{model2.add(MaxPooling2D((2,2)))} \)
- Output: 50 x 5 x 5
CNN in Keras

```
model2.add(Dense(output_dim=100))
model2.add(Activation('relu'))
model2.add(Dense(output_dim=10))
model2.add(Activation('softmax'))
```

Fully Connected Feedforward

Input: 1 x 28 x 28

Convolution: 25 x 26 x 26
Max Pooling: 25 x 13 x 13
Convolution: 50 x 11 x 11
Max Pooling: 50 x 5 x 5
Flatten: 1250

Output: Fully Connected Feedforward
Softmax

Multi-Class Classification with NN and SoftMax Function

The softmax as

$$\sigma(j) = \frac{\exp(w_j^T x)}{\sum_{k=1}^{K} \exp(w_k^T x)} = \frac{\exp(z_j)}{\sum_{k=1}^{K} \exp(z_k)}$$

https://victorzhou.com/blog/softmax/
Loss Function

- Way to define how good the network is performing
  - In terms of prediction
- Network training (Optimization)
  - Find the best network parameters to minimize the loss

\[ L(W) = \frac{1}{N} \sum_{i=1}^{N} L_i(f(x_i, W), y_i) \]

Total loss for a training set
N samples

- Loss function
- Network parameters
- Ground truth
- Input
- Network
Choosing Proper Loss

Square Error

$$\sum_{i=1}^{10}(y_i - \hat{y}_i)^2$$

Target (one hot encoding)

Softmax

0.6

Loss
Cross-entropy loss

- Binary case (we know this from logistic regression)

\[-(y \log(\hat{y}) + (1 - y) \log(1 - \hat{y}))\]

- General form

\[-\sum_{i=1}^{K} y_i \log(\hat{y}_i)\]

Label \( y \) is a one-hot vector

\( y_i \in [0,1] \)
Loss Function

- Multi-class classification

Cross-Entropy loss for one sample

\[- \sum_{i=1}^{K} y_i \log(\hat{y}_i)\]

Label \(y\) is a one-hot vector \((y_i) \in [0,1]\)

<table>
<thead>
<tr>
<th>computed</th>
<th>targets</th>
<th>correct?</th>
</tr>
</thead>
<tbody>
<tr>
<td>0.3 0.3 0.4</td>
<td>0 0 1 (democrat)</td>
<td>yes</td>
</tr>
<tr>
<td>0.3 0.4 0.3</td>
<td>0 1 0 (republican)</td>
<td>yes</td>
</tr>
<tr>
<td>0.1 0.2 0.7</td>
<td>1 0 0 (other)</td>
<td>no</td>
</tr>
</tbody>
</table>

\[
\mathcal{L} = -((\log(0.3) \times 0) + (\log(0.3) \times 0) + (\log(0.4) \times 1)) = -\log(0.4)
\]

Softmax

\[
\sigma(j) = \frac{\exp(w_j^T x)}{\sum_{k=1}^{K} \exp(w_k^T x)} = \frac{\exp(z_j)}{\sum_{k=1}^{K} \exp(z_k)}
\]
Choosing Proper Loss

![Diagram of neural network with Softmax output and Cross Entropy loss function]

Softmax output:
- \( \hat{y}_1 \) = 0.6
- \( \hat{y}_2 \) = 0
- \( \hat{y}_{10} \) = 0

Target:
- \( y_1 \) = 1
- \( y_2 \) = 0
- \( y_{10} \) = 0

Loss function:
\[ - \sum_{i=1}^{10} y_i \log(\hat{y}_i) \]
Several alternatives: https://keras.io/objectives/
Training

Sample labeled data (batch) → Forward it through the network, get predictions → Back-propagate the errors → Update the network weights

Optimize (min. or max.) objective/cost function $J(\theta)$
Generate error signal that measures difference between predictions and target values

Use error signal to change the weights and get more accurate predictions
Subtracting a fraction of the gradient moves you towards the (local) minimum of the cost function

Gradient Descent

**objective/cost function** $J(\theta)$

$$\theta^\text{new}_j = \theta^\text{old}_j - \alpha \frac{d}{d\theta_j^\text{old}} J(\theta)$$  \hspace{1cm} Update each element of $\theta$

$$\theta^\text{new} = \theta^\text{old} - \alpha \nabla_\theta J(\theta)$$  \hspace{1cm} Matrix notation for all parameters

learning rate

Recursively apply **chain rule** though each node
• Rectified Linear Unit (ReLU)

\[
\begin{align*}
\sigma(z) & \quad a = z \\
& \quad a = 0 \\
& \quad z
\end{align*}
\]

[Xavier Glorot, AISTATS’11]  
[Andrew L. Maas, ICML’13]  
[Kaiming He, arXiv’15]

Vanishing gradient problem
ReLU - variant

**Leaky ReLU**

\[ a = z \]

\[ a = 0.01z \]

**Parametric ReLU**

\[ a = z \]

\[ a = \alpha z \]

\( \alpha \) also learned by gradient descent
Stochastic Gradient Descent

- Gradient over entire dataset is impractical
- Better to take quick, noisy steps
- Estimate gradient over a mini-batch of examples
Mini-batch

Randomly initialize network parameters

Pick the 1\textsuperscript{st} batch
\[ L' = l^1 + l^{31} + \ldots \]
Update parameters once

Pick the 2\textsuperscript{nd} batch
\[ L'' = l^2 + l^{16} + \ldots \]
Update parameters once

Until all mini-batches have been picked

We do not really minimize total loss!

one epoch

Repeat the above process
Mini-batch

model.fit(x_train, y_train, batch_size=100, nb_epoch=20)

- Pick the 1st batch
  \[ L' = l^1 + l^{131} + \ldots \]
  Update parameters once

- Pick the 2nd batch
  \[ L'' = l^2 + l^{16} + \ldots \]
  Update parameters once

- Until all mini-batches have been picked

100 examples in a mini-batch
Repeat 20 times

one epoch
Other optimizers

- Adagrad [John Duchi, JMLR’11]
- RMSprop
  - https://www.youtube.com/watch?v=O3sxAc4hxZU
- Adadelta [Matthew D. Zeiler, arXiv’12]
- “No more pesky learning rates” [Tom Schaul, arXiv’12]
- AdaSecant [Caglar Gulcehre, arXiv’14]
- Adam [Diederik P. Kingma, ICLR’15]
- Nadam
How to pick the learning rate?

- Too big = diverge, too small = slow convergence
- No “one learning rate to rule them all”
- Start from a high value and keep cutting by half if model diverges
- Learning rate schedule: decay learning rate over time
http://cs231n.github.io/assets/nn3/learningrates.jpeg
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 $\rightarrow$ big penalty for big weights

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

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

Validation

Training

Time to stop training

Number of epochs
Batch Normalization

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

---

Batch Normalization

The provided code snippet demonstrates the use of Batch Normalization in a Keras model:

```python
# 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
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
Reading Material


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

- CNNs, Part 2: Training a Convolutional Neural Network (Keras implementation)

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

- Mnist classification (CNN Keras)

- Bag of Tricks for Image Classification with Convolutional Neural Networks
Data Augmentation

- Why data augmentation -> increase training data

Credit: Andrew Ng
Data Augmentation

Color shifting

Credit: Andrew Ng
Data Augmentation

- **Color space conversion**

<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

More on data augmentation

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

- https://github.com/AgaMiko/data-augmentation-review
Case study

- ImageNet challenge
Case study
Case study

GoogLeNet/Inception

Inception module
Case study

Residual Networks
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
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 tasks:

- Semantic segmentation
- Object detection
- Instance segmentation
- Person keypoint detection

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

<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/
Applications

- Classification
- Detection
- Action recognition (spatial –temporal)
- Segmentation
- Image generation
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=5tvmMX8r_OM&list=PLtBw6njQRU-rwp5__7C0o1Vt26ZgjG9NI&index=1

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