APS360 Fundamentals of AI

Lisa Zhang

Lecture 8; Feb 1, 2019

Agenda: First hour

- Neural Network Features
- Preventing Overfitting
  - Data Augmentation
  - Regularization
  - Dropout

Agenda: Second hour

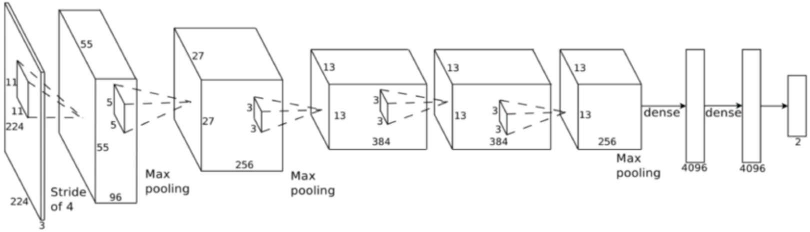
- CNN Architectures
- Fully Convolutional Networks

Neural Network Features

AlexNet from last class

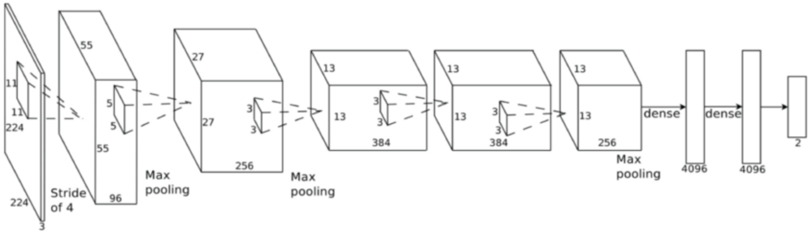
```
import torchvision.models

alexNet = torchvision.models.alexnet(pretrained=True)
alexNet.features
alexNet.classifier
```

AlexNet architecture



- **Convolutional part**: `features`
- **Fully-connected part**: `classifier`

AlexNet features



- Each layer we computes a different *representation* of the input
- These *representations* are better-suited (to the classification task) than the input representation
- These *representations* turns out to be useful to other tasks!

Assignment 3

In assignment 3, we will use the pre-trained `AlexNet.features` network:

- Find the AlexNet features for our gesture image
- Use the features as input to a classification network of our own

... the idea of applying knowledge gained from solving one problem to another problem is called **transfer learning**.

Assignment 3 (other thoughts)

- You might find `torchvision.datasets.ImageFolder` useful
- Train/Validation/Test split
  - Random split across all images?
  - Evenly split the letters?
  - Split across users?

Preventing Overfitting

Overfitting and Underfitting

- Detecting underfitting is much harder than detecting overfitting
- Generally, we want to get to a point where we overfit, then

Strategies to prevent overfitting

- More data set (expensive, often not feasible)
- Use a smaller network (requires starting over)
- Weight-sharing - as in convolutional neural networks
- **Early stopping** - stop training at an earlier epoch

Other strategies

- Data Augmentation
- Data Normalization
- Weight Decay
- Model Averaging
- Dropout

Data Augmentation

Make small alternations to the data that you have to get new data

- Flip each image horizontally or vertically (e.g. for cats vs dogs, not for gesture recognition)
- Shift each pixel a little to the left or right
- Rotate the images a little
- Add noise to the image
- Combination

Data Normalization

Normalize the pixel intensities of an image

```
transforms.Normalize((0.5, 0.5, 0.5), (0.5, 0.5, 0.5))
```

Remove features of the image that we know are unrelated to the task we want to perform.

Penalizing Large Weights

Penalize **large weights**, by adding a term (e.g. $\sum_k w_k^2$) to the loss function

Why?

Penalizing Large Weights

Penalize **large weights**, by adding a term (e.g. $\sum_k w_k^2$) to the loss function

Why?

Because large weights mean that the prediction relies **a lot** on the content of one pixel

Weight Decay

- L^1 regularization: add a term $\sum_k |w_k|$ to the loss function
  - Mathematically, this term encourages weights to be exactly 0
- L^2 regularization: add a term $\sum_k w_k^2$ to the loss function
  - Mathematically, in each iteration the weight is pushed towards 0
- Combination of L^1 and L^2 regularization: add a term $\sum_k |w_k| + w_k^2$ to the loss function

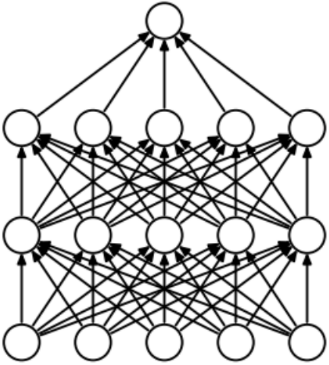
Model Averaging

To prevent overfitting, build **many** models, and average their predictions.

Each model use a slightly different architecture, or different initial weights.

Dropout

Randomly “remove” a portion of neurons from each training iteration:



(a) Standard Neural Net (b) After applying dropout.

A different set of neurons are “removed” in a different iteration.

All neurons are used during test time (for evaluation and for making actual predictions)

Why dropout

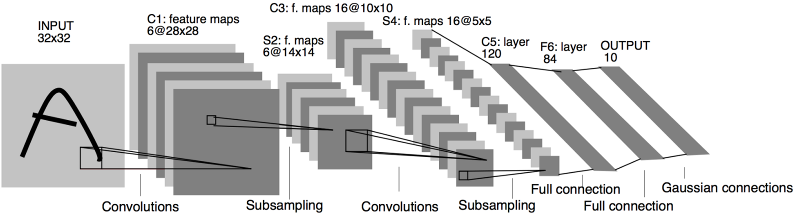
- Prevent weights from depending on each other.
- Encourage each hidden unit to learn "more independent" features.
- Is actually a form of model averaging: averaging over all possible connections.

CNN Architectures

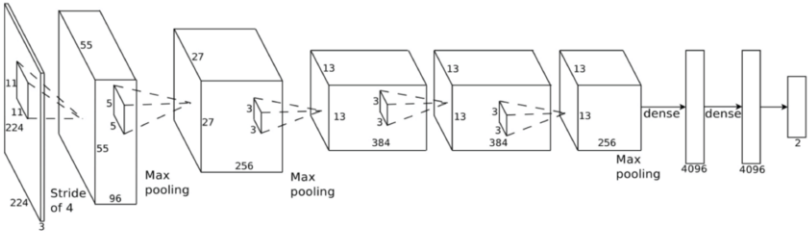
Named Architectures

- LeNext
- AlexNet
- VGG
- ResNet

LeNet

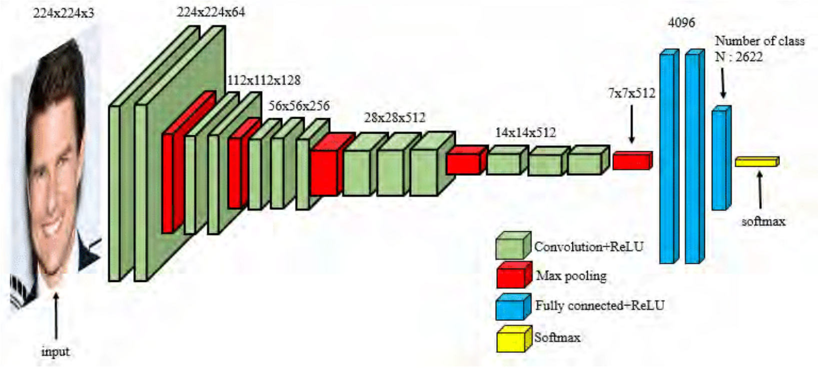

INPUT
32x32
C1: feature maps
6@28x28
S2: f. maps
6@14x14
C3: f. maps 16@10x10
S4: f. maps 16@5x5
C5: layer
120
F6: layer
84
OUTPUT
10
Convolutions
Subsampling
Convolutions
Subsampling
Full connection
Full connection
Gaussian connections

AlexNet



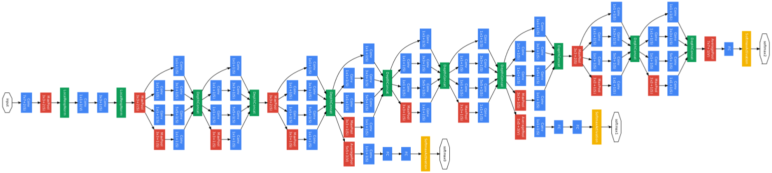
11
11
224
224
3
55
55
96
Stride
of 4
5
5
Max
pooling
27
27
256
3
3
Max
pooling
13
13
384
3
3
13
13
384
3
3
13
13
256
Max
pooling
dense
dense
4096
4096
2

VGG



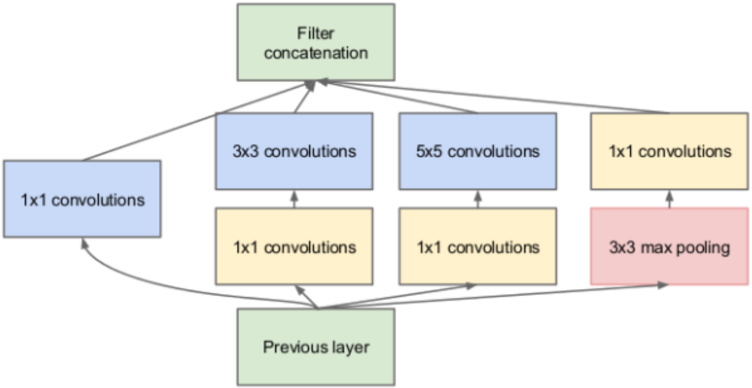
224x224x3
224x224x64
112x112x128
56x56x256
28x28x512
14x14x512
7x7x512
4096
Number of class
N : 2622
softmax
input
Convolution+ReLU
Max pooling
Fully connected+ReLU
Softmax

GoogleLeNet (Inception)

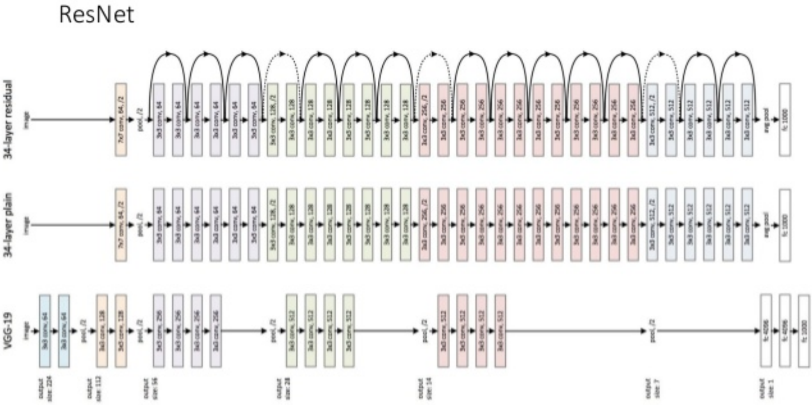


Basic idea: repeated modules

Inception Module

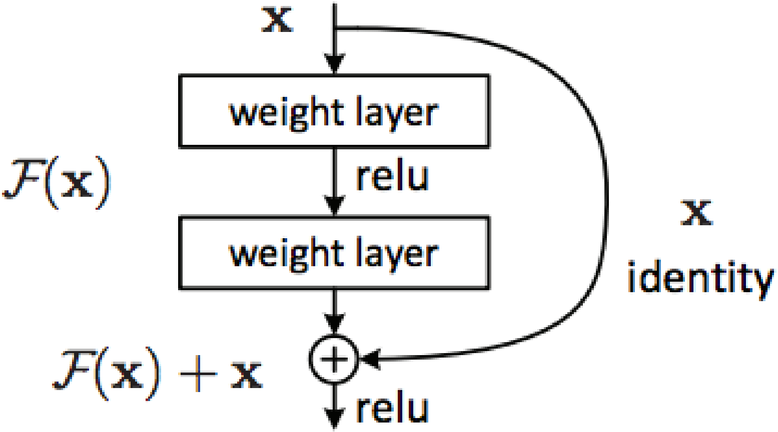


ResNet

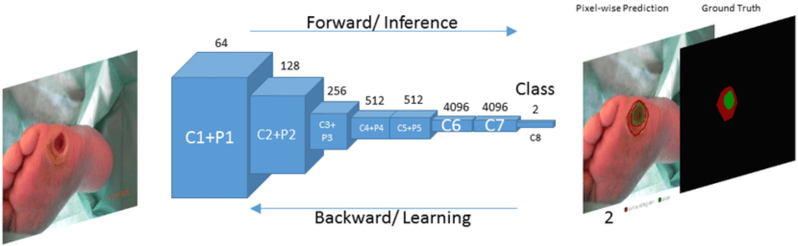


Basic idea: skip connections

ResNet Basic Block



Fully Convolutional Networks



Idea: do away with fully-connected layers

`Image from "Fully Convolutional Networks for Diabetic Foot Ulcer Segmentation"`

Why avoid fully connected layers?

- So that the neural network can take arbitrary dimension images as input

Instead of fully connected layers..

- Use a convolution layer with the same kernel size as hidden unit size and no padding
- Use global average-pooling