Convolutional Neural Networks

HW 6

Convolution

- conv2d: Convolution masks applied to all elements in a stack
- depthwise_conv2d: Convolution mask is applied to each stack, independently (resulting stack depth:
  input stack depth x # filters)
- conv3d: 3D convolution for 3D images

Pooling

- Average vs max pooling
- Stack elements are pooled independently
  - So: output stack depth is the same as input depth
- Not uncommon to use pooling to reduce image size by a factor of 2 along each dimenson:
  - filter_size = 2
  - Stride = 2

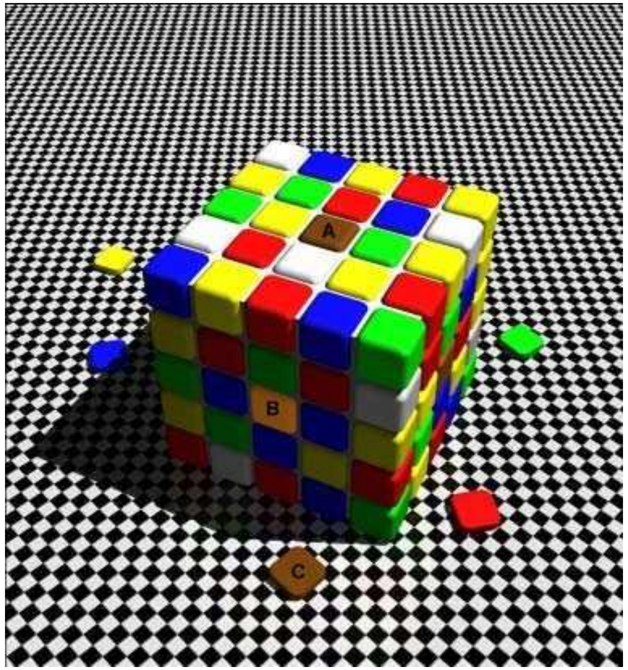
Notes

- Convolution and average pooling (in and of themselves) are linear operators
- Composition of multiple linear operators can be expressed as a single linear operator
- Likewise, a single linear operator can be decomposed into multiple linear operators
- Large convolutional filters (7x7, 9x9, …) are expensive to compute
    - Can be equivalently captured as a sequence of smaller filters

Deep Networks for Image Recognition

- Images are composed of large numbers of pixels
- A particular pixel value can vary a lot:
  - Color, illumination
- Objects can vary a lot
  - Size, orientation, perspective

Individual pixels are irrelevant…



http://brainden.com/color-illusions.htm

Deep Networks for Image Recognition

If individual pixels are irrelevant, then what is important?

Finding Abstractions

First level abstractions:

- Edges (with orientation)
- Bars
- Spots
- Corners

Finding Abstractions

Spatial scale is important, too:

- An edge might be multiple pixels in width or have some arbitrary length, but we still want to recognize it under these variations
- Pooling layers give us a way to shrink the images so that the same low-level filters can be applied at different spatial scales

Finding Abstractions

Second level abstractions:

- Curves
- Constellations of first-level filters

Beyond the Primitives…

How should the primitives be combined to form more of a semantic representation (dog, cat, grandma, etc.)?

- It is unclear how to write these “filters” down
- The deep learning approach is to let the learning algorithm handle this…

Beyond the Primitives…

How should the primitives be combined to form more of a semantic representation (dog, cat, grandma, etc.)?

- After providing the primitives in the first layers of our deep network, employ dense layers to allow for arbitrary combinations of the primitives

Overfitting

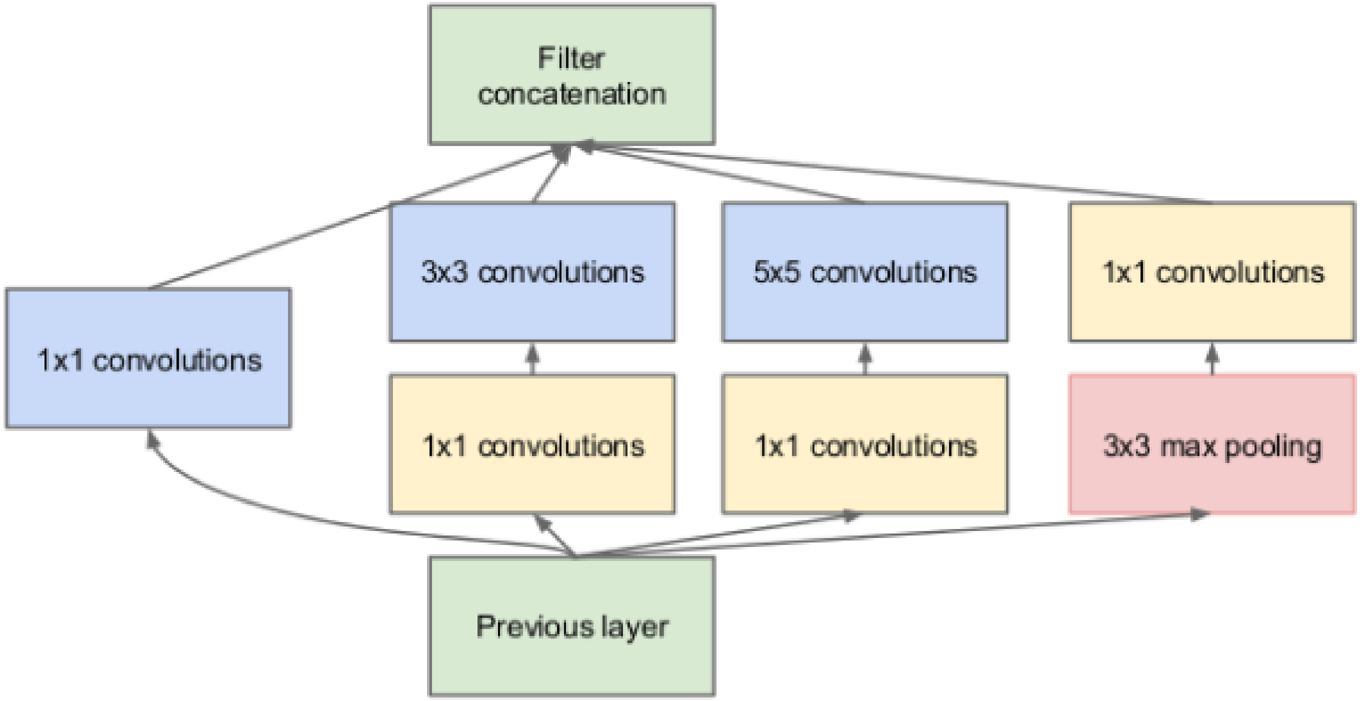
Convolutional filters involve very few parameters, but the deep layers (especially when deep stacks are their inputs) have lots of parameters

- Same approaches to overfitting still apply
  - Regularization, dropout, max-norm, diverse training set …
- Local response normalization: force corresponding units across a stack to compete with one-another
  - A highly active neuron suppresses the activity of others in nearby stack elements
  - Forces the different filters to take on very different representations

Inception Modules

Ambiguous as to the right scale and type of processing: so, do it all…

- Stack concatenation: concatenate the 4 stacks together into a larger stack
- 1x1 vs 3x3 vs 5x5 gives us different scaling
- Can create with a single method call!

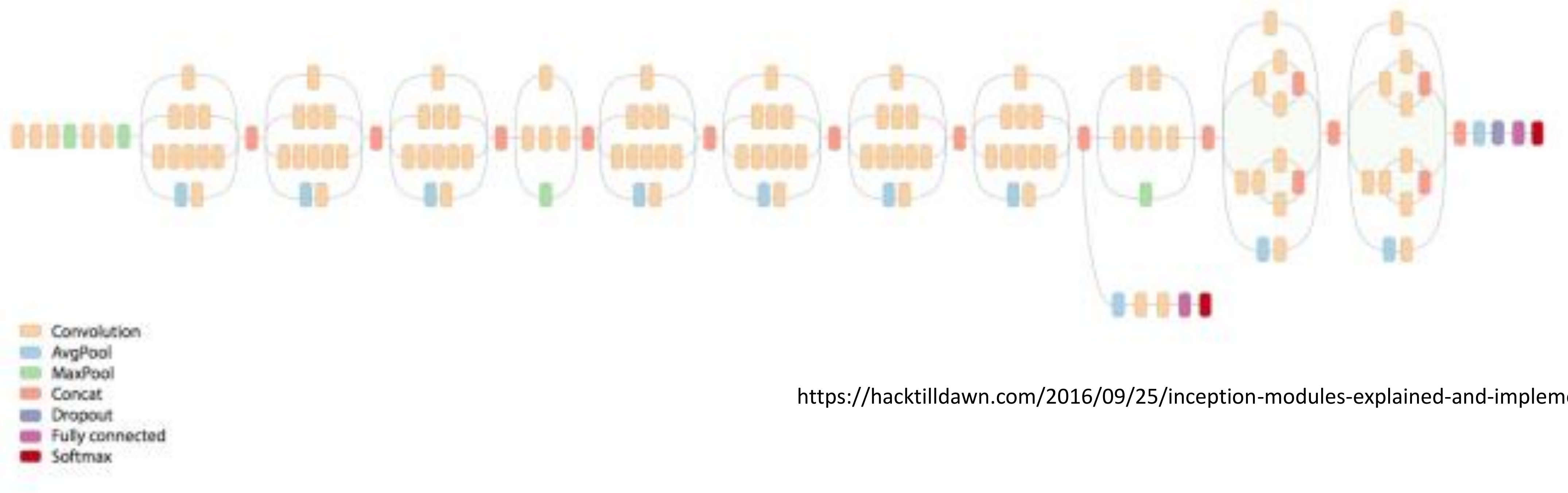


https://hacktilldawn.com/2016/09/25/inception-modules-explained-and-implemented/

1x1 “Convolutional” Filters

- No longer combining neighboring pixels
- But: we are still combining the corresponding pixels from the different stack elements
- In particular: if the new filter produces fewer stack elements, this forces a certain degree of compression across the input stack elements

Full Inception Network (V3)



https://hacktilldawn.com/2016/09/25/inception-modules-explained-and-implemented/

Note: Multiple "exit points"

Skip Connections

We would like to not commit to a particular spatial scale ahead of time

- Deep layers receive inputs from multiple convolutional layers
- As you saw in HW 5, we can use skip connections to capture the large-scale aspects of the function & then rely on the “unskipped” layers to handle the fine details

Practicalities (HW 6)

- Now have 6 conditions available on the cluster
- My network:
  - Structure essentially the same
  - Now using dropout
  - Training with 4+4 objects with every other image, but in stochastic mini-batches (400 images at a time)
  - Validation on the 5th objects & every 10th image

HW 6

- Overfitting is a challenge
- It is possible to get lucky with one object each for the validation set. I recommend trying at least two different training/validation object partitions
- Most interesting learning is happening in the first 100-200 epochs
- Lots of tweaking, but now getting AUC > 0.5 for the validation set

Transition from Convolutional Layers to Dense Layers

Simplest solution: reshape the convolutional layer into one linear layer, then provide as input to your dense layer (as usual)

```
conv2_reshape =
        tf.reshape(conv2, (-1, size_r*size_c*nfilters2))
```

- Other than the number of samples, have to know the shape of the resulting Tensor at network construction time
- tf.shape() will tell you Tensor shape at ***runtime*** (which is too late for this purpose)

Using tf.metrics

AUC Initialization

```
auc, update_op=
    tf.metrics.auc(y, prediction, name="auc")

running_vars =
    tf.get_collection(tf.GraphKeys.LOCAL_VARIABLES,
        scope=name+"/"+"auc")

running_vars_initializer =
    tf.variables_initializer(var_list=running_vars,
        name="initializer_auc")
```

Run-Time

```
sess.run(initializer_auc)

sess.run(update_op_auc,
    feed_dict=feed_dict_validation)

mse_validation, auc_validation =
    sess.run([mse, auc],
    feed_dict=feed_dict_validation)
```