Convolutional Neural Networks

Deep Learning Lecture 4

Samuel Cheng

School of ECE
University of Oklahoma

Spring, 2017

Table of Contents

1. Review

2. Babysitting your learning job

3. Overview and history of CNN

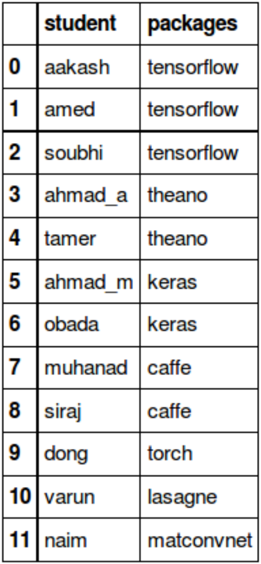
4. CNN basic

5. Case study

6. Some CNN tricks

7. Conclusions

Presentation order



	student	packages
0	aakash	tensorflow
1	amed	tensorflow
2	soubhi	tensorflow
3	ahmad_a	theano
4	tamer	theano
5	ahmad_m	keras
6	obada	keras
7	muhanad	caffe
8	siraj	caffe
9	dong	torch
10	varun	lasagne
11	naim	matconvnet

Logistics

- HW1 is due today
- 5% per day penalty (of HW1) starting tomorrow
- Naim is the winner for the first HW with 3% overall bonus
  - As extra “bonus” to the winner, I would like him to present his solution in class next Friday ($10 \sim 20$ minutes). Emphasized on **surprises** and **lesson learned**
  - No need to be comprehensive
  - HW1 won’t be accepted after his presentation

Review

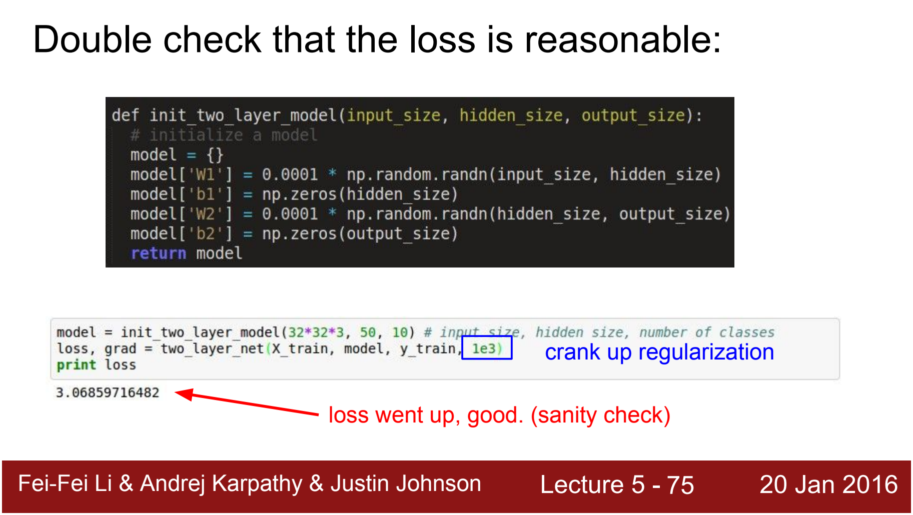
In the last class, we discussed

- BP
- Weight initialization
- Batch normalization
- Dropout
- More optimization tricks
    - Nesterov accelerated gradient descent
    - RMSProp
    - Adam

Today

- Left out from last lecture: some remarks on babysitting your training process
- Convolutional neural network (CNN)

Debugging optimizer



Double check that the loss is reasonable:

```
def init_two_layer_model(input_size, hidden_size, output_size):
  # initialize a model
  model = {}
  model['W1'] = 0.0001 * np.random.randn(input_size, hidden_size)
  model['b1'] = np.zeros(hidden_size)
  model['W2'] = 0.0001 * np.random.randn(hidden_size, output_size)
  model['b2'] = np.zeros(output_size)
  return model
```

```
model = init_two_layer_model(32*32*3, 50, 10) # input_size, hidden_size, number of classes
loss, grad = two_layer_net(X_train, model, y_train, 1e3)
print loss
```

```
3.06859716482
```

crank up regularization

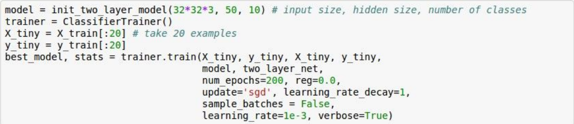
loss went up, good. (sanity check)

Fei-Fei Li & Andrej Karpathy & Justin Johnson Lecture 5 - 75 20 Jan 2016

Debugging optimizer

Lets try to train now…

Tip: Make sure that you can overfit very small portion of the training data



```
model = init_two_layer_model(32*32*3, 50, 10) # input size, hidden size, number of classes
trainer = ClassifierTrainer()
X_tiny = X_train[:20] # take 20 examples
y_tiny = y_train[:20]
best_model, stats = trainer.train(X_tiny, y_tiny, X_tiny, y_tiny,
                                  model, two_layer_net,
                                  num_epochs=200, reg=0.0,
                                  update='sgd', learning_rate_decay=1,
                                  sample_batches = False,
                                  learning_rate=1e-3, verbose=True)
```

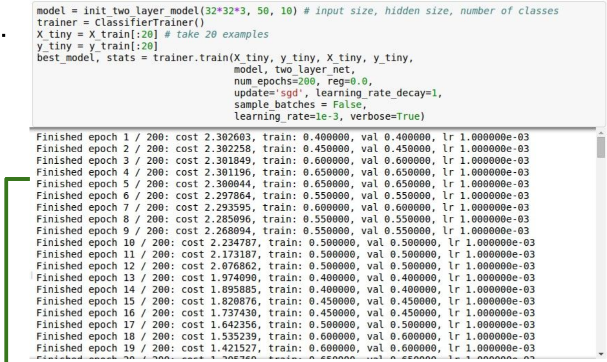
The above code:

- take the first 20 examples from CIFAR-10
- turn off regularization (reg = 0.0)
- use simple vanilla ‘sgd’

Debugging optimizer

Lets try to train now…

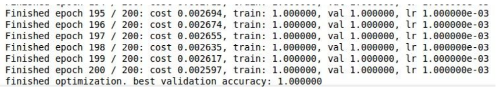
Tip: Make sure that you can overfit very small portion of the training data



```
model = init_two_layer_model(32*32*3, 50, 10) # input size, hidden size, number of classes
trainer = ClassifierTrainer()
X_tiny = X_train[:20] # take 20 examples
y_tiny = y_train[:20]
best_model, stats = trainer.train(X_tiny, y_tiny, X_tiny, y_tiny,
                                  model, two_layer_net,
                                  num_epochs=200, reg=0.0,
                                  update='sgd', learning_rate_decay=1,
                                  sample_batches = False,
                                  learning_rate=1e-3, verbose=True)
```

```
Finished epoch 1 / 200: cost 2.302603, train: 0.400000, val 0.400000, lr 1.000000e-03
Finished epoch 2 / 200: cost 2.302258, train: 0.450000, val 0.450000, lr 1.000000e-03
Finished epoch 3 / 200: cost 2.301849, train: 0.600000, val 0.600000, lr 1.000000e-03
Finished epoch 4 / 200: cost 2.301196, train: 0.650000, val 0.650000, lr 1.000000e-03
Finished epoch 5 / 200: cost 2.300044, train: 0.650000, val 0.650000, lr 1.000000e-03
Finished epoch 6 / 200: cost 2.297864, train: 0.550000, val 0.550000, lr 1.000000e-03
Finished epoch 7 / 200: cost 2.293595, train: 0.600000, val 0.600000, lr 1.000000e-03
Finished epoch 8 / 200: cost 2.285096, train: 0.550000, val 0.550000, lr 1.000000e-03
Finished epoch 9 / 200: cost 2.268094, train: 0.550000, val 0.550000, lr 1.000000e-03
Finished epoch 10 / 200: cost 2.234787, train: 0.500000, val 0.500000, lr 1.000000e-03
Finished epoch 11 / 200: cost 2.173187, train: 0.500000, val 0.500000, lr 1.000000e-03
Finished epoch 12 / 200: cost 2.076862, train: 0.500000, val 0.500000, lr 1.000000e-03
Finished epoch 13 / 200: cost 1.974090, train: 0.400000, val 0.400000, lr 1.000000e-03
Finished epoch 14 / 200: cost 1.895885, train: 0.400000, val 0.400000, lr 1.000000e-03
Finished epoch 15 / 200: cost 1.820876, train: 0.450000, val 0.450000, lr 1.000000e-03
Finished epoch 16 / 200: cost 1.737430, train: 0.450000, val 0.450000, lr 1.000000e-03
Finished epoch 17 / 200: cost 1.642356, train: 0.500000, val 0.500000, lr 1.000000e-03
Finished epoch 18 / 200: cost 1.535239, train: 0.600000, val 0.600000, lr 1.000000e-03
Finished epoch 19 / 200: cost 1.421527, train: 0.600000, val 0.600000, lr 1.000000e-03
Finished epoch 20 / 200: cost 1.395760, train: 0.650000, val 0.650000, lr 1.000000e-03
```

Very small loss, train accuracy 1.00, nice!



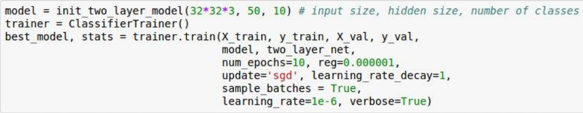
```
Finished epoch 194 / 200: cost 0.002715, train: 1.000000, val 1.000000, lr 1.000000e-03
Finished epoch 195 / 200: cost 0.002694, train: 1.000000, val 1.000000, lr 1.000000e-03
Finished epoch 196 / 200: cost 0.002674, train: 1.000000, val 1.000000, lr 1.000000e-03
Finished epoch 197 / 200: cost 0.002655, train: 1.000000, val 1.000000, lr 1.000000e-03
Finished epoch 198 / 200: cost 0.002635, train: 1.000000, val 1.000000, lr 1.000000e-03
Finished epoch 199 / 200: cost 0.002617, train: 1.000000, val 1.000000, lr 1.000000e-03
Finished epoch 200 / 200: cost 0.002597, train: 1.000000, val 1.000000, lr 1.000000e-03
finished optimization. best validation accuracy: 1.000000
```

Fei-Fei Li & Andrej Karpathy & Justin Johnson Lecture 5 - 77 20 Jan 2016

Debugging optimizer

Lets try to train now…

I like to start with small regularization and find learning rate that makes the loss go down.

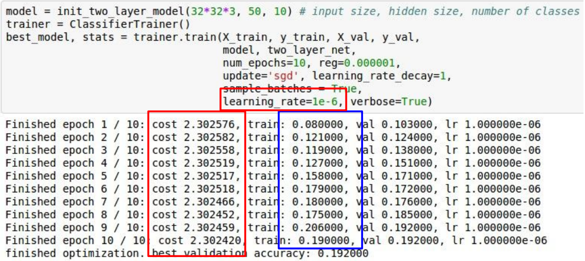


```
model = init_two_layer_model(32*32*3, 50, 10) # input size, hidden size, number of classes
trainer = ClassifierTrainer()
best_model, stats = trainer.train(X_train, y_train, X_val, y_val,
                                  model, two_layer_net,
                                  num_epochs=10, reg=0.000001,
                                  update='sgd', learning_rate_decay=1,
                                  sample_batches = True,
                                  learning_rate=1e-6, verbose=True)
```

Debugging optimizer

Lets try to train now…

I like to start with small regularization and find learning rate that makes the loss go down.



```
model = init_two_layer_model(32*32*3, 50, 10) # input size, hidden size, number of classes
trainer = ClassifierTrainer()
best_model, stats = trainer.train(X_train, y_train, X_val, y_val,
                                  model, two_layer_net,
                                  num_epochs=10, reg=0.000001,
                                  update='sgd', learning_rate_decay=1,
                                  ~~sample_batches = True,~~
                                  learning_rate=1e-6, verbose=True)
```

```
Finished epoch 1 / 10: cost 2.302576, train: 0.080000, val 0.103000, lr 1.000000e-06
Finished epoch 2 / 10: cost 2.302582, train: 0.121000, val 0.124000, lr 1.000000e-06
Finished epoch 3 / 10: cost 2.302558, train: 0.119000, val 0.138000, lr 1.000000e-06
Finished epoch 4 / 10: cost 2.302519, train: 0.127000, val 0.151000, lr 1.000000e-06
Finished epoch 5 / 10: cost 2.302517, train: 0.158000, val 0.171000, lr 1.000000e-06
Finished epoch 6 / 10: cost 2.302518, train: 0.179000, val 0.172000, lr 1.000000e-06
Finished epoch 7 / 10: cost 2.302466, train: 0.180000, val 0.176000, lr 1.000000e-06
Finished epoch 8 / 10: cost 2.302452, train: 0.175000, val 0.185000, lr 1.000000e-06
Finished epoch 9 / 10: cost 2.302459, train: 0.206000, val 0.192000, lr 1.000000e-06
Finished epoch 10 / 10: cost 2.302420, train: 0.190000, val 0.192000, lr 1.000000e-06
finished optimization. best validation accuracy: 0.192000
```

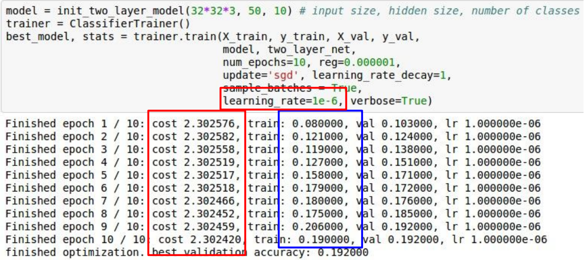
Loss barely changing

Debugging optimizer

Lets try to train now…

I like to start with small regularization and find learning rate that makes the loss go down.

loss not going down: learning rate too low



```
model = init_two_layer_model(32*32*3, 50, 10) # input size, hidden size, number of classes
trainer = ClassifierTrainer()
best_model, stats = trainer.train(X_train, y_train, X_val, y_val,
                                  model, two_layer_net,
                                  num_epochs=10, reg=0.000001,
                                  update='sgd', learning_rate_decay=1,
                                  sample_batches = True,
                                  learning_rate=1e-6, verbose=True)
```

```
Finished epoch 1 / 10: cost 2.302576, train: 0.080000, val 0.103000, lr 1.000000e-06
Finished epoch 2 / 10: cost 2.302582, train: 0.121000, val 0.124000, lr 1.000000e-06
Finished epoch 3 / 10: cost 2.302558, train: 0.119000, val 0.138000, lr 1.000000e-06
Finished epoch 4 / 10: cost 2.302519, train: 0.127000, val 0.151000, lr 1.000000e-06
Finished epoch 5 / 10: cost 2.302517, train: 0.158000, val 0.171000, lr 1.000000e-06
Finished epoch 6 / 10: cost 2.302518, train: 0.179000, val 0.172000, lr 1.000000e-06
Finished epoch 7 / 10: cost 2.302466, train: 0.180000, val 0.176000, lr 1.000000e-06
Finished epoch 8 / 10: cost 2.302452, train: 0.175000, val 0.185000, lr 1.000000e-06
Finished epoch 9 / 10: cost 2.302459, train: 0.206000, val 0.192000, lr 1.000000e-06
Finished epoch 10 / 10: cost 2.302420, train: 0.190000, val 0.192000, lr 1.000000e-06
finished optimization. best validation accuracy: 0.192000
```

Loss barely changing: Learning rate is probably too low

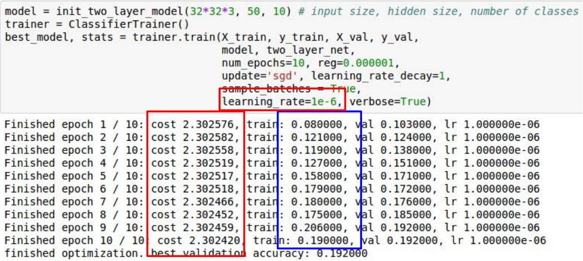
Fei-Fei Li & Andrej Karpathy & Justin Johnson Lecture 5 - 80 20 Jan 2016

Debugging optimizer

Lets try to train now…

I like to start with small regularization and find learning rate that makes the loss go down.

loss not going down: learning rate too low



```
model = init_two_layer_model(32*32*3, 50, 10) # input size, hidden size, number of classes
trainer = ClassifierTrainer()
best_model, stats = trainer.train(X_train, y_train, X_val, y_val,
                                  model, two_layer_net,
                                  num_epochs=10, reg=0.000001,
                                  update='sgd', learning_rate_decay=1,
                                  sample_batches = True,
                                  learning_rate=1e-6, verbose=True)
```

```
Finished epoch 1 / 10: cost 2.302576, train: 0.080000, val 0.103000, lr 1.000000e-06
Finished epoch 2 / 10: cost 2.302582, train: 0.121000, val 0.124000, lr 1.000000e-06
Finished epoch 3 / 10: cost 2.302558, train: 0.119000, val 0.138000, lr 1.000000e-06
Finished epoch 4 / 10: cost 2.302519, train: 0.127000, val 0.151000, lr 1.000000e-06
Finished epoch 5 / 10: cost 2.302517, train: 0.158000, val 0.171000, lr 1.000000e-06
Finished epoch 6 / 10: cost 2.302518, train: 0.179000, val 0.172000, lr 1.000000e-06
Finished epoch 7 / 10: cost 2.302466, train: 0.180000, val 0.176000, lr 1.000000e-06
Finished epoch 8 / 10: cost 2.302452, train: 0.175000, val 0.185000, lr 1.000000e-06
Finished epoch 9 / 10: cost 2.302459, train: 0.206000, val 0.192000, lr 1.000000e-06
Finished epoch 10 / 10: cost 2.302420, train: 0.190000, val 0.192000, lr 1.000000e-06
finished optimization. best validation accuracy: 0.192000
```

Loss barely changing: Learning rate is probably too low

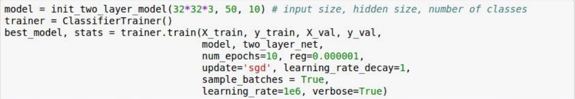
Notice train/val accuracy goes to 20% though, what's up with that? (remember this is softmax)

Debugging optimizer

Lets try to train now…

I like to start with small regularization and find learning rate that makes the loss go down.

loss not going down:
learning rate too low



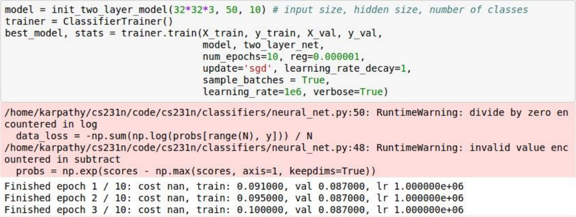
```
model = init_two_layer_model(32*32*3, 50, 10) # input size, hidden size, number of classes
trainer = ClassifierTrainer()
best_model, stats = trainer.train(X_train, y_train, X_val, y_val,
                                  model, two_layer_net,
                                  num_epochs=10, reg=0.000001,
                                  update='sgd', learning_rate_decay=1,
                                  sample_batches = True,
                                  learning_rate=1e6, verbose=True)
```

Okay now lets try learning rate 1e6. What could possibly go wrong?

Debugging optimizer

Lets try to train now…

I like to start with small regularization and find learning rate that makes the loss go down.



```
model = init_two_layer_model(32*32*3, 50, 10) # input size, hidden size, number of classes
trainer = ClassifierTrainer()
best_model, stats = trainer.train(X_train, y_train, X_val, y_val,
                                  model, two_layer_net,
                                  num_epochs=10, reg=0.000001,
                                  update='sgd', learning_rate_decay=1,
                                  sample_batches = True,
                                  learning_rate=1e6, verbose=True)
```

```
/home/karpathy/cs231n/code/cs231n/classifiers/neural_net.py:50: RuntimeWarning: divide by zero en
countered in log
  data_loss = -np.sum(np.log(probs[range(N), y])) / N
/home/karpathy/cs231n/code/cs231n/classifiers/neural_net.py:48: RuntimeWarning: invalid value enc
ountered in subtract
  probs = np.exp(scores - np.max(scores, axis=1, keepdims=True))
Finished epoch 1 / 10: cost nan, train: 0.091000, val 0.087000, lr 1.000000e+06
Finished epoch 2 / 10: cost nan, train: 0.095000, val 0.087000, lr 1.000000e+06
Finished epoch 3 / 10: cost nan, train: 0.100000, val 0.087000, lr 1.000000e+06
```

loss not going down:
learning rate too low
loss exploding:
learning rate too high

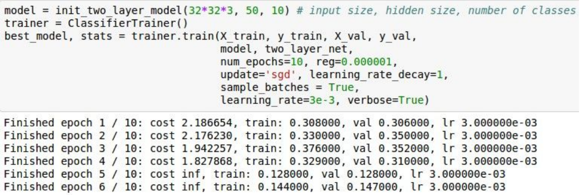
cost: NaN almost always means high learning rate...

Fei-Fei Li & Andrej Karpathy & Justin Johnson Lecture 5 - 83 20 Jan 2016

Debugging optimizer

Lets try to train now…

I like to start with small regularization and find learning rate that makes the loss go down.



```
model = init_two_layer_model(32*32*3, 50, 10) # input size, hidden size, number of classes
trainer = ClassifierTrainer()
best_model, stats = trainer.train(X_train, y_train, X_val, y_val,
                                  model, two_layer_net,
                                  num_epochs=10, reg=0.000001,
                                  update='sgd', learning_rate_decay=1,
                                  sample_batches = True,
                                  learning_rate=3e-3, verbose=True)
```

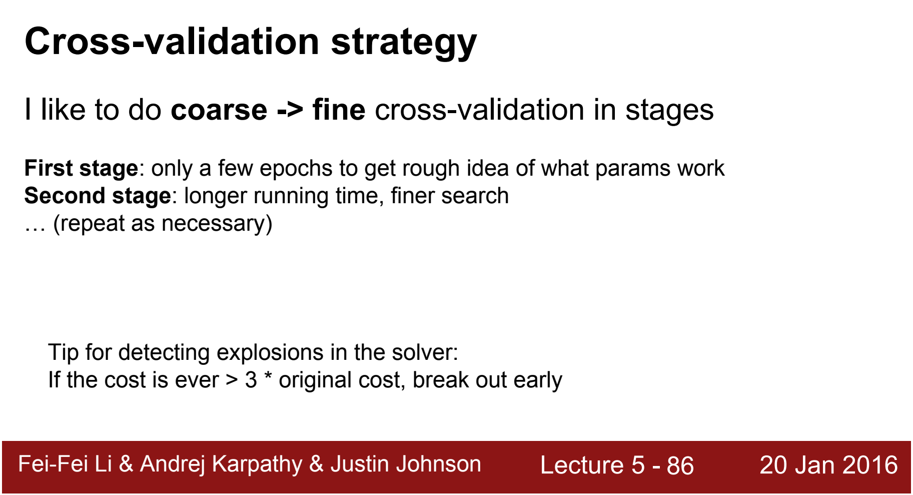
```
Finished epoch 1 / 10: cost 2.186654, train: 0.308000, val 0.306000, lr 3.000000e-03
Finished epoch 2 / 10: cost 2.176230, train: 0.330000, val 0.350000, lr 3.000000e-03
Finished epoch 3 / 10: cost 1.942257, train: 0.376000, val 0.352000, lr 3.000000e-03
Finished epoch 4 / 10: cost 1.827868, train: 0.329000, val 0.310000, lr 3.000000e-03
Finished epoch 5 / 10: cost inf, train: 0.128000, val 0.128000, lr 3.000000e-03
Finished epoch 6 / 10: cost inf, train: 0.144000, val 0.147000, lr 3.000000e-03
```

3e-3 is still too high. Cost explodes….

loss not going down:
learning rate too low
loss exploding:
learning rate too high

=> Rough range for learning rate we should be cross-validating is somewhere [1e-3 … 1e-5]

Hyperparameter optimization



Cross-validation strategy

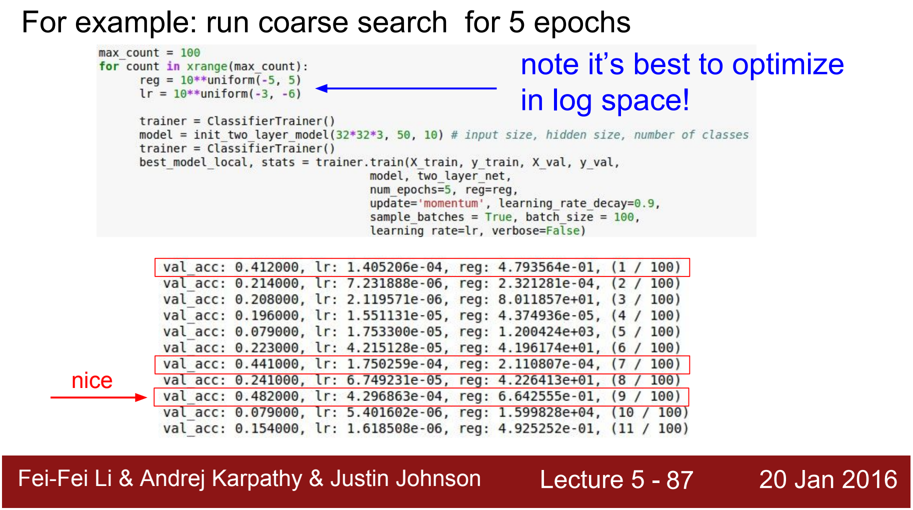
I like to do **coarse -> fine** cross-validation in stages

First stage: only a few epochs to get rough idea of what params work
Second stage: longer running time, finer search
… (repeat as necessary)

Tip for detecting explosions in the solver:
If the cost is ever > 3 * original cost, break out early

Fei-Fei Li & Andrej Karpathy & Justin Johnson Lecture 5 - 86 20 Jan 2016

Hyperparameter optimization



For example: run coarse search for 5 epochs

```
max_count = 100
for count in xrange(max_count):
      reg = 10**uniform(-5, 5)
      lr = 10**uniform(-3, -6)

      trainer = ClassifierTrainer()
      model = init_two_layer_model(32*32*3, 50, 10) # input size, hidden size, number of classes
      trainer = ClassifierTrainer()
      best_model_local, stats = trainer.train(X_train, y_train, X_val, y_val,
                                   model, two_layer_net,
                                   num_epochs=5, reg=reg,
                                   update='momentum', learning_rate_decay=0.9,
                                   sample_batches = True, batch_size = 100,
                                   learning_rate=lr, verbose=False)
```

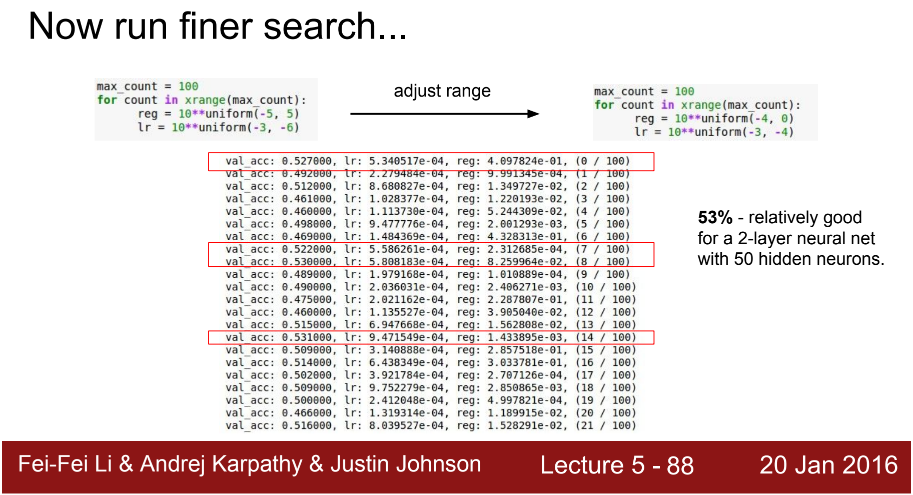
note it's best to optimize in log space!

```
val_acc: 0.412000, lr: 1.405206e-04, reg: 4.793564e-01, (1 / 100)
val_acc: 0.214000, lr: 7.231888e-06, reg: 2.321281e-04, (2 / 100)
val_acc: 0.208000, lr: 2.119571e-06, reg: 8.011857e+01, (3 / 100)
val_acc: 0.196000, lr: 1.551131e-05, reg: 4.374936e-05, (4 / 100)
val_acc: 0.079000, lr: 1.753300e-05, reg: 1.200424e+03, (5 / 100)
val_acc: 0.223000, lr: 4.215128e-05, reg: 4.196174e+01, (6 / 100)
val_acc: 0.441000, lr: 1.750259e-04, reg: 2.110807e-04, (7 / 100)
val_acc: 0.241000, lr: 6.749231e-05, reg: 4.226413e+01, (8 / 100)
val_acc: 0.482000, lr: 4.296863e-04, reg: 6.642555e-01, (9 / 100)
val_acc: 0.079000, lr: 5.401602e-06, reg: 1.599828e+04, (10 / 100)
val_acc: 0.154000, lr: 1.618508e-06, reg: 4.925252e-01, (11 / 100)
```

nice

Fei-Fei Li & Andrej Karpathy & Justin Johnson   Lecture 5 - 87   20 Jan 2016

Hyperparameter optimization



Now run finer search...

```
max_count = 100
for count in xrange(max_count):
      reg = 10**uniform(-5, 5)
      lr = 10**uniform(-3, -6)
```

adjust range

```
max_count = 100
for count in xrange(max_count):
      reg = 10**uniform(-4, 0)
      lr = 10**uniform(-3, -4)
```

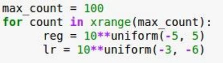
```
val_acc: 0.527000, lr: 5.340517e-04, reg: 4.097824e-01, (0 / 100)
val_acc: 0.492000, lr: 2.279484e-04, reg: 9.991345e-04, (1 / 100)
val_acc: 0.512000, lr: 8.680827e-04, reg: 1.349727e-02, (2 / 100)
val_acc: 0.461000, lr: 1.028377e-04, reg: 1.220193e-02, (3 / 100)
val_acc: 0.460000, lr: 1.113730e-04, reg: 5.244309e-02, (4 / 100)
val_acc: 0.498000, lr: 9.477776e-04, reg: 2.001293e-03, (5 / 100)
val_acc: 0.469000, lr: 1.484369e-04, reg: 4.328313e-01, (6 / 100)
val_acc: 0.522000, lr: 5.586261e-04, reg: 2.312685e-04, (7 / 100)
val_acc: 0.530000, lr: 5.808183e-04, reg: 8.259964e-02, (8 / 100)
val_acc: 0.489000, lr: 1.979168e-04, reg: 1.010889e-04, (9 / 100)
val_acc: 0.490000, lr: 2.036031e-04, reg: 2.406271e-03, (10 / 100)
val_acc: 0.475000, lr: 2.021162e-04, reg: 2.287807e-01, (11 / 100)
val_acc: 0.460000, lr: 1.135527e-04, reg: 3.905040e-02, (12 / 100)
val_acc: 0.515000, lr: 6.947668e-04, reg: 1.562808e-02, (13 / 100)
val_acc: 0.531000, lr: 9.471549e-04, reg: 1.433895e-03, (14 / 100)
val_acc: 0.509000, lr: 3.140888e-04, reg: 2.857518e-01, (15 / 100)
val_acc: 0.514000, lr: 6.438349e-04, reg: 3.033781e-01, (16 / 100)
val_acc: 0.502000, lr: 3.921784e-04, reg: 2.707126e-04, (17 / 100)
val_acc: 0.509000, lr: 9.752279e-04, reg: 2.850865e-03, (18 / 100)
val_acc: 0.500000, lr: 2.412048e-04, reg: 4.997821e-04, (19 / 100)
val_acc: 0.466000, lr: 1.319314e-04, reg: 1.189915e-02, (20 / 100)
val_acc: 0.516000, lr: 8.039527e-04, reg: 1.528291e-02, (21 / 100)
```

53% - relatively good for a 2-layer neural net with 50 hidden neurons.

Fei-Fei Li & Andrej Karpathy & Justin Johnson   Lecture 5 - 88   20 Jan 2016

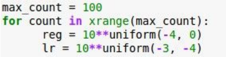
Hyperparameter optimization

Now run finer search...

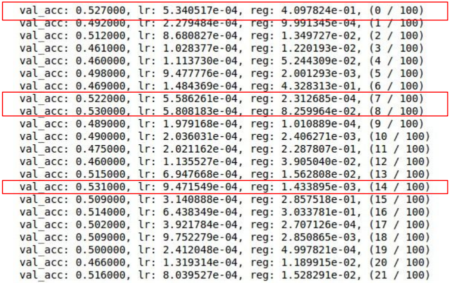


```
max_count = 100
for count in xrange(max_count):
      reg = 10**uniform(-5, 5)
      lr = 10**uniform(-3, -6)
```

adjust range



```
max_count = 100
for count in xrange(max_count):
      reg = 10**uniform(-4, 0)
      lr = 10**uniform(-3, -4)
```

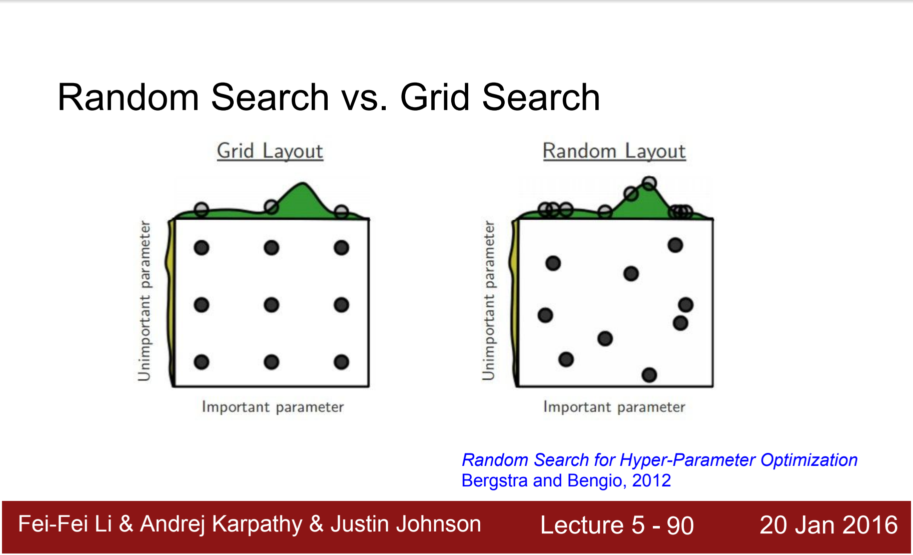


```
val_acc: 0.527000, lr: 5.340517e-04, reg: 4.097824e-01, (0 / 100)
val_acc: 0.492000, lr: 2.279484e-04, reg: 9.991345e-04, (1 / 100)
val_acc: 0.512000, lr: 8.680827e-04, reg: 1.349727e-02, (2 / 100)
val_acc: 0.461000, lr: 1.028377e-04, reg: 1.220193e-02, (3 / 100)
val_acc: 0.460000, lr: 1.113730e-04, reg: 5.244309e-02, (4 / 100)
val_acc: 0.498000, lr: 9.477776e-04, reg: 2.001293e-03, (5 / 100)
val_acc: 0.469000, lr: 1.484369e-04, reg: 4.328313e-01, (6 / 100)
val_acc: 0.522000, lr: 5.586261e-04, reg: 2.312685e-04, (7 / 100)
val_acc: 0.530000, lr: 5.808183e-04, reg: 8.259964e-02, (8 / 100)
val_acc: 0.489000, lr: 1.979168e-04, reg: 1.010889e-04, (9 / 100)
val_acc: 0.490000, lr: 2.036031e-04, reg: 2.406271e-03, (10 / 100)
val_acc: 0.475000, lr: 2.021162e-04, reg: 2.287807e-01, (11 / 100)
val_acc: 0.460000, lr: 1.135527e-04, reg: 3.905040e-02, (12 / 100)
val_acc: 0.515000, lr: 6.947668e-04, reg: 1.562808e-02, (13 / 100)
val_acc: 0.531000, lr: 9.471549e-04, reg: 1.433895e-03, (14 / 100)
val_acc: 0.509000, lr: 3.140888e-04, reg: 2.857518e-01, (15 / 100)
val_acc: 0.514000, lr: 6.438349e-04, reg: 3.033781e-01, (16 / 100)
val_acc: 0.502000, lr: 3.921784e-04, reg: 2.707126e-04, (17 / 100)
val_acc: 0.509000, lr: 9.752279e-04, reg: 2.850865e-03, (18 / 100)
val_acc: 0.500000, lr: 2.412048e-04, reg: 4.997821e-04, (19 / 100)
val_acc: 0.466000, lr: 1.319314e-04, reg: 1.189915e-02, (20 / 100)
val_acc: 0.516000, lr: 8.039527e-04, reg: 1.528291e-02, (21 / 100)
```

53% - relatively good for a 2-layer neural net with 50 hidden neurons.

But this best cross-validation result is worrying. Why?

Hyperparameter optimization



Random Search vs. Grid Search

Random Search for Hyper-Parameter Optimization
Bergstra and Bengio, 2012

Fei-Fei Li & Andrej Karpathy & Justin Johnson Lecture 5 - 90 20 Jan 2016

Conclusions of last lecture

- BP is just chain rule in calculus
- Use ReLU. Never use Sigmoid (use Tanh instead)
- Input preprocessing is no longer very important
    - Do subtract mean
    - Whitening and normalizing are not much needed
- Weight initialization on the other hand is extremely important for deep networks
- Use batch normalization if you can
- Use dropout
- Use Adam (or maybe RMSprop) for optimizer. If you don't have much data, can consider LBFGS
- Need to babysit your learning for real-world problems
- Never use grid search for tuning your hyperparameters

Conclusions of last lecture

- BP is just chain rule in calculus
- Use ReLU. Never use Sigmoid (use Tanh instead)
- Input preprocessing is no longer very important
    - Do subtract mean
    - Whitening and normalizing are not much needed
- Weight initialization on the other hand is extremely important for deep networks
- Use batch normalization if you can
- Use dropout
- Use Adam (or maybe RMSprop) for optimizer. If you don't have much data, can consider LBFGS
- Need to babysit your learning for real-world problems
- Never use grid search for tuning your hyperparameters

Conclusions of last lecture

- BP is just chain rule in calculus
- Use ReLU. Never use Sigmoid (use Tanh instead)
- Input preprocessing is no longer very important
    - Do subtract mean
    - Whitening and normalizing are not much needed
- Weight initialization on the other hand is extremely important for deep networks
- Use batch normalization if you can
- Use dropout
- Use Adam (or maybe RMSprop) for optimizer. If you don't have much data, can consider LBFGS
- Need to babysit your learning for real-world problems
- Never use grid search for tuning your hyperparameters

Conclusions of last lecture

- BP is just chain rule in calculus
- Use ReLU. Never use Sigmoid (use Tanh instead)
- Input preprocessing is no longer very important
  - Do subtract mean
  - Whitening and normalizing are not much needed
- Weight initialization on the other hand is extremely important for deep networks
- Use batch normalization if you can
- Use dropout
- Use Adam (or maybe RMSprop) for optimizer. If you don't have much data, can consider LBFGS
- Need to babysit your learning for real-world problems
- Never use grid search for tuning your hyperparameters

Conclusions of last lecture

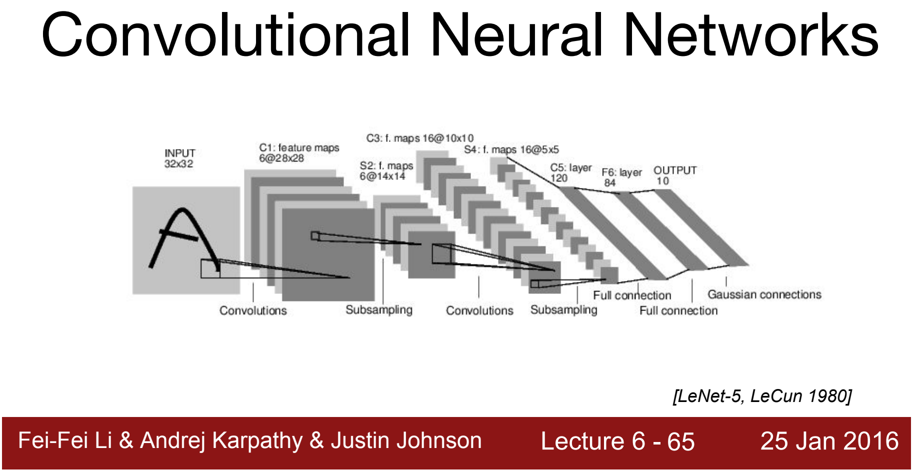
- BP is just chain rule in calculus
- Use ReLU. Never use Sigmoid (use Tanh instead)
- Input preprocessing is no longer very important
  - Do subtract mean
  - Whitening and normalizing are not much needed
- Weight initialization on the other hand is extremely important for deep networks
- Use batch normalization if you can
- Use dropout
- Use Adam (or maybe RMSprop) for optimizer. If you don't have much data, can consider LBFGS
- Need to babysit your learning for real-world problems
- Never use grid search for tuning your hyperparameters

Conclusions of last lecture

- BP is just chain rule in calculus
- Use ReLU. Never use Sigmoid (use Tanh instead)
- Input preprocessing is no longer very important
  - Do subtract mean
  - Whitening and normalizing are not much needed
- Weight initialization on the other hand is extremely important for deep networks
- Use batch normalization if you can
- Use dropout
- Use Adam (or maybe RMSprop) for optimizer. If you don't have much data, can consider LBFGS
- Need to babysit your learning for real-world problems
- Never use grid search for tuning your hyperparameters

Conclusions of last lecture

- BP is just chain rule in calculus
- Use ReLU. Never use Sigmoid (use Tanh instead)
- Input preprocessing is no longer very important
  - Do subtract mean
  - Whitening and normalizing are not much needed
- Weight initialization on the other hand is extremely important for deep networks
- Use batch normalization if you can
- Use dropout
- Use Adam (or maybe RMSprop) for optimizer. If you don't have much data, can consider LBFGS
- Need to babysit your learning for real-world problems
- Never use grid search for tuning your hyperparameters

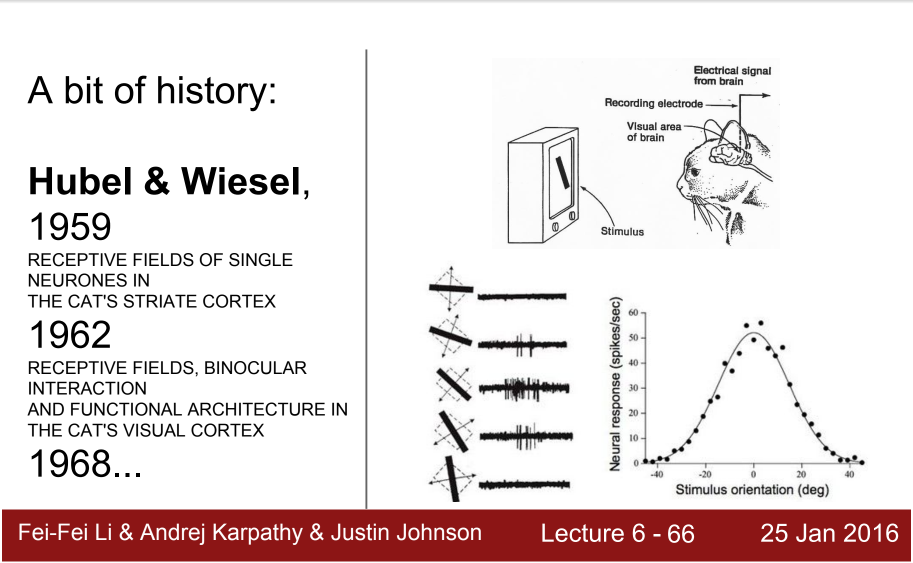


Convolutional Neural Networks

[LeNet-5, LeCun 1980]

Fei-Fei Li & Andrej Karpathy & Justin Johnson Lecture 6 - 65 25 Jan 2016

CNN history



A bit of history:

Hubel & Wiesel,

1959

RECEPTIVE FIELDS OF SINGLE NEURONES IN THE CAT'S STRIATE CORTEX

1962

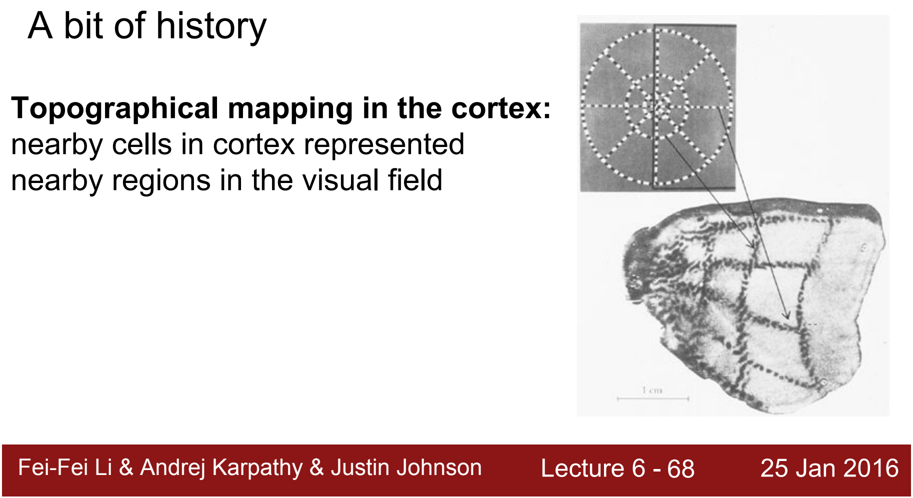
RECEPTIVE FIELDS, BINOCULAR INTERACTION AND FUNCTIONAL ARCHITECTURE IN THE CAT'S VISUAL CORTEX

1968...

Electrical signal from brain
Recording electrode
Visual area of brain
Stimulus
Neural response (spikes/sec)
60
50
40
30
20
10
0
-40
-20
0
20
40
Stimulus orientation (deg)

Fei-Fei Li & Andrej Karpathy & Justin Johnson Lecture 6 - 66 25 Jan 2016

CNN history

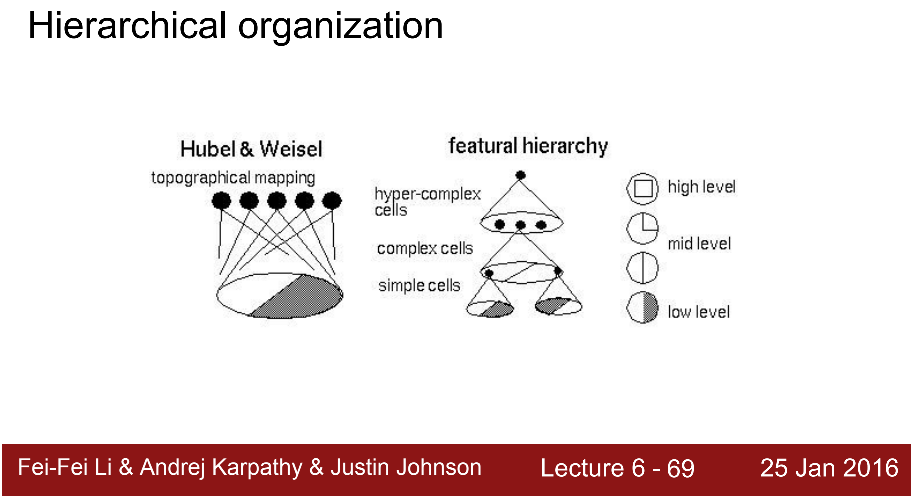


A bit of history

Topographical mapping in the cortex:
nearby cells in cortex represented
nearby regions in the visual field

Fei-Fei Li & Andrej Karpathy & Justin Johnson Lecture 6 - 68 25 Jan 2016

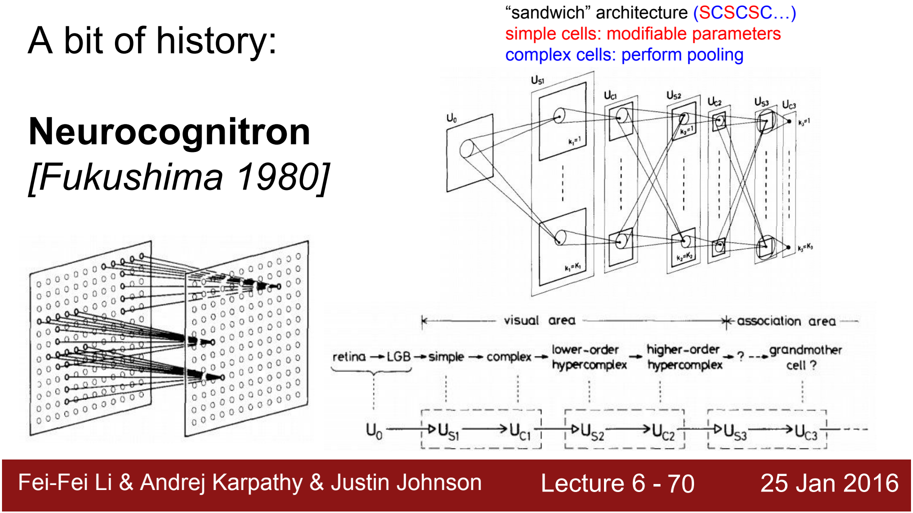
CNN history



Hierarchical organization

Fei-Fei Li & Andrej Karpathy & Justin Johnson Lecture 6 - 69 25 Jan 2016

CNN history



A bit of history:

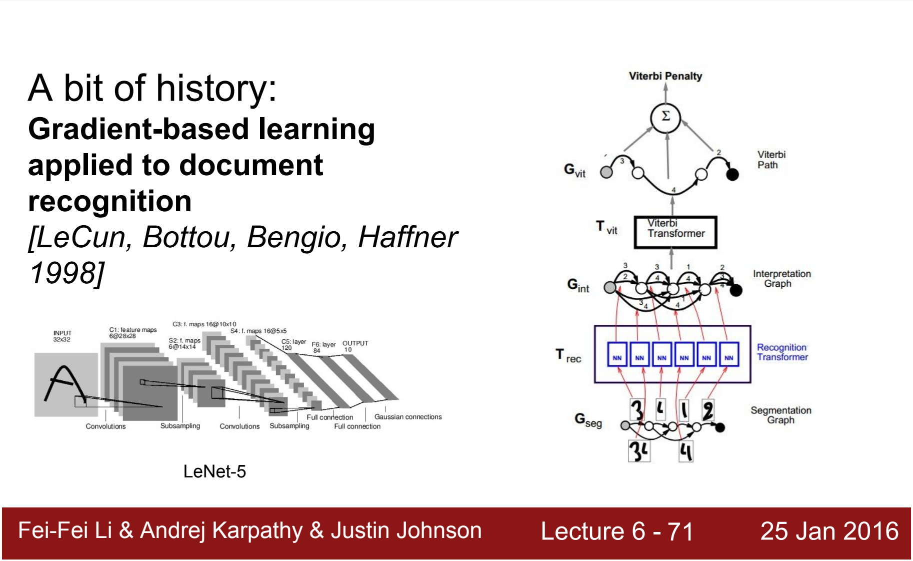
Neurocognitron
[Fukushima 1980]

“sandwich” architecture (SCSCSC…)
simple cells: modifiable parameters
complex cells: perform pooling

U_0
U_{S1}
U_{C1}
U_{S2}
U_{C2}
U_{S3}
U_{C3}
$k_1=1$
$k_1=K_1$
$k_2=1$
$k_2=K_2$
$k_3=1$
$k_3=K_3$
visual area
association area
retina → LGB → simple → complex → lower-order hypercomplex → higher-order hypercomplex → ? --→ grandmother cell ?
$U_0 \rightarrow U_{S1} \rightarrow U_{C1} \rightarrow U_{S2} \rightarrow U_{C2} \rightarrow U_{S3} \rightarrow U_{C3}$

Fei-Fei Li & Andrej Karpathy & Justin Johnson   Lecture 6 - 70   25 Jan 2016

CNN history



A bit of history:

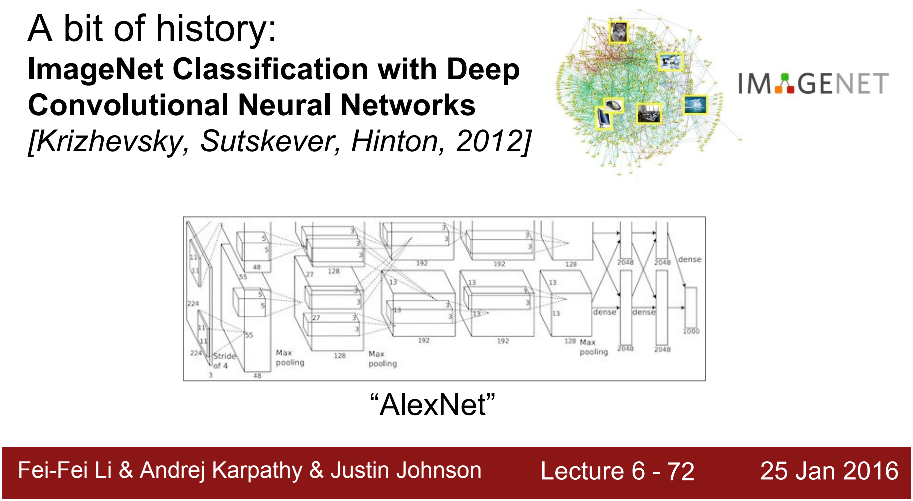
Gradient-based learning applied to document recognition

[LeCun, Bottou, Bengio, Haffner 1998]

LeNet-5

Fei-Fei Li & Andrej Karpathy & Justin Johnson Lecture 6 - 71 25 Jan 2016

CNN today



A bit of history:

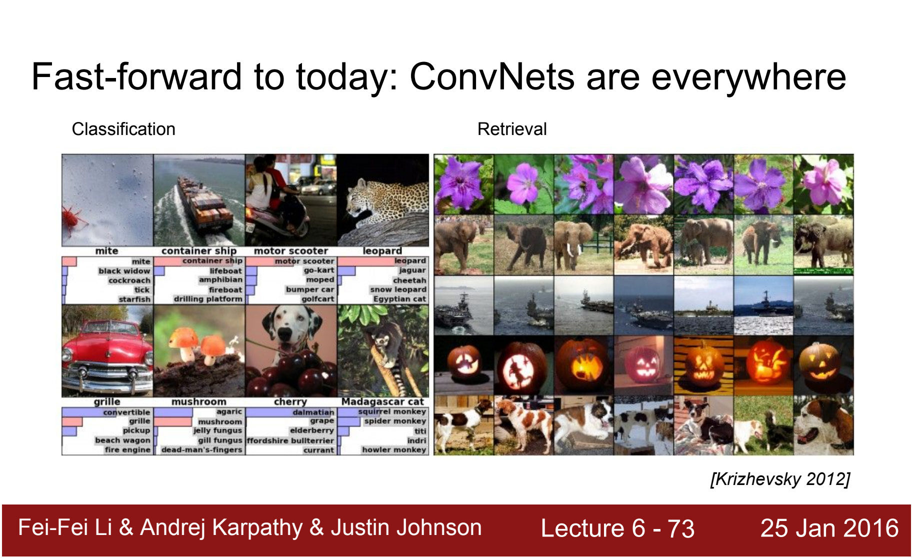
ImageNet Classification with Deep Convolutional Neural Networks

[Krizhevsky, Sutskever, Hinton, 2012]

"AlexNet"

Fei-Fei Li & Andrej Karpathy & Justin Johnson Lecture 6 - 72 25 Jan 2016

CNN today



Fast-forward to today: ConvNets are everywhere

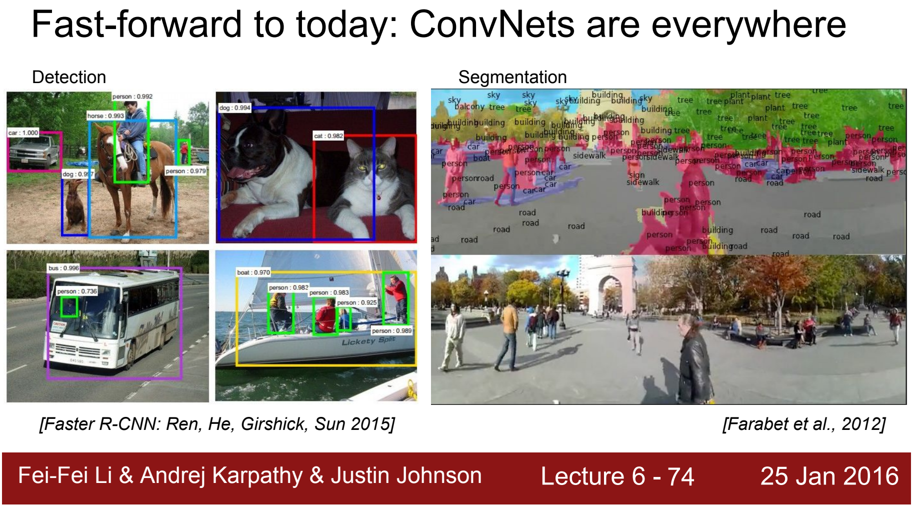
Classification

Retrieval

[Krizhevsky 2012]

Fei-Fei Li & Andrej Karpathy & Justin Johnson Lecture 6 - 73 25 Jan 2016

CNN today



Fast-forward to today: ConvNets are everywhere

Detection

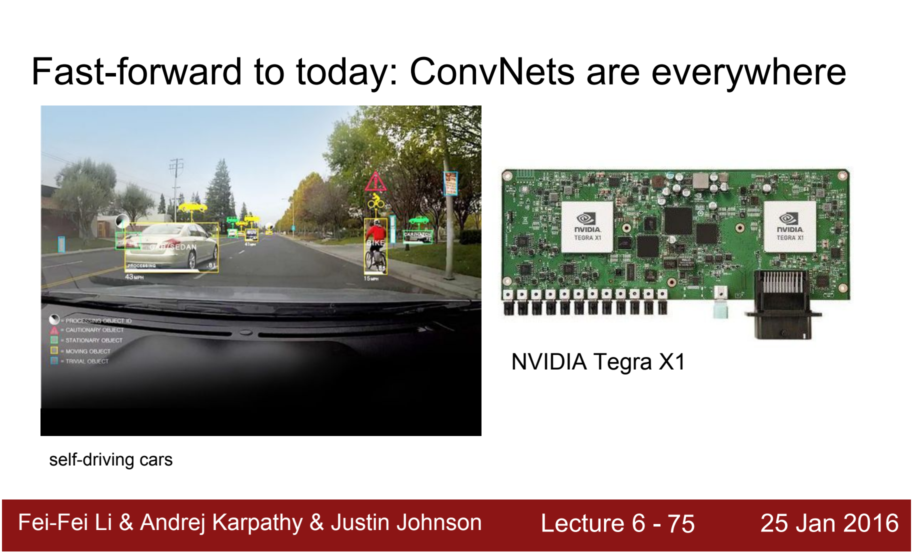
Segmentation

[Faster R-CNN: Ren, He, Girshick, Sun 2015]

[Farabet et al., 2012]

Fei-Fei Li & Andrej Karpathy & Justin Johnson   Lecture 6 - 74   25 Jan 2016

CNN today



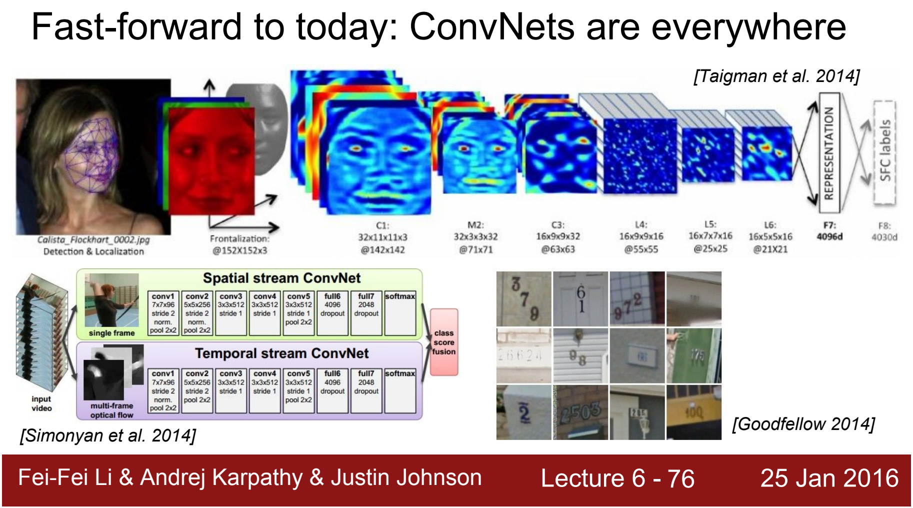
Fast-forward to today: ConvNets are everywhere

self-driving cars

NVIDIA Tegra X1

Fei-Fei Li & Andrej Karpathy & Justin Johnson   Lecture 6 - 75   25 Jan 2016

CNN today



Fast-forward to today: ConvNets are everywhere

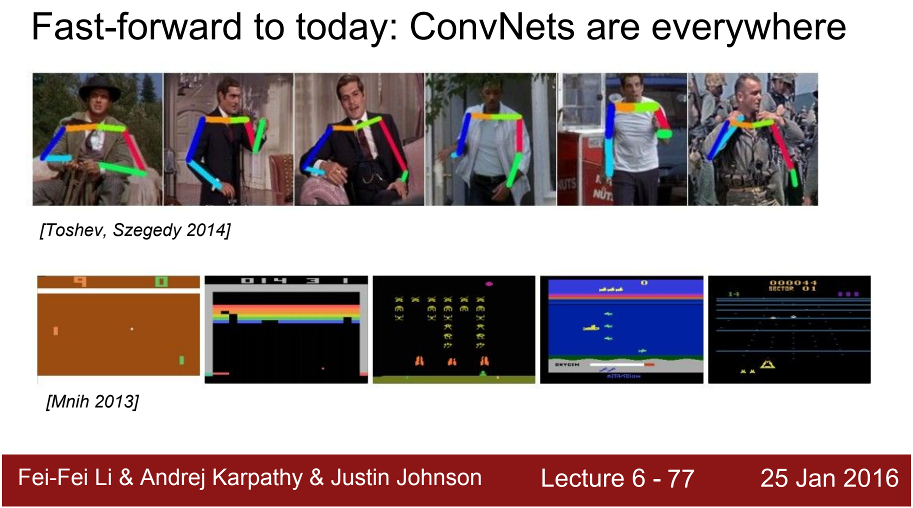
[Taigman et al. 2014]

[Simonyan et al. 2014]

[Goodfellow 2014]

Fei-Fei Li & Andrej Karpathy & Justin Johnson Lecture 6 - 76 25 Jan 2016

CNN today



Fast-forward to today: ConvNets are everywhere

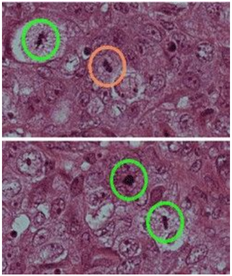
[Toshev, Szegedy 2014]

[Mnih 2013]

Fei-Fei Li & Andrej Karpathy & Justin Johnson Lecture 6 - 77 25 Jan 2016

CNN today

Fast-forward to today: ConvNets are everywhere



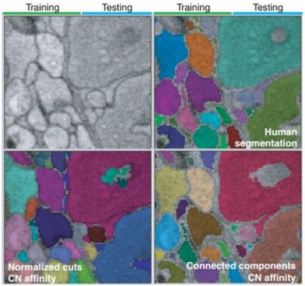
[Ciresan et al. 2013]

[Sermanet et al. 2011]
[Ciresan et al.]

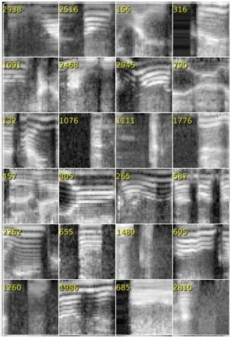
Fei-Fei Li & Andrej Karpathy & Justin Johnson Lecture 6 - 78 25 Jan 2016

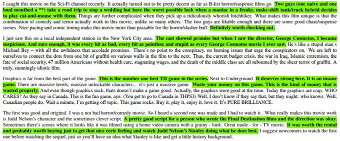
CNN today

Fast-forward to today: ConvNets are everywhere



[Turaga et al., 2010]





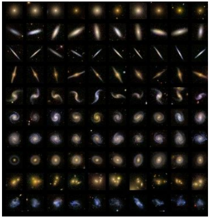
I caught this movie on the Sci-Fi channel recently. It actually turned out to be pretty decent as far as B-list horror/suspense films go. **Two guys (one naive and one loud mouthed a **) take a road trip to stop a wedding but have the worst possible luck when a maniac in a freaky, make-shift tank/truck hybrid decides to play cat-and-mouse with them.** Things are further complicated when they pick up a ridiculously whorish hitchhiker. What makes this film unique is that the combination of comedy and terror actually work in this movie, unlike so many others. The two guys are likable enough and there are some good chase/suspense scenes. Nice pacing and comic timing make this movie more than passable for the horror/slasher buff. **Definitely worth checking out.**

I just saw this on a local independent station in the New York City area. **The cast showed promise but when I saw the director, George Cosmotos, I became suspicious. And sure enough, it was every bit as bad, every bit as pointless and stupid as every George Cosmotos movie I ever saw.** He's like a stupid man's Michael Bey – with all the awfulness that accolade promises. There's no point to the conspiracy, no burning issues that urge the conspirators on. We are left to ourselves to connect the dots from one bit of graffiti on various walls in the film to the next. Thus, the current budget crisis, the war in Iraq, Islamic extremism, the fate of social security, 47 million Americans without health care, stagnating wages, and the death of the middle class are all subsumed by the sheer terror of graffiti. A truly, stunningly idiotic film.

Graphics is far from the best part of the game. **This is the number one best TH game in the series.** Next to Underground. **It deserves strong love. It is an insane game.** There are massive levels, massive unlockable characters... it's just a massive game. **Waste your money on this game. This is the kind of money that is wasted properly.** And even though graphics suck, thats doesn't make a game good. Actually, the graphics were good at the time. Today the graphics are crap. WHO CARES? As they say in Canada, This is the fun game, aye. (You get to go to Canada in THPS3) Well, I don't know if they say that, but they might. who knows. Well, Canadian people do. Wait a minute, I'm getting off topic. This game rocks. Buy it, play it, enjoy it, love it. It's PURE BRILLIANCE.

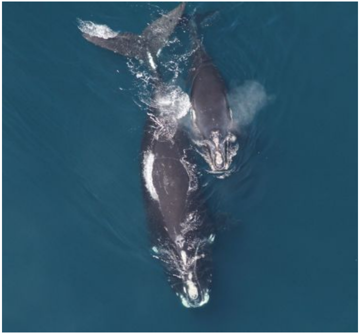
The first was good and original. I was a not bad horror/comedy movie. So I heard a second one was made and I had to watch it . What really makes this movie work is Judd Nelson's character and the sometimes clever script. **A pretty good script for a person who wrote the Final Destination films and the direction was okay.** Sometimes there's scenes where it looks like it was filmed using a home video camera with a grainy - look. Great made - for - TV movie. **It was worth the rental and probably worth buying just to get that nice eerie feeling and watch Judd Nelson's Stanley doing what he does best.** I suggest newcomers to watch the first one before watching the sequel, just so you'll have an idea what Stanley is like and get a little history background.

[Denil et al. 2014]



Fei-Fei Li & Andrej Karpathy & Justin Johnson Lecture 6 - 79 25 Jan 2016

CNN today

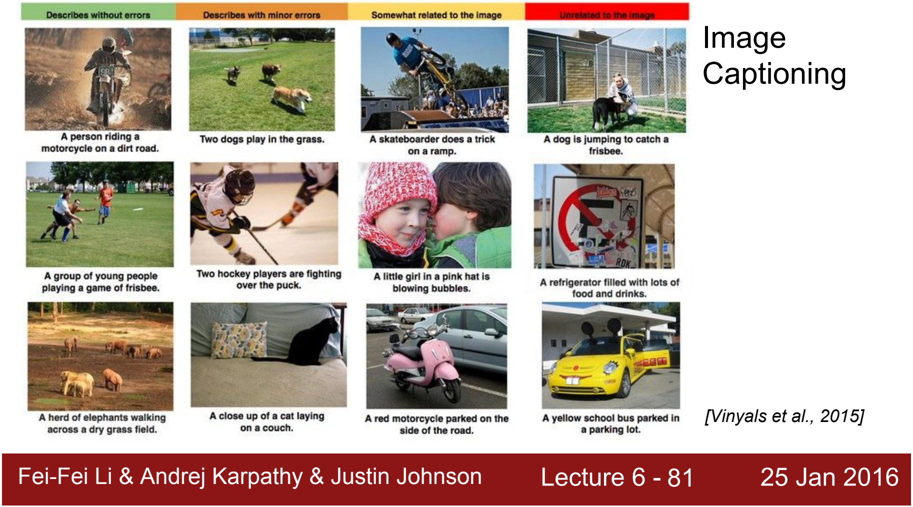


Whale recognition, Kaggle Challenge

Mnih and Hinton, 2010

Fei-Fei Li & Andrej Karpathy & Justin Johnson Lecture 6 - 80 25 Jan 2016

CNN today



Describes without errors	Describes with minor errors	Somewhat related to the image	Unrelated to the image
A person riding a motorcycle on a dirt road.	Two dogs play in the grass.	A skateboarder does a trick on a ramp.	A dog is jumping to catch a frisbee.
A group of young people playing a game of frisbee.	Two hockey players are fighting over the puck.	A little girl in a pink hat is blowing bubbles.	A refrigerator filled with lots of food and drinks.
A herd of elephants walking across a dry grass field.	A close up of a cat laying on a couch.	A red motorcycle parked on the side of the road.	A yellow school bus parked in a parking lot.

Image Captioning

[Vinyals et al., 2015]

Fei-Fei Li & Andrej Karpathy & Justin Johnson Lecture 6 - 81 25 Jan 2016

CNN today

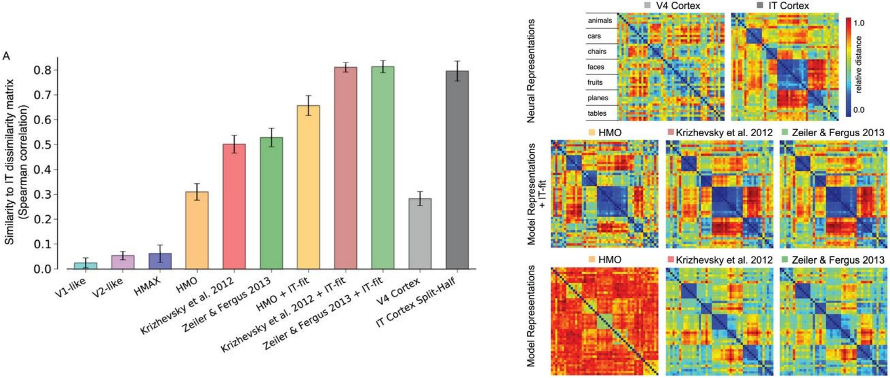
reddit.com/r/deepdream

Fei-Fei Li & Andrej Karpathy & Justin Johnson Lecture 6 - 82 25 Jan 2016

CNN today

Fei-Fei Li & Andrej Karpathy & Justin Johnson Lecture 6 - 83 25 Jan 2016

CNN today

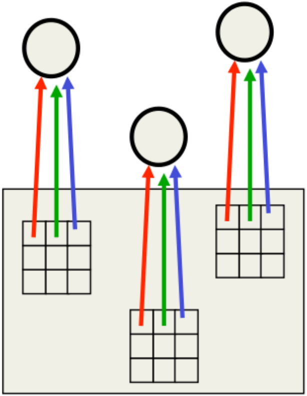


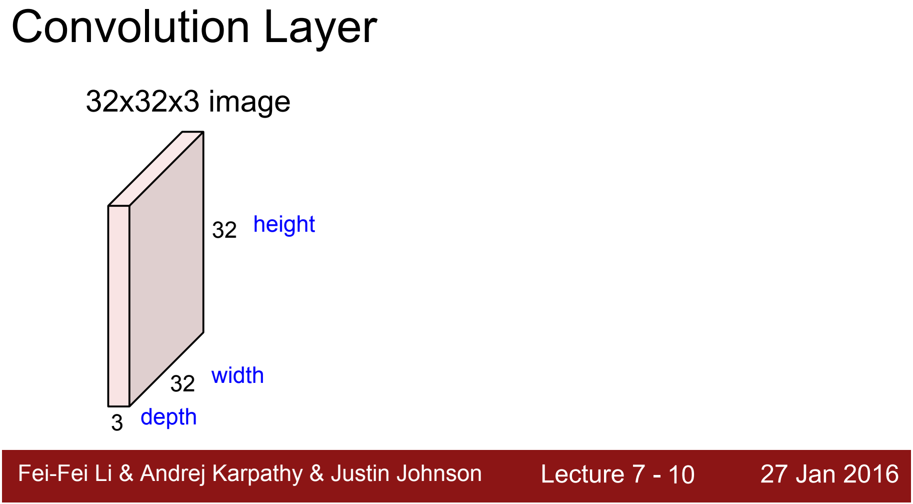
Deep Neural Networks Rival the Representation of Primate IT Cortex for Core Visual Object Recognition [Cadieu et al., 2014]

Fei-Fei Li & Andrej Karpathy & Justin Johnson Lecture 6 - 85 25 Jan 2016

Motivation of CNN

- A same object under different viewpoints is very different in pixel domain
    - A slightly horizontally shifted image has change imperceivable to us but can confuse naive recognition system
- Ideally, we may want to have shift-invariant features
- In practice, if we have local feature suitable for a particular region, the same feature should work well with other region
    - Weight sharing across space → CNN

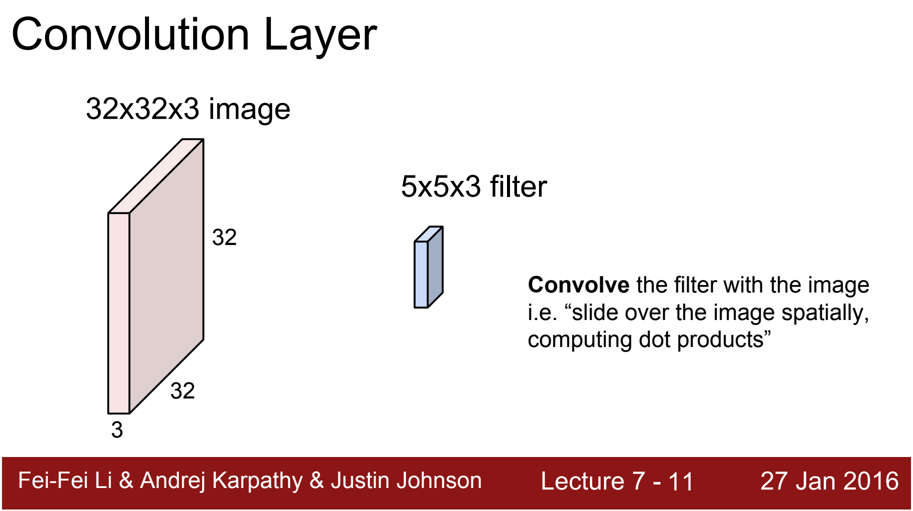




Convolution Layer

32x32x3 image

Fei-Fei Li & Andrej Karpathy & Justin Johnson Lecture 7 - 10 27 Jan 2016



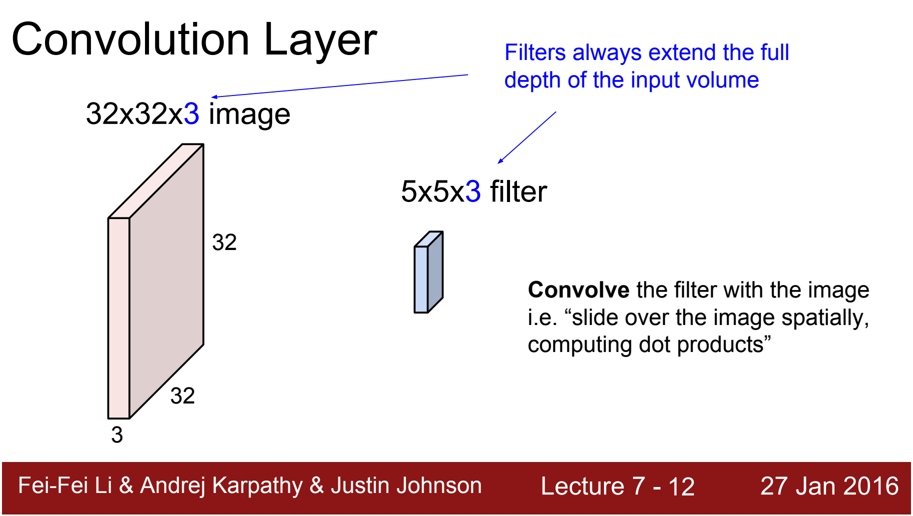
Convolution Layer

32x32x3 image

5x5x3 filter

Convolve the filter with the image i.e. "slide over the image spatially, computing dot products"

Fei-Fei Li & Andrej Karpathy & Justin Johnson Lecture 7 - 11 27 Jan 2016



Convolution Layer

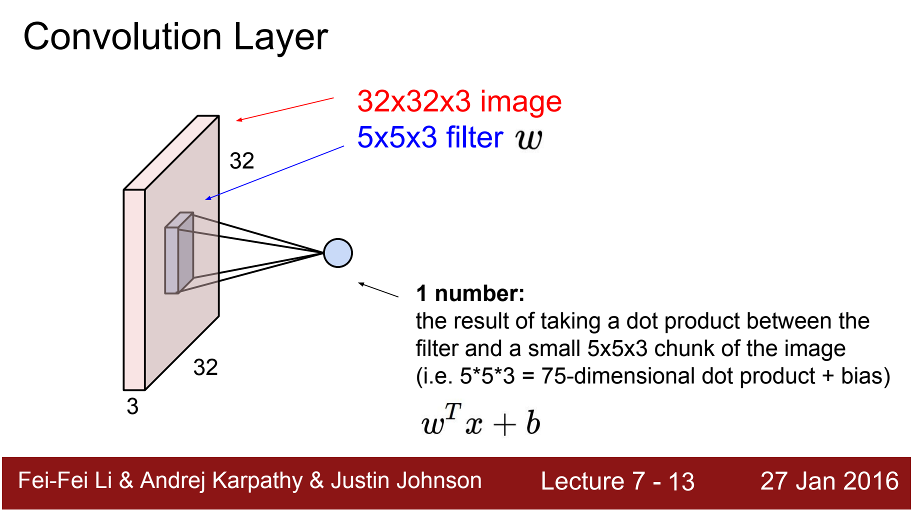
Filters always extend the full depth of the input volume

32x32x3 image

5x5x3 filter

Convolve the filter with the image i.e. “slide over the image spatially, computing dot products”

Fei-Fei Li & Andrej Karpathy & Justin Johnson Lecture 7 - 12 27 Jan 2016



Convolution Layer

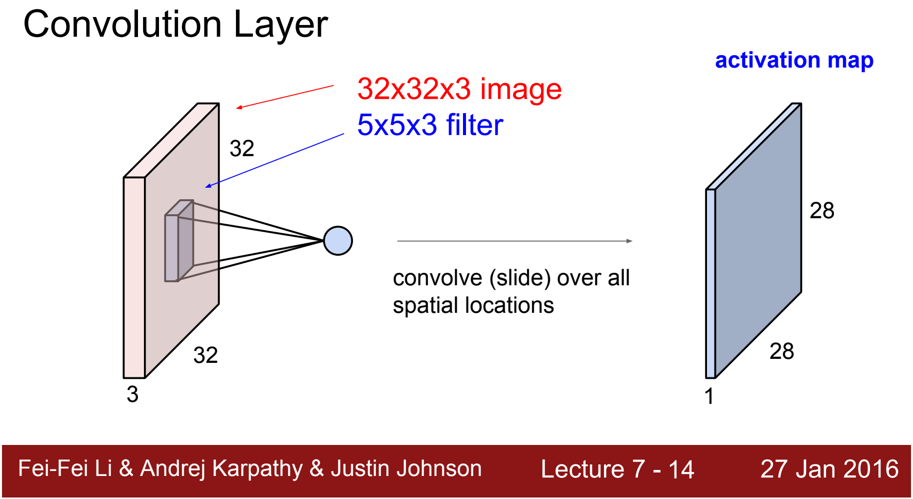
32x32x3 image

5x5x3 filter w

1 number:
the result of taking a dot product between the filter and a small 5x5x3 chunk of the image
(i.e. 5*5*3 = 75-dimensional dot product + bias)

$$w^T x + b$$

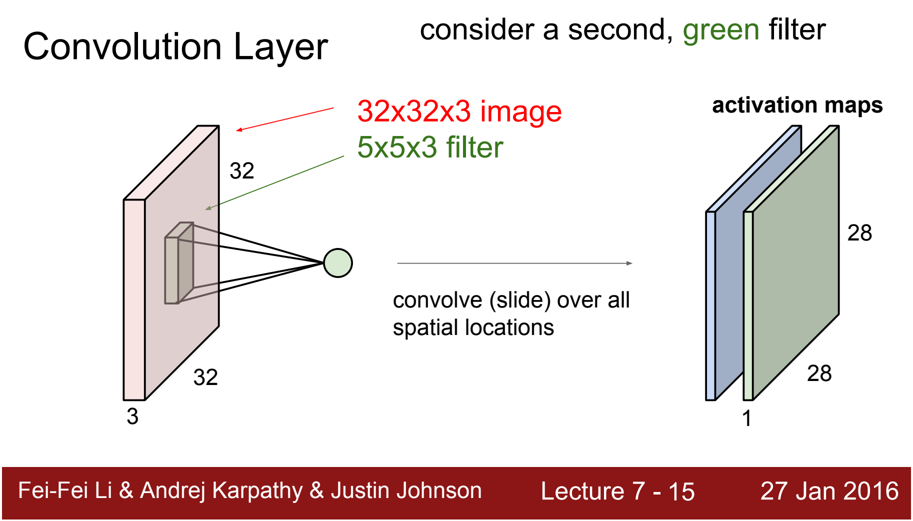
Fei-Fei Li & Andrej Karpathy & Justin Johnson   Lecture 7 - 13   27 Jan 2016



Convolution Layer

convolve (slide) over all spatial locations

Fei-Fei Li & Andrej Karpathy & Justin Johnson Lecture 7 - 14 27 Jan 2016

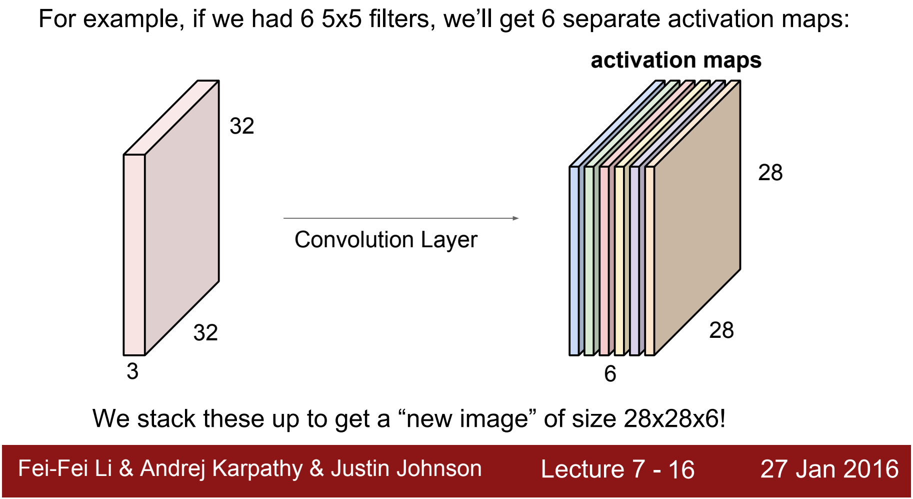


Convolution Layer

consider a second, green filter

convolve (slide) over all spatial locations

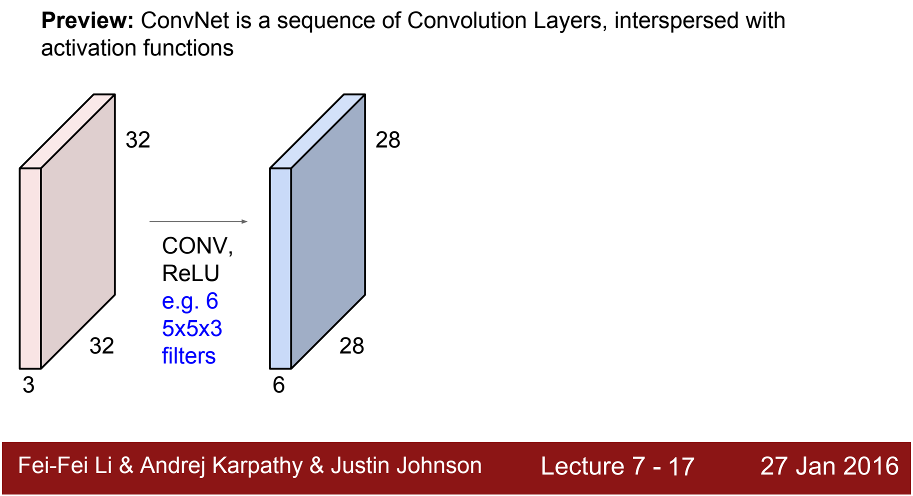
Fei-Fei Li & Andrej Karpathy & Justin Johnson Lecture 7 - 15 27 Jan 2016



For example, if we had 6 5x5 filters, we'll get 6 separate activation maps:

We stack these up to get a "new image" of size 28x28x6!

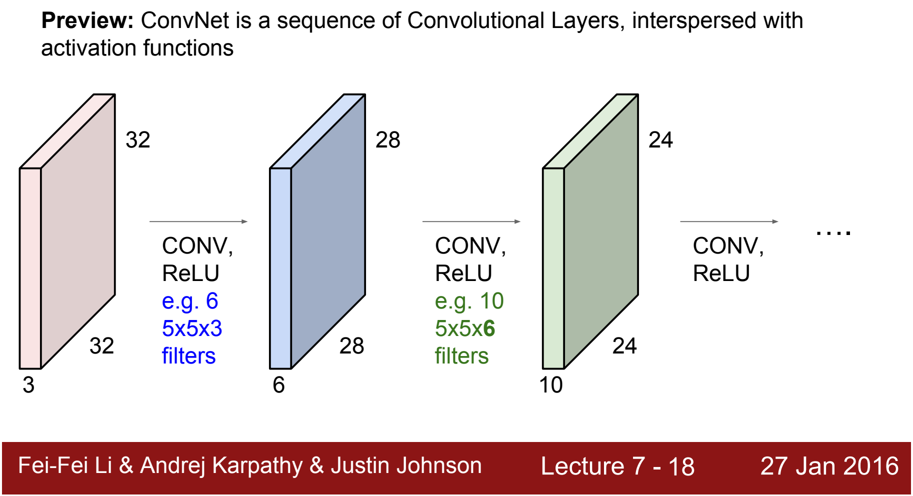
Fei-Fei Li & Andrej Karpathy & Justin Johnson Lecture 7 - 16 27 Jan 2016



Preview: ConvNet is a sequence of Convolution Layers, interspersed with activation functions

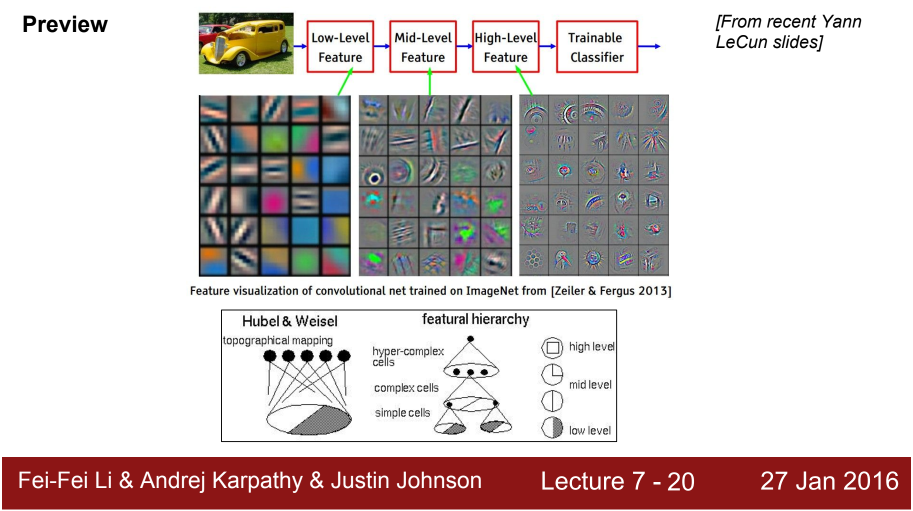
CONV,
ReLU
e.g. 6
5x5x3
filters

Fei-Fei Li & Andrej Karpathy & Justin Johnson Lecture 7 - 17 27 Jan 2016



Preview: ConvNet is a sequence of Convolutional Layers, interspersed with activation functions

Fei-Fei Li & Andrej Karpathy & Justin Johnson Lecture 7 - 18 27 Jan 2016

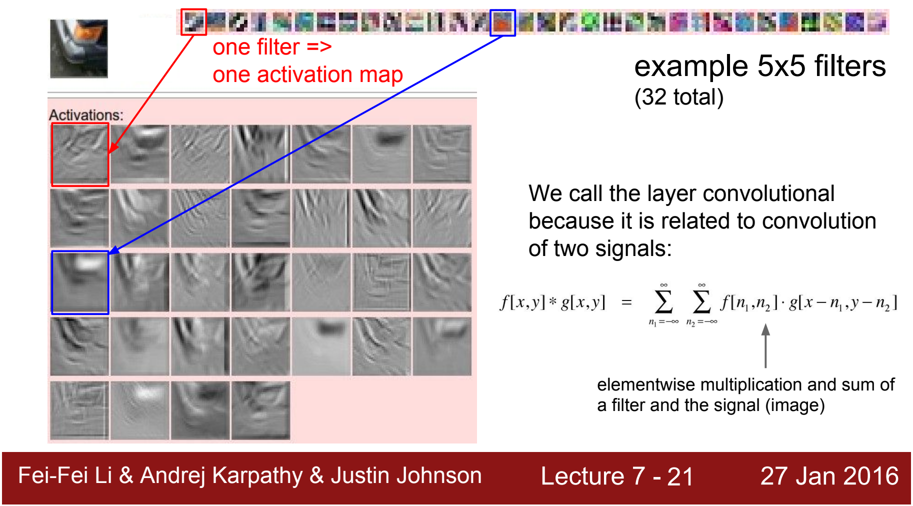


Preview

[From recent Yann LeCun slides]

Feature visualization of convolutional net trained on ImageNet from [Zeiler & Fergus 2013]

Fei-Fei Li & Andrej Karpathy & Justin Johnson Lecture 7 - 20 27 Jan 2016



one filter =>
one activation map

Activations:

example 5x5 filters
(32 total)

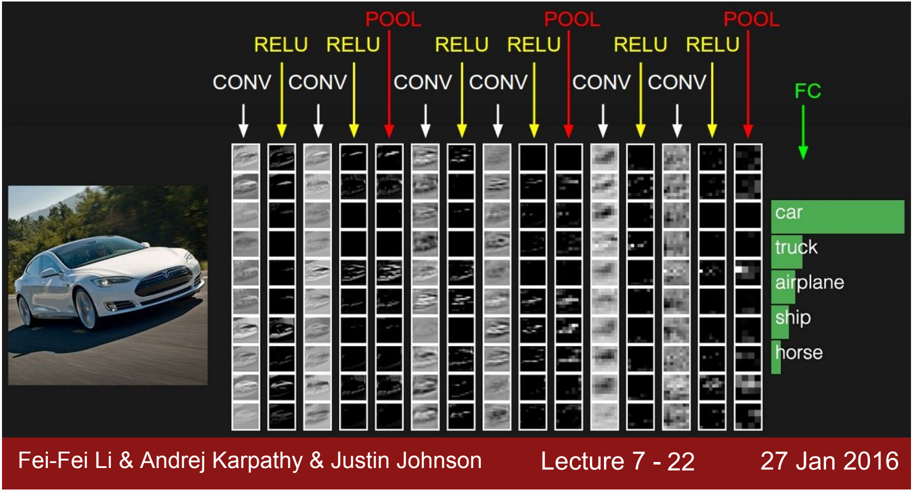
We call the layer convolutional because it is related to convolution of two signals:

$$f[x,y]*g[x,y] = \sum_{n_1=-\infty}^{\infty}\sum_{n_2=-\infty}^{\infty} f[n_1,n_2]\cdot g[x-n_1,y-n_2]$$

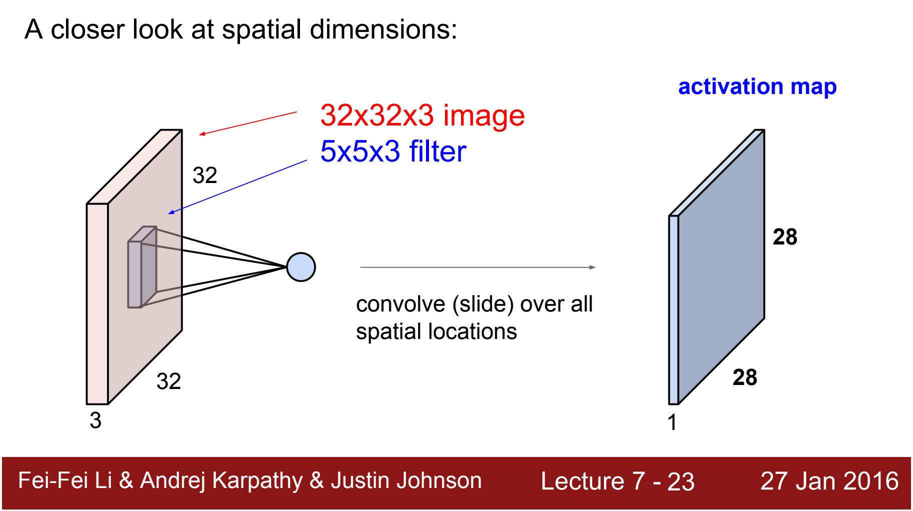
elementwise multiplication and sum of a filter and the signal (image)

Fei-Fei Li & Andrej Karpathy & Justin Johnson Lecture 7 - 21 27 Jan 2016

preview:



Fei-Fei Li & Andrej Karpathy & Justin Johnson Lecture 7 - 22 27 Jan 2016



A closer look at spatial dimensions:

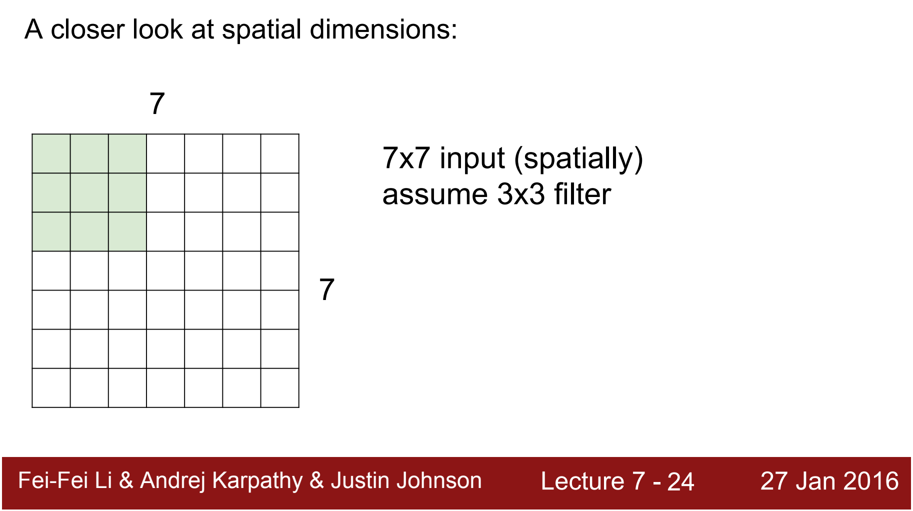
32x32x3 image

5x5x3 filter

activation map

convolve (slide) over all spatial locations

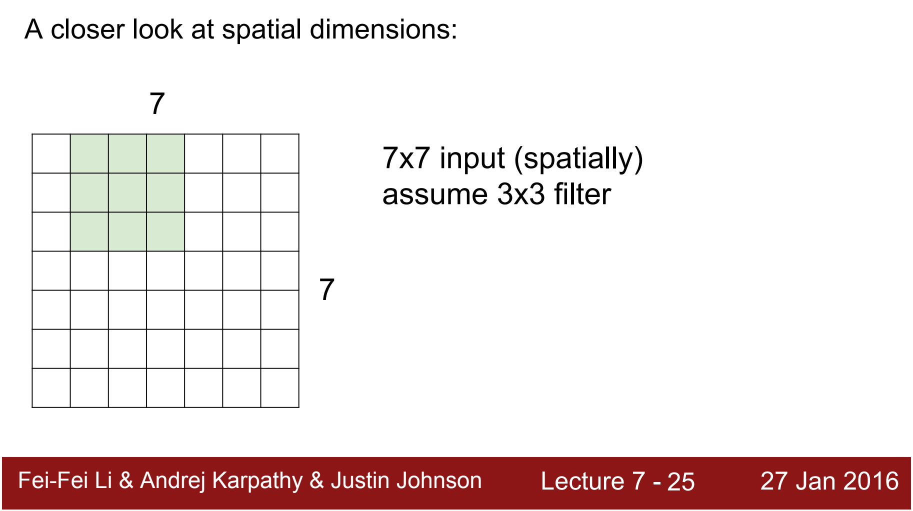
Fei-Fei Li & Andrej Karpathy & Justin Johnson Lecture 7 - 23 27 Jan 2016



A closer look at spatial dimensions:

7x7 input (spatially)
assume 3x3 filter

Fei-Fei Li & Andrej Karpathy & Justin Johnson Lecture 7 - 24 27 Jan 2016

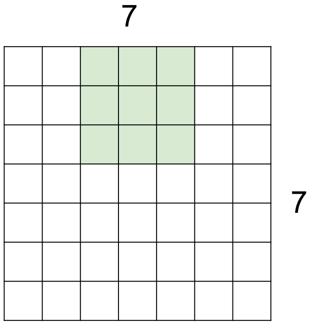


A closer look at spatial dimensions:

7x7 input (spatially)
assume 3x3 filter

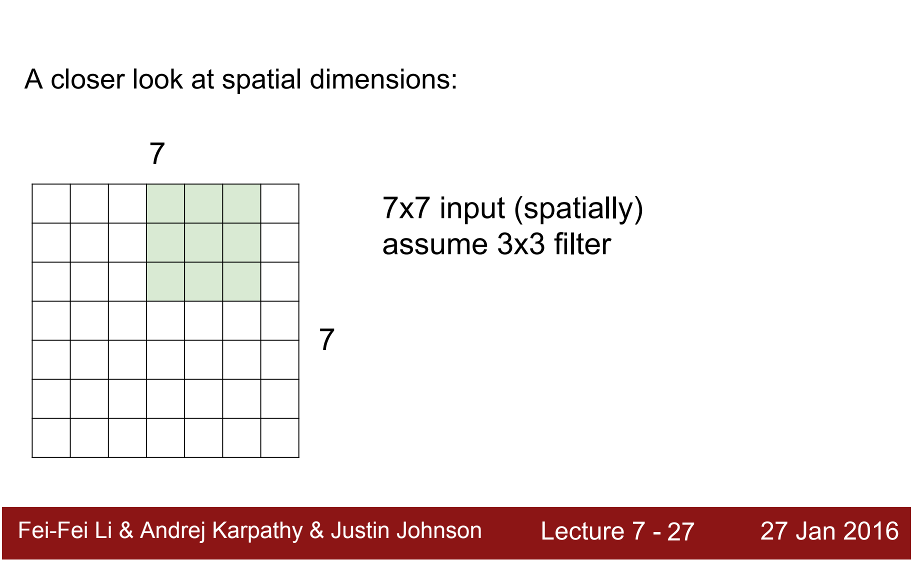
Fei-Fei Li & Andrej Karpathy & Justin Johnson Lecture 7 - 25 27 Jan 2016

A closer look at spatial dimensions:



7x7 input (spatially)
assume 3x3 filter

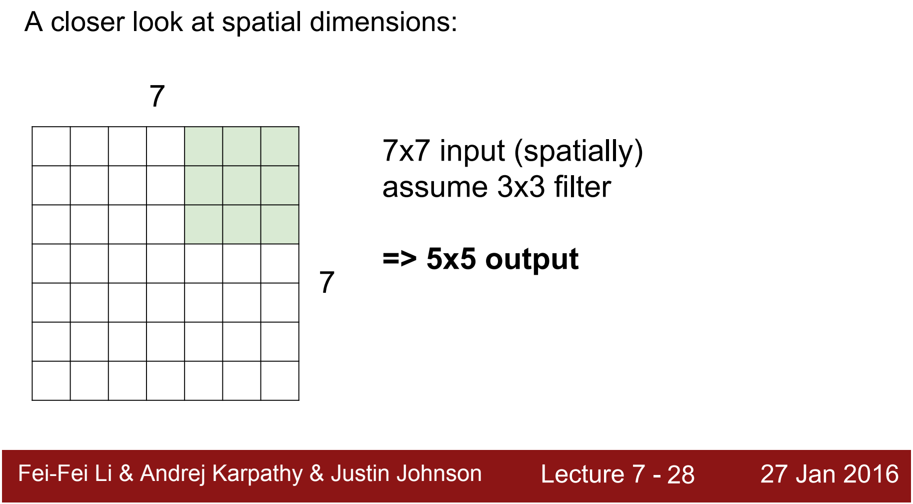
Fei-Fei Li & Andrej Karpathy & Justin Johnson Lecture 7 - 26 27 Jan 2016



A closer look at spatial dimensions:

7x7 input (spatially)
assume 3x3 filter

Fei-Fei Li & Andrej Karpathy & Justin Johnson Lecture 7 - 27 27 Jan 2016

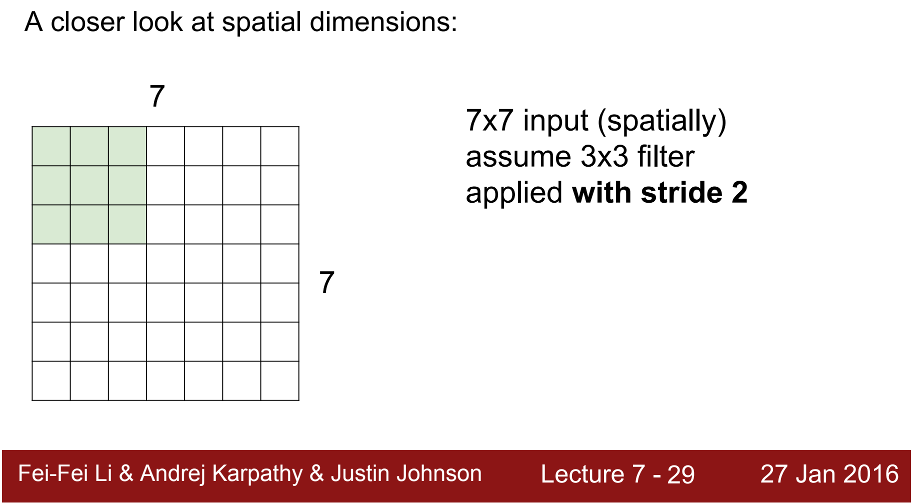


A closer look at spatial dimensions:

7x7 input (spatially)
assume 3x3 filter

=> 5x5 output

Fei-Fei Li & Andrej Karpathy & Justin Johnson Lecture 7 - 28 27 Jan 2016

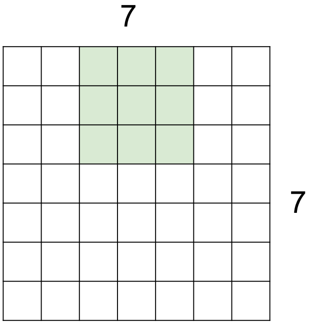


A closer look at spatial dimensions:

7x7 input (spatially)
assume 3x3 filter
applied **with stride 2**

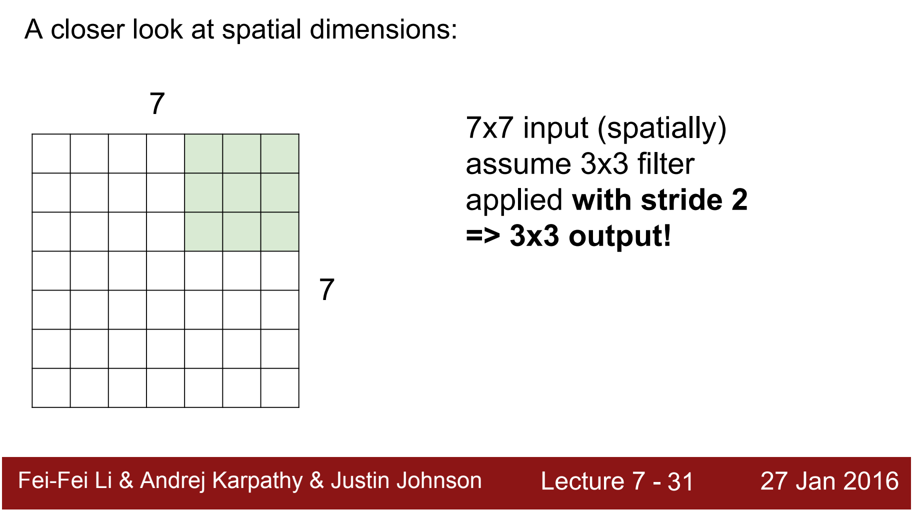
Fei-Fei Li & Andrej Karpathy & Justin Johnson Lecture 7 - 29 27 Jan 2016

A closer look at spatial dimensions:



7x7 input (spatially)
assume 3x3 filter
applied **with stride 2**

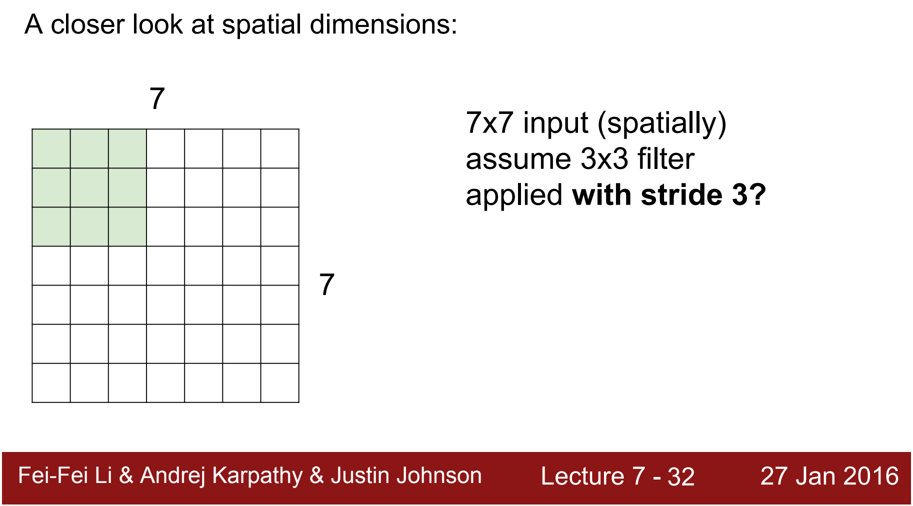
Fei-Fei Li & Andrej Karpathy & Justin Johnson Lecture 7 - 30 27 Jan 2016



A closer look at spatial dimensions:

7x7 input (spatially)
assume 3x3 filter
applied **with stride 2**
=> 3x3 output!

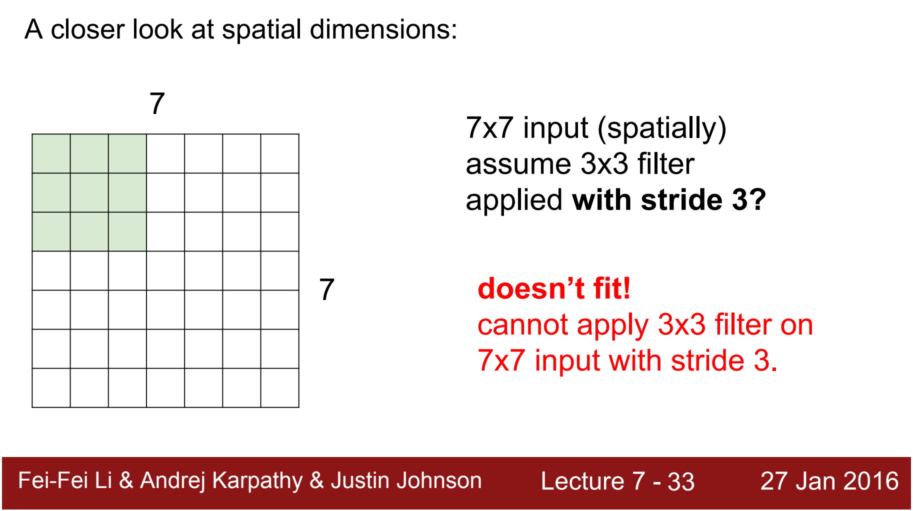
Fei-Fei Li & Andrej Karpathy & Justin Johnson Lecture 7 - 31 27 Jan 2016



A closer look at spatial dimensions:

7x7 input (spatially)
assume 3x3 filter
applied **with stride 3?**

Fei-Fei Li & Andrej Karpathy & Justin Johnson Lecture 7 - 32 27 Jan 2016

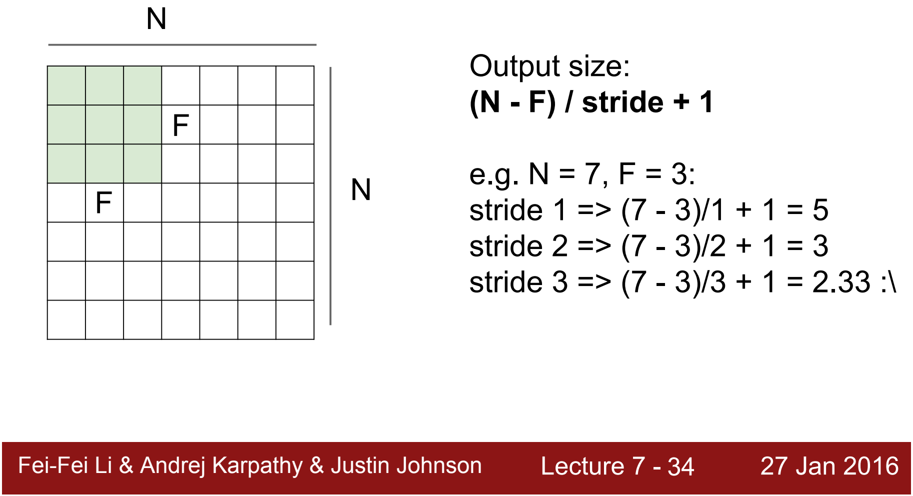


A closer look at spatial dimensions:

7x7 input (spatially)
assume 3x3 filter
applied **with stride 3?**

doesn’t fit!
cannot apply 3x3 filter on
7x7 input with stride 3.

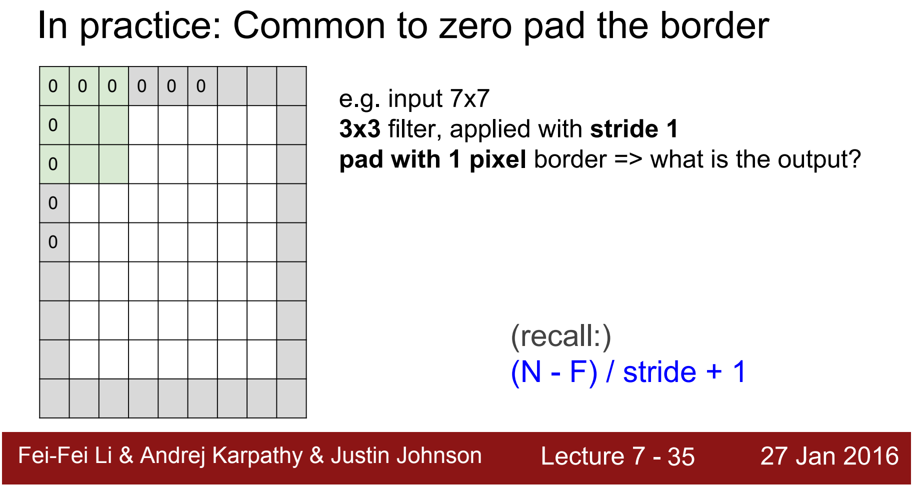
Fei-Fei Li & Andrej Karpathy & Justin Johnson Lecture 7 - 33 27 Jan 2016



Output size:
(N - F) / stride + 1

e.g. N = 7, F = 3:
stride 1 => (7 - 3)/1 + 1 = 5
stride 2 => (7 - 3)/2 + 1 = 3
stride 3 => (7 - 3)/3 + 1 = 2.33 :\

Fei-Fei Li & Andrej Karpathy & Justin Johnson   Lecture 7 - 34   27 Jan 2016



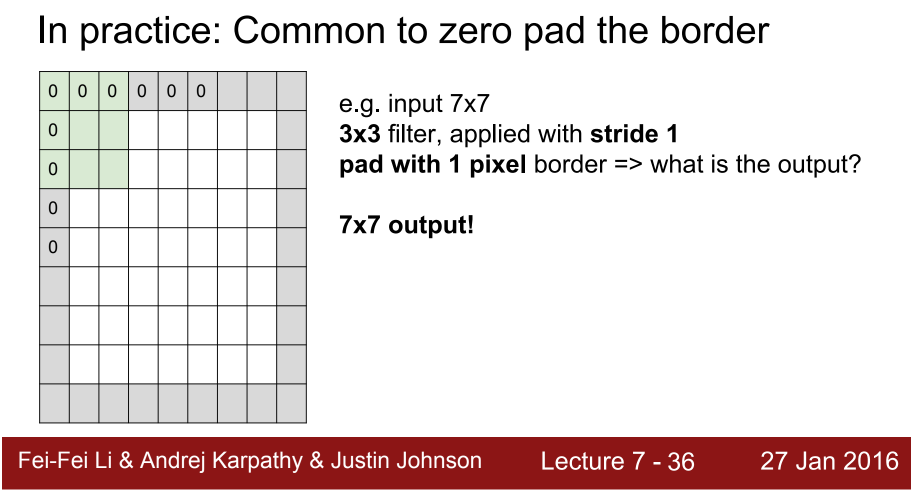
In practice: Common to zero pad the border

0	0	0	0	0	0			
0								
0								
0								
0								

e.g. input 7x7
3x3 filter, applied with **stride 1**
pad with 1 pixel border => what is the output?

(recall:)
(N - F) / stride + 1

Fei-Fei Li & Andrej Karpathy & Justin Johnson Lecture 7 - 35 27 Jan 2016



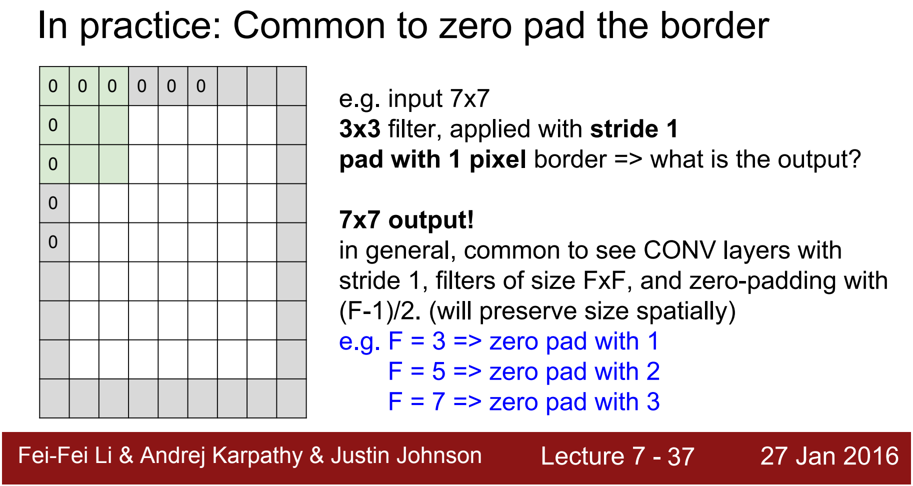
In practice: Common to zero pad the border

0	0	0	0	0	0			
0								
0								
0								
0								

e.g. input 7x7
3x3 filter, applied with **stride 1**
pad with 1 pixel border => what is the output?

7x7 output!

Fei-Fei Li & Andrej Karpathy & Justin Johnson Lecture 7 - 36 27 Jan 2016



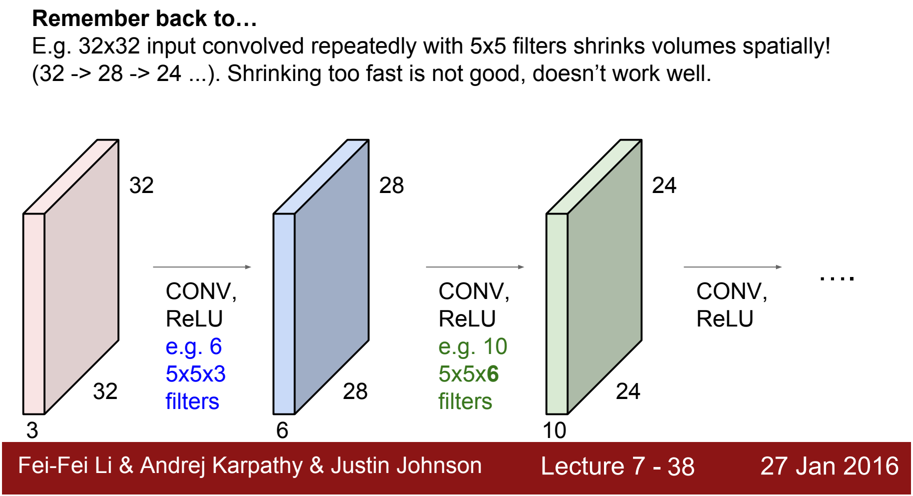
In practice: Common to zero pad the border

0	0	0	0	0	0			
0								
0								
0								
0								

e.g. input 7x7
3x3 filter, applied with **stride 1**
pad with 1 pixel border => what is the output?

7x7 output!
in general, common to see CONV layers with stride 1, filters of size FxF, and zero-padding with (F-1)/2. (will preserve size spatially)
e.g. F = 3 => zero pad with 1
F = 5 => zero pad with 2
F = 7 => zero pad with 3

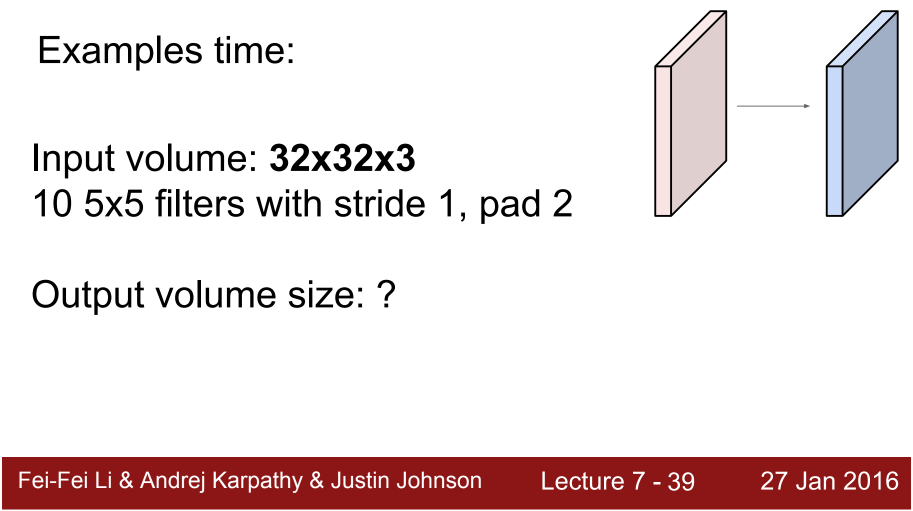
Fei-Fei Li & Andrej Karpathy & Justin Johnson Lecture 7 - 37 27 Jan 2016



Remember back to…

E.g. 32x32 input convolved repeatedly with 5x5 filters shrinks volumes spatially! (32 -> 28 -> 24 ...). Shrinking too fast is not good, doesn't work well.

Fei-Fei Li & Andrej Karpathy & Justin Johnson Lecture 7 - 38 27 Jan 2016

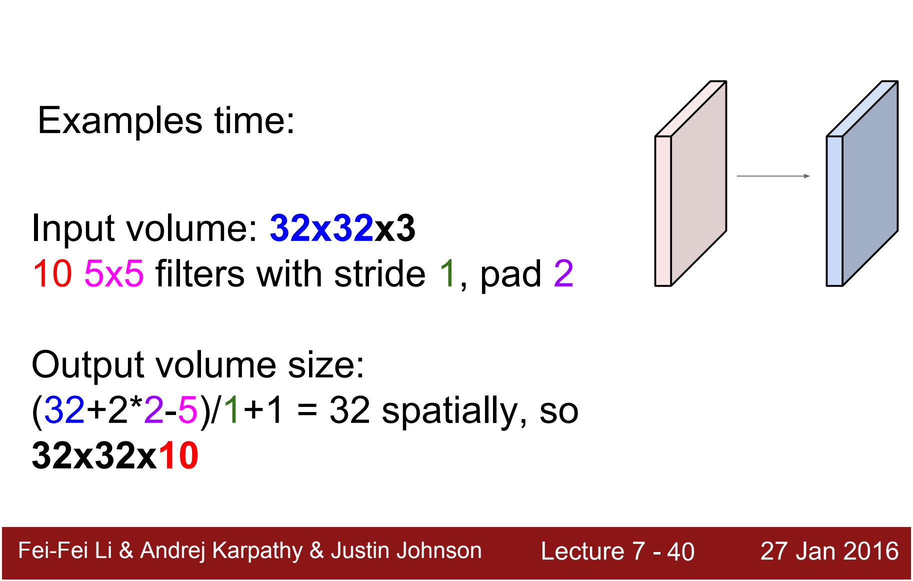


Examples time:

Input volume: **32x32x3**
10 5x5 filters with stride 1, pad 2

Output volume size: ?

Fei-Fei Li & Andrej Karpathy & Justin Johnson Lecture 7 - 39 27 Jan 2016



Examples time:

Input volume: **32x32x3**

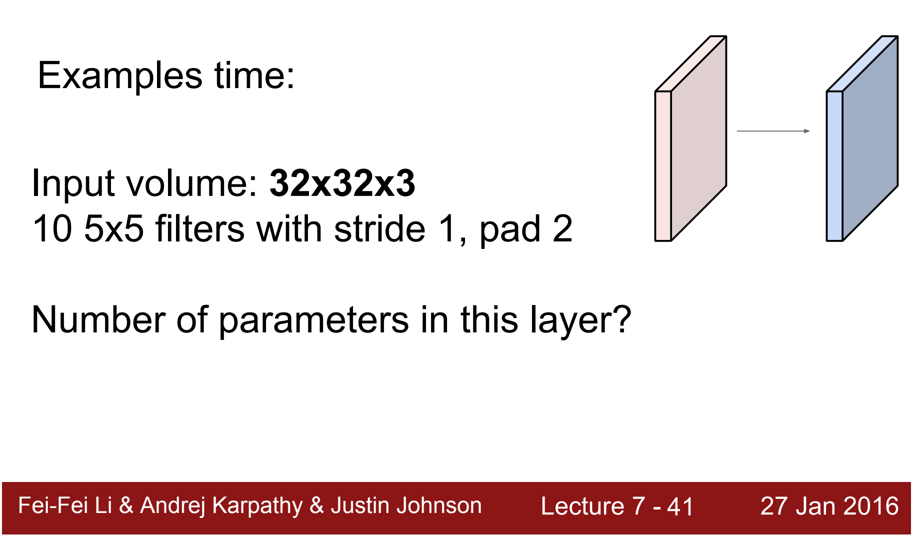
10 5x5 filters with stride 1, pad 2

Output volume size:

(32+2*2-5)/1+1 = 32 spatially, so

32x32x10

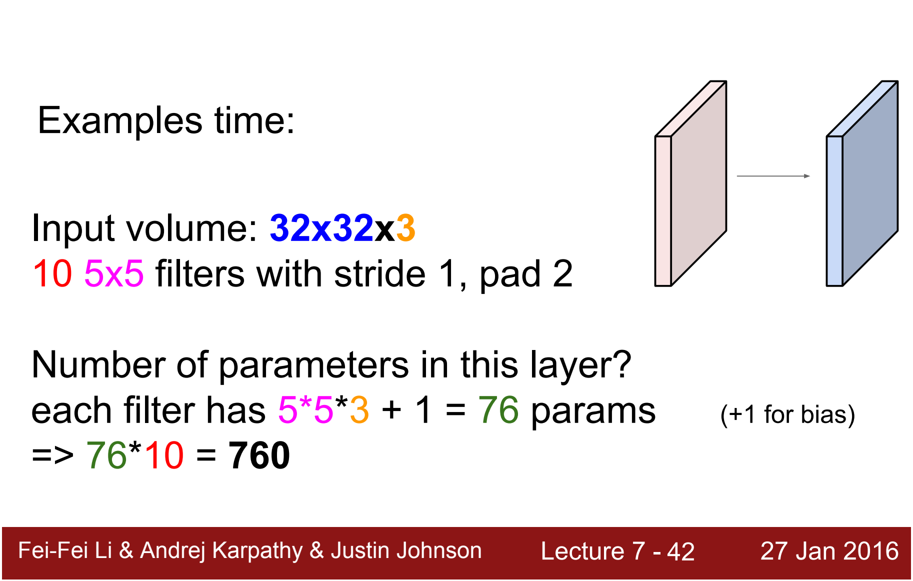
Fei-Fei Li & Andrej Karpathy & Justin Johnson Lecture 7 - 40 27 Jan 2016



Examples time:

Input volume: **32x32x3**
10 5x5 filters with stride 1, pad 2

Number of parameters in this layer?



Examples time:

Input volume: **32x32x3**

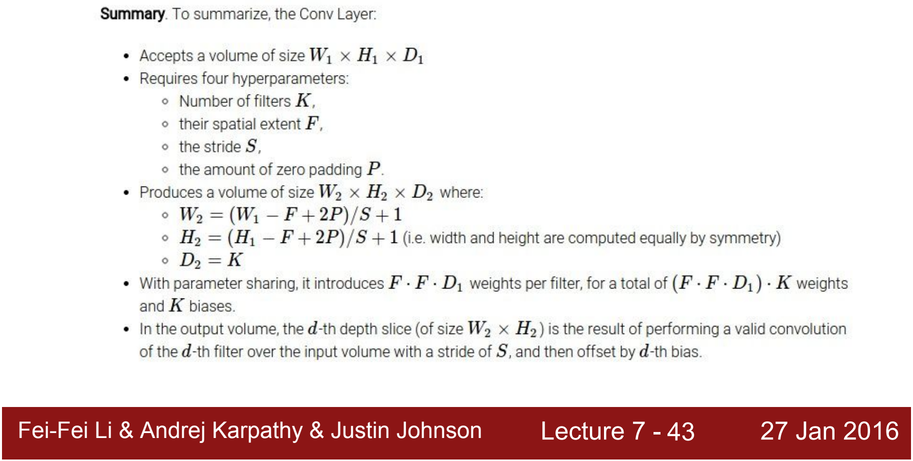
10 5x5 filters with stride 1, pad 2

Number of parameters in this layer?

each filter has 5*5*3 + 1 = 76 params (+1 for bias)

=> 76*10 = **760**

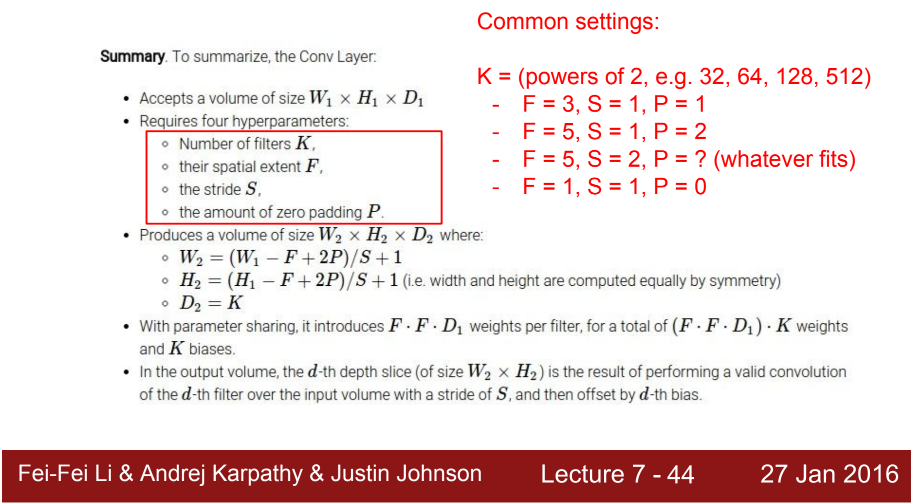
Fei-Fei Li & Andrej Karpathy & Justin Johnson Lecture 7 - 42 27 Jan 2016



Summary. To summarize, the Conv Layer:

- Accepts a volume of size $W_1 \times H_1 \times D_1$
- Requires four hyperparameters:
  - Number of filters K,
  - their spatial extent F,
  - the stride S,
  - the amount of zero padding P.
- Produces a volume of size $W_2 \times H_2 \times D_2$ where:
  - $W_2 = (W_1 - F + 2P)/S + 1$
  - $H_2 = (H_1 - F + 2P)/S + 1$ (i.e. width and height are computed equally by symmetry)
  - $D_2 = K$
- With parameter sharing, it introduces $F \cdot F \cdot D_1$ weights per filter, for a total of $(F \cdot F \cdot D_1) \cdot K$ weights and K biases.
- In the output volume, the d-th depth slice (of size $W_2 \times H_2$) is the result of performing a valid convolution of the d-th filter over the input volume with a stride of S, and then offset by d-th bias.

Fei-Fei Li & Andrej Karpathy & Justin Johnson Lecture 7 - 43 27 Jan 2016



Summary. To summarize, the Conv Layer:

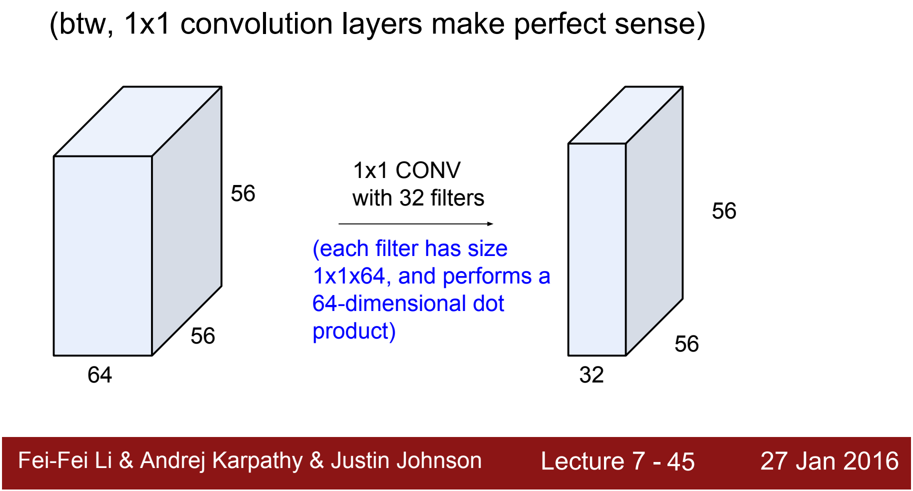
- Accepts a volume of size $W_1 \times H_1 \times D_1$
- Requires four hyperparameters:
  - Number of filters K,
  - their spatial extent F,
  - the stride S,
  - the amount of zero padding P.
- Produces a volume of size $W_2 \times H_2 \times D_2$ where:
  - $W_2 = (W_1 - F + 2P)/S + 1$
  - $H_2 = (H_1 - F + 2P)/S + 1$ (i.e. width and height are computed equally by symmetry)
  - $D_2 = K$
- With parameter sharing, it introduces $F \cdot F \cdot D_1$ weights per filter, for a total of $(F \cdot F \cdot D_1) \cdot K$ weights and K biases.
- In the output volume, the d-th depth slice (of size $W_2 \times H_2$) is the result of performing a valid convolution of the d-th filter over the input volume with a stride of S, and then offset by d-th bias.

Common settings:

K = (powers of 2, e.g. 32, 64, 128, 512)

- F = 3, S = 1, P = 1
- F = 5, S = 1, P = 2
- F = 5, S = 2, P = ? (whatever fits)
- F = 1, S = 1, P = 0

Fei-Fei Li & Andrej Karpathy & Justin Johnson Lecture 7 - 44 27 Jan 2016

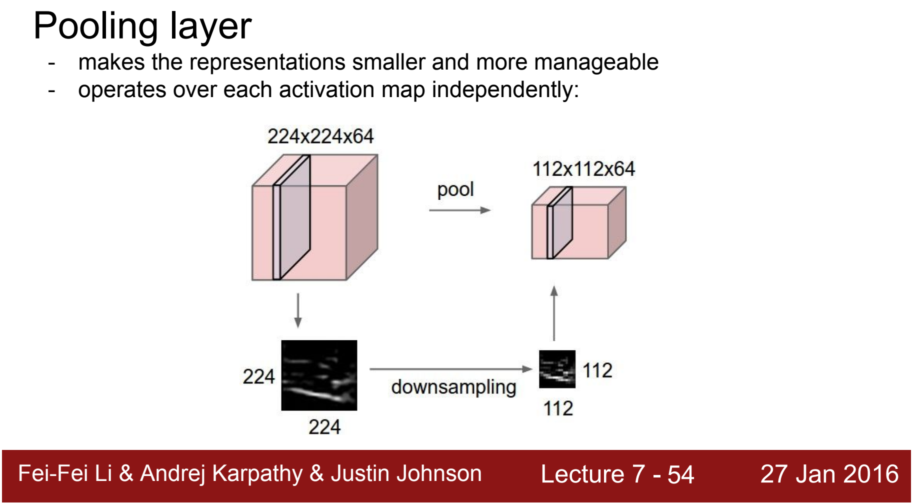


(btw, 1x1 convolution layers make perfect sense)

1x1 CONV
with 32 filters

(each filter has size 1x1x64, and performs a 64-dimensional dot product)

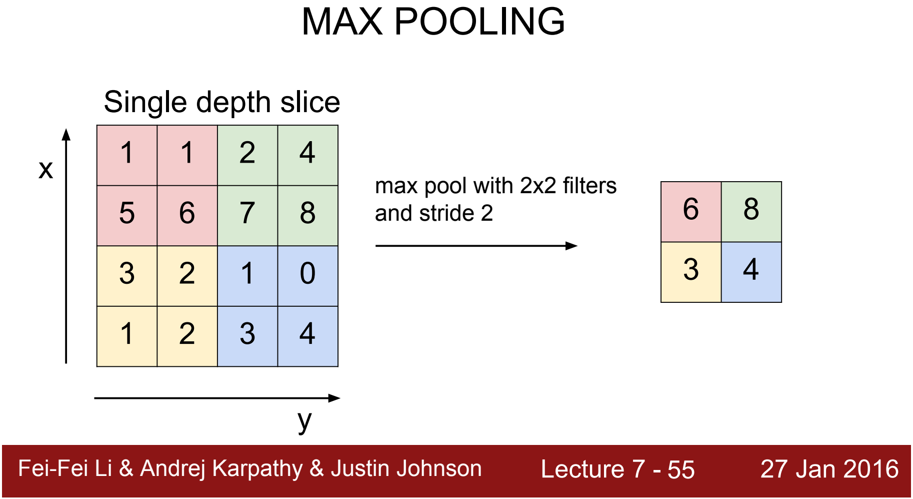
Fei-Fei Li & Andrej Karpathy & Justin Johnson Lecture 7 - 45 27 Jan 2016



Pooling layer

- makes the representations smaller and more manageable
- operates over each activation map independently:

Fei-Fei Li & Andrej Karpathy & Justin Johnson Lecture 7 - 54 27 Jan 2016



MAX POOLING

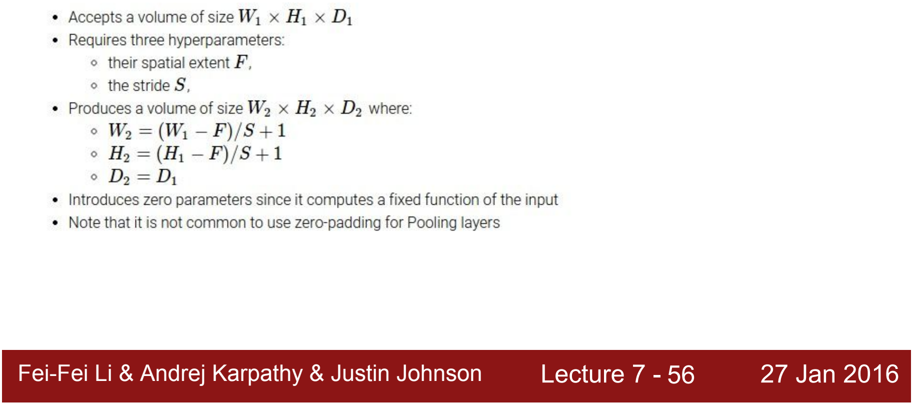
Single depth slice

1	1	2	4
5	6	7	8
3	2	1	0
1	2	3	4

max pool with 2x2 filters and stride 2

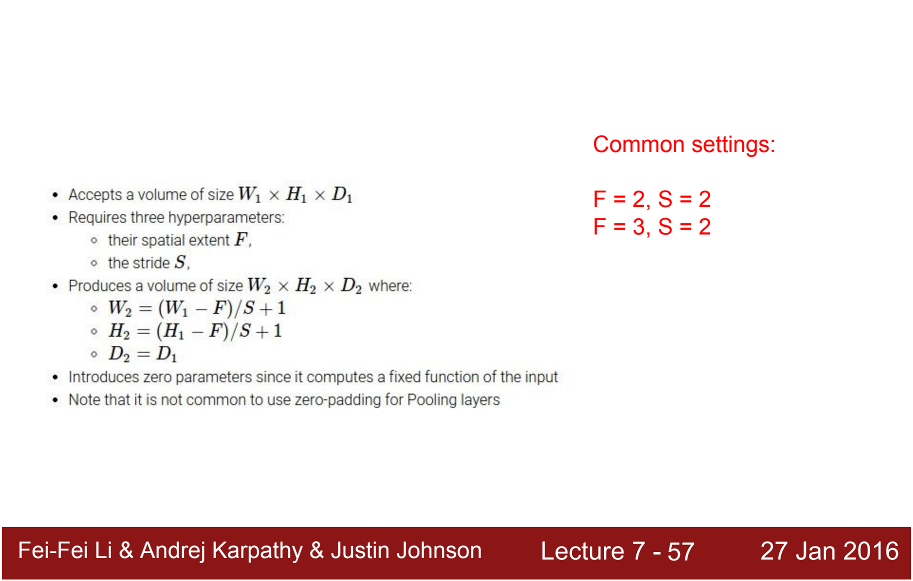
6	8
3	4

Fei-Fei Li & Andrej Karpathy & Justin Johnson   Lecture 7 - 55   27 Jan 2016



- Accepts a volume of size $W_1 \times H_1 \times D_1$
- Requires three hyperparameters:
  - their spatial extent F,
  - the stride S,
- Produces a volume of size $W_2 \times H_2 \times D_2$ where:
  - $W_2 = (W_1 - F)/S + 1$
  - $H_2 = (H_1 - F)/S + 1$
  - $D_2 = D_1$
- Introduces zero parameters since it computes a fixed function of the input
- Note that it is not common to use zero-padding for Pooling layers

Fei-Fei Li & Andrej Karpathy & Justin Johnson Lecture 7 - 56 27 Jan 2016

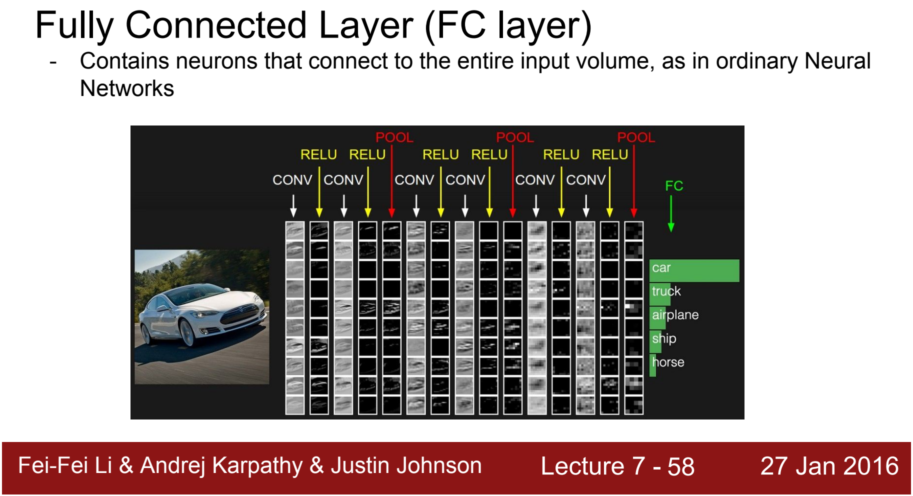


- Accepts a volume of size $W_1 \times H_1 \times D_1$
- Requires three hyperparameters:
  - their spatial extent F,
  - the stride S,
- Produces a volume of size $W_2 \times H_2 \times D_2$ where:
  - $W_2 = (W_1 - F)/S + 1$
  - $H_2 = (H_1 - F)/S + 1$
  - $D_2 = D_1$
- Introduces zero parameters since it computes a fixed function of the input
- Note that it is not common to use zero-padding for Pooling layers

Common settings:

F = 2, S = 2
F = 3, S = 2

Fei-Fei Li & Andrej Karpathy & Justin Johnson Lecture 7 - 57 27 Jan 2016



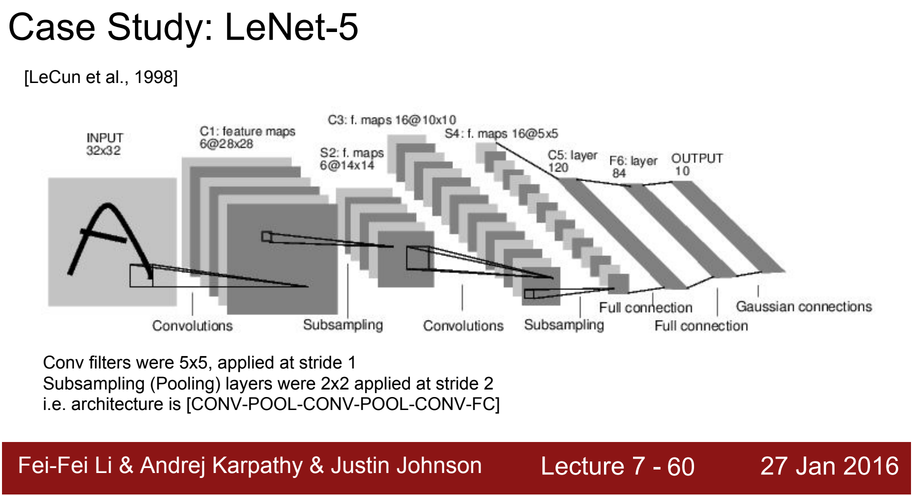
Fully Connected Layer (FC layer)

- Contains neurons that connect to the entire input volume, as in ordinary Neural Networks

Fei-Fei Li & Andrej Karpathy & Justin Johnson Lecture 7 - 58 27 Jan 2016

Demo

ConvNetJS cifar10 demo

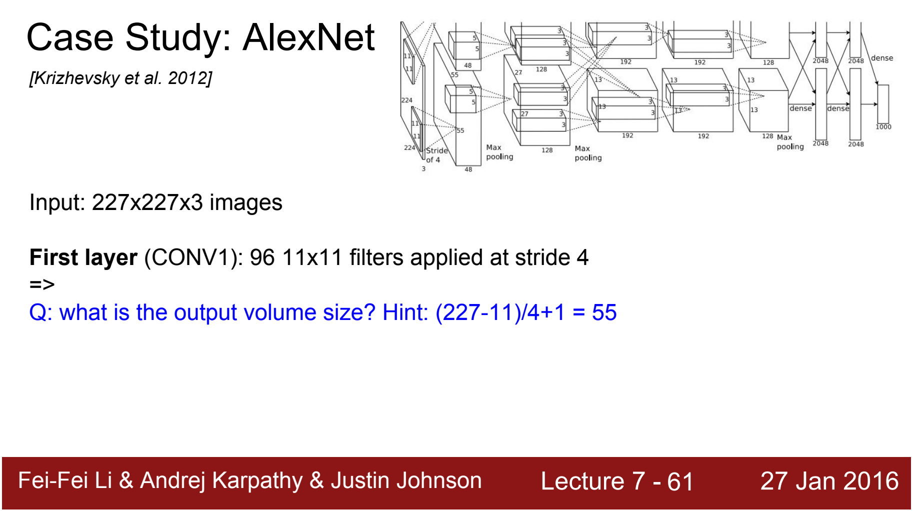


Case Study: LeNet-5

[LeCun et al., 1998]

Conv filters were 5x5, applied at stride 1
Subsampling (Pooling) layers were 2x2 applied at stride 2
i.e. architecture is [CONV-POOL-CONV-POOL-CONV-FC]

Fei-Fei Li & Andrej Karpathy & Justin Johnson Lecture 7 - 60 27 Jan 2016



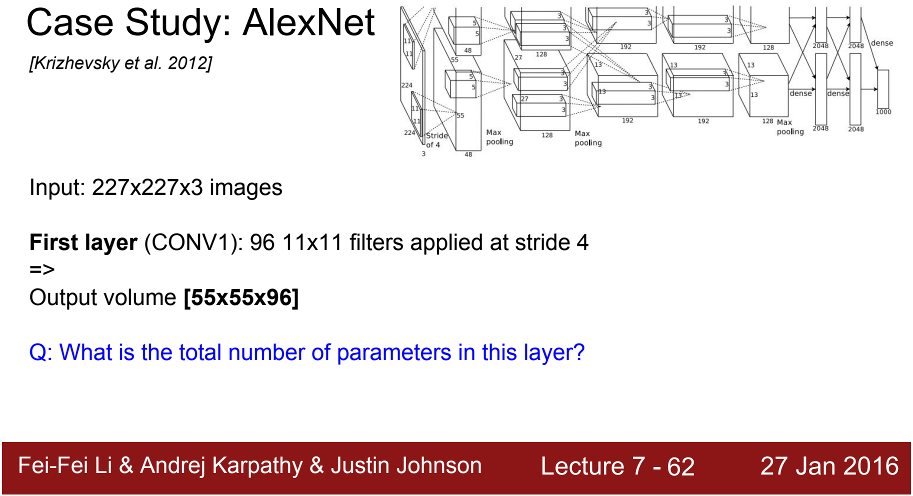
Case Study: AlexNet

[Krizhevsky et al. 2012]

Input: 227x227x3 images

First layer (CONV1): 96 11x11 filters applied at stride 4
=>
Q: what is the output volume size? Hint: (227-11)/4+1 = 55

Fei-Fei Li & Andrej Karpathy & Justin Johnson Lecture 7 - 61 27 Jan 2016



Case Study: AlexNet

[Krizhevsky et al. 2012]

Input: 227x227x3 images

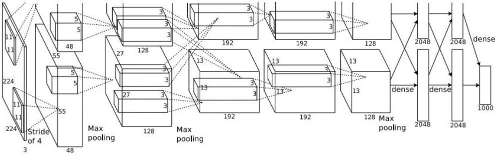
First layer (CONV1): 96 11x11 filters applied at stride 4
=>
Output volume **[55x55x96]**

Q: What is the total number of parameters in this layer?

Fei-Fei Li & Andrej Karpathy & Justin Johnson Lecture 7 - 62 27 Jan 2016

Case Study: AlexNet

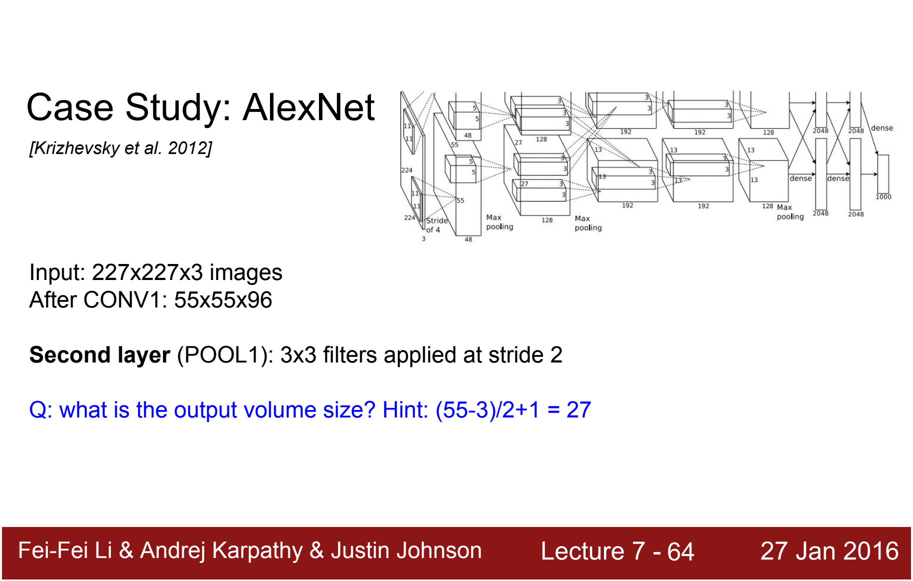
[Krizhevsky et al. 2012]



Input: 227x227x3 images

First layer (CONV1): 96 11x11 filters applied at stride 4
=>
Output volume **[55x55x96]**
Parameters: (11*11*3)*96 = **35K**

Fei-Fei Li & Andrej Karpathy & Justin Johnson Lecture 7 - 63 27 Jan 2016



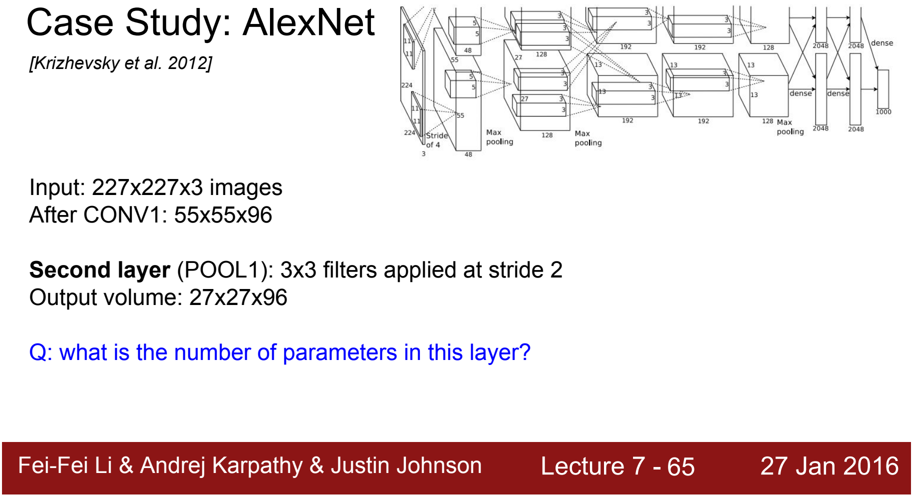
Case Study: AlexNet

[Krizhevsky et al. 2012]

Input: 227x227x3 images
After CONV1: 55x55x96

Second layer (POOL1): 3x3 filters applied at stride 2

Q: what is the output volume size? Hint: (55-3)/2+1 = 27



Case Study: AlexNet

[Krizhevsky et al. 2012]

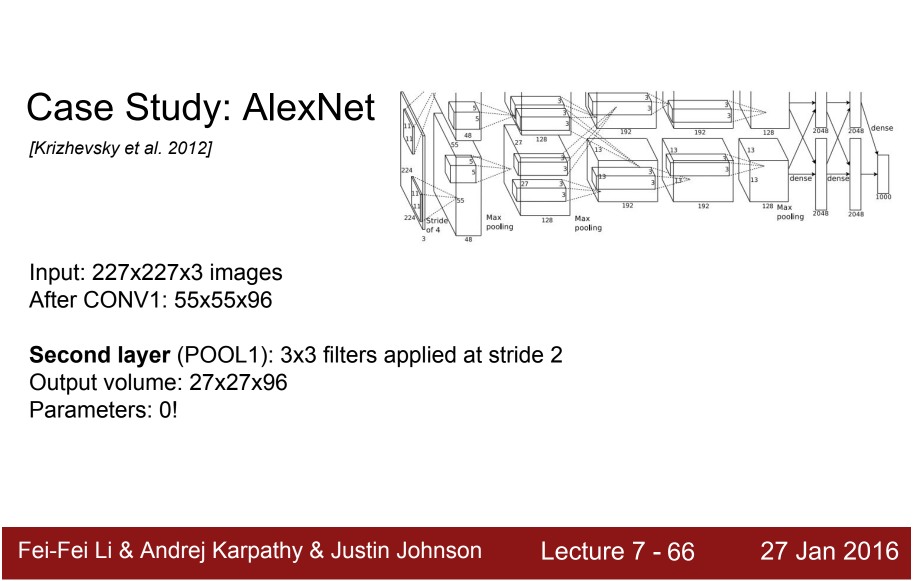
224
224
3
11
11
Stride of 4
55
48
5
5
27
128
Max pooling
3
13
192
128
Max pooling
dense
2048
1000

Input: 227x227x3 images
After CONV1: 55x55x96

Second layer (POOL1): 3x3 filters applied at stride 2
Output volume: 27x27x96

Q: what is the number of parameters in this layer?

Fei-Fei Li & Andrej Karpathy & Justin Johnson Lecture 7 - 65 27 Jan 2016



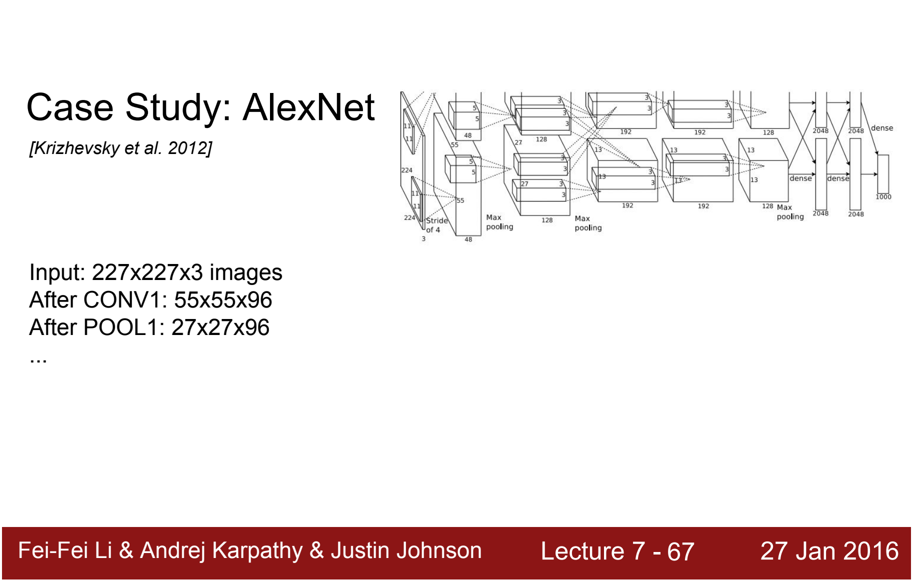
Case Study: AlexNet

[Krizhevsky et al. 2012]

Input: 227x227x3 images
After CONV1: 55x55x96

Second layer (POOL1): 3x3 filters applied at stride 2
Output volume: 27x27x96
Parameters: 0!

Fei-Fei Li & Andrej Karpathy & Justin Johnson Lecture 7 - 66 27 Jan 2016

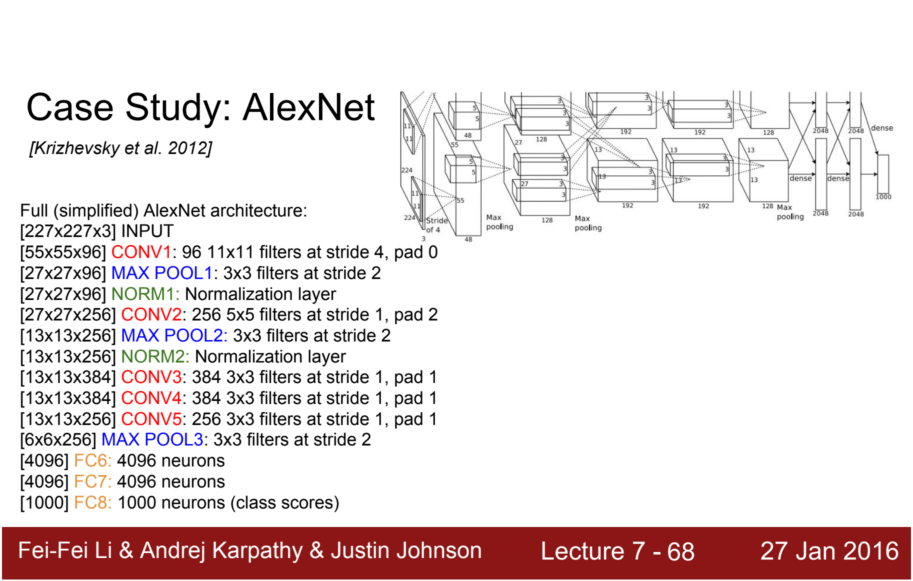


Case Study: AlexNet

[Krizhevsky et al. 2012]

Input: 227x227x3 images
After CONV1: 55x55x96
After POOL1: 27x27x96
...

Fei-Fei Li & Andrej Karpathy & Justin Johnson Lecture 7 - 67 27 Jan 2016

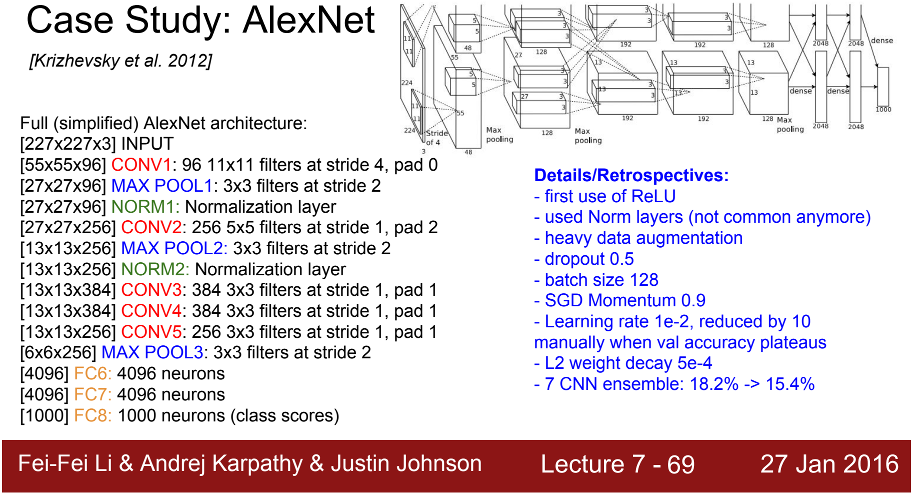

224
224
11
11
11
11
Stride
of 4
3
55
55
5
5
48
48
27
27
128
128
3
3
13
13
192
192
192
192
128
128
Max
pooling
Max
pooling
Max
pooling
dense
dense
dense
2048
2048
2048
2048
1000

Case Study: AlexNet

[Krizhevsky et al. 2012]

Full (simplified) AlexNet architecture:
[227x227x3] INPUT
[55x55x96] CONV1: 96 11x11 filters at stride 4, pad 0
[27x27x96] MAX POOL1: 3x3 filters at stride 2
[27x27x96] NORM1: Normalization layer
[27x27x256] CONV2: 256 5x5 filters at stride 1, pad 2
[13x13x256] MAX POOL2: 3x3 filters at stride 2
[13x13x256] NORM2: Normalization layer
[13x13x384] CONV3: 384 3x3 filters at stride 1, pad 1
[13x13x384] CONV4: 384 3x3 filters at stride 1, pad 1
[13x13x256] CONV5: 256 3x3 filters at stride 1, pad 1
[6x6x256] MAX POOL3: 3x3 filters at stride 2
[4096] FC6: 4096 neurons
[4096] FC7: 4096 neurons
[1000] FC8: 1000 neurons (class scores)

Fei-Fei Li & Andrej Karpathy & Justin Johnson Lecture 7 - 68 27 Jan 2016



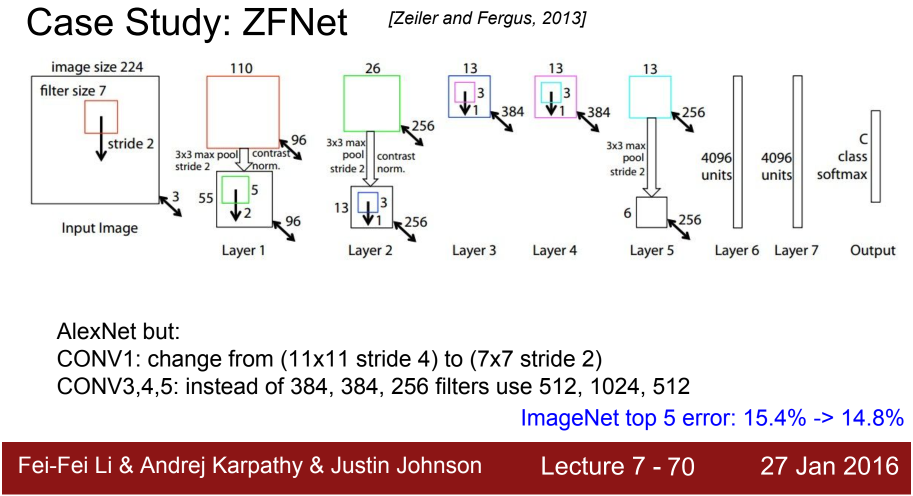
Case Study: AlexNet

[Krizhevsky et al. 2012]

Full (simplified) AlexNet architecture:
[227x227x3] INPUT
[55x55x96] CONV1: 96 11x11 filters at stride 4, pad 0
[27x27x96] MAX POOL1: 3x3 filters at stride 2
[27x27x96] NORM1: Normalization layer
[27x27x256] CONV2: 256 5x5 filters at stride 1, pad 2
[13x13x256] MAX POOL2: 3x3 filters at stride 2
[13x13x256] NORM2: Normalization layer
[13x13x384] CONV3: 384 3x3 filters at stride 1, pad 1
[13x13x384] CONV4: 384 3x3 filters at stride 1, pad 1
[13x13x256] CONV5: 256 3x3 filters at stride 1, pad 1
[6x6x256] MAX POOL3: 3x3 filters at stride 2
[4096] FC6: 4096 neurons
[4096] FC7: 4096 neurons
[1000] FC8: 1000 neurons (class scores)

Details/Retrospectives:
- first use of ReLU
- used Norm layers (not common anymore)
- heavy data augmentation
- dropout 0.5
- batch size 128
- SGD Momentum 0.9
- Learning rate 1e-2, reduced by 10 manually when val accuracy plateaus
- L2 weight decay 5e-4
- 7 CNN ensemble: 18.2% -> 15.4%

Fei-Fei Li & Andrej Karpathy & Justin Johnson Lecture 7 - 69 27 Jan 2016



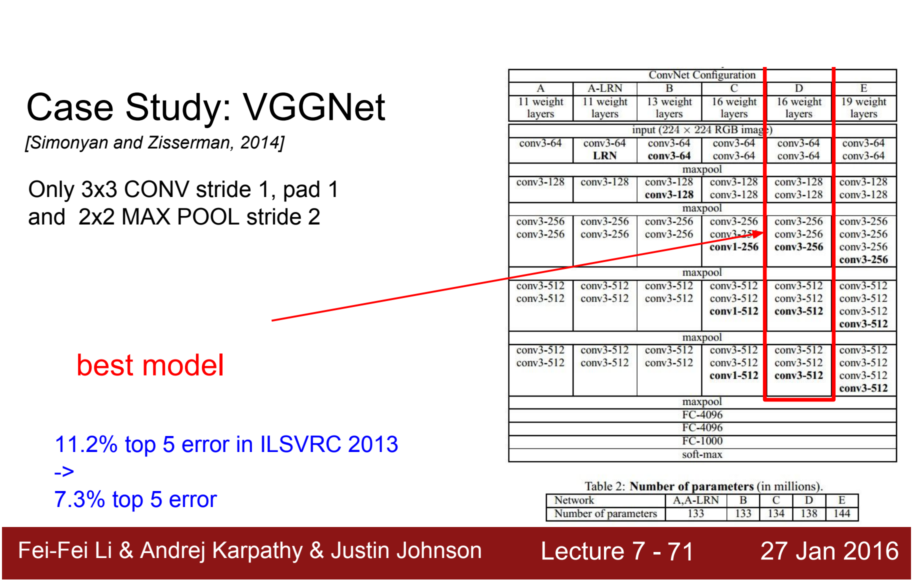
Case Study: ZFNet

[Zeiler and Fergus, 2013]

AlexNet but:
CONV1: change from (11x11 stride 4) to (7x7 stride 2)
CONV3,4,5: instead of 384, 384, 256 filters use 512, 1024, 512

ImageNet top 5 error: 15.4% -> 14.8%

Fei-Fei Li & Andrej Karpathy & Justin Johnson Lecture 7 - 70 27 Jan 2016



Case Study: VGGNet

[Simonyan and Zisserman, 2014]

Only 3x3 CONV stride 1, pad 1
and 2x2 MAX POOL stride 2

best model

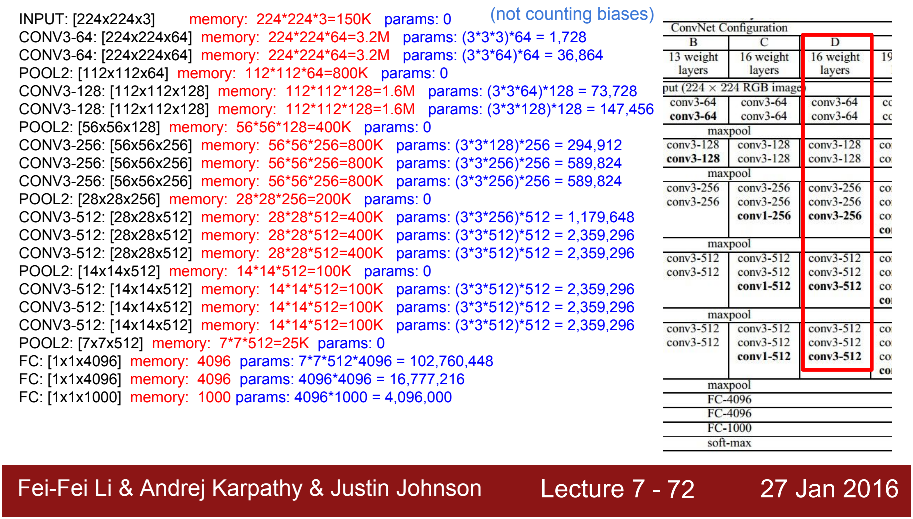
11.2% top 5 error in ILSVRC 2013
->
7.3% top 5 error

ConvNet Configuration					
A	A-LRN	B	C	D	E
11 weight layers	11 weight layers	13 weight layers	16 weight layers	16 weight layers	19 weight layers
input (224 × 224 RGB image)					
conv3-64	conv3-64 **LRN**	conv3-64 **conv3-64**	conv3-64 conv3-64	conv3-64 conv3-64	conv3-64 conv3-64
maxpool					
conv3-128	conv3-128	conv3-128 **conv3-128**	conv3-128 conv3-128	conv3-128 conv3-128	conv3-128 conv3-128
maxpool					
conv3-256 conv3-256	conv3-256 conv3-256	conv3-256 conv3-256	conv3-256 conv3-256 **conv1-256**	conv3-256 conv3-256 **conv3-256**	conv3-256 conv3-256 conv3-256 **conv3-256**
maxpool					
conv3-512 conv3-512	conv3-512 conv3-512	conv3-512 conv3-512	conv3-512 conv3-512 **conv1-512**	conv3-512 conv3-512 **conv3-512**	conv3-512 conv3-512 conv3-512 **conv3-512**
maxpool					
conv3-512 conv3-512	conv3-512 conv3-512	conv3-512 conv3-512	conv3-512 conv3-512 **conv1-512**	conv3-512 conv3-512 **conv3-512**	conv3-512 conv3-512 conv3-512 **conv3-512**
maxpool					
FC-4096					
FC-4096					
FC-1000					
soft-max					

Table 2: **Number of parameters** (in millions).

Network	A,A-LRN	B	C	D	E
Number of parameters	133	133	134	138	144

Fei-Fei Li & Andrej Karpathy & Justin Johnson Lecture 7 - 71 27 Jan 2016

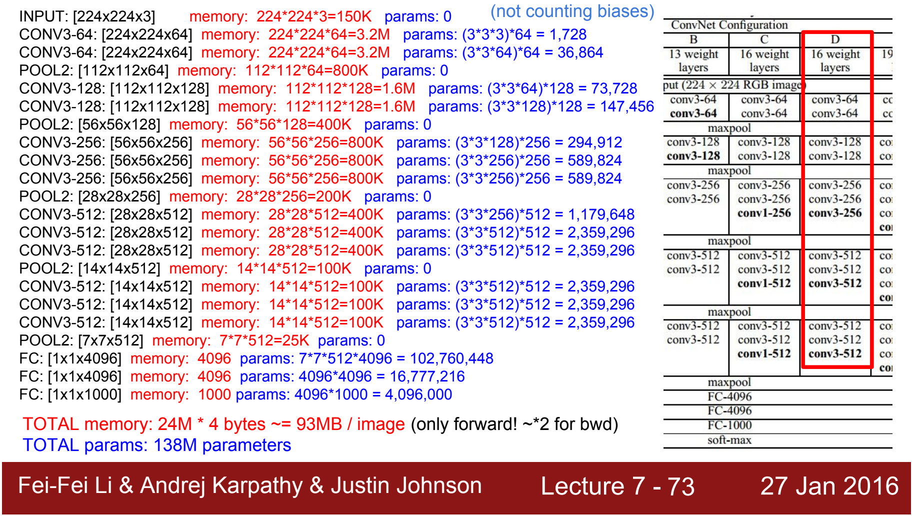


(not counting biases)

INPUT: [224x224x3] memory: 224*224*3=150K params: 0
CONV3-64: [224x224x64] memory: 224*224*64=3.2M params: (3*3*3)*64 = 1,728
CONV3-64: [224x224x64] memory: 224*224*64=3.2M params: (3*3*64)*64 = 36,864
POOL2: [112x112x64] memory: 112*112*64=800K params: 0
CONV3-128: [112x112x128] memory: 112*112*128=1.6M params: (3*3*64)*128 = 73,728
CONV3-128: [112x112x128] memory: 112*112*128=1.6M params: (3*3*128)*128 = 147,456
POOL2: [56x56x128] memory: 56*56*128=400K params: 0
CONV3-256: [56x56x256] memory: 56*56*256=800K params: (3*3*128)*256 = 294,912
CONV3-256: [56x56x256] memory: 56*56*256=800K params: (3*3*256)*256 = 589,824
CONV3-256: [56x56x256] memory: 56*56*256=800K params: (3*3*256)*256 = 589,824
POOL2: [28x28x256] memory: 28*28*256=200K params: 0
CONV3-512: [28x28x512] memory: 28*28*512=400K params: (3*3*256)*512 = 1,179,648
CONV3-512: [28x28x512] memory: 28*28*512=400K params: (3*3*512)*512 = 2,359,296
CONV3-512: [28x28x512] memory: 28*28*512=400K params: (3*3*512)*512 = 2,359,296
POOL2: [14x14x512] memory: 14*14*512=100K params: 0
CONV3-512: [14x14x512] memory: 14*14*512=100K params: (3*3*512)*512 = 2,359,296
CONV3-512: [14x14x512] memory: 14*14*512=100K params: (3*3*512)*512 = 2,359,296
CONV3-512: [14x14x512] memory: 14*14*512=100K params: (3*3*512)*512 = 2,359,296
POOL2: [7x7x512] memory: 7*7*512=25K params: 0
FC: [1x1x4096] memory: 4096 params: 7*7*512*4096 = 102,760,448
FC: [1x1x4096] memory: 4096 params: 4096*4096 = 16,777,216
FC: [1x1x1000] memory: 1000 params: 4096*1000 = 4,096,000

ConvNet Configuration			
B	C	D	
13 weight layers	16 weight layers	16 weight layers	19
put (224 × 224 RGB image)			
conv3-64 **conv3-64**	conv3-64 conv3-64	conv3-64 conv3-64	cc cc
maxpool			
conv3-128 **conv3-128**	conv3-128 conv3-128	conv3-128 conv3-128	co co
maxpool			
conv3-256 conv3-256	conv3-256 conv3-256 **conv1-256**	conv3-256 conv3-256 **conv3-256**	co co co **co**
maxpool			
conv3-512 conv3-512	conv3-512 conv3-512 **conv1-512**	conv3-512 conv3-512 **conv3-512**	co co co **co**
maxpool			
conv3-512 conv3-512	conv3-512 conv3-512 **conv1-512**	conv3-512 conv3-512 **conv3-512**	co co co **co**
maxpool			
FC-4096			
FC-4096			
FC-1000			
soft-max			

Fei-Fei Li & Andrej Karpathy & Justin Johnson Lecture 7 - 72 27 Jan 2016



(not counting biases)

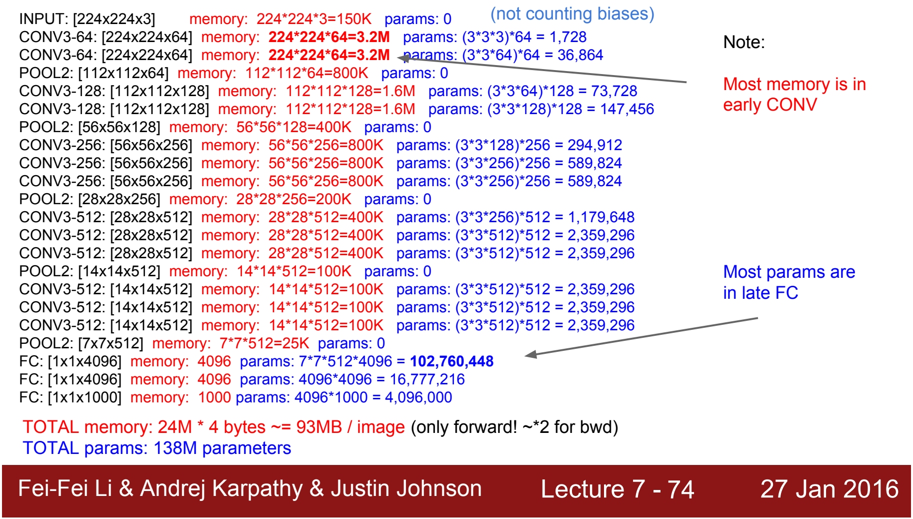
INPUT: [224x224x3]	memory: 224*224*3=150K	params: 0
CONV3-64: [224x224x64]	memory: 224*224*64=3.2M	params: (3*3*3)*64 = 1,728
CONV3-64: [224x224x64]	memory: 224*224*64=3.2M	params: (3*3*64)*64 = 36,864
POOL2: [112x112x64]	memory: 112*112*64=800K	params: 0
CONV3-128: [112x112x128]	memory: 112*112*128=1.6M	params: (3*3*64)*128 = 73,728
CONV3-128: [112x112x128]	memory: 112*112*128=1.6M	params: (3*3*128)*128 = 147,456
POOL2: [56x56x128]	memory: 56*56*128=400K	params: 0
CONV3-256: [56x56x256]	memory: 56*56*256=800K	params: (3*3*128)*256 = 294,912
CONV3-256: [56x56x256]	memory: 56*56*256=800K	params: (3*3*256)*256 = 589,824
CONV3-256: [56x56x256]	memory: 56*56*256=800K	params: (3*3*256)*256 = 589,824
POOL2: [28x28x256]	memory: 28*28*256=200K	params: 0
CONV3-512: [28x28x512]	memory: 28*28*512=400K	params: (3*3*256)*512 = 1,179,648
CONV3-512: [28x28x512]	memory: 28*28*512=400K	params: (3*3*512)*512 = 2,359,296
CONV3-512: [28x28x512]	memory: 28*28*512=400K	params: (3*3*512)*512 = 2,359,296
POOL2: [14x14x512]	memory: 14*14*512=100K	params: 0
CONV3-512: [14x14x512]	memory: 14*14*512=100K	params: (3*3*512)*512 = 2,359,296
CONV3-512: [14x14x512]	memory: 14*14*512=100K	params: (3*3*512)*512 = 2,359,296
CONV3-512: [14x14x512]	memory: 14*14*512=100K	params: (3*3*512)*512 = 2,359,296
POOL2: [7x7x512]	memory: 7*7*512=25K	params: 0
FC: [1x1x4096]	memory: 4096	params: 7*7*512*4096 = 102,760,448
FC: [1x1x4096]	memory: 4096	params: 4096*4096 = 16,777,216
FC: [1x1x1000]	memory: 1000	params: 4096*1000 = 4,096,000

TOTAL memory: 24M * 4 bytes ~= 93MB / image (only forward! ~*2 for bwd)

TOTAL params: 138M parameters

ConvNet Configuration			
B	C	D	
13 weight layers	16 weight layers	16 weight layers	19
put (224 × 224 RGB image)			
conv3-64 **conv3-64**	conv3-64 conv3-64	conv3-64 conv3-64	cc cc
maxpool			
conv3-128 **conv3-128**	conv3-128 conv3-128	conv3-128 conv3-128	co co
maxpool			
conv3-256 conv3-256	conv3-256 conv3-256 **conv1-256**	conv3-256 conv3-256 **conv3-256**	co co co **co**
maxpool			
conv3-512 conv3-512	conv3-512 conv3-512 **conv1-512**	conv3-512 conv3-512 **conv3-512**	co co co **co**
maxpool			
conv3-512 conv3-512	conv3-512 conv3-512 **conv1-512**	conv3-512 conv3-512 **conv3-512**	co co co **co**
maxpool			
FC-4096			
FC-4096			
FC-1000			
soft-max			

Fei-Fei Li & Andrej Karpathy & Justin Johnson Lecture 7 - 73 27 Jan 2016



(not counting biases)

INPUT: [224x224x3] memory: 224*224*3=150K params: 0
CONV3-64: [224x224x64] memory: **224*224*64=3.2M** params: (3*3*3)*64 = 1,728
CONV3-64: [224x224x64] memory: **224*224*64=3.2M** params: (3*3*64)*64 = 36,864
POOL2: [112x112x64] memory: 112*112*64=800K params: 0
CONV3-128: [112x112x128] memory: 112*112*128=1.6M params: (3*3*64)*128 = 73,728
CONV3-128: [112x112x128] memory: 112*112*128=1.6M params: (3*3*128)*128 = 147,456
POOL2: [56x56x128] memory: 56*56*128=400K params: 0
CONV3-256: [56x56x256] memory: 56*56*256=800K params: (3*3*128)*256 = 294,912
CONV3-256: [56x56x256] memory: 56*56*256=800K params: (3*3*256)*256 = 589,824
CONV3-256: [56x56x256] memory: 56*56*256=800K params: (3*3*256)*256 = 589,824
POOL2: [28x28x256] memory: 28*28*256=200K params: 0
CONV3-512: [28x28x512] memory: 28*28*512=400K params: (3*3*256)*512 = 1,179,648
CONV3-512: [28x28x512] memory: 28*28*512=400K params: (3*3*512)*512 = 2,359,296
CONV3-512: [28x28x512] memory: 28*28*512=400K params: (3*3*512)*512 = 2,359,296
POOL2: [14x14x512] memory: 14*14*512=100K params: 0
CONV3-512: [14x14x512] memory: 14*14*512=100K params: (3*3*512)*512 = 2,359,296
CONV3-512: [14x14x512] memory: 14*14*512=100K params: (3*3*512)*512 = 2,359,296
CONV3-512: [14x14x512] memory: 14*14*512=100K params: (3*3*512)*512 = 2,359,296
POOL2: [7x7x512] memory: 7*7*512=25K params: 0
FC: [1x1x4096] memory: 4096 params: 7*7*512*4096 = **102,760,448**
FC: [1x1x4096] memory: 4096 params: 4096*4096 = 16,777,216
FC: [1x1x1000] memory: 1000 params: 4096*1000 = 4,096,000

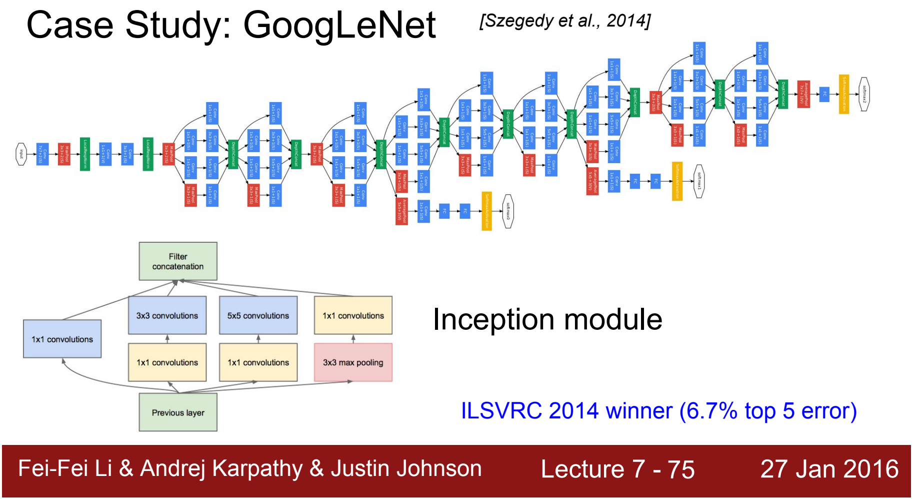
Note:

Most memory is in early CONV

Most params are in late FC

TOTAL memory: 24M * 4 bytes ~= 93MB / image (only forward! ~*2 for bwd)
TOTAL params: 138M parameters

Fei-Fei Li & Andrej Karpathy & Justin Johnson Lecture 7 - 74 27 Jan 2016



Case Study: GoogLeNet

[Szegedy et al., 2014]

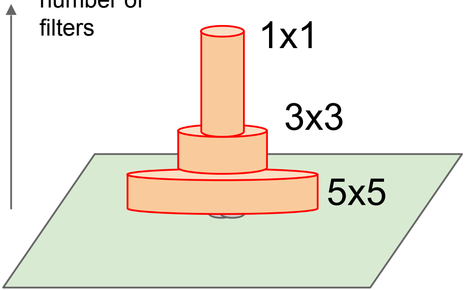
Inception module

ILSVRC 2014 winner (6.7% top 5 error)

Fei-Fei Li & Andrej Karpathy & Justin Johnson Lecture 7 - 75 27 Jan 2016

Slides from Fisher Yu

Schematic view (naive version)

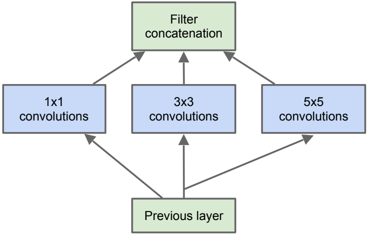


number of filters

1x1

3x3

5x5



Filter concatenation

1x1 convolutions

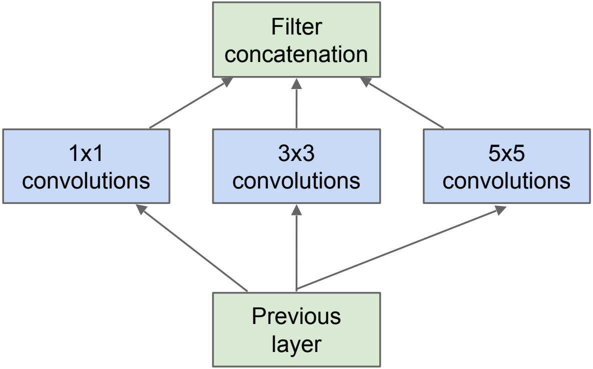
3x3 convolutions

5x5 convolutions

Previous layer

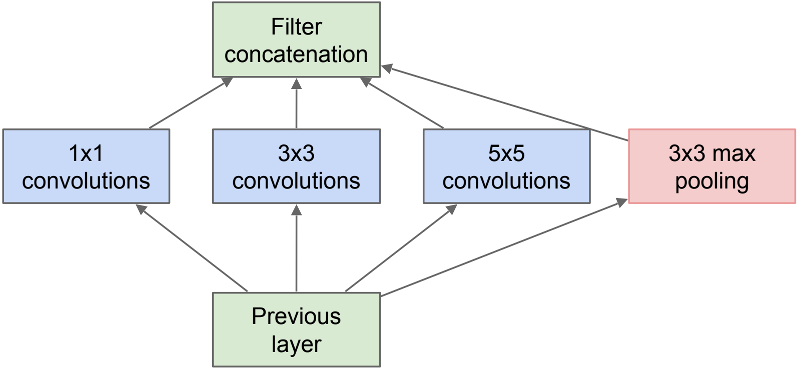
Slides from Fisher Yu

Naive idea



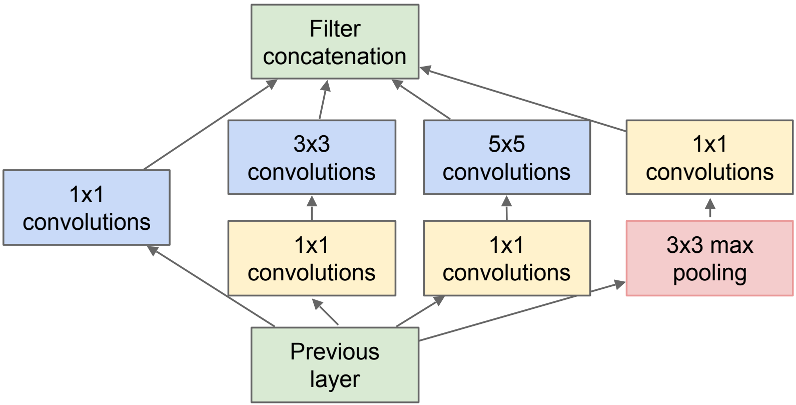
Slides from Fisher Yu

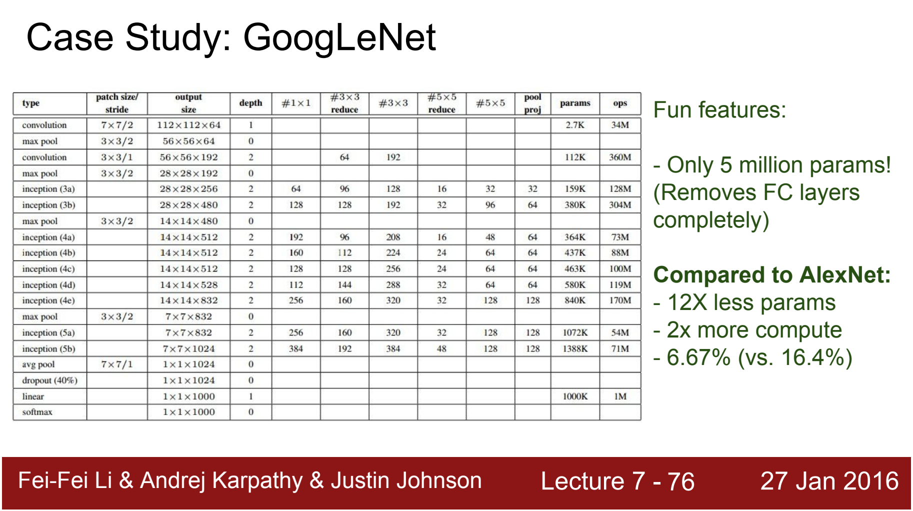
Naive idea (**does not work!**)



Slides from Fisher Yu

Inception module





Case Study: GoogLeNet

type	patch size/ stride	output size	depth	#1×1	#3×3 reduce	#3×3	#5×5 reduce	#5×5	pool proj	params	ops
convolution	7×7/2	112×112×64	1							2.7K	34M
max pool	3×3/2	56×56×64	0								
convolution	3×3/1	56×56×192	2		64	192				112K	360M
max pool	3×3/2	28×28×192	0								
inception (3a)		28×28×256	2	64	96	128	16	32	32	159K	128M
inception (3b)		28×28×480	2	128	128	192	32	96	64	380K	304M
max pool	3×3/2	14×14×480	0								
inception (4a)		14×14×512	2	192	96	208	16	48	64	364K	73M
inception (4b)		14×14×512	2	160	112	224	24	64	64	437K	88M
inception (4c)		14×14×512	2	128	128	256	24	64	64	463K	100M
inception (4d)		14×14×528	2	112	144	288	32	64	64	580K	119M
inception (4e)		14×14×832	2	256	160	320	32	128	128	840K	170M
max pool	3×3/2	7×7×832	0								
inception (5a)		7×7×832	2	256	160	320	32	128	128	1072K	54M
inception (5b)		7×7×1024	2	384	192	384	48	128	128	1388K	71M
avg pool	7×7/1	1×1×1024	0								
dropout (40%)		1×1×1024	0								
linear		1×1×1000	1							1000K	1M
softmax		1×1×1000	0								

Fun features:

- Only 5 million params! (Removes FC layers completely)

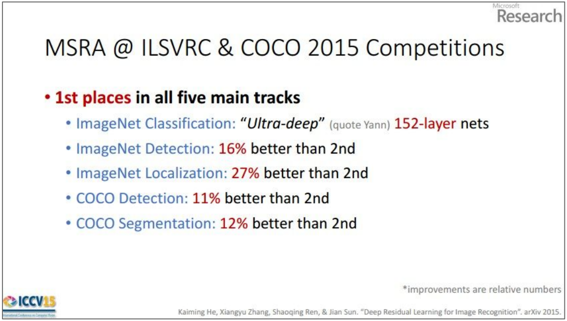
Compared to AlexNet:
- 12X less params
- 2x more compute
- 6.67% (vs. 16.4%)

Fei-Fei Li & Andrej Karpathy & Justin Johnson Lecture 7 - 76 27 Jan 2016

Case Study: ResNet

[He et al., 2015]

ILSVRC 2015 winner (3.6% top 5 error)



Microsoft Research

MSRA @ ILSVRC & COCO 2015 Competitions

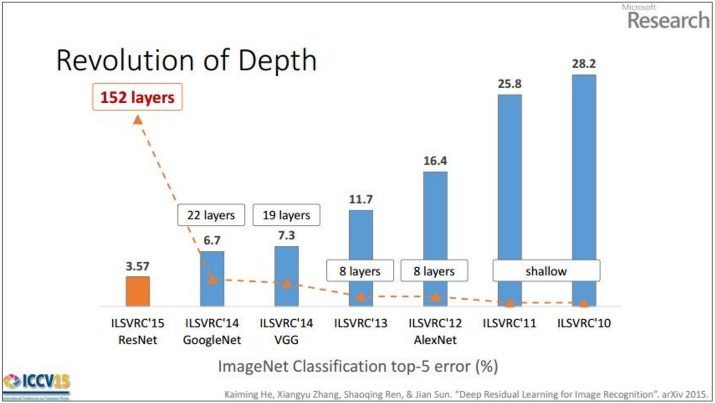
- **1st places in all five main tracks**
  - ImageNet Classification: *"Ultra-deep"* (quote Yann) 152-layer nets
  - ImageNet Detection: 16% better than 2nd
  - ImageNet Localization: 27% better than 2nd
  - COCO Detection: 11% better than 2nd
  - COCO Segmentation: 12% better than 2nd

*improvements are relative numbers

ICCV15

Kaiming He, Xiangyu Zhang, Shaoqing Ren, & Jian Sun. "Deep Residual Learning for Image Recognition". arXiv 2015.

Slide from Kaiming He's recent presentation https://www.youtube.com/watch?v=1PGLj-uKT1w

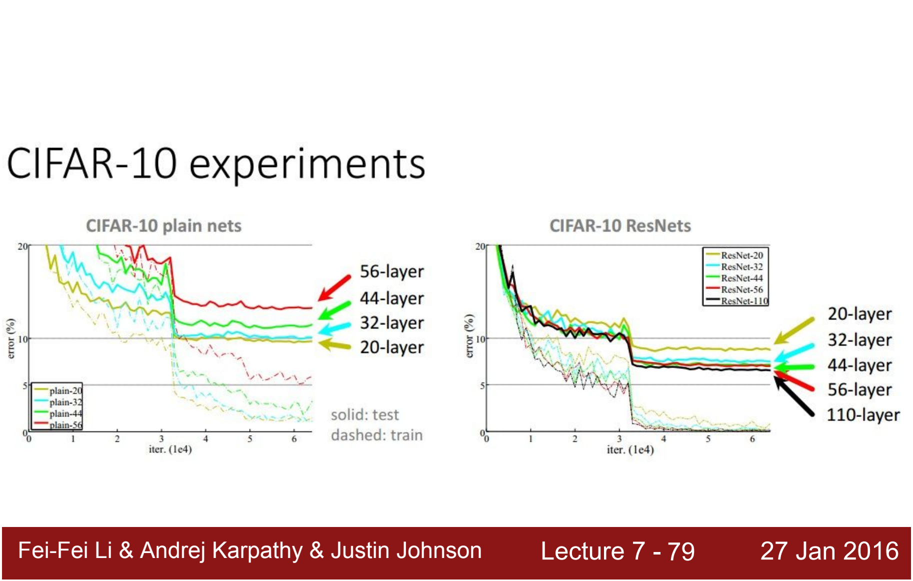


Revolution of Depth

Kaiming He, Xiangyu Zhang, Shaoqing Ren, & Jian Sun. "Deep Residual Learning for Image Recognition". arXiv 2015.

(slide from Kaiming He's recent presentation)

Fei-Fei Li & Andrej Karpathy & Justin Johnson Lecture 7 - 78 27 Jan 2016



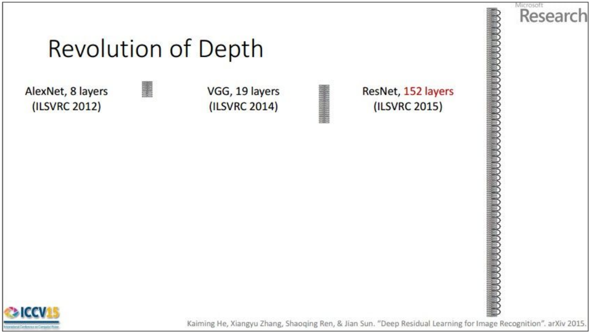
CIFAR-10 experiments

Fei-Fei Li & Andrej Karpathy & Justin Johnson Lecture 7 - 79 27 Jan 2016

Case Study: ResNet

[He et al., 2015]

ILSVRC 2015 winner (3.6% top 5 error)



Revolution of Depth

AlexNet, 8 layers
(ILSVRC 2012)

VGG, 19 layers
(ILSVRC 2014)

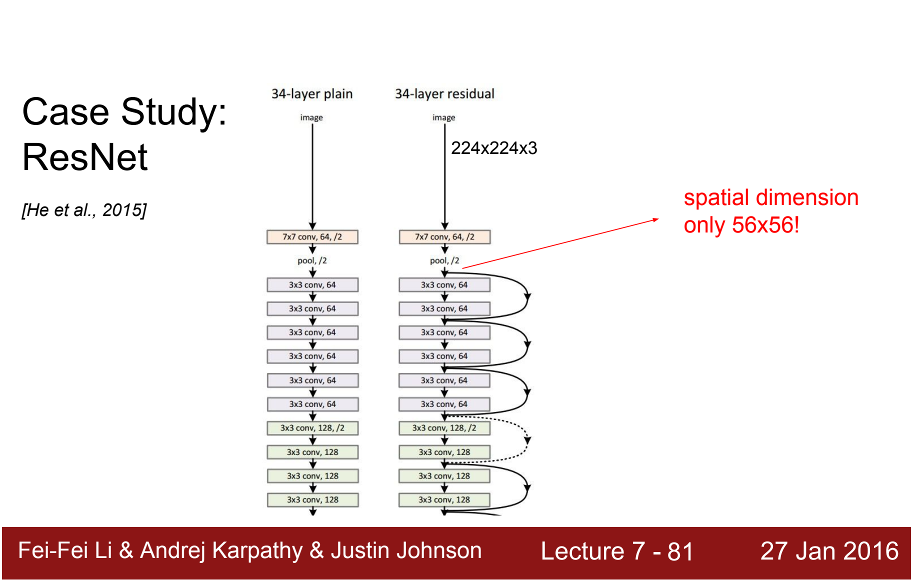
ResNet, 152 layers
(ILSVRC 2015)

Kaiming He, Xiangyu Zhang, Shaoqing Ren, & Jian Sun. "Deep Residual Learning for Image Recognition". arXiv 2015.

(slide from Kaiming He's recent presentation)

2-3 weeks of training on 8 GPU machine

at runtime: faster than a VGGNet! (even though it has 8x more layers)

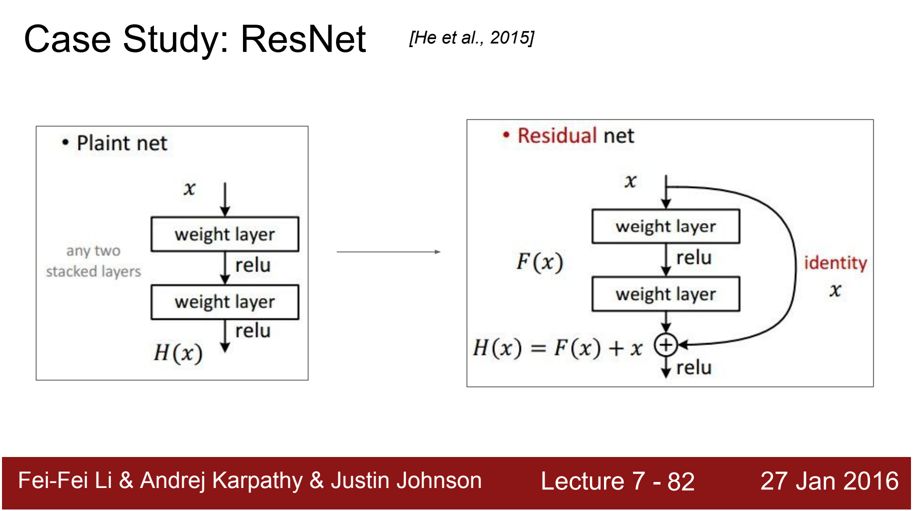


Case Study: ResNet

[He et al., 2015]

spatial dimension only 56x56!

Fei-Fei Li & Andrej Karpathy & Justin Johnson Lecture 7 - 81 27 Jan 2016



Case Study: ResNet *[He et al., 2015]*

- Plaint net

x

weight layer

relu

weight layer

relu

$H(x)$

any two stacked layers

- Residual net

x

weight layer

relu

weight layer

$F(x)$

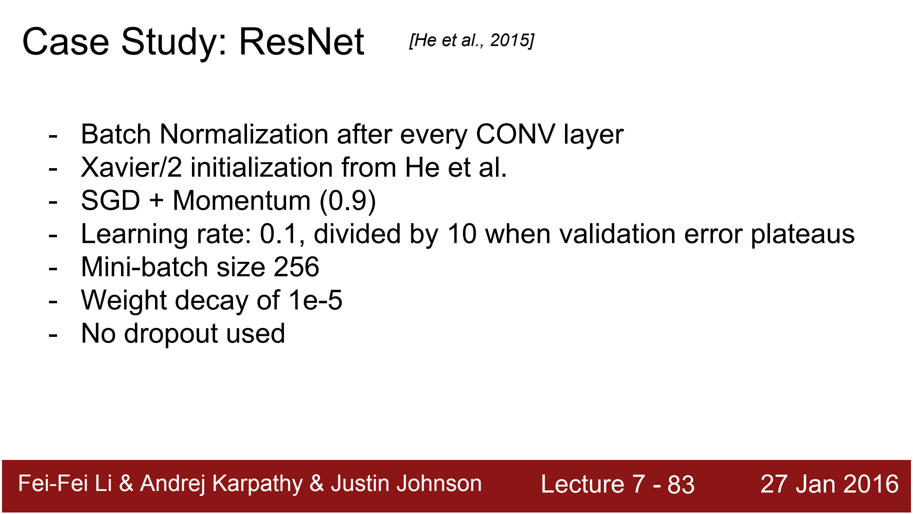
identity

x

$H(x) = F(x) + x$

relu

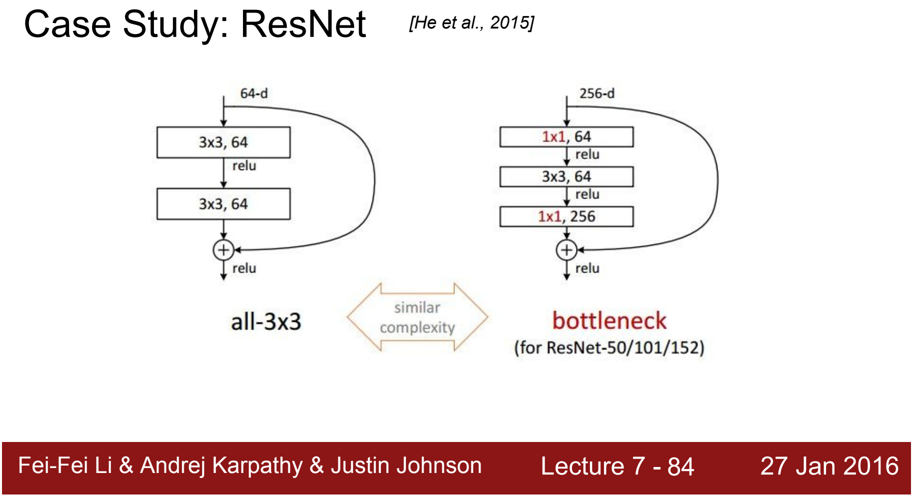
Fei-Fei Li & Andrej Karpathy & Justin Johnson Lecture 7 - 82 27 Jan 2016



Case Study: ResNet *[He et al., 2015]*

- Batch Normalization after every CONV layer
- Xavier/2 initialization from He et al.
- SGD + Momentum (0.9)
- Learning rate: 0.1, divided by 10 when validation error plateaus
- Mini-batch size 256
- Weight decay of 1e-5
- No dropout used

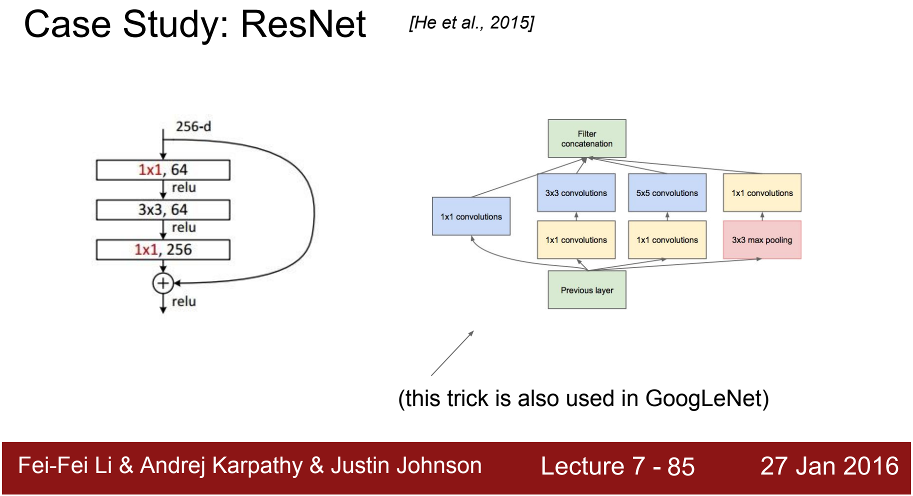
Fei-Fei Li & Andrej Karpathy & Justin Johnson Lecture 7 - 83 27 Jan 2016



Case Study: ResNet

[He et al., 2015]

Fei-Fei Li & Andrej Karpathy & Justin Johnson Lecture 7 - 84 27 Jan 2016

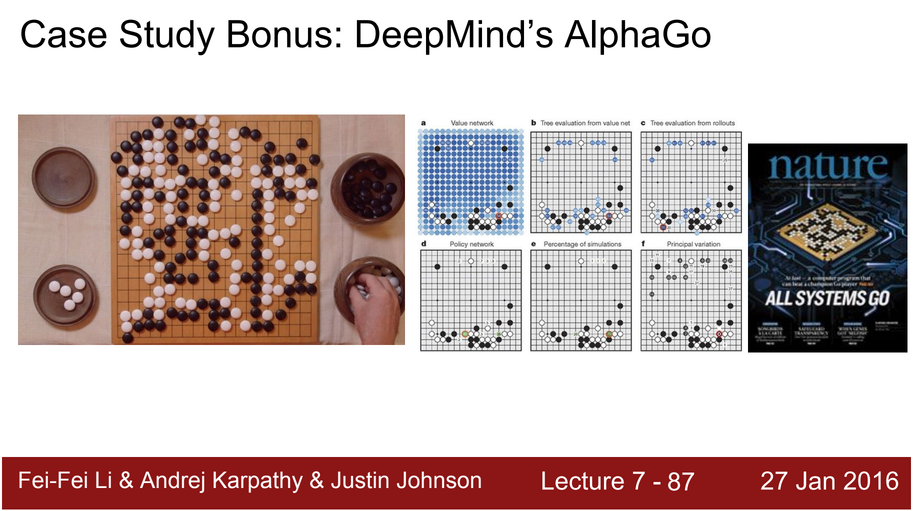


Case Study: ResNet

[He et al., 2015]

(this trick is also used in GoogLeNet)

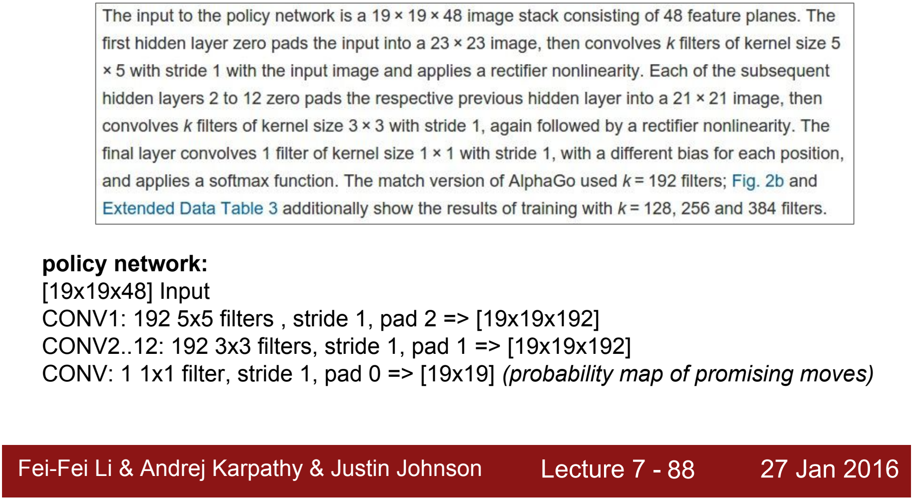
Fei-Fei Li & Andrej Karpathy & Justin Johnson Lecture 7 - 85 27 Jan 2016



Case Study Bonus: DeepMind’s AlphaGo

a Value network
b Tree evaluation from value net
c Tree evaluation from rollouts
d Policy network
e Percentage of simulations
f Principal variation
nature
ALL SYSTEMS GO

Fei-Fei Li & Andrej Karpathy & Justin Johnson Lecture 7 - 87 27 Jan 2016



The input to the policy network is a 19 × 19 × 48 image stack consisting of 48 feature planes. The first hidden layer zero pads the input into a 23 × 23 image, then convolves *k* filters of kernel size 5 × 5 with stride 1 with the input image and applies a rectifier nonlinearity. Each of the subsequent hidden layers 2 to 12 zero pads the respective previous hidden layer into a 21 × 21 image, then convolves *k* filters of kernel size 3 × 3 with stride 1, again followed by a rectifier nonlinearity. The final layer convolves 1 filter of kernel size 1 × 1 with stride 1, with a different bias for each position, and applies a softmax function. The match version of AlphaGo used *k* = 192 filters; Fig. 2b and Extended Data Table 3 additionally show the results of training with *k* = 128, 256 and 384 filters.

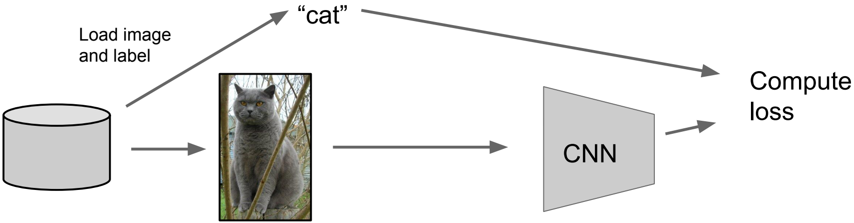
policy network:
[19x19x48] Input
CONV1: 192 5x5 filters , stride 1, pad 2 => [19x19x192]
CONV2..12: 192 3x3 filters, stride 1, pad 1 => [19x19x192]
CONV: 1 1x1 filter, stride 1, pad 0 => [19x19] *(probability map of promising moves)*

Fei-Fei Li & Andrej Karpathy & Justin Johnson Lecture 7 - 88 27 Jan 2016

Some CNN tricks

- Data augmentation
- Transfer learning
- Use of small filters
- Implementing CNN efficiently
- Use of GPUs
- About floating point precision

Data Augmentation



Load image and label

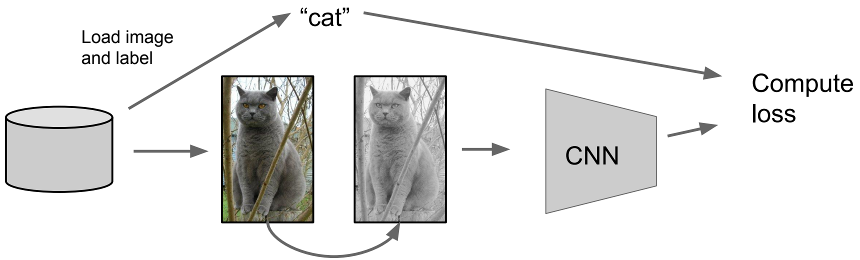
“cat”

CNN

Compute loss

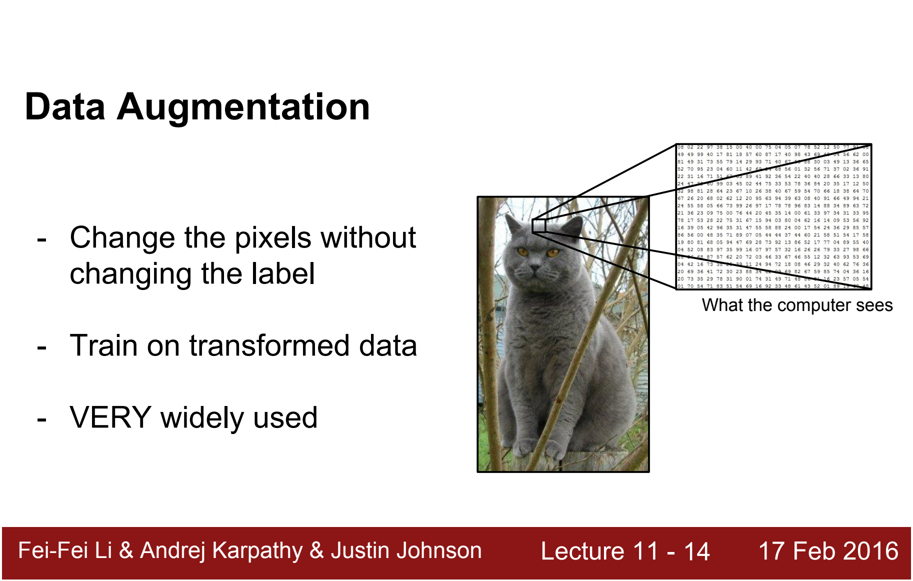
Fei-Fei Li & Andrej Karpathy & Justin Johnson Lecture 11 - 12 17 Feb 2016

Data Augmentation



Transform image

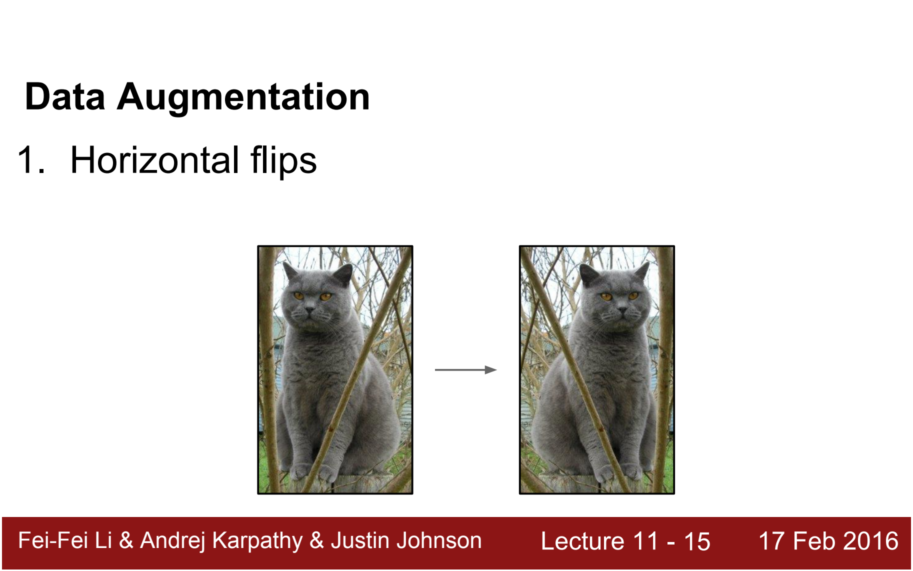
Fei-Fei Li & Andrej Karpathy & Justin Johnson Lecture 11 - 13 17 Feb 2016



Data Augmentation

- Change the pixels without changing the label
- Train on transformed data
- VERY widely used

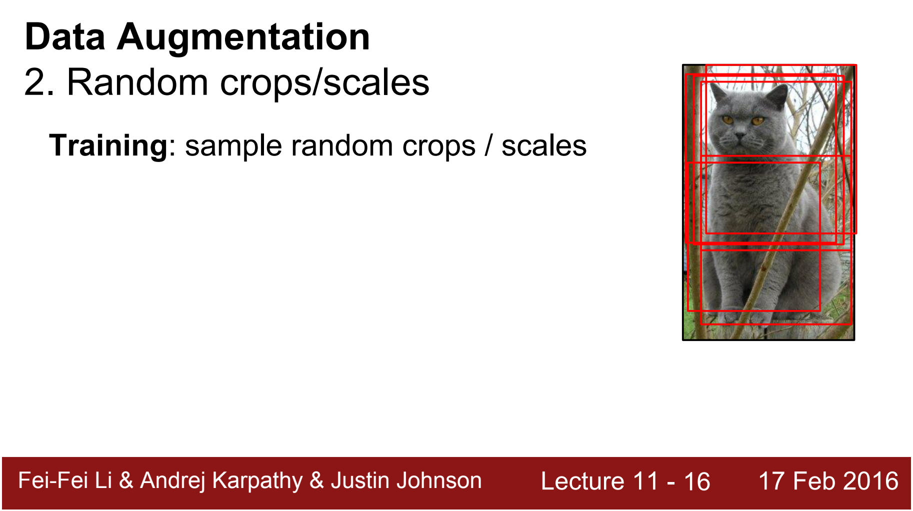
Fei-Fei Li & Andrej Karpathy & Justin Johnson Lecture 11 - 14 17 Feb 2016



Data Augmentation

1. Horizontal flips

Fei-Fei Li & Andrej Karpathy & Justin Johnson Lecture 11 - 15 17 Feb 2016

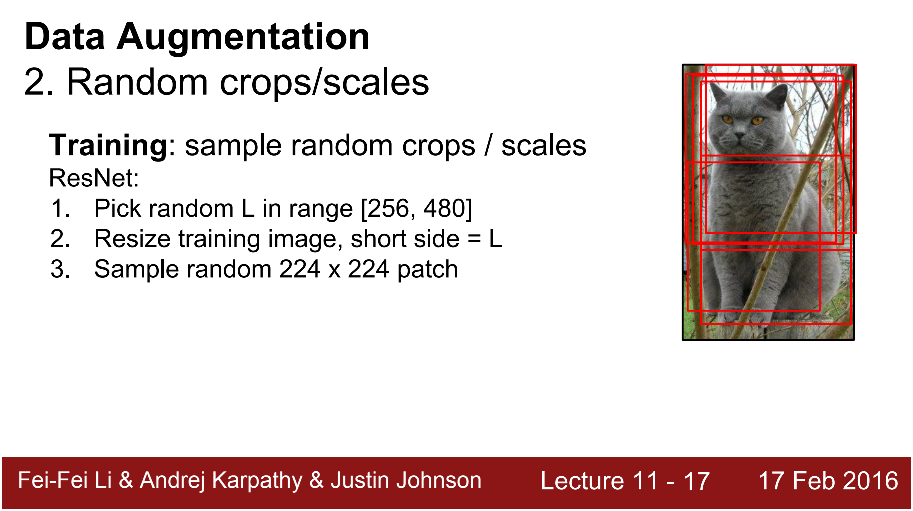


Data Augmentation

2. Random crops/scales

Training: sample random crops / scales

Fei-Fei Li & Andrej Karpathy & Justin Johnson Lecture 11 - 16 17 Feb 2016



Data Augmentation

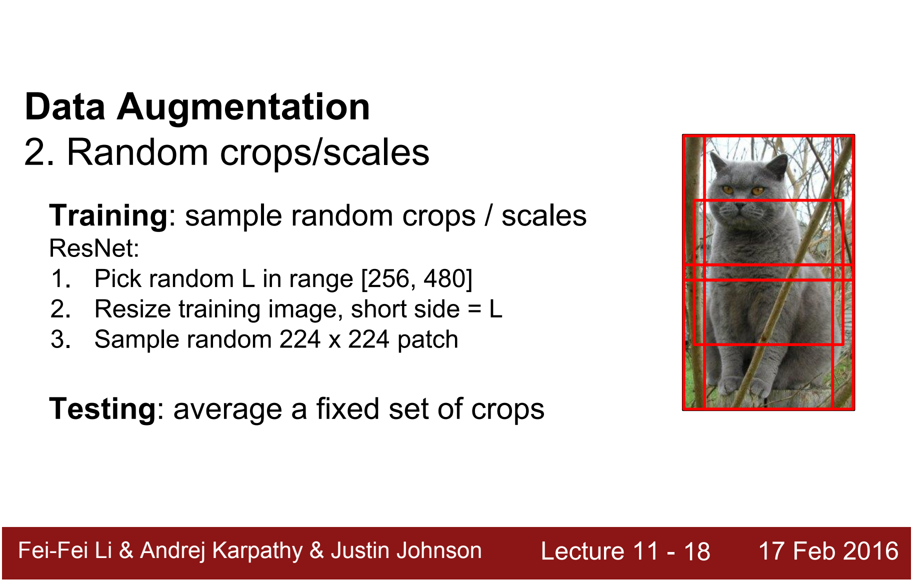
2. Random crops/scales

Training: sample random crops / scales

ResNet:

1. Pick random L in range [256, 480]
2. Resize training image, short side = L
3. Sample random 224 x 224 patch

Fei-Fei Li & Andrej Karpathy & Justin Johnson Lecture 11 - 17 17 Feb 2016



Data Augmentation

2. Random crops/scales

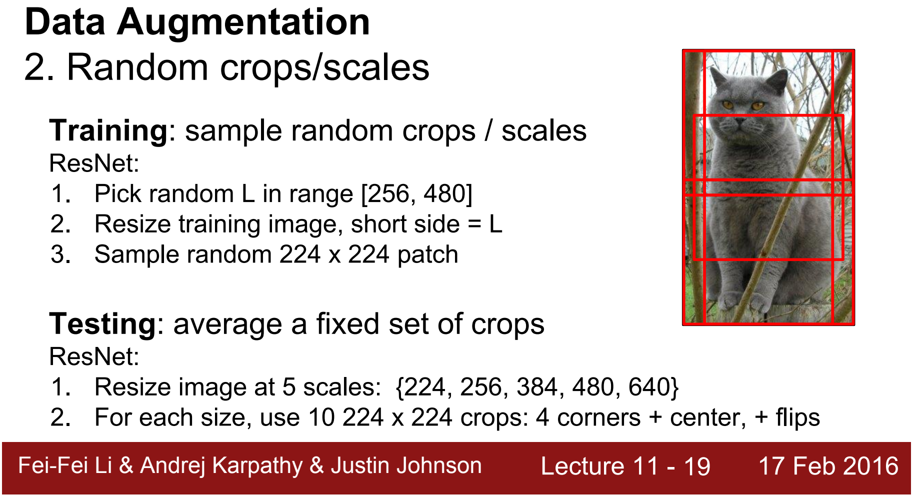
Training: sample random crops / scales

ResNet:

1. Pick random L in range [256, 480]
2. Resize training image, short side = L
3. Sample random 224 x 224 patch

Testing: average a fixed set of crops

Fei-Fei Li & Andrej Karpathy & Justin Johnson Lecture 11 - 18 17 Feb 2016



Data Augmentation

2. Random crops/scales

Training: sample random crops / scales

ResNet:

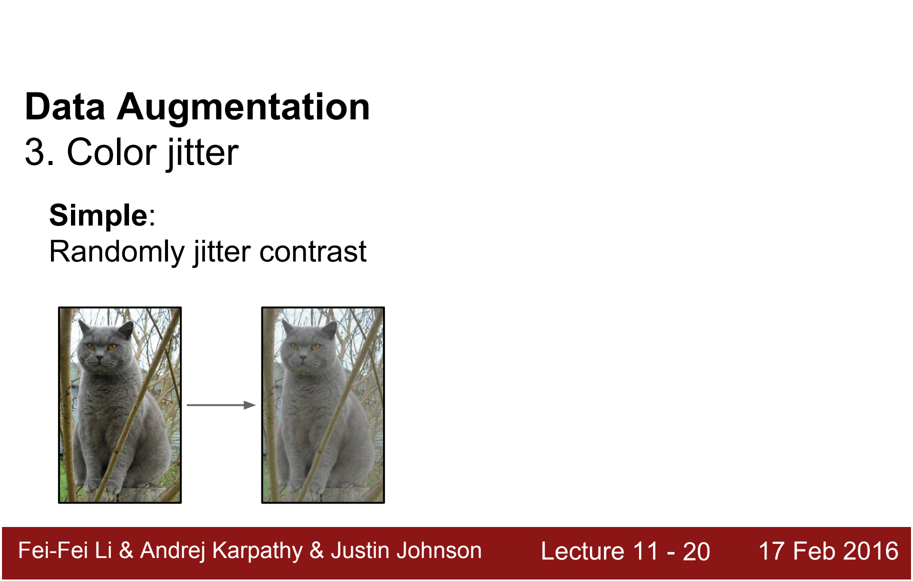
1. Pick random L in range [256, 480]
2. Resize training image, short side = L
3. Sample random 224 x 224 patch

Testing: average a fixed set of crops

ResNet:

1. Resize image at 5 scales: {224, 256, 384, 480, 640}
2. For each size, use 10 224 x 224 crops: 4 corners + center, + flips

Fei-Fei Li & Andrej Karpathy & Justin Johnson Lecture 11 - 19 17 Feb 2016

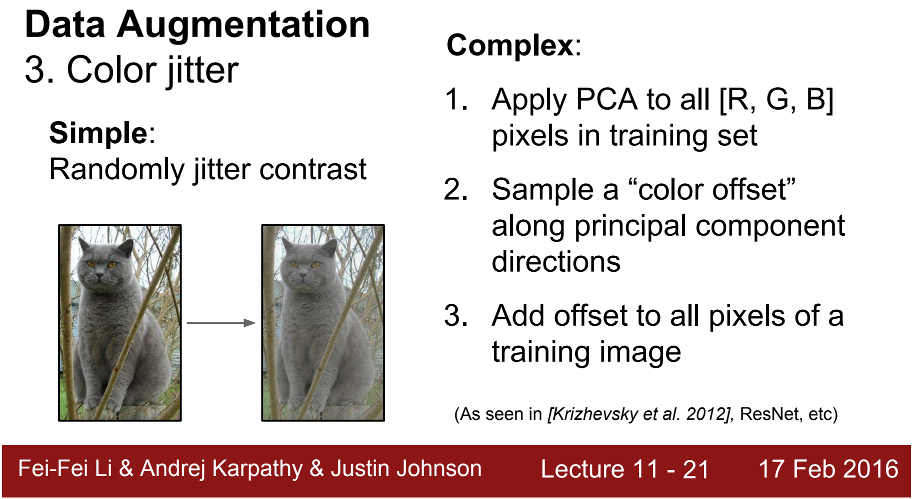


Data Augmentation

3. Color jitter

Simple:
Randomly jitter contrast

Fei-Fei Li & Andrej Karpathy & Justin Johnson Lecture 11 - 20 17 Feb 2016



Data Augmentation

3. Color jitter

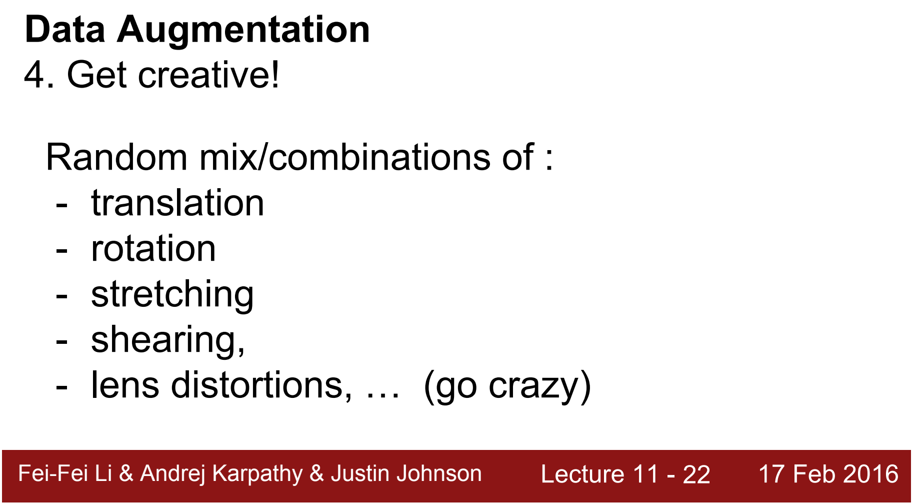
Simple:
Randomly jitter contrast

Complex:

1. Apply PCA to all [R, G, B] pixels in training set
2. Sample a “color offset” along principal component directions
3. Add offset to all pixels of a training image

(As seen in *[Krizhevsky et al. 2012],* ResNet, etc)

Fei-Fei Li & Andrej Karpathy & Justin Johnson Lecture 11 - 21 17 Feb 2016



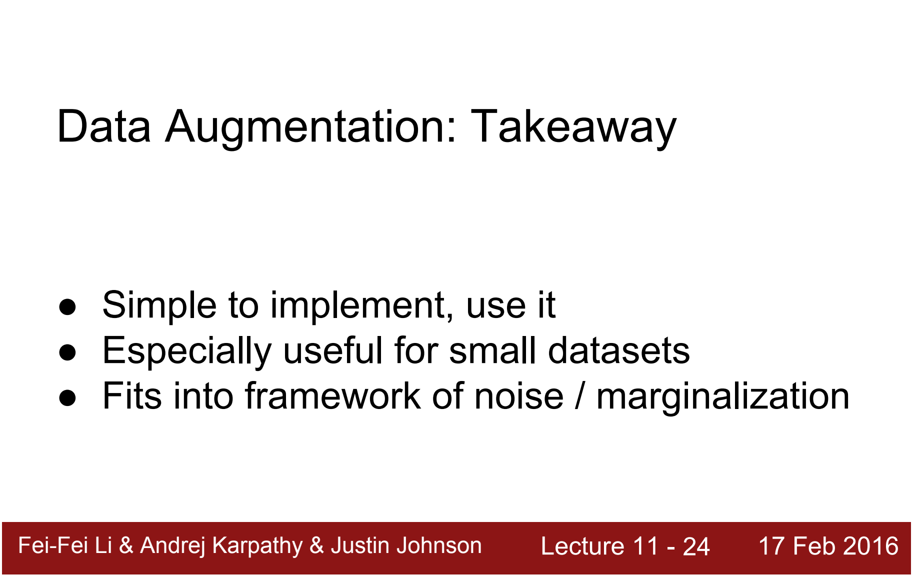
Data Augmentation

4. Get creative!

Random mix/combinations of :

- translation
- rotation
- stretching
- shearing,
- lens distortions, … (go crazy)

Fei-Fei Li & Andrej Karpathy & Justin Johnson Lecture 11 - 22 17 Feb 2016

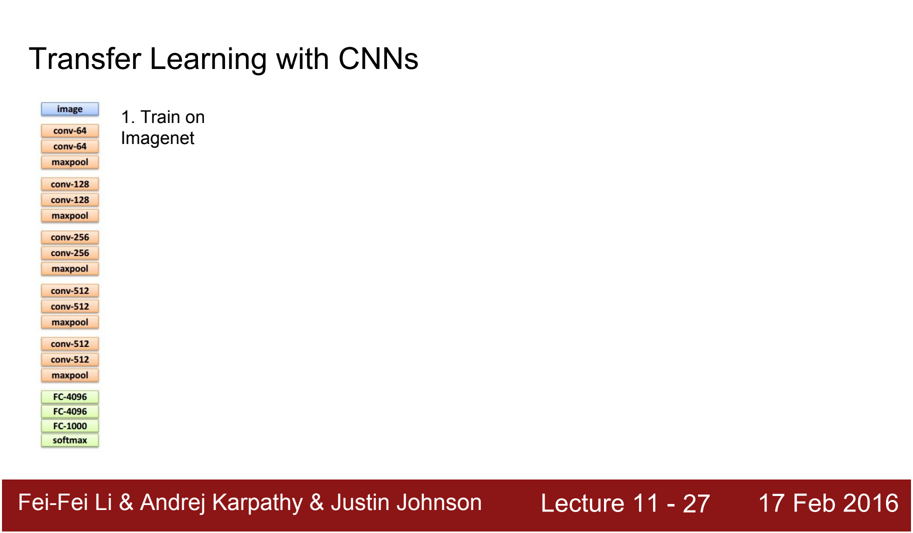


Data Augmentation: Takeaway

- Simple to implement, use it
- Especially useful for small datasets
- Fits into framework of noise / marginalization

Fei-Fei Li & Andrej Karpathy & Justin Johnson Lecture 11 - 24 17 Feb 2016

Don't necesarily need lots of data for CNN



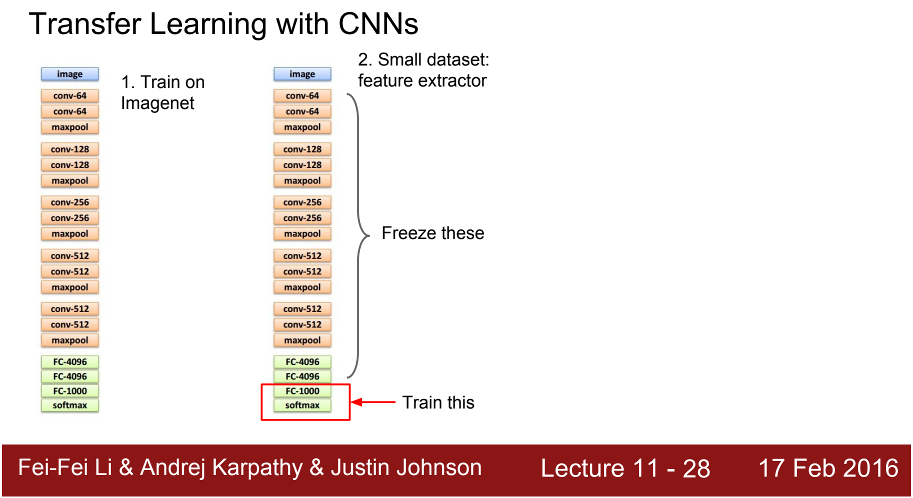
Transfer Learning with CNNs

image
conv-64
conv-64
maxpool
conv-128
conv-128
maxpool
conv-256
conv-256
maxpool
conv-512
conv-512
maxpool
conv-512
conv-512
maxpool
FC-4096
FC-4096
FC-1000
softmax

1. Train on Imagenet

Fei-Fei Li & Andrej Karpathy & Justin Johnson Lecture 11 - 27 17 Feb 2016

Don’t necesarily need lots of data for CNN



Transfer Learning with CNNs

1. Train on Imagenet

image
conv-64
conv-64
maxpool
conv-128
conv-128
maxpool
conv-256
conv-256
maxpool
conv-512
conv-512
maxpool
conv-512
conv-512
maxpool
FC-4096
FC-4096
FC-1000
softmax

2. Small dataset: feature extractor

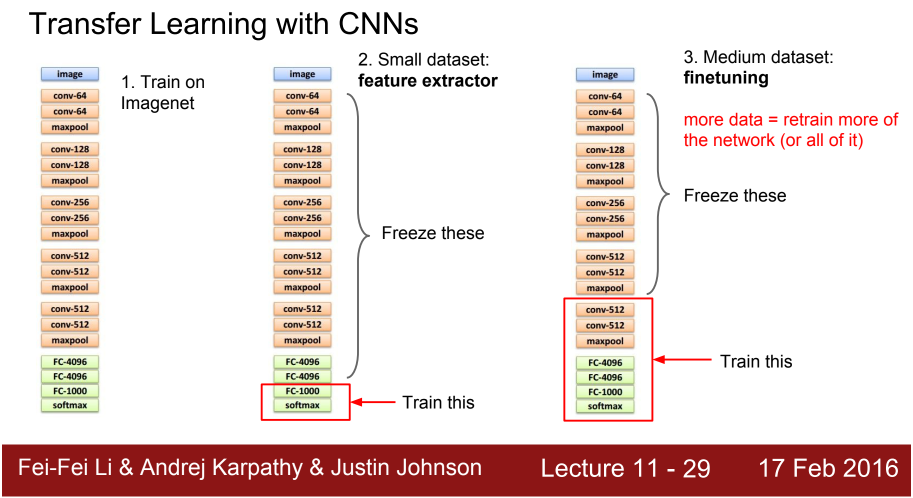
image
conv-64
conv-64
maxpool
conv-128
conv-128
maxpool
conv-256
conv-256
maxpool
conv-512
conv-512
maxpool
conv-512
conv-512
maxpool
FC-4096
FC-4096
FC-1000
softmax

Freeze these

Train this

Fei-Fei Li & Andrej Karpathy & Justin Johnson Lecture 11 - 28 17 Feb 2016

Don’t necesarily need lots of data for CNN



Transfer Learning with CNNs

1. Train on Imagenet

image
conv-64
conv-64
maxpool
conv-128
conv-128
maxpool
conv-256
conv-256
maxpool
conv-512
conv-512
maxpool
conv-512
conv-512
maxpool
FC-4096
FC-4096
FC-1000
softmax

2. Small dataset: **feature extractor**

image
conv-64
conv-64
maxpool
conv-128
conv-128
maxpool
conv-256
conv-256
maxpool
conv-512
conv-512
maxpool
conv-512
conv-512
maxpool
FC-4096
FC-4096
FC-1000
softmax

Freeze these

Train this

3. Medium dataset: **finetuning**

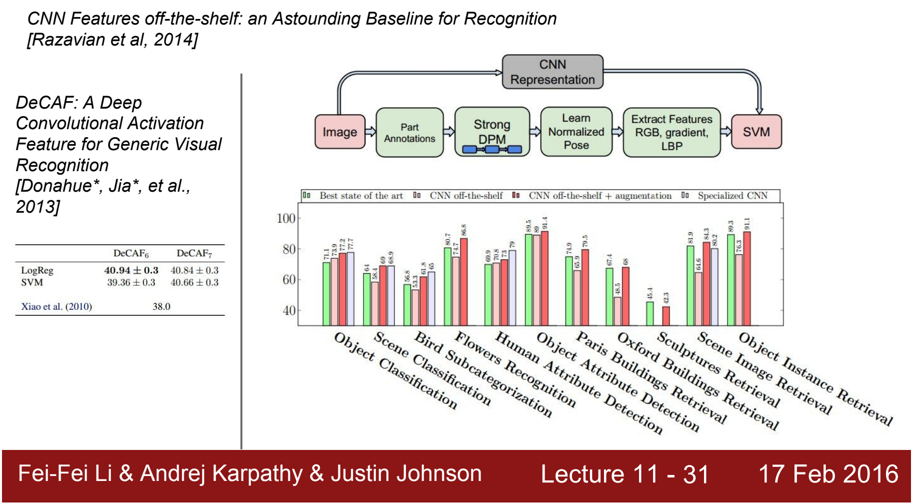
more data = retrain more of the network (or all of it)

image
conv-64
conv-64
maxpool
conv-128
conv-128
maxpool
conv-256
conv-256
maxpool
conv-512
conv-512
maxpool
conv-512
conv-512
maxpool
FC-4096
FC-4096
FC-1000
softmax

Freeze these

Train this

Fei-Fei Li & Andrej Karpathy & Justin Johnson Lecture 11 - 29 17 Feb 2016

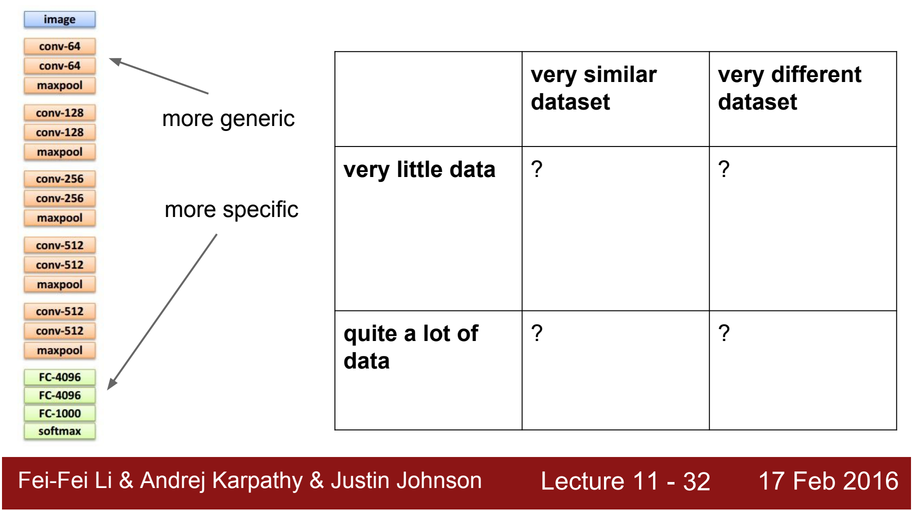


CNN Features off-the-shelf: an Astounding Baseline for Recognition [Razavian et al, 2014]

DeCAF: A Deep Convolutional Activation Feature for Generic Visual Recognition [Donahue, Jia*, et al., 2013]*

	DeCAF$_6$	DeCAF$_7$
LogReg	**40.94 ± 0.3**	40.84 ± 0.3
SVM	39.36 ± 0.3	40.66 ± 0.3
Xiao et al. (2010)	38.0	

Fei-Fei Li & Andrej Karpathy & Justin Johnson   Lecture 11 - 31   17 Feb 2016



	very similar dataset	**very different dataset**
very little data	?	?
quite a lot of data	?	?

Fei-Fei Li & Andrej Karpathy & Justin Johnson Lecture 11 - 32 17 Feb 2016

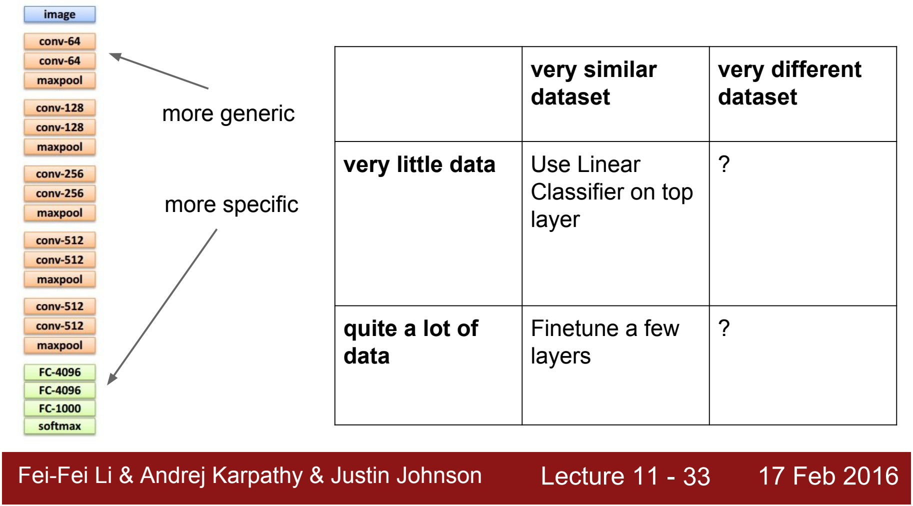


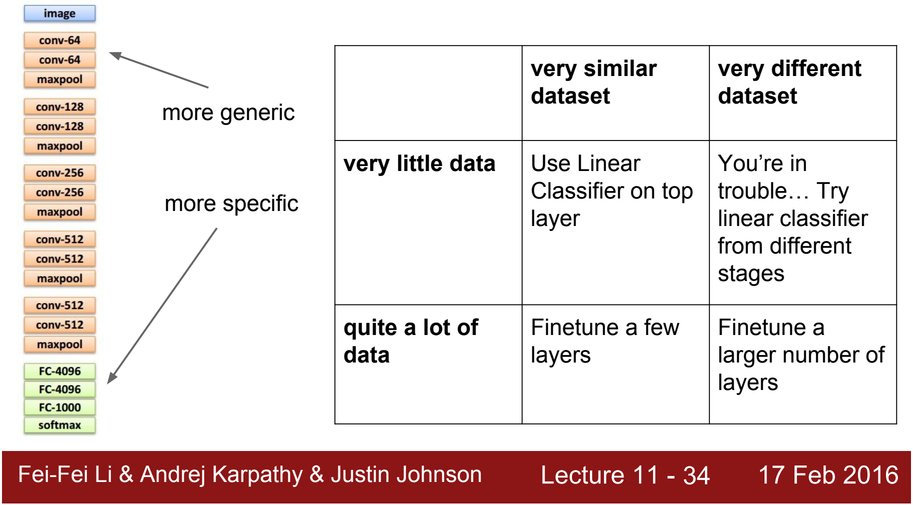
image
conv-64
conv-64
maxpool
conv-128
conv-128
maxpool
conv-256
conv-256
maxpool
conv-512
conv-512
maxpool
conv-512
conv-512
maxpool
FC-4096
FC-4096
FC-1000
softmax

more generic

more specific

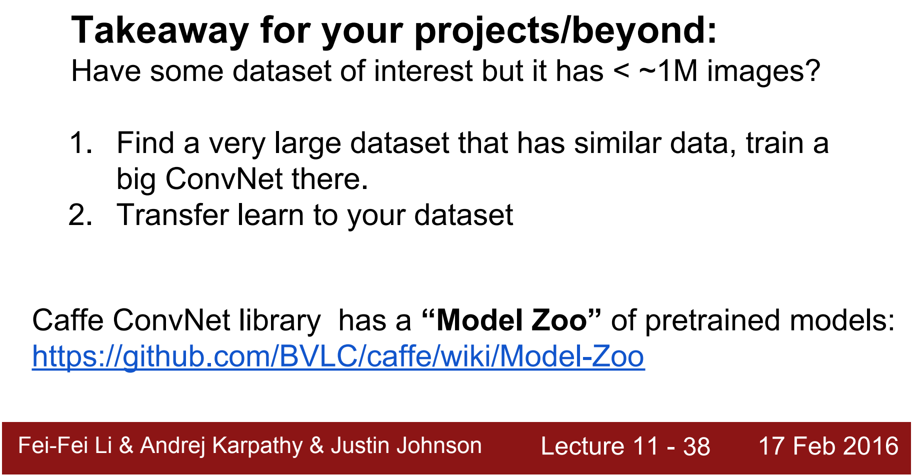
	very similar dataset	**very different dataset**
very little data	Use Linear Classifier on top layer	?
quite a lot of data	Finetune a few layers	?

Fei-Fei Li & Andrej Karpathy & Justin Johnson   Lecture 11 - 33   17 Feb 2016



	very similar dataset	**very different dataset**
very little data	Use Linear Classifier on top layer	You're in trouble… Try linear classifier from different stages
quite a lot of data	Finetune a few layers	Finetune a larger number of layers

Fei-Fei Li & Andrej Karpathy & Justin Johnson Lecture 11 - 34 17 Feb 2016



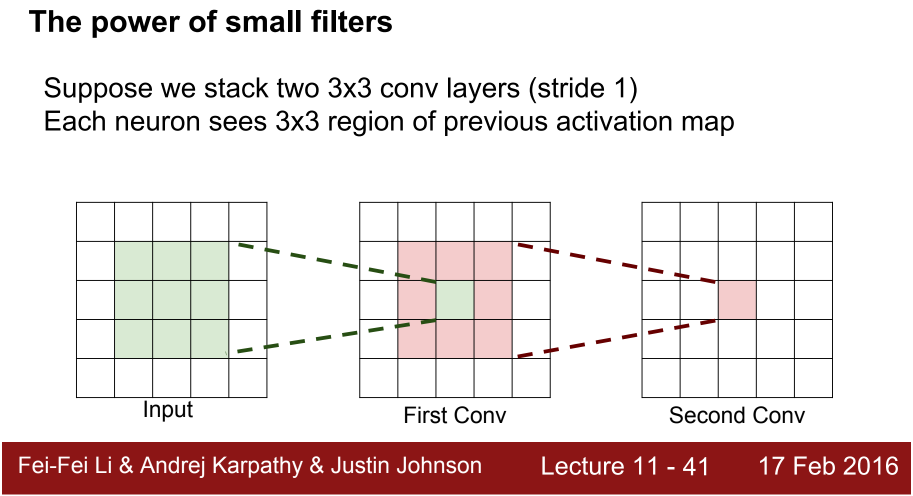
Takeaway for your projects/beyond:

Have some dataset of interest but it has < ~1M images?

1. Find a very large dataset that has similar data, train a big ConvNet there.
2. Transfer learn to your dataset

Caffe ConvNet library has a “**Model Zoo**” of pretrained models:
https://github.com/BVLC/caffe/wiki/Model-Zoo

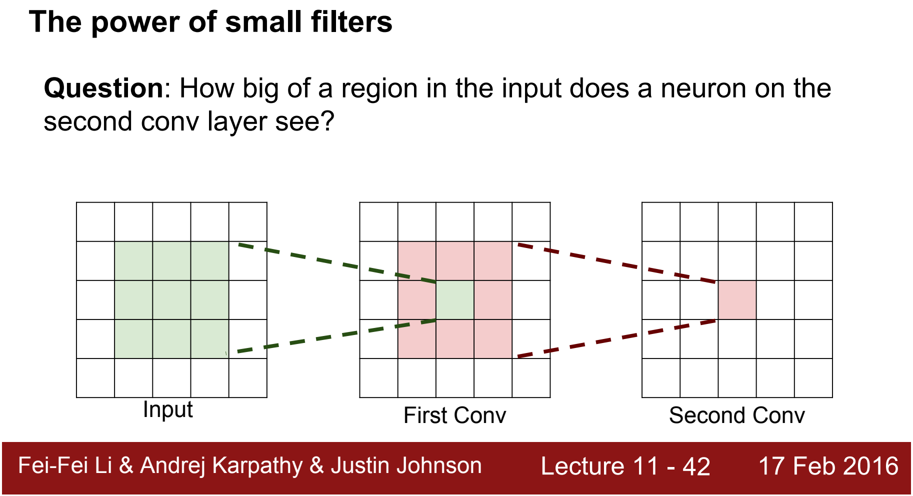
Fei-Fei Li & Andrej Karpathy & Justin Johnson Lecture 11 - 38 17 Feb 2016



The power of small filters

Suppose we stack two 3x3 conv layers (stride 1)
Each neuron sees 3x3 region of previous activation map

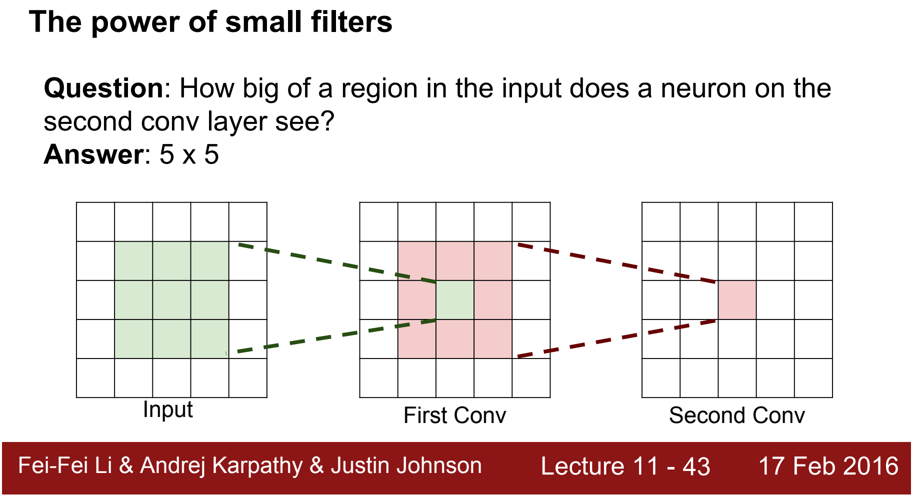
Fei-Fei Li & Andrej Karpathy & Justin Johnson Lecture 11 - 41 17 Feb 2016



The power of small filters

Question: How big of a region in the input does a neuron on the second conv layer see?

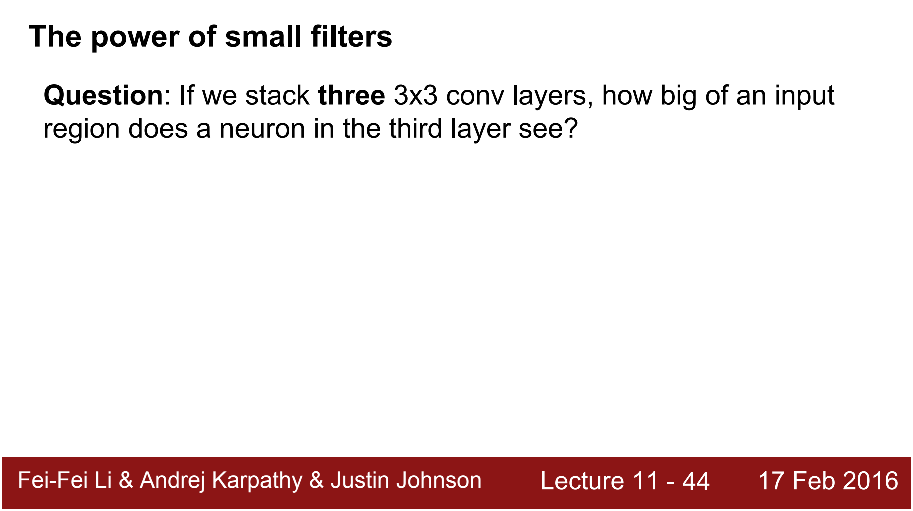
Fei-Fei Li & Andrej Karpathy & Justin Johnson Lecture 11 - 42 17 Feb 2016



The power of small filters

Question: How big of a region in the input does a neuron on the second conv layer see?
Answer: 5 x 5

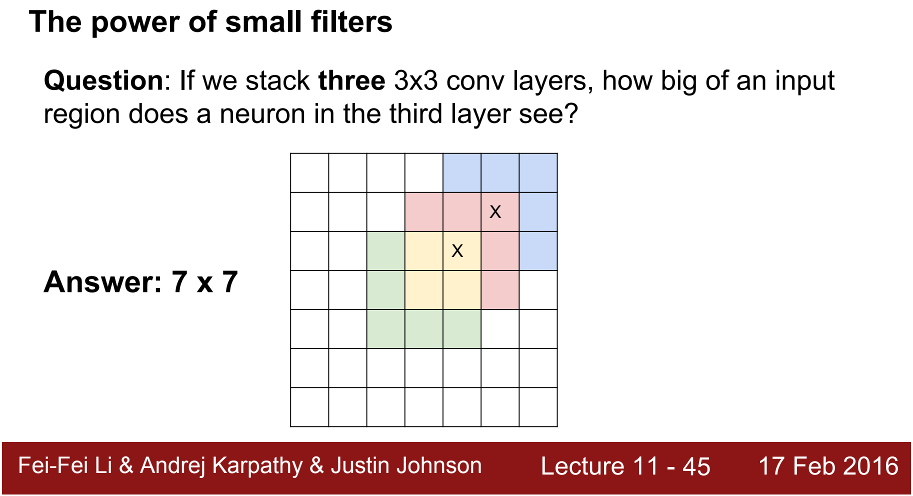
Fei-Fei Li & Andrej Karpathy & Justin Johnson Lecture 11 - 43 17 Feb 2016



The power of small filters

Question: If we stack **three** 3x3 conv layers, how big of an input region does a neuron in the third layer see?

Fei-Fei Li & Andrej Karpathy & Justin Johnson Lecture 11 - 44 17 Feb 2016

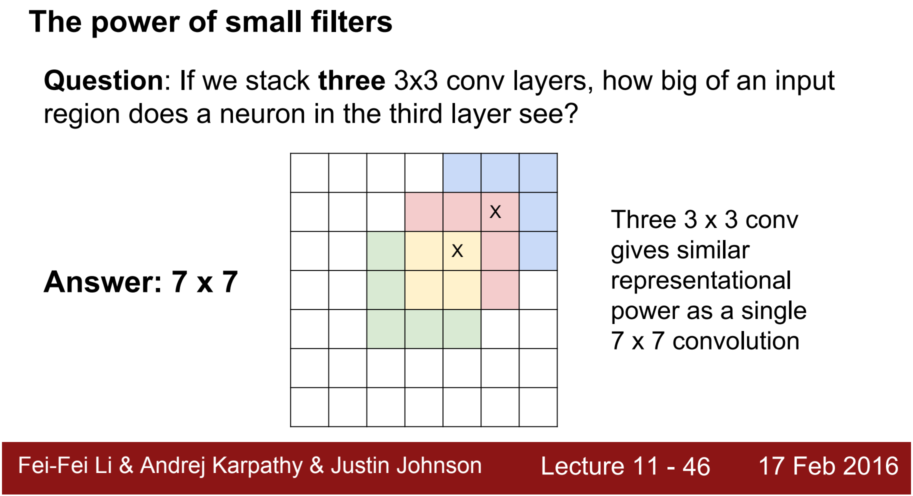


The power of small filters

Question: If we stack **three** 3x3 conv layers, how big of an input region does a neuron in the third layer see?

Answer: 7 x 7

Fei-Fei Li & Andrej Karpathy & Justin Johnson Lecture 11 - 45 17 Feb 2016



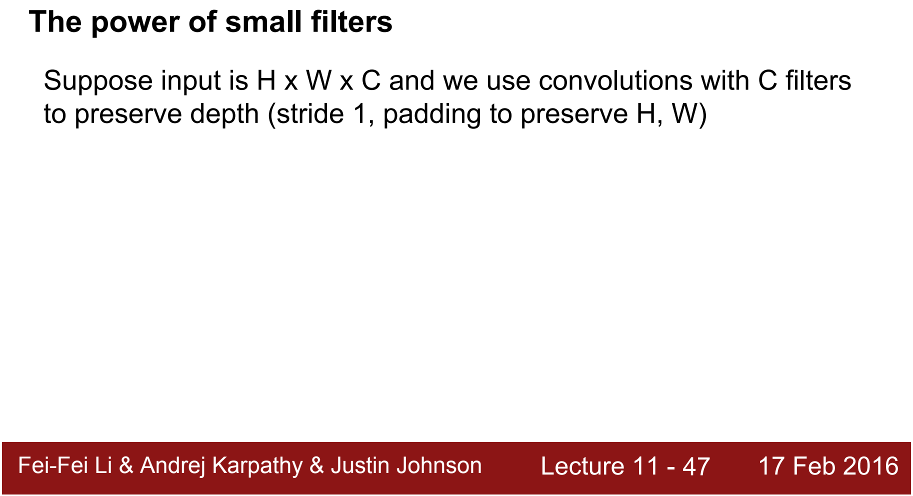
The power of small filters

Question: If we stack **three** 3x3 conv layers, how big of an input region does a neuron in the third layer see?

Answer: 7 x 7

Three 3 x 3 conv gives similar representational power as a single 7 x 7 convolution

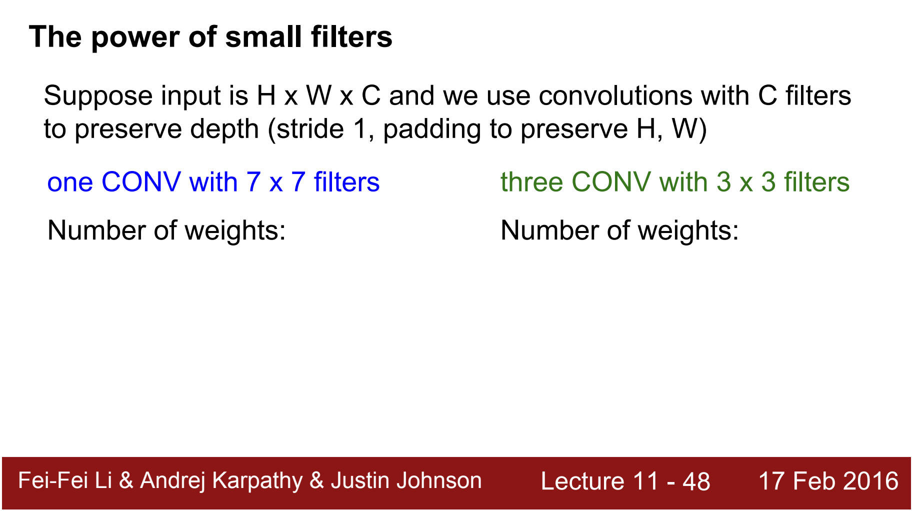
Fei-Fei Li & Andrej Karpathy & Justin Johnson Lecture 11 - 46 17 Feb 2016



The power of small filters

Suppose input is H x W x C and we use convolutions with C filters to preserve depth (stride 1, padding to preserve H, W)

Fei-Fei Li & Andrej Karpathy & Justin Johnson Lecture 11 - 47 17 Feb 2016

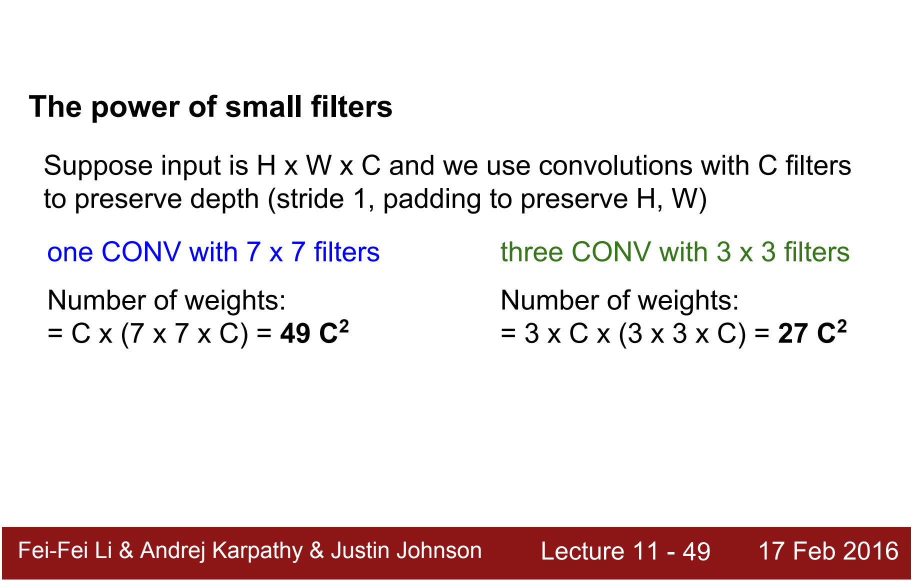


The power of small filters

Suppose input is H x W x C and we use convolutions with C filters to preserve depth (stride 1, padding to preserve H, W)

one CONV with 7 x 7 filters	three CONV with 3 x 3 filters
Number of weights:	Number of weights:

Fei-Fei Li & Andrej Karpathy & Justin Johnson Lecture 11 - 48 17 Feb 2016



The power of small filters

Suppose input is H x W x C and we use convolutions with C filters to preserve depth (stride 1, padding to preserve H, W)

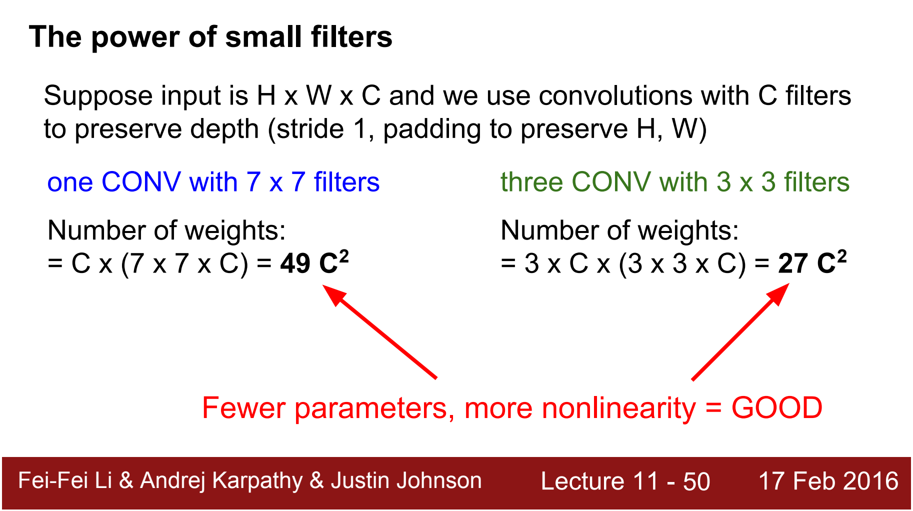
one CONV with 7 x 7 filters

Number of weights:
= C x (7 x 7 x C) = $\mathbf{49\ C^2}$

three CONV with 3 x 3 filters

Number of weights:
= 3 x C x (3 x 3 x C) = $\mathbf{27\ C^2}$

Fei-Fei Li & Andrej Karpathy & Justin Johnson Lecture 11 - 49 17 Feb 2016



The power of small filters

Suppose input is H x W x C and we use convolutions with C filters to preserve depth (stride 1, padding to preserve H, W)

one CONV with 7 x 7 filters

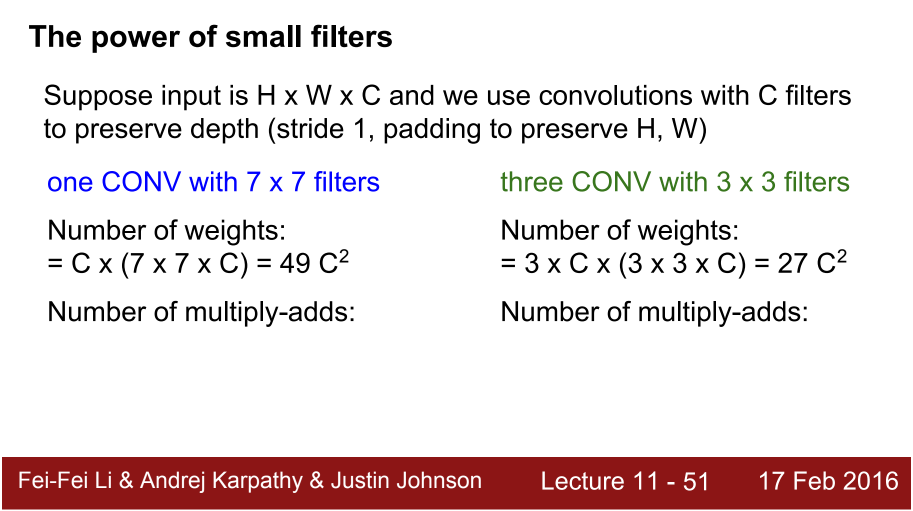
Number of weights:
= C x (7 x 7 x C) = **49 C^2**

three CONV with 3 x 3 filters

Number of weights:
= 3 x C x (3 x 3 x C) = **27 C^2**

Fewer parameters, more nonlinearity = GOOD

Fei-Fei Li & Andrej Karpathy & Justin Johnson Lecture 11 - 50 17 Feb 2016



The power of small filters

Suppose input is H x W x C and we use convolutions with C filters to preserve depth (stride 1, padding to preserve H, W)

one CONV with 7 x 7 filters

Number of weights:
= C x (7 x 7 x C) = 49 C^2

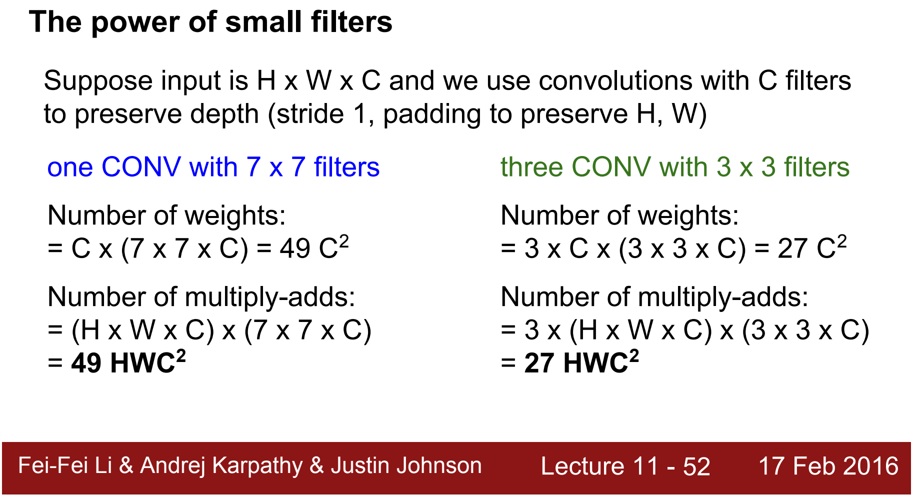
Number of multiply-adds:

three CONV with 3 x 3 filters

Number of weights:
= 3 x C x (3 x 3 x C) = 27 C^2

Number of multiply-adds:

Fei-Fei Li & Andrej Karpathy & Justin Johnson Lecture 11 - 51 17 Feb 2016



The power of small filters

Suppose input is H x W x C and we use convolutions with C filters to preserve depth (stride 1, padding to preserve H, W)

one CONV with 7 x 7 filters

Number of weights:
= C x (7 x 7 x C) = 49 C^2

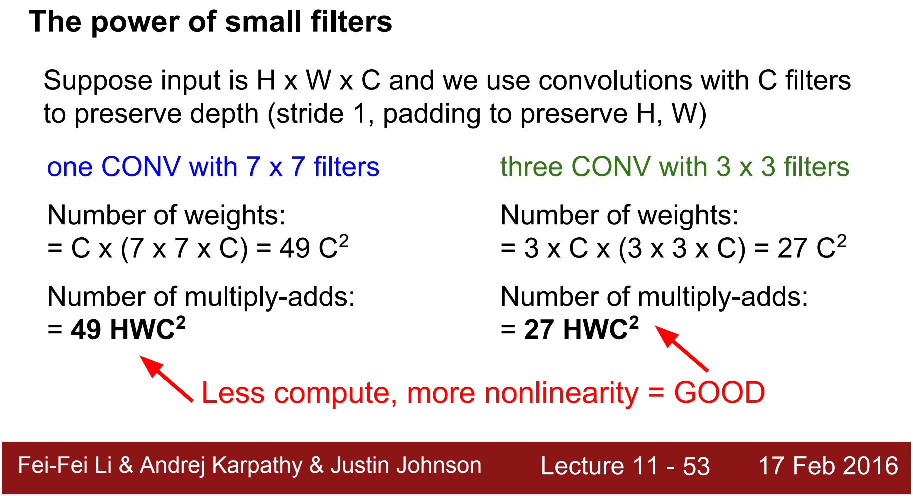
Number of multiply-adds:
= (H x W x C) x (7 x 7 x C)
= **49 HWC^2**

three CONV with 3 x 3 filters

Number of weights:
= 3 x C x (3 x 3 x C) = 27 C^2

Number of multiply-adds:
= 3 x (H x W x C) x (3 x 3 x C)
= **27 HWC^2**

Fei-Fei Li & Andrej Karpathy & Justin Johnson Lecture 11 - 52 17 Feb 2016



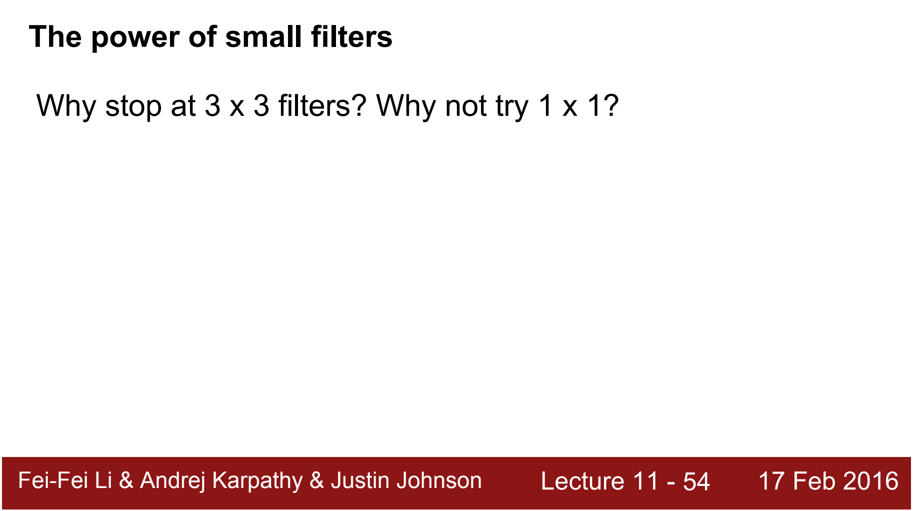
The power of small filters

Suppose input is H x W x C and we use convolutions with C filters to preserve depth (stride 1, padding to preserve H, W)

one CONV with 7 x 7 filters	three CONV with 3 x 3 filters
Number of weights: = C x (7 x 7 x C) = 49 C^2	Number of weights: = 3 x C x (3 x 3 x C) = 27 C^2
Number of multiply-adds: = **49 HWC^2**	Number of multiply-adds: = **27 HWC^2**

Less compute, more nonlinearity = GOOD

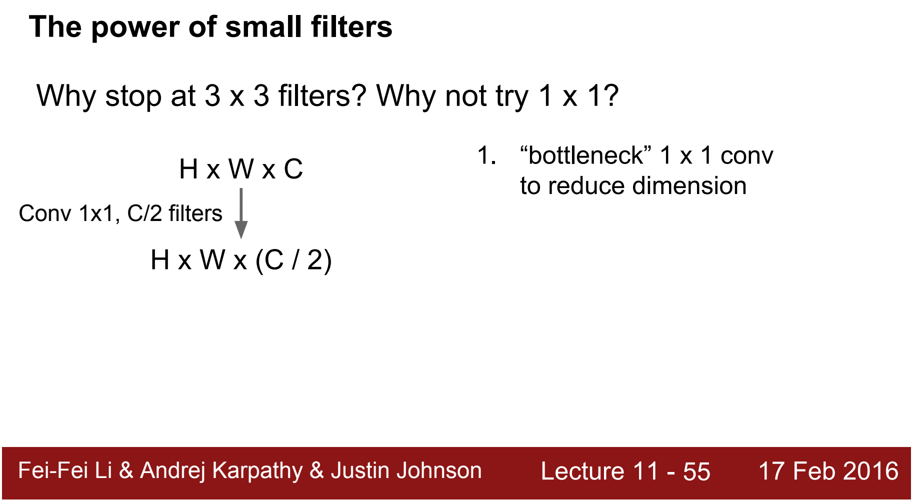
Fei-Fei Li & Andrej Karpathy & Justin Johnson Lecture 11 - 53 17 Feb 2016



The power of small filters

Why stop at 3 x 3 filters? Why not try 1 x 1?

Fei-Fei Li & Andrej Karpathy & Justin Johnson Lecture 11 - 54 17 Feb 2016



The power of small filters

Why stop at 3 x 3 filters? Why not try 1 x 1?

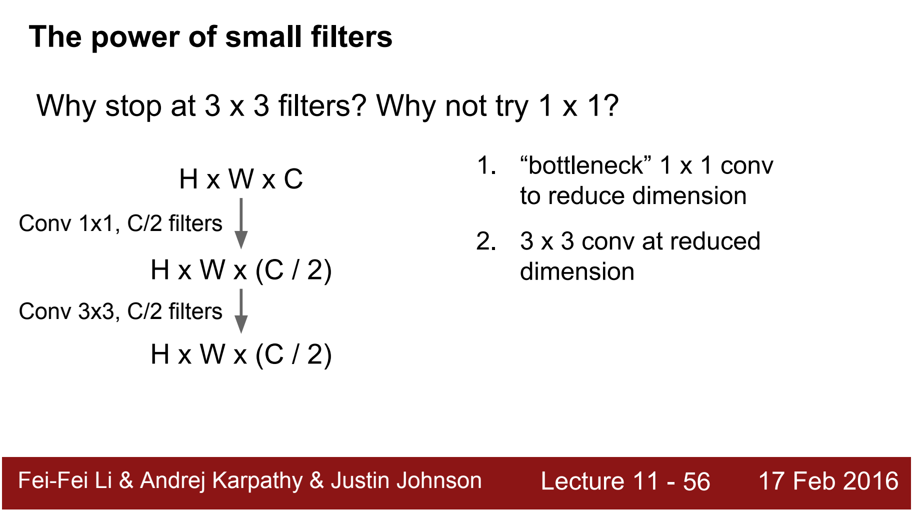
H x W x C

Conv 1x1, C/2 filters

H x W x (C / 2)

1. “bottleneck” 1 x 1 conv to reduce dimension

Fei-Fei Li & Andrej Karpathy & Justin Johnson Lecture 11 - 55 17 Feb 2016



The power of small filters

Why stop at 3 x 3 filters? Why not try 1 x 1?

H x W x C

Conv 1x1, C/2 filters

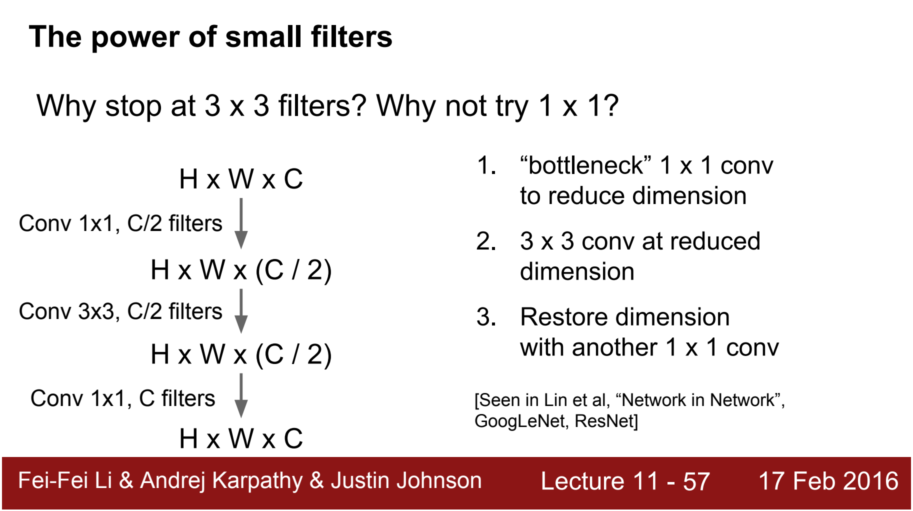
H x W x (C / 2)

Conv 3x3, C/2 filters

H x W x (C / 2)

1. “bottleneck” 1 x 1 conv to reduce dimension
2. 3 x 3 conv at reduced dimension

Fei-Fei Li & Andrej Karpathy & Justin Johnson Lecture 11 - 56 17 Feb 2016



The power of small filters

Why stop at 3 x 3 filters? Why not try 1 x 1?

H x W x C

Conv 1x1, C/2 filters

H x W x (C / 2)

Conv 3x3, C/2 filters

H x W x (C / 2)

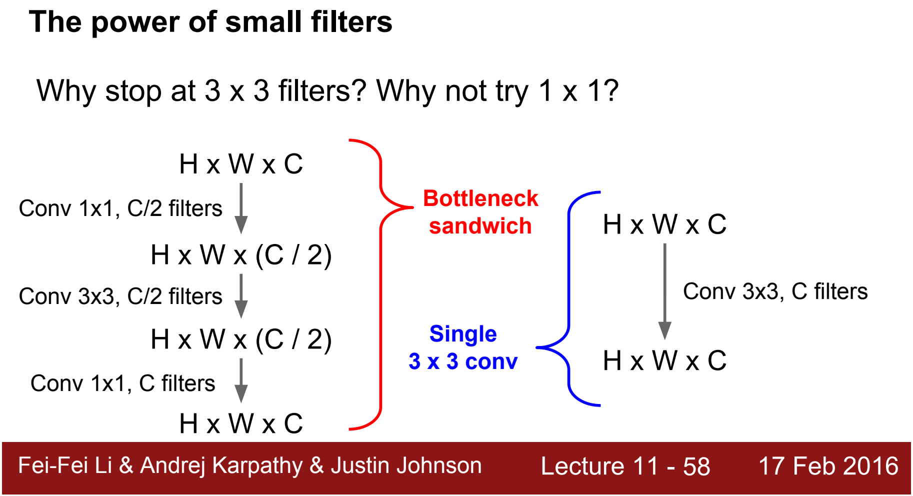
Conv 1x1, C filters

H x W x C

1. “bottleneck” 1 x 1 conv to reduce dimension
2. 3 x 3 conv at reduced dimension
3. Restore dimension with another 1 x 1 conv

[Seen in Lin et al, “Network in Network”, GoogLeNet, ResNet]

Fei-Fei Li & Andrej Karpathy & Justin Johnson Lecture 11 - 57 17 Feb 2016



The power of small filters

Why stop at 3 x 3 filters? Why not try 1 x 1?

H x W x C

Conv 1x1, C/2 filters

H x W x (C / 2)

Conv 3x3, C/2 filters

H x W x (C / 2)

Conv 1x1, C filters

H x W x C

Bottleneck sandwich

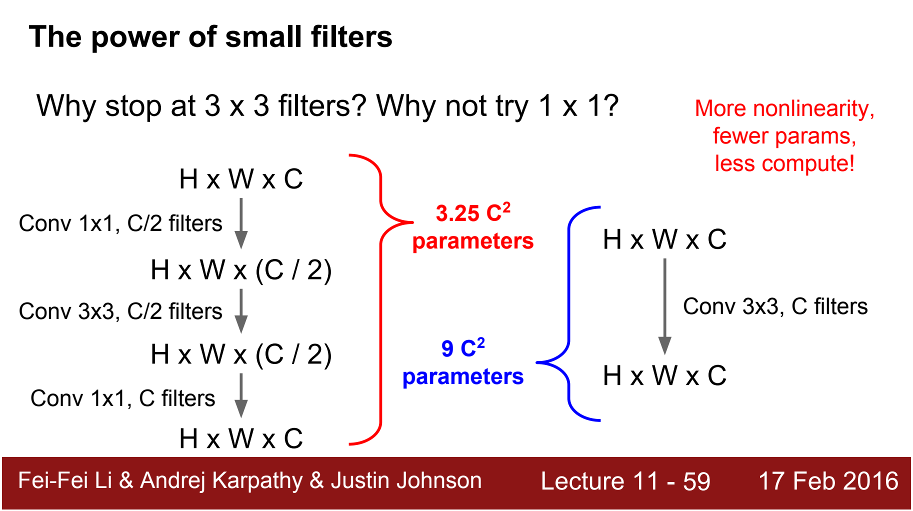
Single 3 x 3 conv

H x W x C

Conv 3x3, C filters

H x W x C

Fei-Fei Li & Andrej Karpathy & Justin Johnson Lecture 11 - 58 17 Feb 2016



The power of small filters

Why stop at 3 x 3 filters? Why not try 1 x 1?

More nonlinearity, fewer params, less compute!

H x W x C

Conv 1x1, C/2 filters

H x W x (C / 2)

Conv 3x3, C/2 filters

H x W x (C / 2)

Conv 1x1, C filters

H x W x C

3.25 C^2 parameters

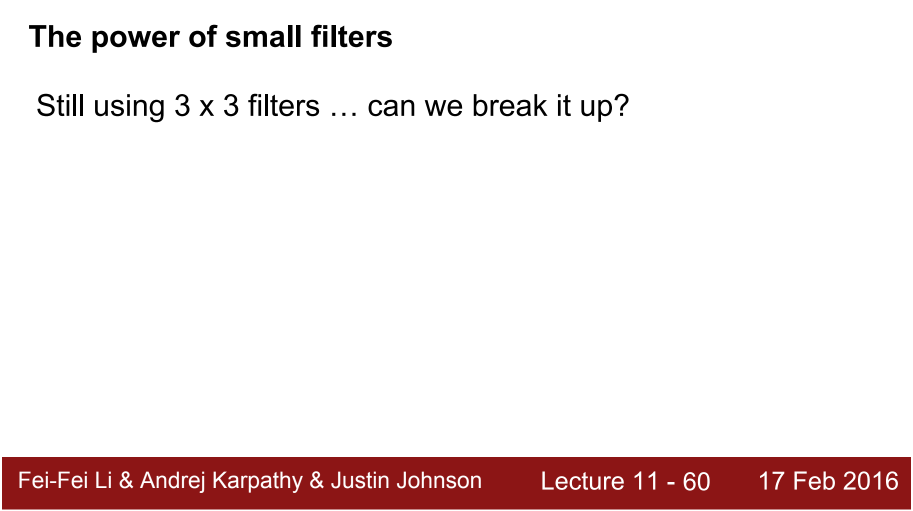
9 C^2 parameters

H x W x C

Conv 3x3, C filters

H x W x C

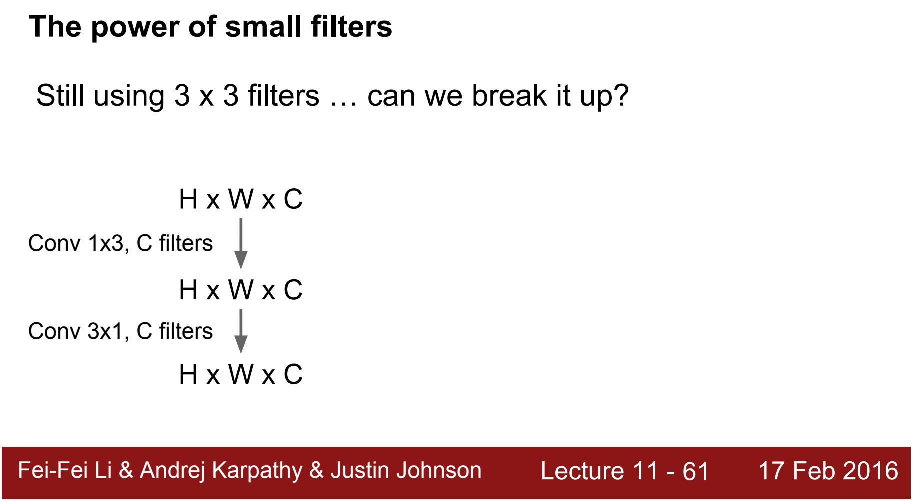
Fei-Fei Li & Andrej Karpathy & Justin Johnson   Lecture 11 - 59   17 Feb 2016



The power of small filters

Still using 3 x 3 filters … can we break it up?

Fei-Fei Li & Andrej Karpathy & Justin Johnson Lecture 11 - 60 17 Feb 2016



The power of small filters

Still using 3 x 3 filters … can we break it up?

H x W x C

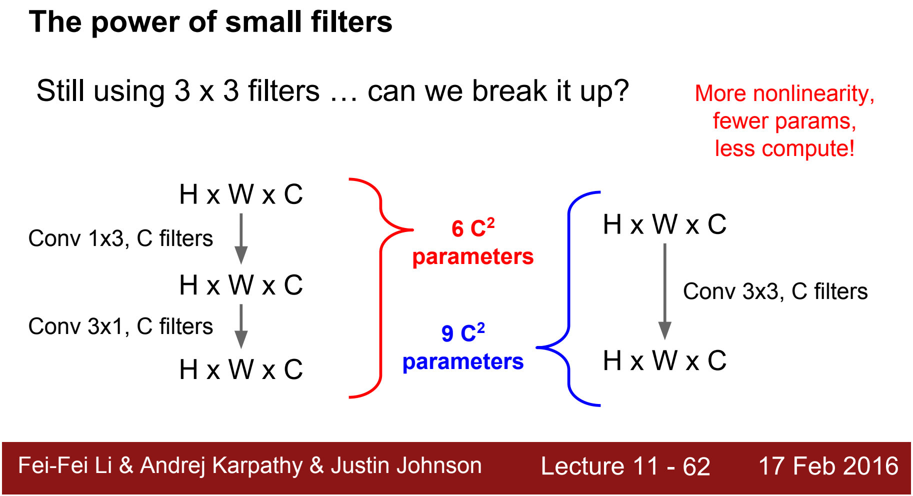
Conv 1x3, C filters

H x W x C

Conv 3x1, C filters

H x W x C

Fei-Fei Li & Andrej Karpathy & Justin Johnson Lecture 11 - 61 17 Feb 2016



The power of small filters

Still using 3 x 3 filters … can we break it up?

More nonlinearity, fewer params, less compute!

H x W x C

Conv 1x3, C filters

H x W x C

Conv 3x1, C filters

H x W x C

$6\ C^2$ parameters

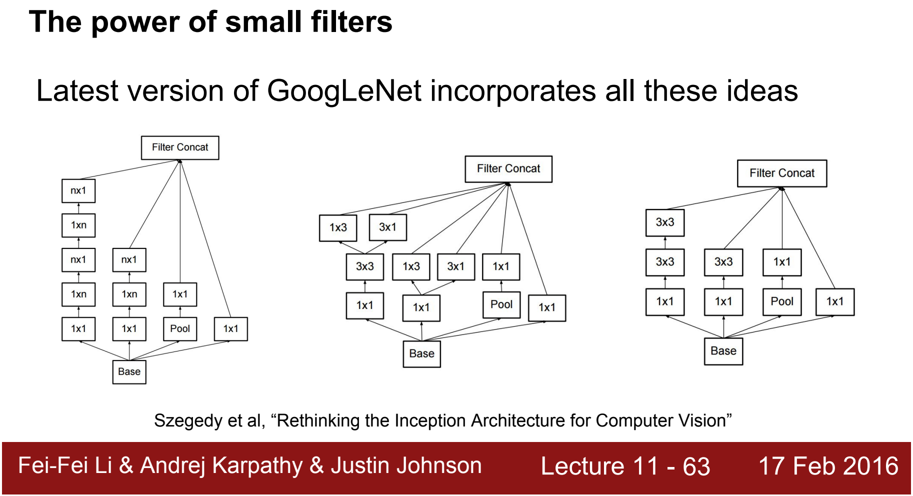
$9\ C^2$ parameters

H x W x C

Conv 3x3, C filters

H x W x C

Fei-Fei Li & Andrej Karpathy & Justin Johnson Lecture 11 - 62 17 Feb 2016

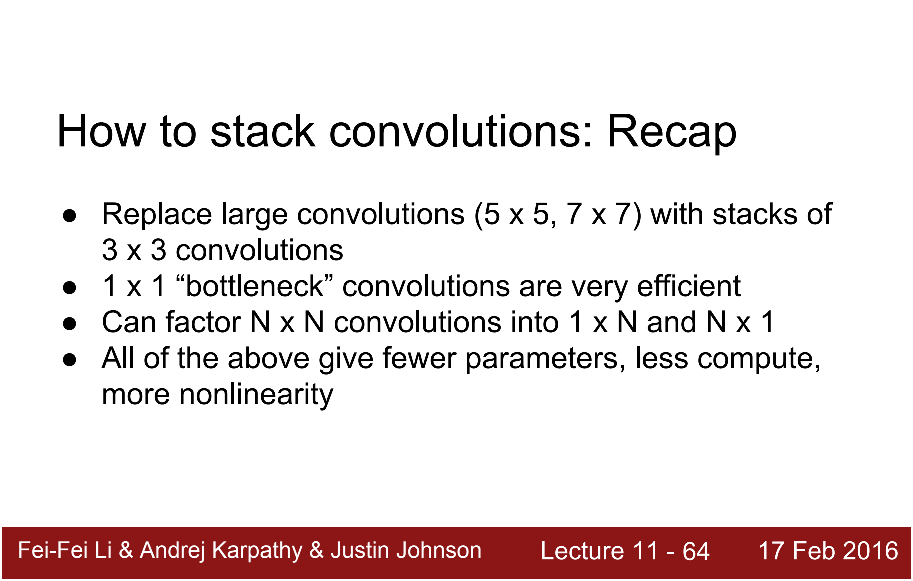


The power of small filters

Latest version of GoogLeNet incorporates all these ideas

Szegedy et al, “Rethinking the Inception Architecture for Computer Vision”

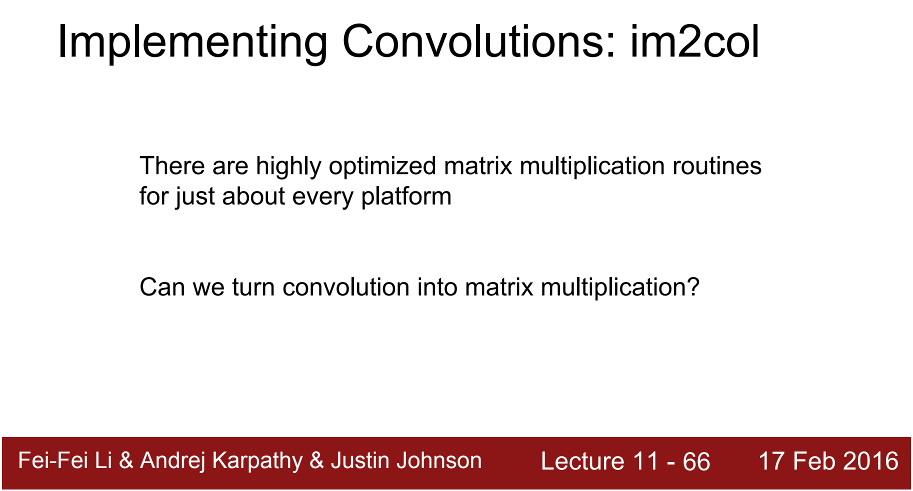
Fei-Fei Li & Andrej Karpathy & Justin Johnson Lecture 11 - 63 17 Feb 2016



How to stack convolutions: Recap

- Replace large convolutions (5 x 5, 7 x 7) with stacks of 3 x 3 convolutions
- 1 x 1 "bottleneck" convolutions are very efficient
- Can factor N x N convolutions into 1 x N and N x 1
- All of the above give fewer parameters, less compute, more nonlinearity

Fei-Fei Li & Andrej Karpathy & Justin Johnson Lecture 11 - 64 17 Feb 2016

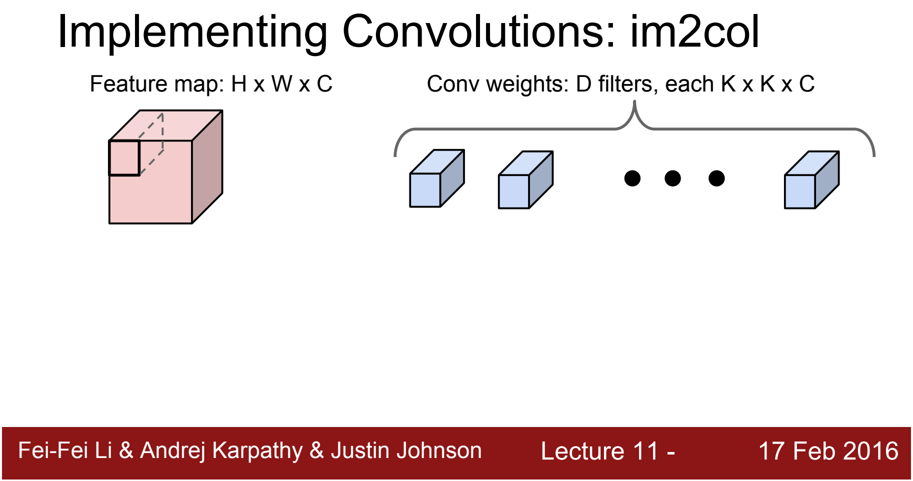


Implementing Convolutions: im2col

There are highly optimized matrix multiplication routines for just about every platform

Can we turn convolution into matrix multiplication?

Fei-Fei Li & Andrej Karpathy & Justin Johnson Lecture 11 - 66 17 Feb 2016



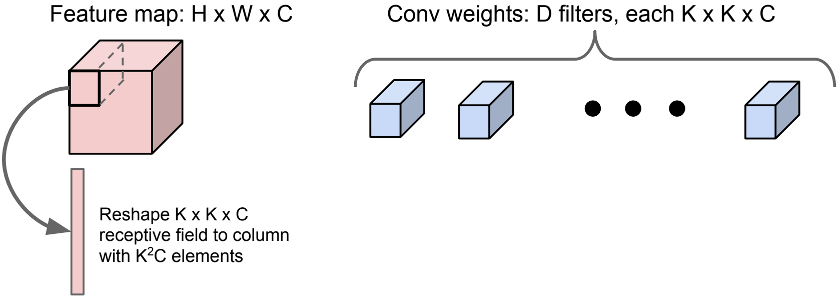
Implementing Convolutions: im2col

Feature map: H x W x C

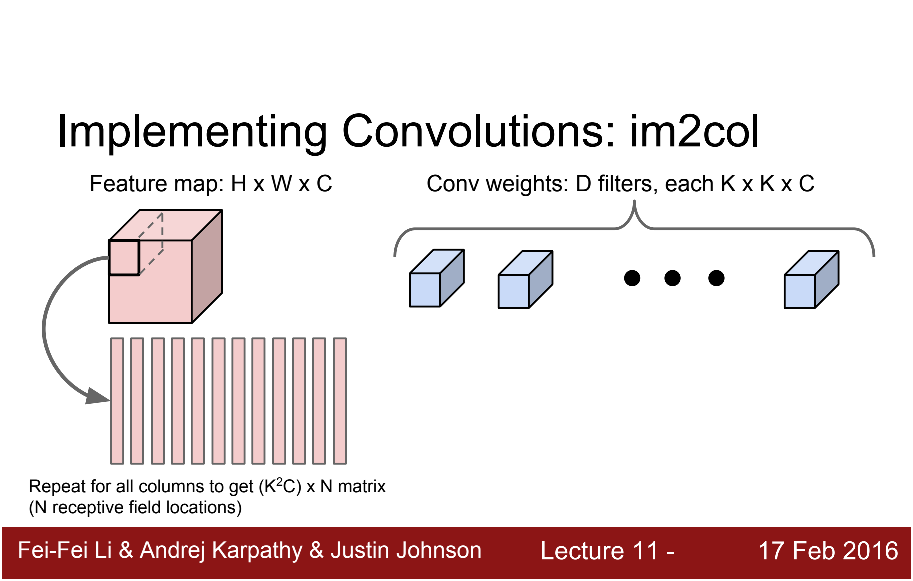
Conv weights: D filters, each K x K x C

Fei-Fei Li & Andrej Karpathy & Justin Johnson Lecture 11 - 17 Feb 2016

Implementing Convolutions: im2col


Feature map: H x W x C
Conv weights: D filters, each K x K x C
Reshape K x K x C receptive field to column with K²C elements

Fei-Fei Li & Andrej Karpathy & Justin Johnson Lecture 11 - 17 Feb 2016



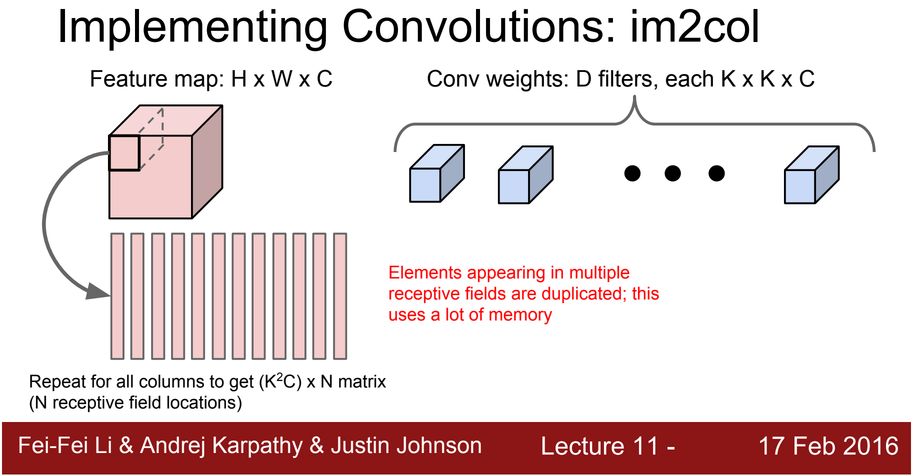
Implementing Convolutions: im2col

Feature map: H x W x C

Conv weights: D filters, each K x K x C

Repeat for all columns to get (K^2C) x N matrix
(N receptive field locations)

Fei-Fei Li & Andrej Karpathy & Justin Johnson Lecture 11 - 17 Feb 2016



Implementing Convolutions: im2col

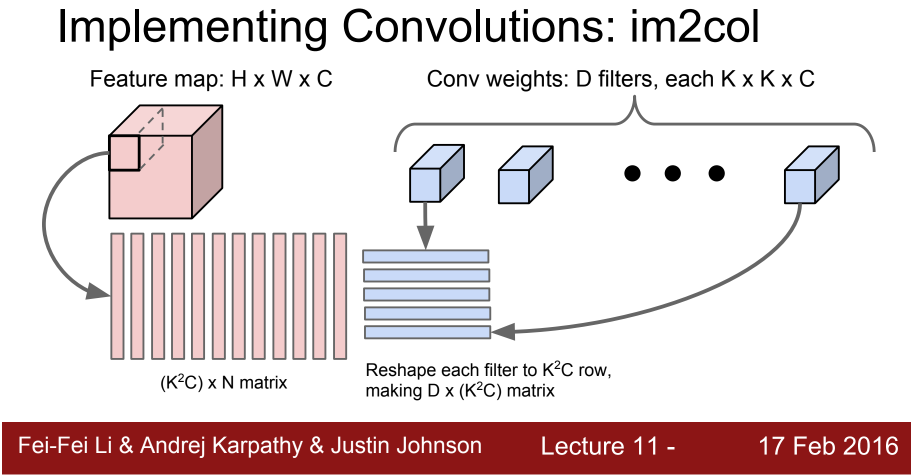
Feature map: H x W x C

Conv weights: D filters, each K x K x C

Elements appearing in multiple receptive fields are duplicated; this uses a lot of memory

Repeat for all columns to get (K^2C) x N matrix
(N receptive field locations)

Fei-Fei Li & Andrej Karpathy & Justin Johnson Lecture 11 - 17 Feb 2016



Implementing Convolutions: im2col

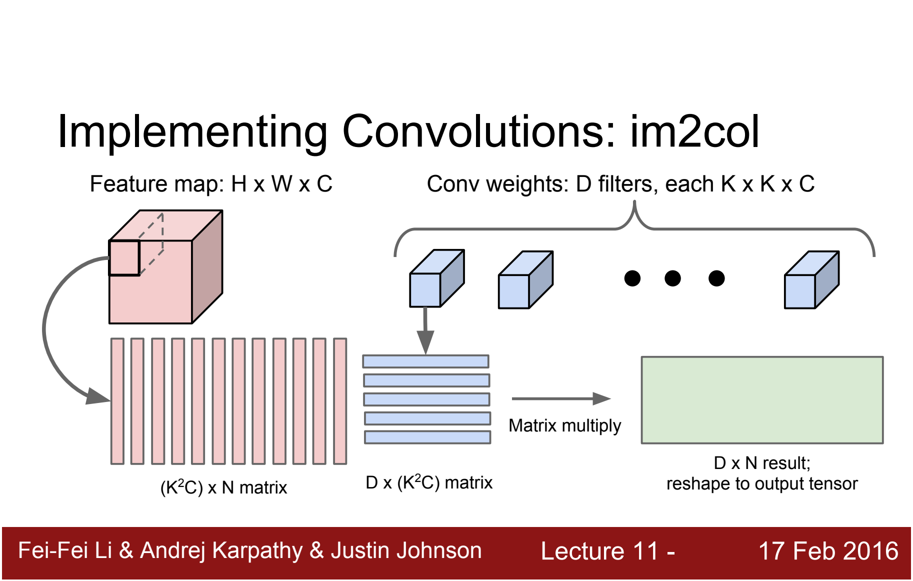
Feature map: H x W x C

Conv weights: D filters, each K x K x C

(K^{2}C) x N matrix

Reshape each filter to K^{2}C row, making D x (K^{2}C) matrix

Fei-Fei Li & Andrej Karpathy & Justin Johnson Lecture 11 - 17 Feb 2016



Implementing Convolutions: im2col

Feature map: H x W x C

Conv weights: D filters, each K x K x C

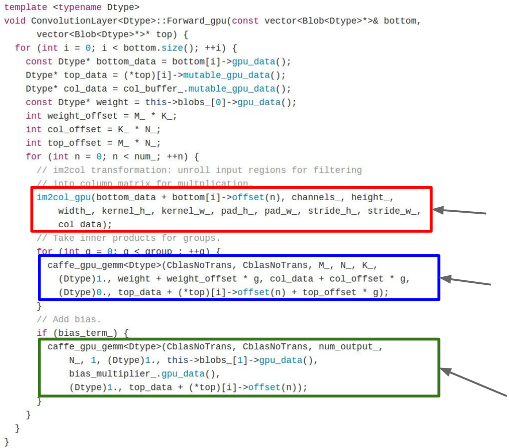
(K^2C) x N matrix

D x (K^2C) matrix

Matrix multiply

D x N result;
reshape to output tensor

Fei-Fei Li & Andrej Karpathy & Justin Johnson Lecture 11 - 17 Feb 2016



```
template <typename Dtype>
void ConvolutionLayer<Dtype>::Forward_gpu(const vector<Blob<Dtype>*>& bottom,
      vector<Blob<Dtype>*>* top) {
  for (int i = 0; i < bottom.size(); ++i) {
    const Dtype* bottom_data = bottom[i]->gpu_data();
    Dtype* top_data = (*top)[i]->mutable_gpu_data();
    Dtype* col_data = col_buffer_.mutable_gpu_data();
    const Dtype* weight = this->blobs_[0]->gpu_data();
    int weight_offset = M_ * K_;
    int col_offset = K_ * N_;
    int top_offset = M_ * N_;
    for (int n = 0; n < num_; ++n) {
      // im2col transformation: unroll input regions for filtering
      // into column matrix for multplication.
      im2col_gpu(bottom_data + bottom[i]->offset(n), channels_, height_,
          width_, kernel_h_, kernel_w_, pad_h_, pad_w_, stride_h_, stride_w_,
          col_data);
      // Take inner products for groups.
      for (int g = 0; g < group_; ++g) {
        caffe_gpu_gemm<Dtype>(CblasNoTrans, CblasNoTrans, M_, N_, K_,
          (Dtype)1., weight + weight_offset * g, col_data + col_offset * g,
          (Dtype)0., top_data + (*top)[i]->offset(n) + top_offset * g);
      }
      // Add bias.
      if (bias_term_) {
        caffe_gpu_gemm<Dtype>(CblasNoTrans, CblasNoTrans, num_output_,
            N_, 1, (Dtype)1., this->blobs_[1]->gpu_data(),
            bias_multiplier_.gpu_data(),
            (Dtype)1., top_data + (*top)[i]->offset(n));
      }
    }
  }
}
```

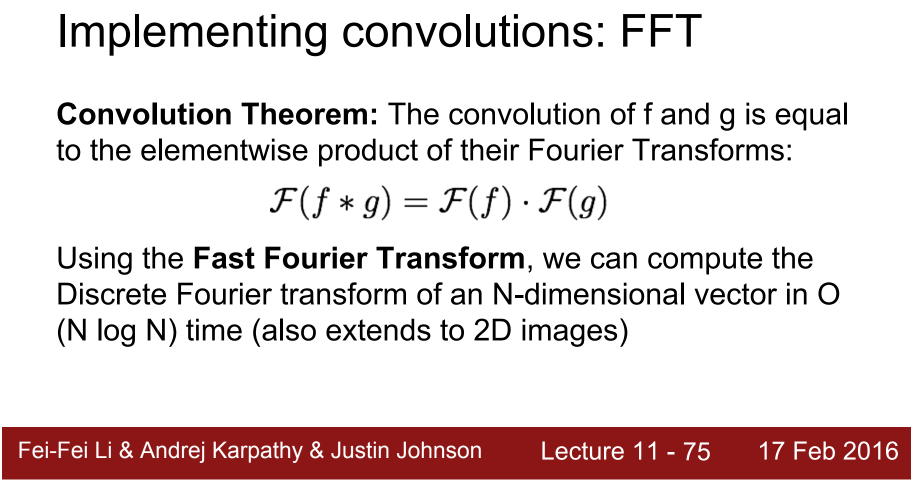
Case study:
CONV forward in Caffe library

im2col

matrix multiply: call to cuBLAS

bias offset

Fei-Fei Li & Andrej Karpathy & Justin Johnson Lecture 11 - 73 17 Feb 2016



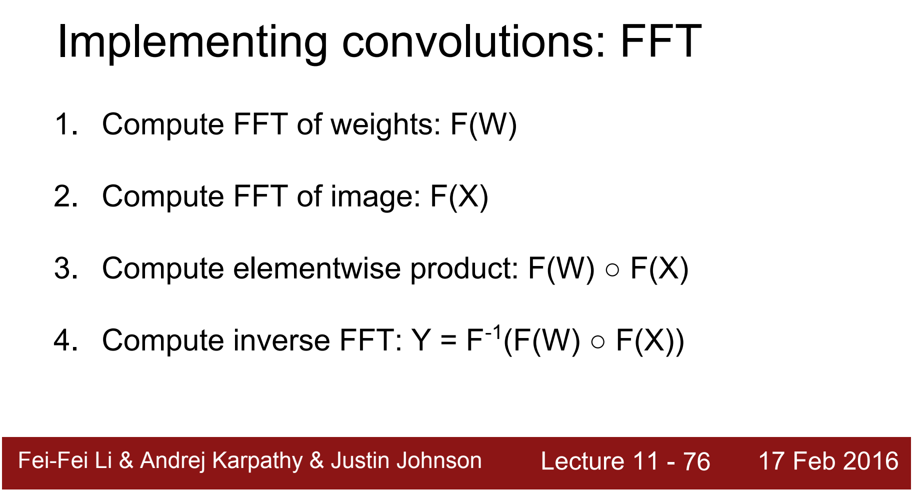
Implementing convolutions: FFT

Convolution Theorem: The convolution of f and g is equal to the elementwise product of their Fourier Transforms:

$$\mathcal{F}(f * g) = \mathcal{F}(f) \cdot \mathcal{F}(g)$$

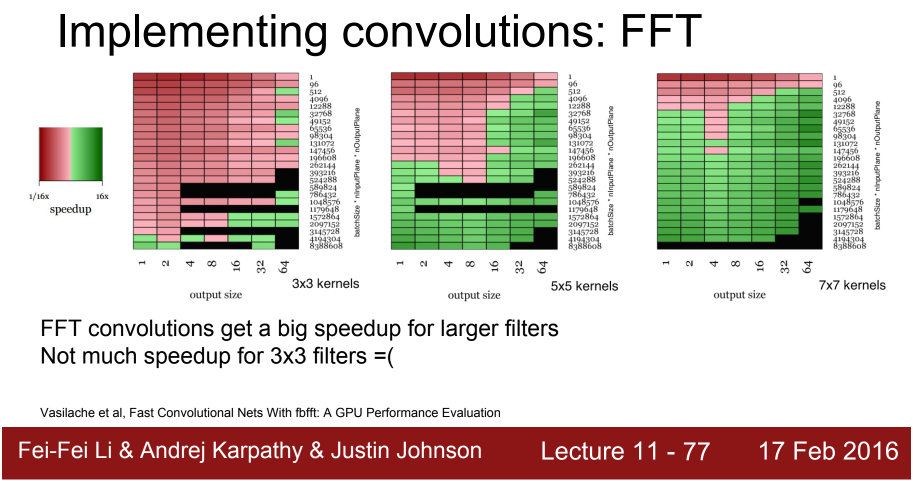
Using the **Fast Fourier Transform**, we can compute the Discrete Fourier transform of an N-dimensional vector in O (N log N) time (also extends to 2D images)

Fei-Fei Li & Andrej Karpathy & Justin Johnson Lecture 11 - 75 17 Feb 2016



Implementing convolutions: FFT

1. Compute FFT of weights: $F(W)$
2. Compute FFT of image: $F(X)$
3. Compute elementwise product: $F(W) \circ F(X)$
4. Compute inverse FFT: $Y = F^{-1}(F(W) \circ F(X))$



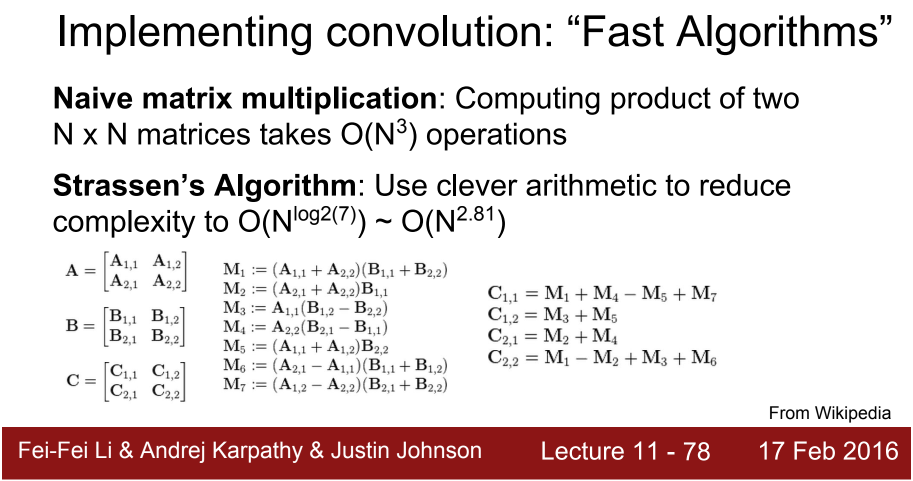
Implementing convolutions: FFT

FFT convolutions get a big speedup for larger filters

Not much speedup for 3x3 filters =(

Vasilache et al, Fast Convolutional Nets With fbfft: A GPU Performance Evaluation

Fei-Fei Li & Andrej Karpathy & Justin Johnson Lecture 11 - 77 17 Feb 2016



Implementing convolution: “Fast Algorithms”

Naive matrix multiplication: Computing product of two N x N matrices takes O(N^3) operations

Strassen’s Algorithm: Use clever arithmetic to reduce complexity to O(N$^{\log2(7)}$) ~ O(N$^{2.81}$)

$$A = \begin{bmatrix} A_{1,1} & A_{1,2} \\ A_{2,1} & A_{2,2} \end{bmatrix}$$

$$B = \begin{bmatrix} B_{1,1} & B_{1,2} \\ B_{2,1} & B_{2,2} \end{bmatrix}$$

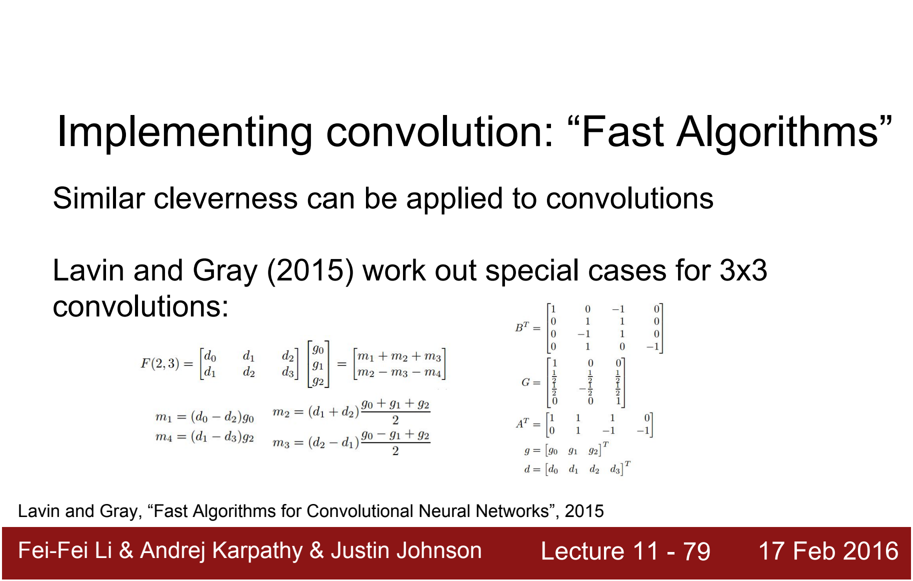
$$C = \begin{bmatrix} C_{1,1} & C_{1,2} \\ C_{2,1} & C_{2,2} \end{bmatrix}$$

$$\begin{aligned}
M_1 &:= (A_{1,1} + A_{2,2})(B_{1,1} + B_{2,2}) \\
M_2 &:= (A_{2,1} + A_{2,2})B_{1,1} \\
M_3 &:= A_{1,1}(B_{1,2} - B_{2,2}) \\
M_4 &:= A_{2,2}(B_{2,1} - B_{1,1}) \\
M_5 &:= (A_{1,1} + A_{1,2})B_{2,2} \\
M_6 &:= (A_{2,1} - A_{1,1})(B_{1,1} + B_{1,2}) \\
M_7 &:= (A_{1,2} - A_{2,2})(B_{2,1} + B_{2,2})
\end{aligned}$$

$$\begin{aligned}
C_{1,1} &= M_1 + M_4 - M_5 + M_7 \\
C_{1,2} &= M_3 + M_5 \\
C_{2,1} &= M_2 + M_4 \\
C_{2,2} &= M_1 - M_2 + M_3 + M_6
\end{aligned}$$

From Wikipedia

Fei-Fei Li & Andrej Karpathy & Justin Johnson Lecture 11 - 78 17 Feb 2016



Implementing convolution: “Fast Algorithms”

Similar cleverness can be applied to convolutions

Lavin and Gray (2015) work out special cases for 3x3 convolutions:

$$F(2,3) = \begin{bmatrix} d_0 & d_1 & d_2 \\ d_1 & d_2 & d_3 \end{bmatrix} \begin{bmatrix} g_0 \\ g_1 \\ g_2 \end{bmatrix} = \begin{bmatrix} m_1 + m_2 + m_3 \\ m_2 - m_3 - m_4 \end{bmatrix}$$

$$m_1 = (d_0 - d_2)g_0 \qquad m_2 = (d_1 + d_2)\frac{g_0 + g_1 + g_2}{2}$$
$$m_4 = (d_1 - d_3)g_2 \qquad m_3 = (d_2 - d_1)\frac{g_0 - g_1 + g_2}{2}$$

$$B^T = \begin{bmatrix} 1 & 0 & -1 & 0 \\ 0 & 1 & 1 & 0 \\ 0 & -1 & 1 & 0 \\ 0 & 1 & 0 & -1 \end{bmatrix}$$

$$G = \begin{bmatrix} 1 & 0 & 0 \\ \frac{1}{2} & \frac{1}{2} & \frac{1}{2} \\ \frac{1}{2} & -\frac{1}{2} & \frac{1}{2} \\ 0 & 0 & 1 \end{bmatrix}$$

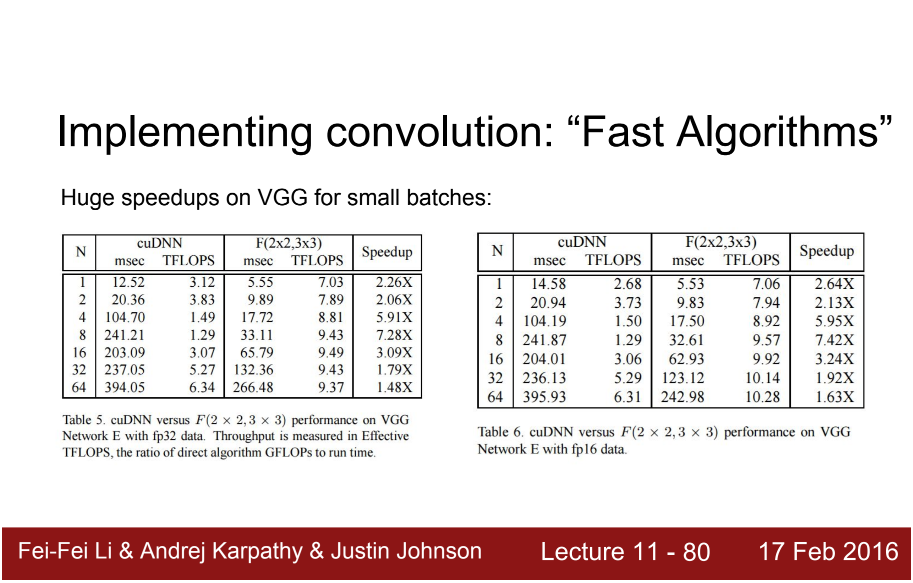
$$A^T = \begin{bmatrix} 1 & 1 & 1 & 0 \\ 0 & 1 & -1 & -1 \end{bmatrix}$$

$$g = \begin{bmatrix} g_0 & g_1 & g_2 \end{bmatrix}^T$$

$$d = \begin{bmatrix} d_0 & d_1 & d_2 & d_3 \end{bmatrix}^T$$

Lavin and Gray, “Fast Algorithms for Convolutional Neural Networks”, 2015

Fei-Fei Li & Andrej Karpathy & Justin Johnson Lecture 11 - 79 17 Feb 2016



Implementing convolution: “Fast Algorithms”

Huge speedups on VGG for small batches:

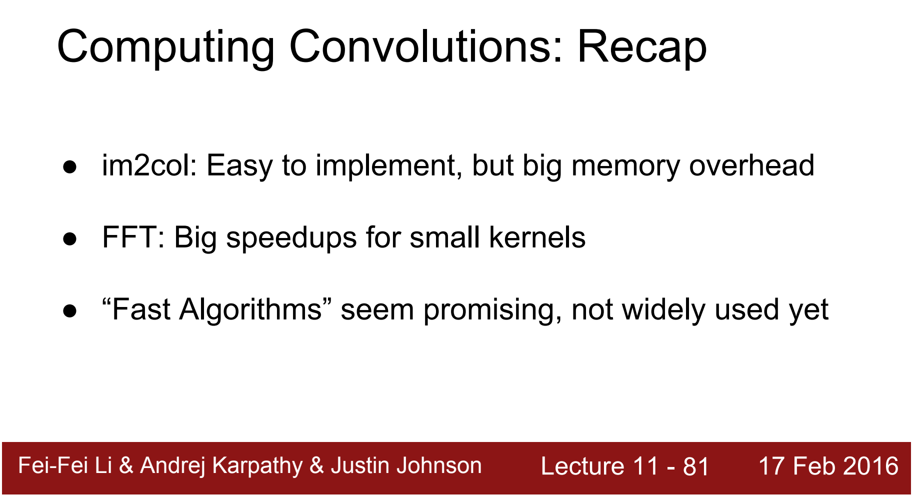
N	cuDNN		F(2x2,3x3)		Speedup
	msec	TFLOPS	msec	TFLOPS	
1	12.52	3.12	5.55	7.03	2.26X
2	20.36	3.83	9.89	7.89	2.06X
4	104.70	1.49	17.72	8.81	5.91X
8	241.21	1.29	33.11	9.43	7.28X
16	203.09	3.07	65.79	9.49	3.09X
32	237.05	5.27	132.36	9.43	1.79X
64	394.05	6.34	266.48	9.37	1.48X

Table 5. cuDNN versus $F(2 \times 2, 3 \times 3)$ performance on VGG Network E with fp32 data. Throughput is measured in Effective TFLOPS, the ratio of direct algorithm GFLOPs to run time.

N	cuDNN		F(2x2,3x3)		Speedup
	msec	TFLOPS	msec	TFLOPS	
1	14.58	2.68	5.53	7.06	2.64X
2	20.94	3.73	9.83	7.94	2.13X
4	104.19	1.50	17.50	8.92	5.95X
8	241.87	1.29	32.61	9.57	7.42X
16	204.01	3.06	62.93	9.92	3.24X
32	236.13	5.29	123.12	10.14	1.92X
64	395.93	6.31	242.98	10.28	1.63X

Table 6. cuDNN versus $F(2 \times 2, 3 \times 3)$ performance on VGG Network E with fp16 data.

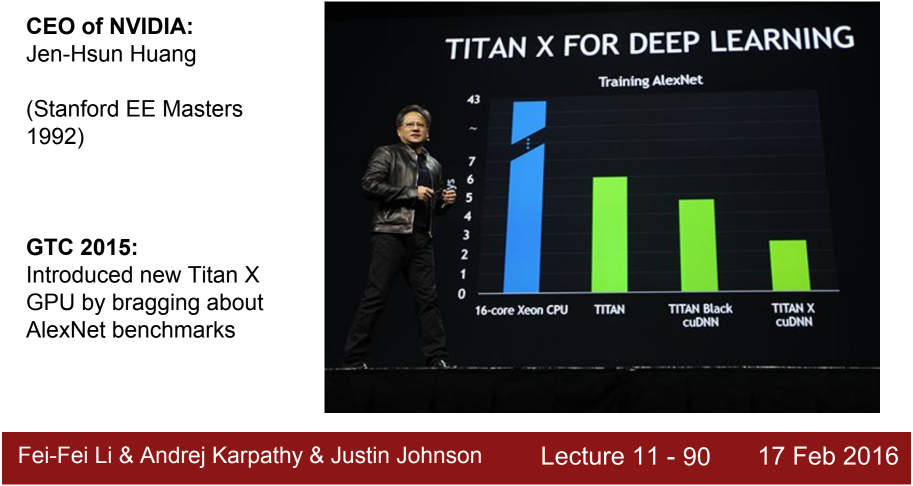
Fei-Fei Li & Andrej Karpathy & Justin Johnson Lecture 11 - 80 17 Feb 2016



Computing Convolutions: Recap

- im2col: Easy to implement, but big memory overhead
- FFT: Big speedups for small kernels
- “Fast Algorithms” seem promising, not widely used yet

Fei-Fei Li & Andrej Karpathy & Justin Johnson Lecture 11 - 81 17 Feb 2016

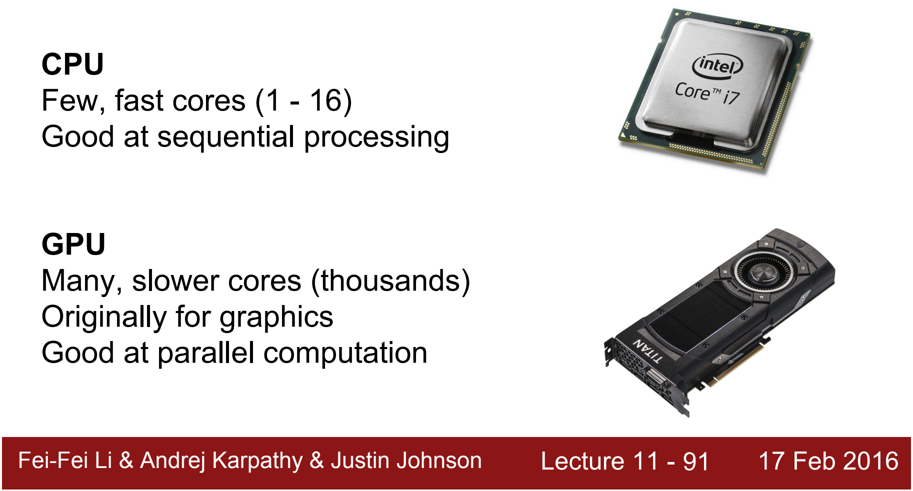


CEO of NVIDIA:
Jen-Hsun Huang

(Stanford EE Masters 1992)

GTC 2015:
Introduced new Titan X GPU by bragging about AlexNet benchmarks

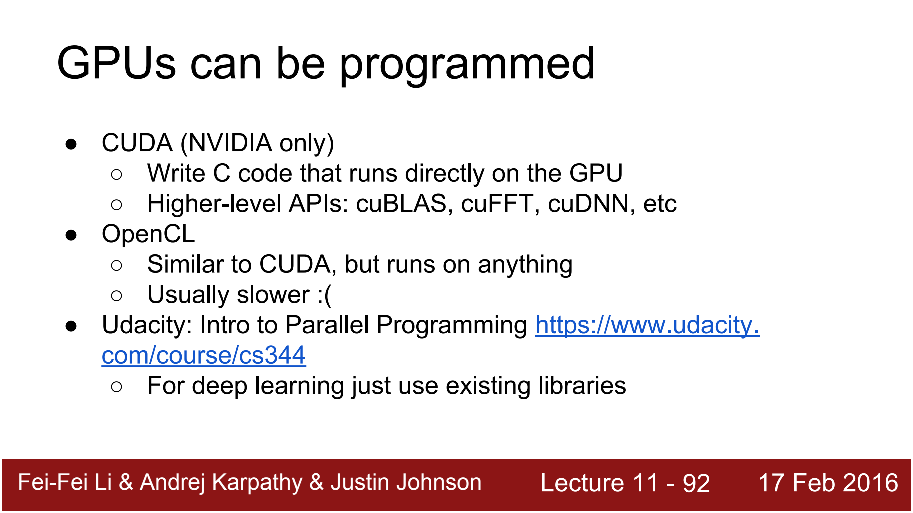
Fei-Fei Li & Andrej Karpathy & Justin Johnson Lecture 11 - 90 17 Feb 2016



CPU
Few, fast cores (1 - 16)
Good at sequential processing

GPU
Many, slower cores (thousands)
Originally for graphics
Good at parallel computation

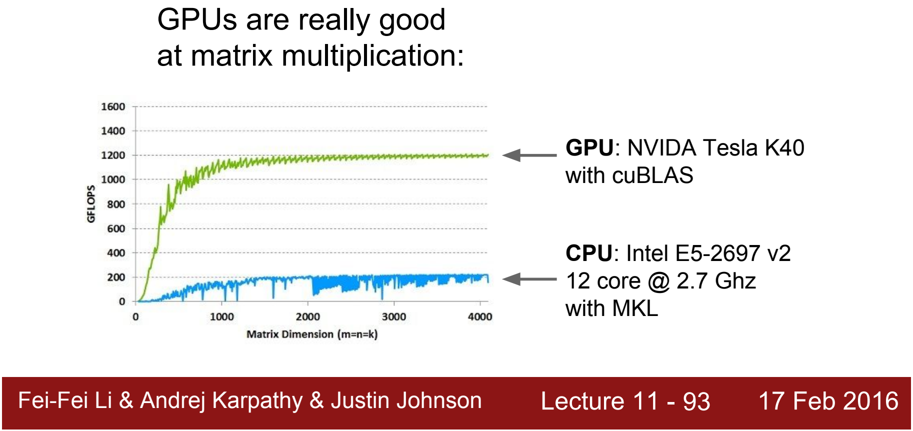
Fei-Fei Li & Andrej Karpathy & Justin Johnson Lecture 11 - 91 17 Feb 2016



GPUs can be programmed

- CUDA (NVIDIA only)
  - ○ Write C code that runs directly on the GPU
  - ○ Higher-level APIs: cuBLAS, cuFFT, cuDNN, etc
- OpenCL
  - ○ Similar to CUDA, but runs on anything
  - ○ Usually slower :(
- Udacity: Intro to Parallel Programming https://www.udacity.com/course/cs344
  - ○ For deep learning just use existing libraries

Fei-Fei Li & Andrej Karpathy & Justin Johnson Lecture 11 - 92 17 Feb 2016

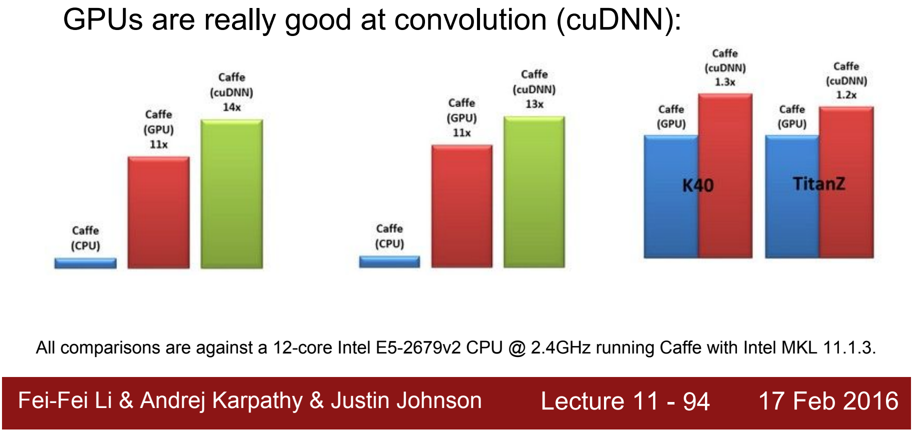


GPUs are really good at matrix multiplication:

GPU: NVIDA Tesla K40 with cuBLAS

CPU: Intel E5-2697 v2 12 core @ 2.7 Ghz with MKL

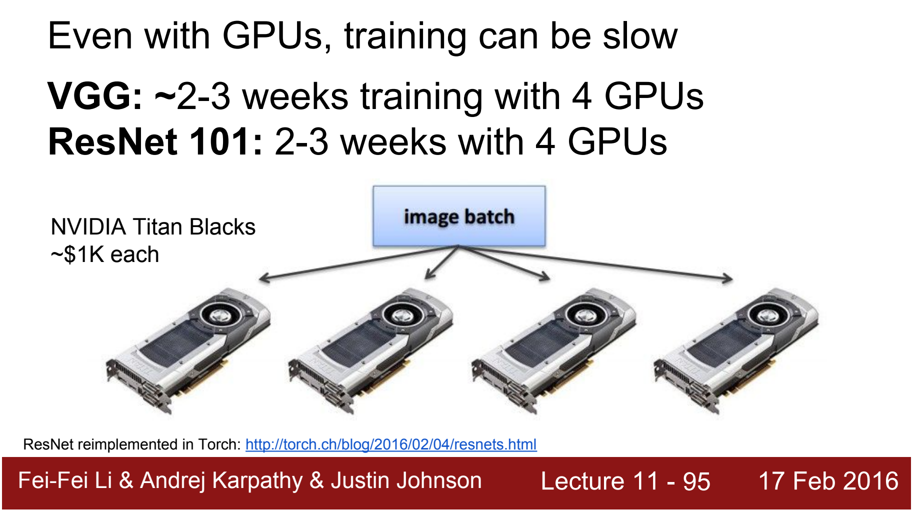
Fei-Fei Li & Andrej Karpathy & Justin Johnson Lecture 11 - 93 17 Feb 2016



GPUs are really good at convolution (cuDNN):

All comparisons are against a 12-core Intel E5-2679v2 CPU @ 2.4GHz running Caffe with Intel MKL 11.1.3.

Fei-Fei Li & Andrej Karpathy & Justin Johnson Lecture 11 - 94 17 Feb 2016



Even with GPUs, training can be slow

VGG: ~2-3 weeks training with 4 GPUs
ResNet 101: 2-3 weeks with 4 GPUs

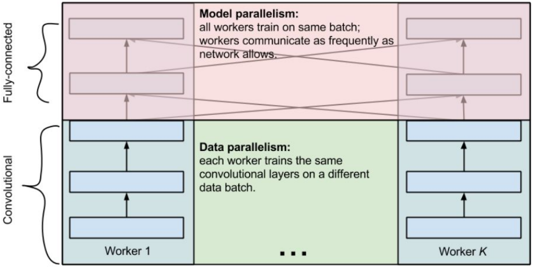
NVIDIA Titan Blacks
~$1K each

image batch

ResNet reimplemented in Torch: http://torch.ch/blog/2016/02/04/resnets.html

Fei-Fei Li & Andrej Karpathy & Justin Johnson Lecture 11 - 95 17 Feb 2016

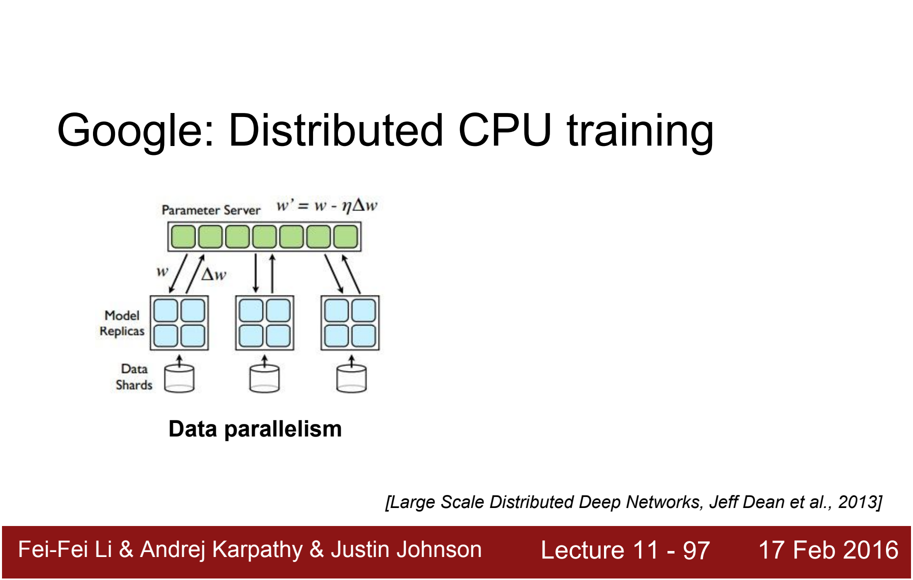
Multi-GPU training: More complex



Model parallelism: all workers train on same batch; workers communicate as frequently as network allows.

Data parallelism: each worker trains the same convolutional layers on a different data batch.

Alex Krizhevsky, “One weird trick for parallelizing convolutional neural networks”

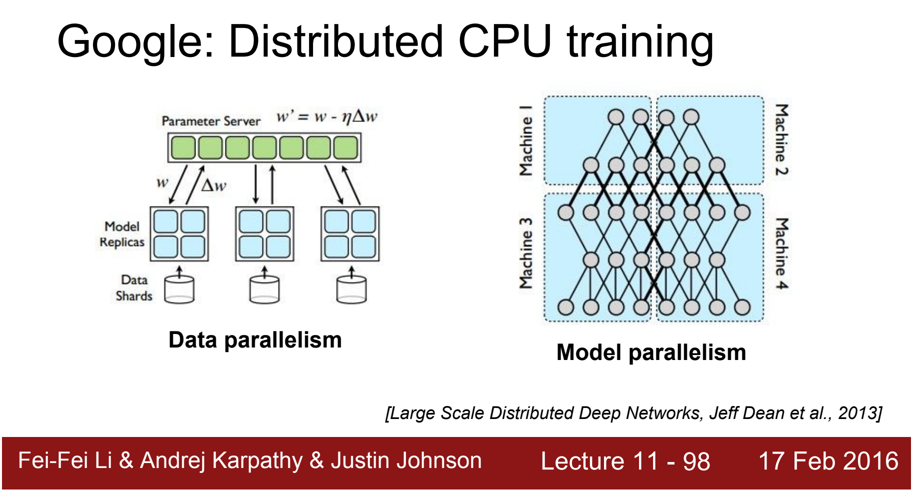


Google: Distributed CPU training

Data parallelism

[Large Scale Distributed Deep Networks, Jeff Dean et al., 2013]

Fei-Fei Li & Andrej Karpathy & Justin Johnson Lecture 11 - 97 17 Feb 2016



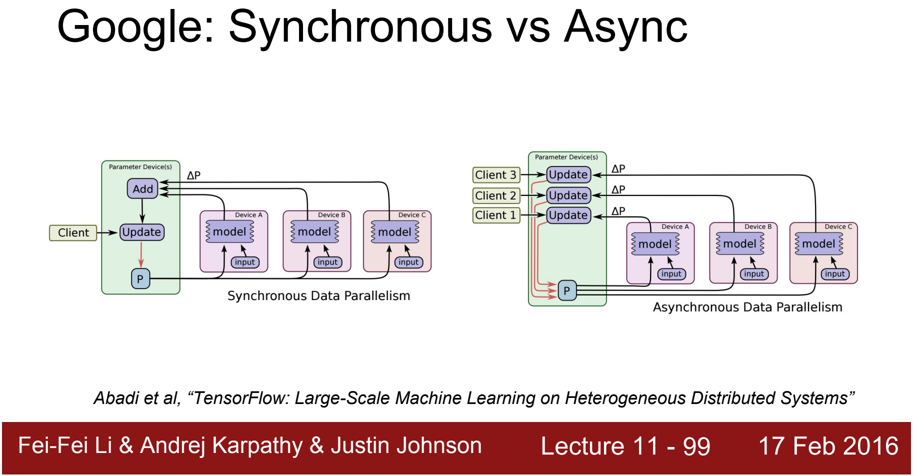
Google: Distributed CPU training

Data parallelism

Model parallelism

[Large Scale Distributed Deep Networks, Jeff Dean et al., 2013]

Fei-Fei Li & Andrej Karpathy & Justin Johnson Lecture 11 - 98 17 Feb 2016



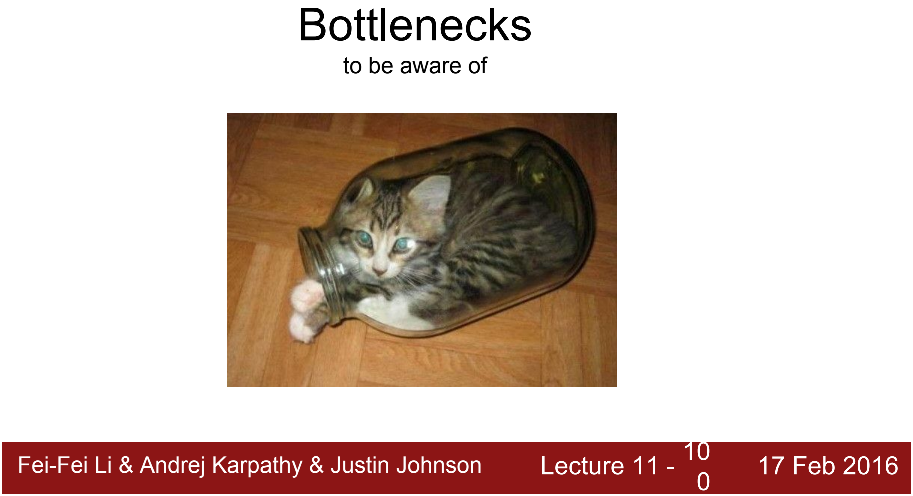
Google: Synchronous vs Async

Synchronous Data Parallelism

Asynchronous Data Parallelism

Abadi et al, "TensorFlow: Large-Scale Machine Learning on Heterogeneous Distributed Systems"

Fei-Fei Li & Andrej Karpathy & Justin Johnson Lecture 11 - 99 17 Feb 2016



Bottlenecks

to be aware of

Fei-Fei Li & Andrej Karpathy & Justin Johnson Lecture 11 - 100 17 Feb 2016

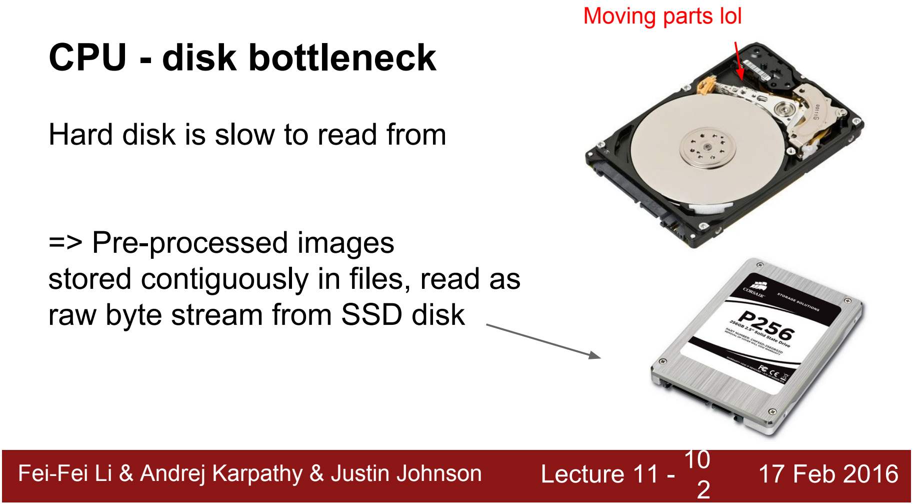
GPU - CPU communication is a bottleneck.
=>

CPU data prefetch+augment thread running

while

GPU performs forward/backward pass

Fei-Fei Li & Andrej Karpathy & Justin Johnson Lecture 11 - 101 17 Feb 2016



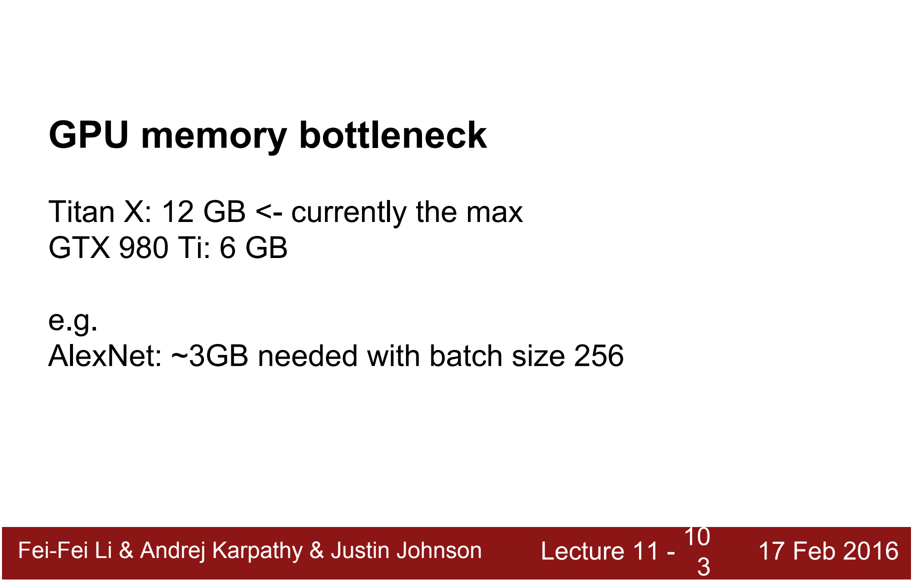
Moving parts lol

CPU - disk bottleneck

Hard disk is slow to read from

=> Pre-processed images stored contiguously in files, read as raw byte stream from SSD disk

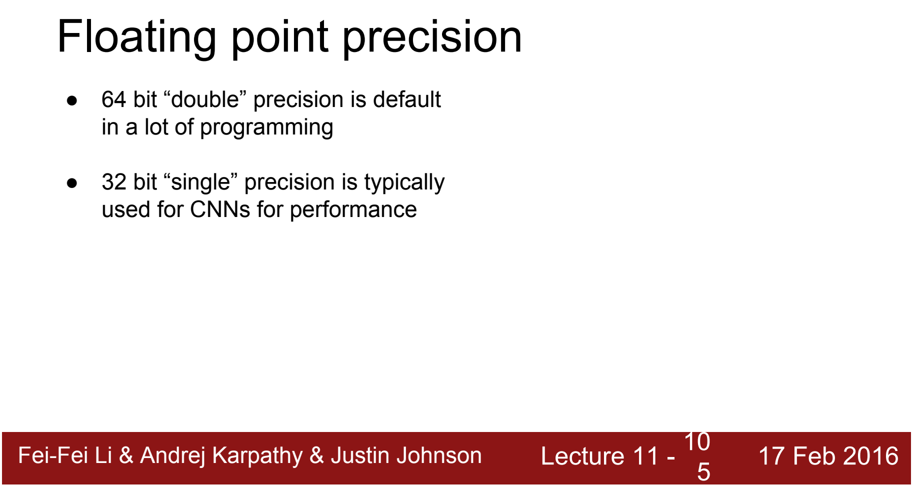
Fei-Fei Li & Andrej Karpathy & Justin Johnson   Lecture 11 - 102   17 Feb 2016



GPU memory bottleneck

Titan X: 12 GB <- currently the max
GTX 980 Ti: 6 GB

e.g.
AlexNet: ~3GB needed with batch size 256



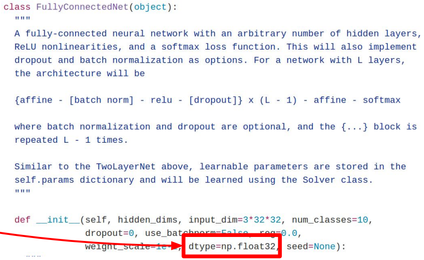
Floating point precision

- 64 bit “double” precision is default in a lot of programming
- 32 bit “single” precision is typically used for CNNs for performance

Fei-Fei Li & Andrej Karpathy & Justin Johnson Lecture 11 - 105 17 Feb 2016

Floating point precision

- 64 bit “double” precision is default in a lot of programming
- 32 bit “single” precision is typically used for CNNs for performance
  - Including cs231n homework!



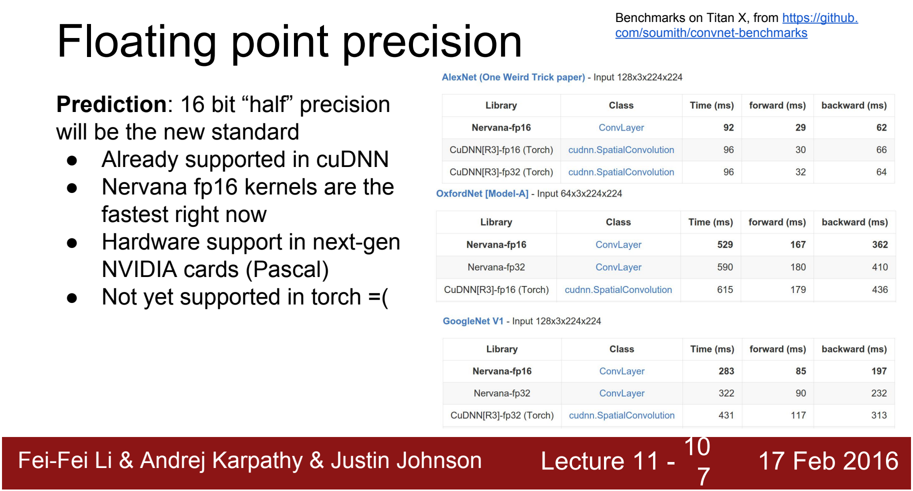
```
class FullyConnectedNet(object):
  """
  A fully-connected neural network with an arbitrary number of hidden layers,
  ReLU nonlinearities, and a softmax loss function. This will also implement
  dropout and batch normalization as options. For a network with L layers,
  the architecture will be

  {affine - [batch norm] - relu - [dropout]} x (L - 1) - affine - softmax

  where batch normalization and dropout are optional, and the {...} block is
  repeated L - 1 times.

  Similar to the TwoLayerNet above, learnable parameters are stored in the
  self.params dictionary and will be learned using the Solver class.
  """

  def __init__(self, hidden_dims, input_dim=3*32*32, num_classes=10,
               dropout=0, use_batchnorm=False, reg=0.0,
               weight_scale=1e-2, dtype=np.float32, seed=None):
```



Floating point precision

Benchmarks on Titan X, from https://github.com/soumith/convnet-benchmarks

Prediction: 16 bit “half” precision will be the new standard

- Already supported in cuDNN
- Nervana fp16 kernels are the fastest right now
- Hardware support in next-gen NVIDIA cards (Pascal)
- Not yet supported in torch =(

AlexNet (One Weird Trick paper) - Input 128x3x224x224

Library	Class	Time (ms)	forward (ms)	backward (ms)
Nervana-fp16	ConvLayer	**92**	**29**	**62**
CuDNN[R3]-fp16 (Torch)	cudnn.SpatialConvolution	96	30	66
CuDNN[R3]-fp32 (Torch)	cudnn.SpatialConvolution	96	32	64

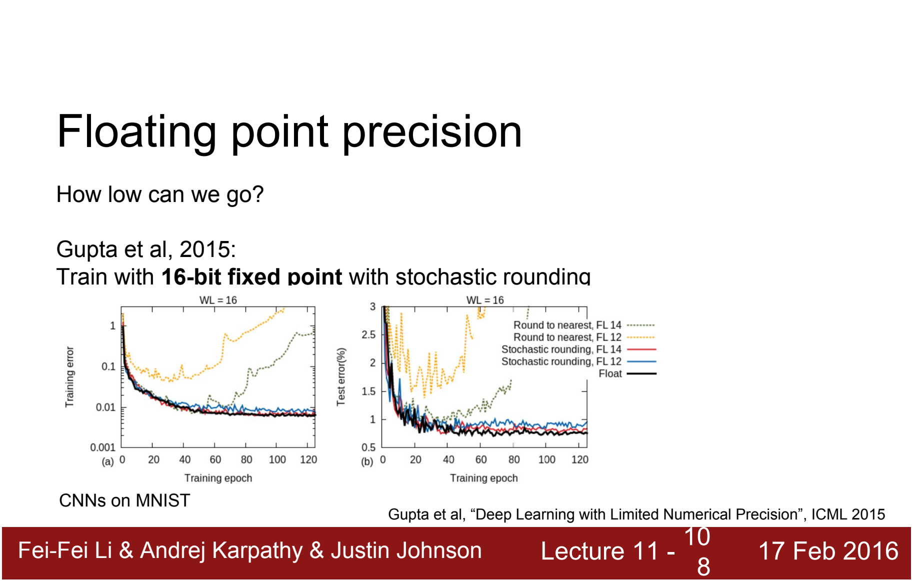
OxfordNet [Model-A] - Input 64x3x224x224

Library	Class	Time (ms)	forward (ms)	backward (ms)
Nervana-fp16	ConvLayer	**529**	**167**	**362**
Nervana-fp32	ConvLayer	590	180	410
CuDNN[R3]-fp16 (Torch)	cudnn.SpatialConvolution	615	179	436

GoogleNet V1 - Input 128x3x224x224

Library	Class	Time (ms)	forward (ms)	backward (ms)
Nervana-fp16	ConvLayer	**283**	**85**	**197**
Nervana-fp32	ConvLayer	322	90	232
CuDNN[R3]-fp32 (Torch)	cudnn.SpatialConvolution	431	117	313

Fei-Fei Li & Andrej Karpathy & Justin Johnson   Lecture 11 - 107   17 Feb 2016



Floating point precision

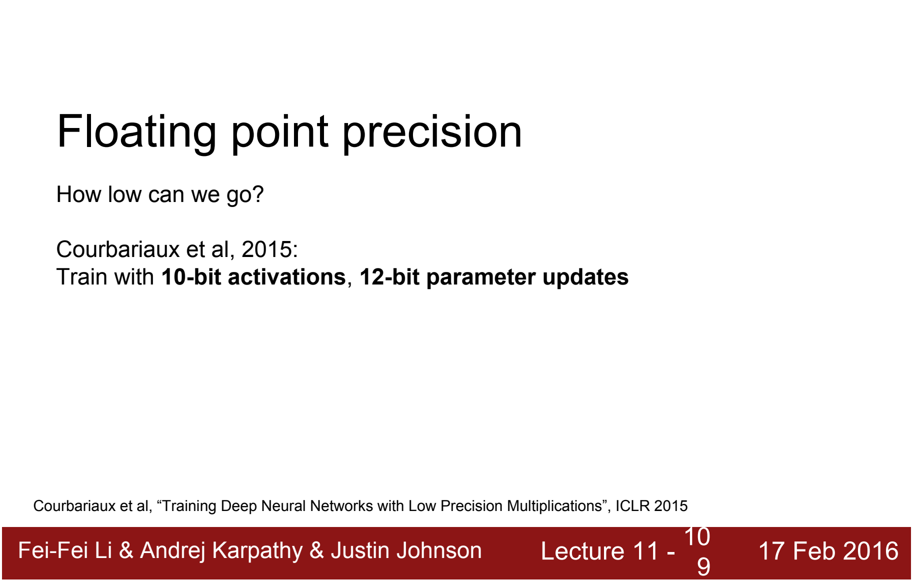
How low can we go?

Gupta et al, 2015:
Train with **16-bit fixed point** with stochastic rounding

CNNs on MNIST

Gupta et al, "Deep Learning with Limited Numerical Precision", ICML 2015

Fei-Fei Li & Andrej Karpathy & Justin Johnson Lecture 11 - 108 17 Feb 2016

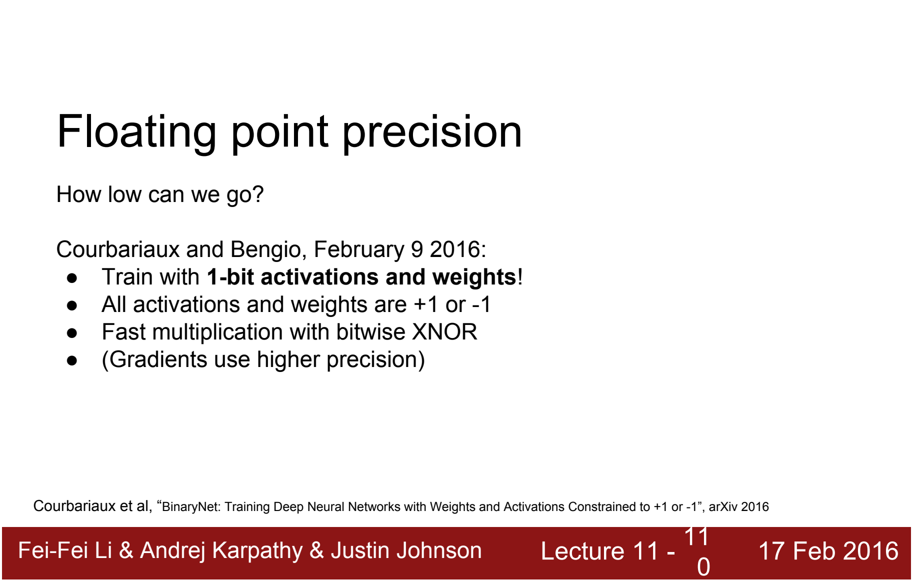


Floating point precision

How low can we go?

Courbariaux et al, 2015:
Train with **10-bit activations**, **12-bit parameter updates**

Courbariaux et al, “Training Deep Neural Networks with Low Precision Multiplications”, ICLR 2015



Floating point precision

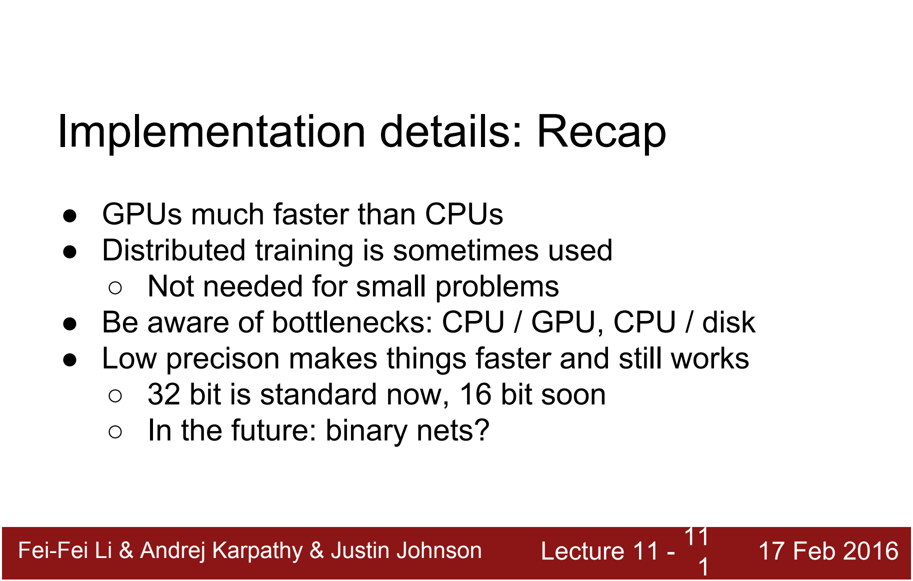
How low can we go?

Courbariaux and Bengio, February 9 2016:

- Train with **1-bit activations and weights**!
- All activations and weights are +1 or -1
- Fast multiplication with bitwise XNOR
- (Gradients use higher precision)

Courbariaux et al, "BinaryNet: Training Deep Neural Networks with Weights and Activations Constrained to +1 or -1", arXiv 2016

Fei-Fei Li & Andrej Karpathy & Justin Johnson Lecture 11 - 110 17 Feb 2016



Implementation details: Recap

- GPUs much faster than CPUs
- Distributed training is sometimes used
  - Not needed for small problems
- Be aware of bottlenecks: CPU / GPU, CPU / disk
- Low precison makes things faster and still works
  - 32 bit is standard now, 16 bit soon
  - In the future: binary nets?

Conclusions

- “Classic” CNN composed of **conv layers**, **pooling layers**, and **fully connected layers**
  - Date back to LeNet-5 by Yann Lecun in 90’s
  - But gaining lots of attention since AlexNet 2012
- Widely used tricks
  - Data augmentation
  - Transfer learning
  - Use of GPUs
- Some recent trends
  - Small filter decomposition
  - Filter output cascading (GoogLeNet)
  - Fast conv layer with “Strassen-like” algorithms
  - Use of lower and lower floating point precision formats

Conclusions

- "Classic" CNN composed of **conv layers**, **pooling layers**, and **fully connected layers**
    - Date back to LeNet-5 by Yann Lecun in 90's
    - But gaining lots of attention since AlexNet 2012
- Widely used tricks
    - Data augmentation
    - Transfer learning
    - Use of GPUs
- Some recent trends
    - Small filter decomposition
    - Filter output cascading (GoogLeNet)
    - Fast conv layer with "Strassen-like" algorithms
    - Use of lower and lower floating point precision formats

Conclusions

- "Classic" CNN composed of **conv layers**, **pooling layers**, and **fully connected layers**
    - Date back to LeNet-5 by Yann Lecun in 90's
    - But gaining lots of attention since AlexNet 2012
- Widely used tricks
    - Data augmentation
    - Transfer learning
    - Use of GPUs
- Some recent trends
    - Small filter decomposition
    - Filter output cascading (GoogLeNet)
    - Fast conv layer with "Strassen-like" algorithms
    - Use of lower and lower floating point precision formats