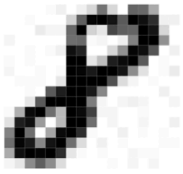
Convolutional Neural Networks

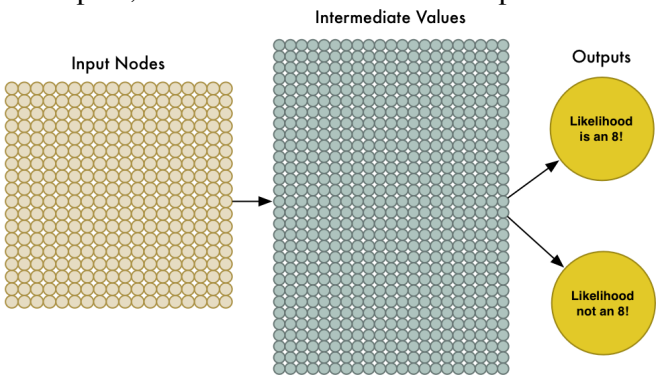
How do we feed images into a neural network, since they take numbers as input? An image is really just a grid of numbers that represent how dark each pixel is:



=

[0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 1, 12, 0, 11, 39, 137, 37, 0, 152, 147, 84, 0, 0, 0, 0,
0, 1, 0, 0, 0, 41, 160, 250, 255, 235, 162, 255, 238, 206, 11, 13, 0, 0, 0, 0, 16, 9, 9, 150, 251, 45, 21, 184, 159, 154, 2
55, 233, 40, 0, 0, 10, 0, 0, 0, 0, 0, 145, 146, 3, 10, 0, 11, 124, 253, 255, 107, 0, 0, 0, 0, 3, 0, 4, 15, 236, 216, 0, 0,
38, 109, 247, 240, 169, 0, 11, 0, 1, 0, 2, 0, 0, 0, 253, 253, 23, 62, 224, 241, 255, 164, 0, 5, 0, 0, 6, 0, 0, 4, 0, 3, 252
, 250, 228, 255, 255, 234, 112, 28, 0, 2, 17, 0, 0, 2, 1, 4, 0, 21, 255, 253, 251, 255, 172, 31, 8, 0, 1, 0, 0, 0, 0, 0, 4,
0, 163, 225, 251, 255, 229, 120, 0, 0, 0, 0, 0, 11, 0, 0, 0, 0, 21, 162, 255, 255, 254, 255, 126, 6, 0, 10, 14, 6, 0, 0, 9
, 0, 3, 79, 242, 255, 141, 66, 255, 245, 189, 7, 8, 0, 0, 5, 0, 0, 0, 0, 26, 221, 237, 98, 0, 67, 251, 255, 144, 0, 8, 0, 0
, 7, 0, 0, 11, 0, 125, 255, 141, 0, 87, 244, 255, 208, 3, 0, 0, 13, 0, 1, 0, 1, 0, 0, 145, 248, 228, 116, 235, 255, 141, 34
, 0, 11, 0, 1, 0, 0, 0, 1, 3, 0, 85, 237, 253, 246, 255, 210, 21, 1, 0, 1, 0, 0, 6, 2, 4, 0, 0, 0, 6, 23, 112, 157, 114, 32
, 0, 0, 0, 0, 2, 0, 8, 0, 7, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0]

To handle the new size of inputs, our network will have 324 input nodes:



There are now two outputs; The first output will predict the likelihood that the image is an “8” and thee second output will predict the likelihood it isn’t an “8”. By having a separate output for each type of object we want to recognize, we can use a neural network to classify objects into groups.

All that’s left is to train the neural network with images of “8”s and not-“8"s so it learns to tell them apart. When we feed in an “8”, we’ll tell it the probability the image is an “8” is 100% and the probability it’s not an “8” is 0%. Vice versa for the counter-example images.
Here’s some of our training data:

Interestingly, the “8” recognizer works well on simple images where the number is in the middle of the image:

But it completely fails to work when the letter isn’t centered. Slight shifts ruin the entire network:

It seems our network only learned the pattern of a perfectly-centered “8”. It has absolutely no idea what an off-center “8” is. It knows exactly one pattern and one pattern only.
That’s not very useful in the real world. Real world problems are never that clean and simple. So we need to figure out how to make our neural network work in cases where the “8” isn’t perfectly centered.

Solutions:

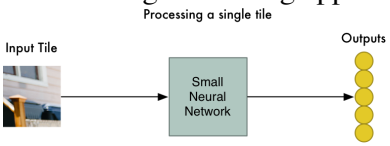
1. Search with a sliding window
    a. Scan all around the image for possible 8’s in smaller sections
        i. Really inefficient
2. More data and deep neural net
    a. Generate synthetic data using original data set
    b. It recognizes an 8 at the bottom as different than the top
    c. Need to determine method that 8 anywhere is an 8

Final Answer: ***Convolution***

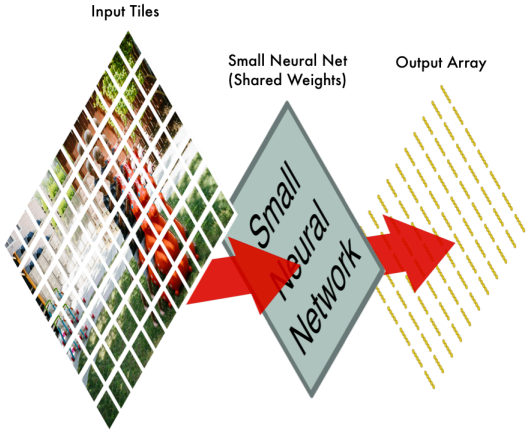
- We need to give the computer translation invariance

How Convolution works

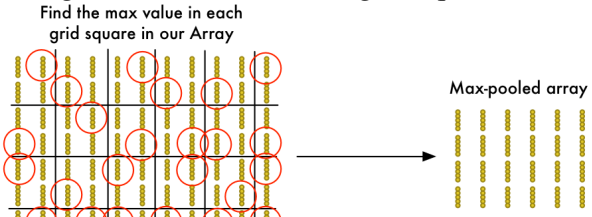
1. Break the image into overlapping image tiles
    a. This produces ~for example~ 100 equally sized tiny image tiles
2. Feed each image tile into a small neural network
    a. Keep same neural network weights for every tile
    b. Treat every image tile equally
    c. If something interesting appears, it will be marked as interesting
    d.

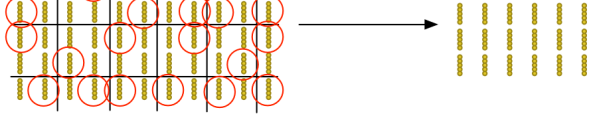


3. Save the results from each tile into a new array
    a. Save result from each tile into a grid in same arrangement as the original image
    b.

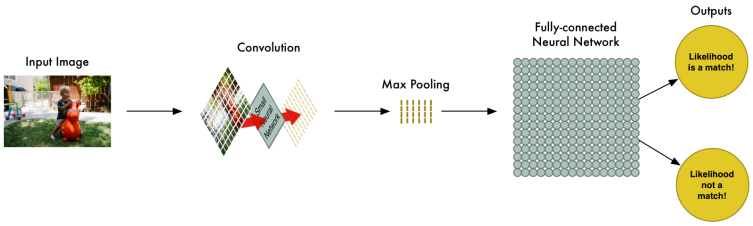


4. Downsampling
    a. Reduce size of the array due to its size
    b. We can use *max pooling*
        i. Taking the max value in each grid-square in the array



ii. 

5. Make a prediction
    a. The array is just numbers, which can be placed into final network, called "Fully connected NN"

b. 

Build a Bird Classifier

Use the following data sets:
CIFAR10 data set – 6,000 Birds and 52,000 non-birds
Caltech-UCSD Birds-200–2011 data set – 12,000 Birds

Although this will work, 72,000 low-res images is still small, one needs missions of large images to get Google level. To build our classifier, we'll use TFLearn. TFlearn is a wrapper around Google's TensorFlow deep learning library that exposes a simplified API. It makes building convolutional neural networks as easy as writing a few lines of code to define the layers of our network.

Here's the code to define and train the network: Go to Folder bird_classifier and run train.py
The script to test is r_u_a_bird.py

The network achieved 95% accuracy but what does that mean?

Our network claims to be 95% accurate. But the devil is in the details. That could mean all sorts of different things.
For example, what if 5% of our training images were birds and the other 95% were not birds? A program that guessed "not a bird" every single time would be 95% accurate! But it would also be 100% useless.
We need to look more closely at the numbers than just the overall accuracy. To judge how good a classification system really is, we need to look closely at how it failed, not just the percentage of the time that it failed.
Instead of thinking about our predictions as "right" and "wrong", let's break them down into four separate categories — True Positives, True Negatives, False Positives, and False Negatives.

Results for 15,000 Validation Images

(6000 images are birds, 9000 images are not birds)

	Predicted 'bird'	Predicted 'not a bird'
Bird	5,450 *True Positives*	550 *False Negatives*
Not a Bird	162 *False Positives*	8,838 *True Negatives*

Precision *If we predicted 'bird', how often was it really a bird?*	97.11% *(True Positives ÷ All Positive Guesses)*
Recall *What percentage of the actual birds did we find?*	90.83% *(True Positives ÷ Total Birds in Dataset)*