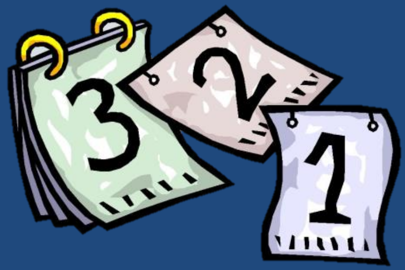
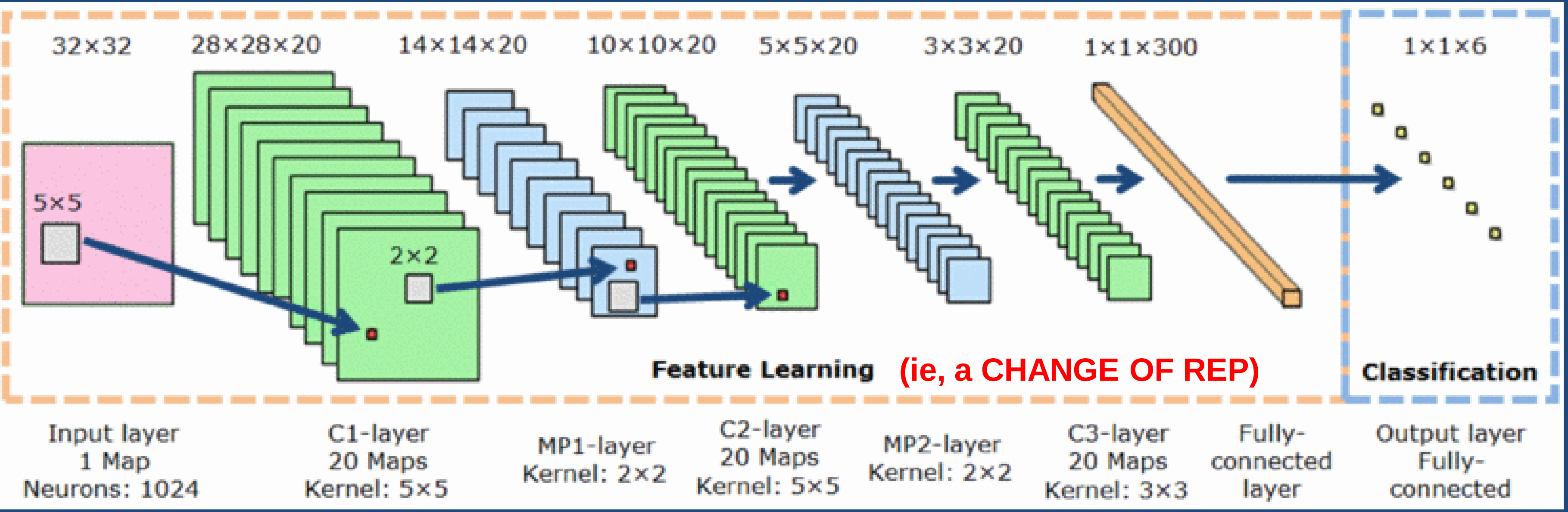
Today's Topics



- Some Lab 3 Tips and Discussion
- Still Need a Lab 3 Partner?
- If Still Working on Lab 2, Complete ASAP!
- Some Advanced Weight-Learning Topics (by Yujia Bao)
- Interpreting ANN Outputs as Probability Distributions and the Cross-Entropy Error Function (might be covered next week)
- Next week: An Overview of Some Deep ML Projects at American Family

Back to Deep ANNs
- Convolution & Max Pooling (Repeat)

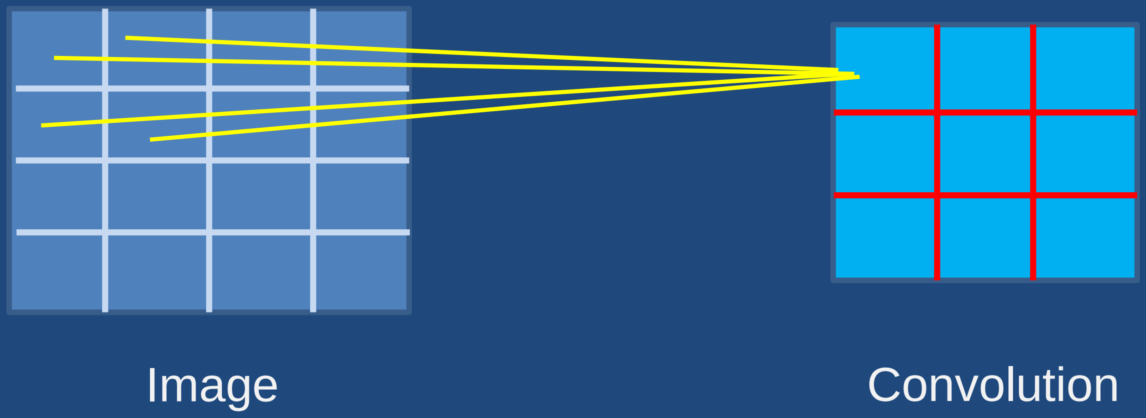
C = Convolution, MP = Max Pooling



My implementation (no dropout yet) of the above topology takes about 1-2 mins per epoch on the provided TRAIN/TUNE/TEST set
(I measure TRAIN, TUNE and TEST accuracy after each epoch)

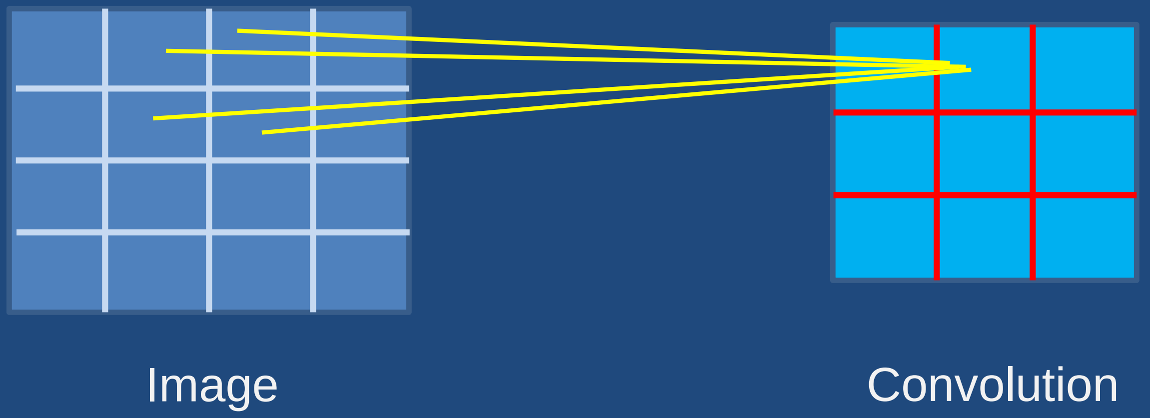
Details: Image to First CONV

- There are MULTIPLE, *INDEPENDENT* <u>plates</u> (so each can learn a different 'derived feature')
- *Within* a plate, there is only <u>one</u> set of weights, shared for all 'starting positions'



Details: Image to First CONV

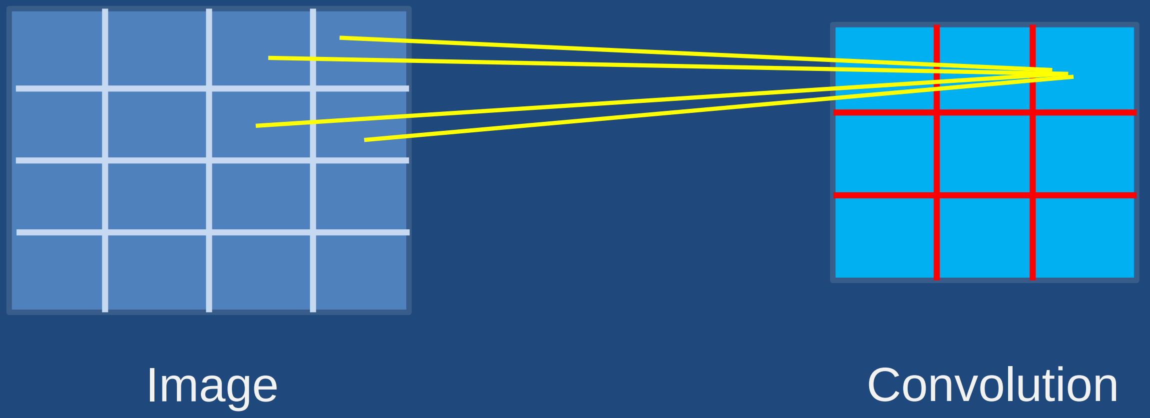
- There are MULTIPLE, *INDEPENDENT* <u>plates</u> (so each can learn a different 'derived feature')
- *Within* a plate, there is only <u>one</u> set of weights, shared for all 'starting positions'



Note that the 'sliding windows' OVERLAP (a design choice)

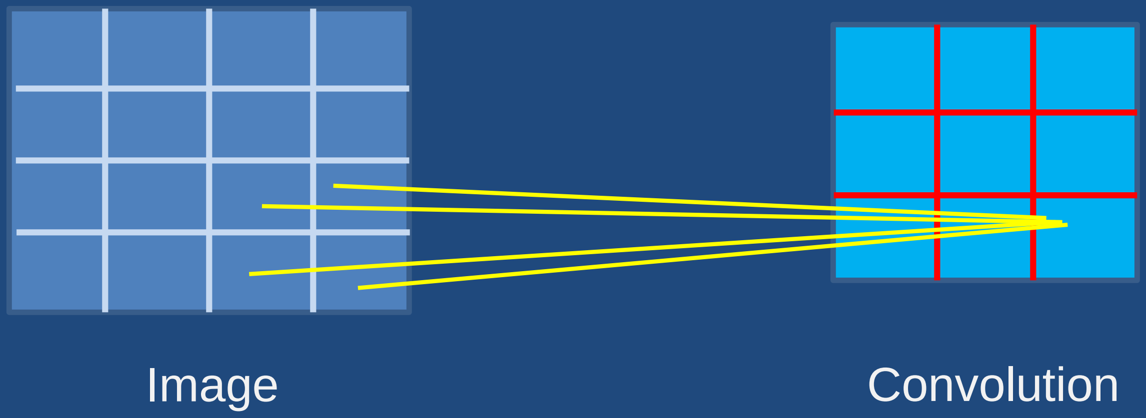
Details: Image to First CONV

- There are MULTIPLE, *INDEPENDENT* plates (so each can learn a different 'derived feature')
- *Within* a plate, there is only one set of weights, shared for all 'starting positions'



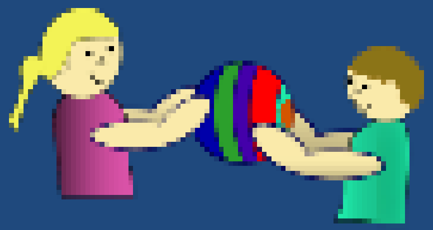
Details: Image to First CONV

- There are MULTIPLE, *INDEPENDENT* plates (so each can learn a different 'derived feature')
- *Within* a plate, there is only one set of weights, shared for all 'starting positions'



CONV Layer Weight Sharing

- The SAME WEIGHTs are used for all ‘sliding window’ locations <u>per plate</u>
- You will only backprop through CONV HUs that are the MAX in some POOL window
- You might want to scale (ie, divide) η by the number of cells in the following POOL layer (ie, the number of MAX’es computed)



function BACK-PROP-LEARNING(*examples*, *network*) **returns** a neural network
  inputs: *examples*, a set of examples, each with input vector $\mathbf{x}$ and output vector $\mathbf{y}$
    network, a multilayer network with L layers, weights $w_{i,j}$, activation function g
  local variables: Δ, a vector of errors, indexed by network node

  repeat
    for each weight $w_{i,j}$ in *network* **do**
      $w_{i,j} \leftarrow$ a small random number
    for each example $(\mathbf{x}, \mathbf{y})$ **in** *examples* **do**
      / Propagate the inputs forward to compute the outputs */*
      for each node i in the input layer **do**
        $a_i \leftarrow x_i$
      for $\ell = 2$ **to** L **do**
        for each node j in layer ℓ **do**
          $in_j \leftarrow \sum_i w_{i,j}\, a_i$
          $a_j \leftarrow g(in_j)$
      / Propagate deltas backward from output layer to input layer */*
      for each node j in the output layer **do**
        $\Delta[j] \leftarrow g'(in_j) \times (y_j - a_j)$
      for $\ell = L - 1$ **to** 1 **do**
        for each node i in layer ℓ **do**
          $\Delta[i] \leftarrow g'(in_i) \sum_j w_{i,j}\, \Delta[j]$
      / Update every weight in network using deltas */*
      for each weight $w_{i,j}$ in *network* **do**
        $w_{i,j} \leftarrow w_{i,j} + \alpha \times a_i \times \Delta[j]$
  until some stopping criterion is satisfied
  return *network*

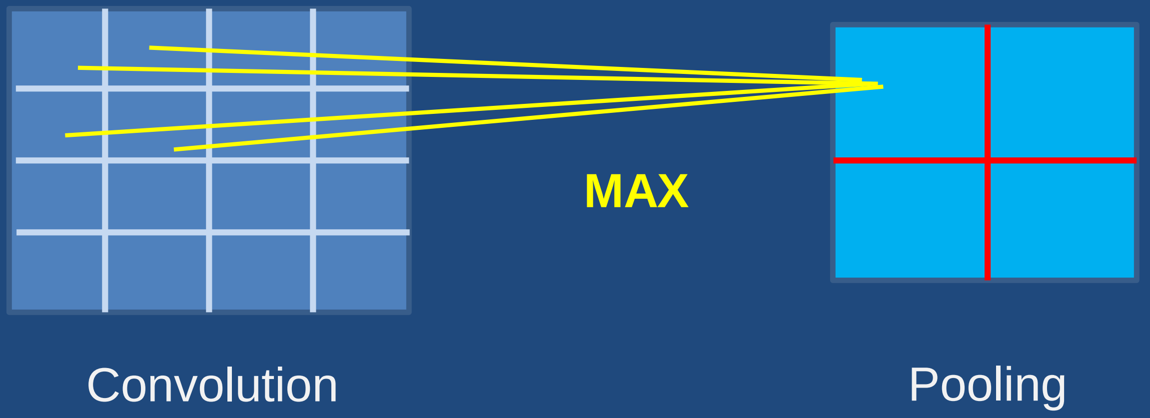
Figure 18.24 The back-propagation algorithm for learning in multilayer networks.

the weight-update rule for the weights between the inputs and the hidden layer is
identical to the update rule for the

From CS 540 textbook, by Russel and Norvig, 3rd ed

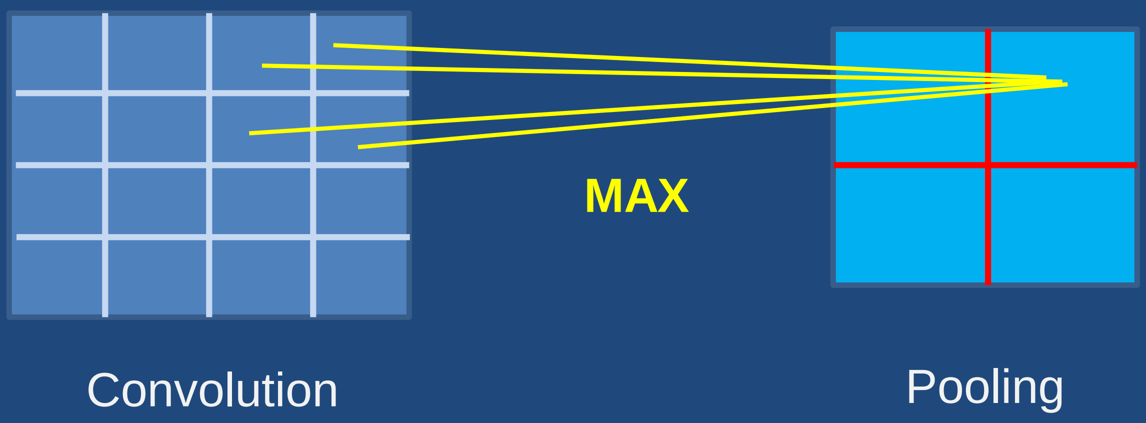
Details: N^{th} CONV to N^{th} POOL

- I decided to have <u>one</u> POOL <u>per</u> CONV
- POOLs compute MAX via code (no wgts)
- In general, could have <u>several</u> POOLs of different sizes per CONV



Details: N^{th} CONV to N^{th} POOL

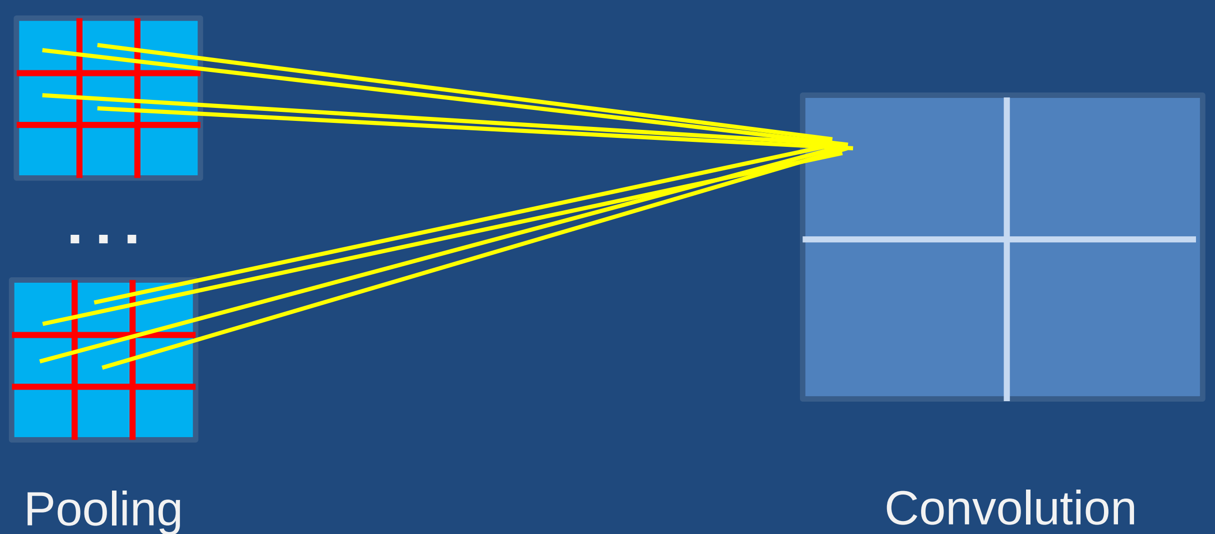
- I *decided* to have <u>one</u> POOL per CONV
- POOLs compute MAX via code (no wgts)
- In general, could have <u>several</u> POOLs of *different sizes* per CONV



Note that the 'sliding windows' DO NOT OVERLAP (a design choice)

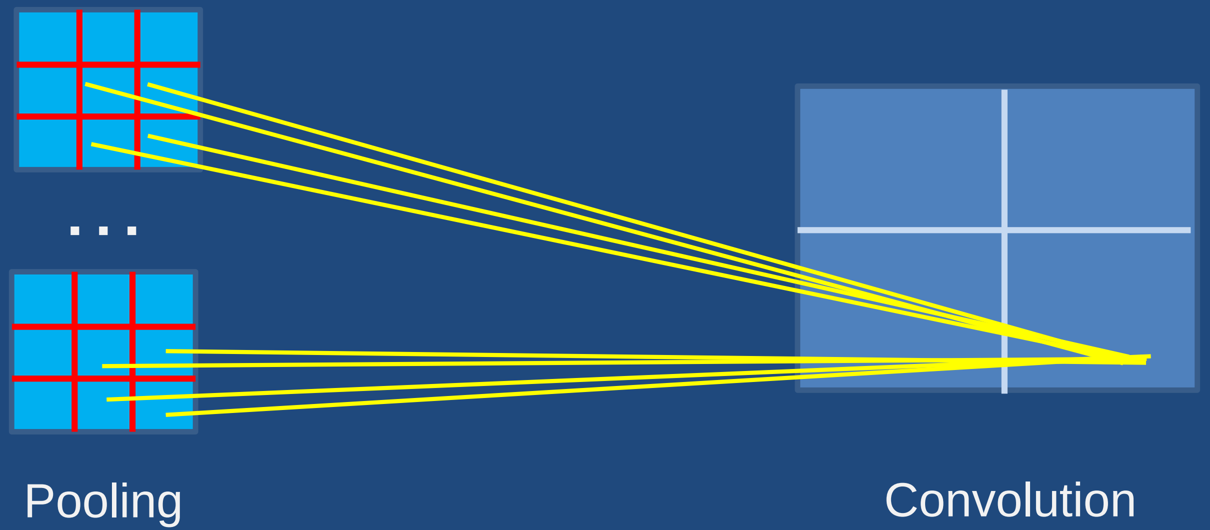
Details: N^{th} POOLs to N^{th} CONV

- We want a CONV layer to look at ALL nodes at previous POOL layer, but only look at the same window
- This allows learning combinations of derived features



Details: N^{th} POOLs to N^{th} CONV

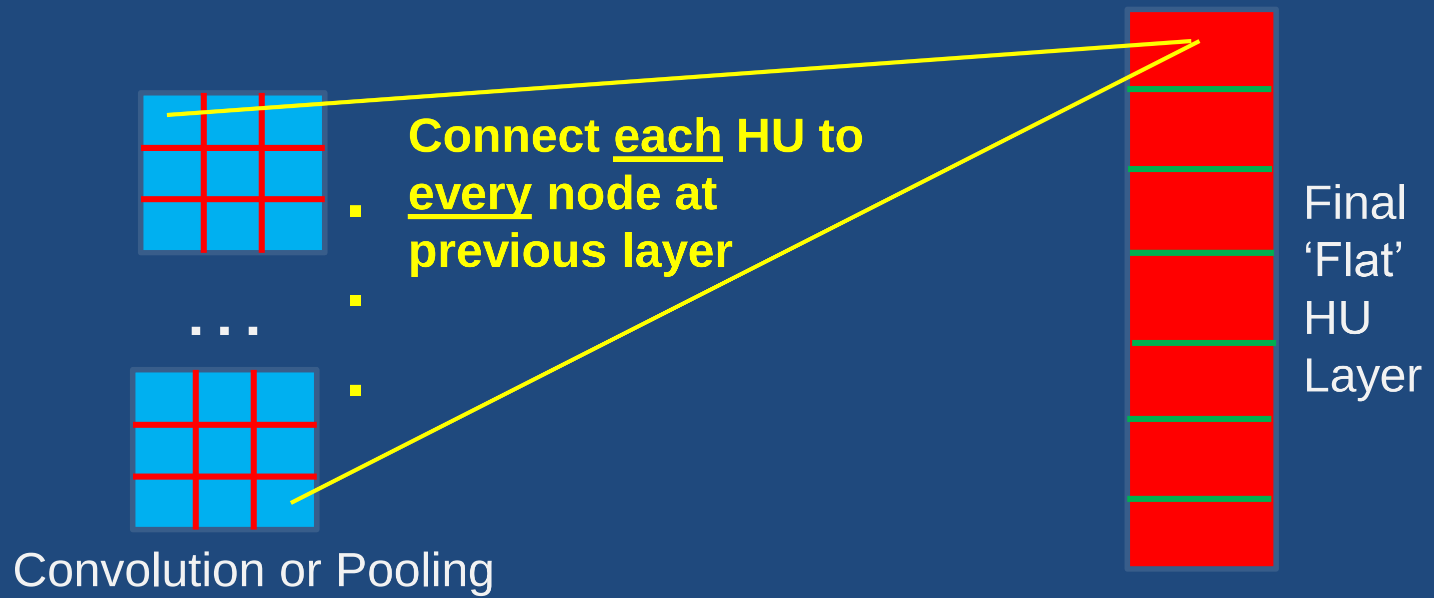
- We want a CONV layer to look at ALL nodes at previous POOL layer, but only look at the same window
- This allows learning combinations of derived features



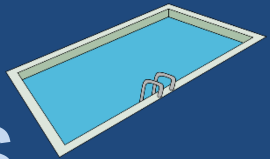
Details: N^{th} CONV to 1 HU Layer

- We want the flat layer of HUs to look all CONV/POOL nodes at final CONV/POOL layer.
- Initially Lab 3 said to use the following, but ok to follow Slide 2's drawing and have a 3rd CONV layer (I did this)

CONV-POOL-CONV-POOL-FLAT-OUTPUT



BP'ing Through POOL Nodes



- My implementation only BPs through the CONV nodes that are the MAX for some POOL window (all others have deviation=0)
- There are no learnable weights (and biases) between CONV and POOL layers (where needed I assume wgt=1, but never change it – ie, the MAX calc is not done via wgt'ed sums and an activation function; it is just Java code)
- POOL nodes compute 'deviations' (ie, the intermediate calc in the BP algorithm), but don't change wgts and biases
- If POOL windows do not overlap (and only 1 POOL plate per CONV plate), a CONV node only sums the deviations from at most one POOL node in the next layer

Pictorially