Waitlisting and Auditing

- I am very happy to hear that people will also be auditing the course!
- And the huge waitlist shows how excited people are about ML
- I will try to keep everyone as informed as possible
- Feel free to ask me questions, but I must prioritize *enrolled* students with my time
- email me (reyancey@ucdavis.edu) to be added to the auditors email list :)
- I will send course updates to people on the waitlist for the first few weeks, as well
- If you are on the waitlist and don't get enrolled, then auditing might be the best option for you
- Im also teaching this in the summer FYI

Practice Quiz Reminders

- Quiz 0 (week 0) will just be an (ungraded) test of OMSI, to make sure you have it working for yourself
- Please, read the OMSI markdown (.md) file very thoroughly
  - I know it is long but it will be worth it (trust me!)
- Test setting up both client and server by yourself
- The Quiz TA can answer any other questions you have about OMSI
- Quiz 1 (week 1) is practice and will be extra credit

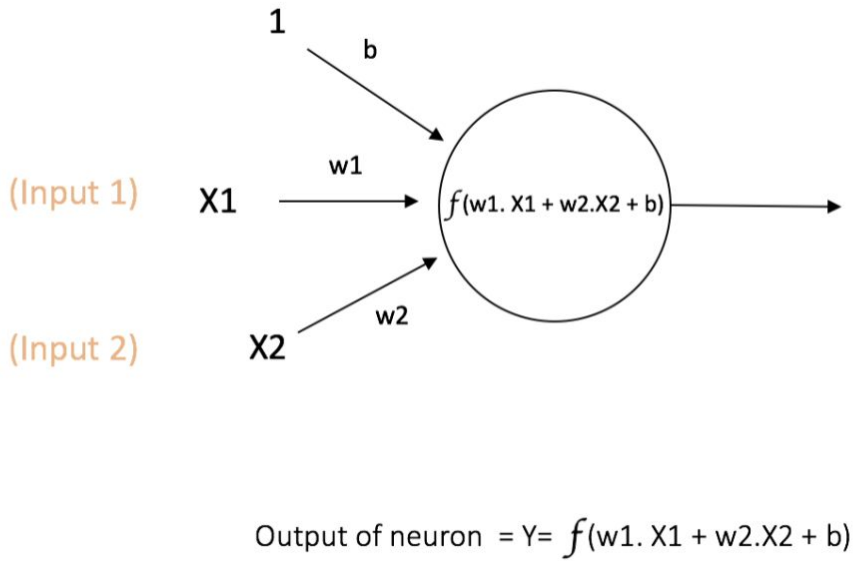
Practice/Test Handin

- Open up a terminal and enter the following to log into CSIF: ssh account_name@pcX.cs.ucdavis.edu
  - where **account_name** is your account name and **X** is the pc you want to login to.
  - Type your password
- create a text file: vi test.txt
- Enter: handin cs171 hw1 test.txt
- It should say: Submitting test.txt... ok

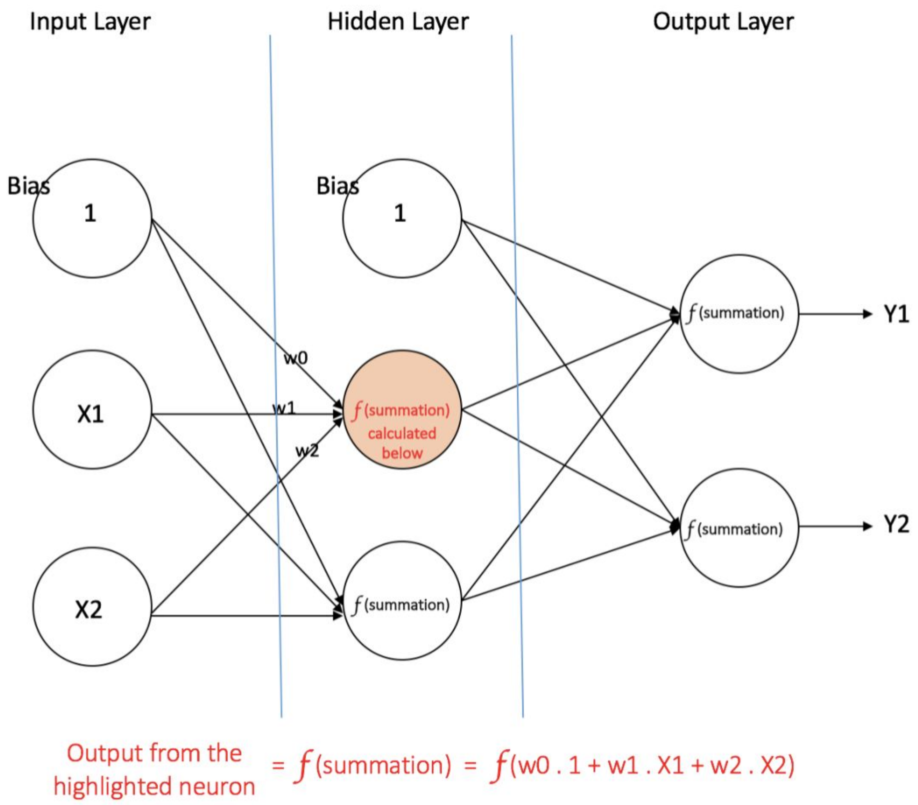
Super Quick Intro to Neural Networks

source:https://ujjwalkarn.me/2016/08/09/quick-intro-neural-networks/

Single Neuron

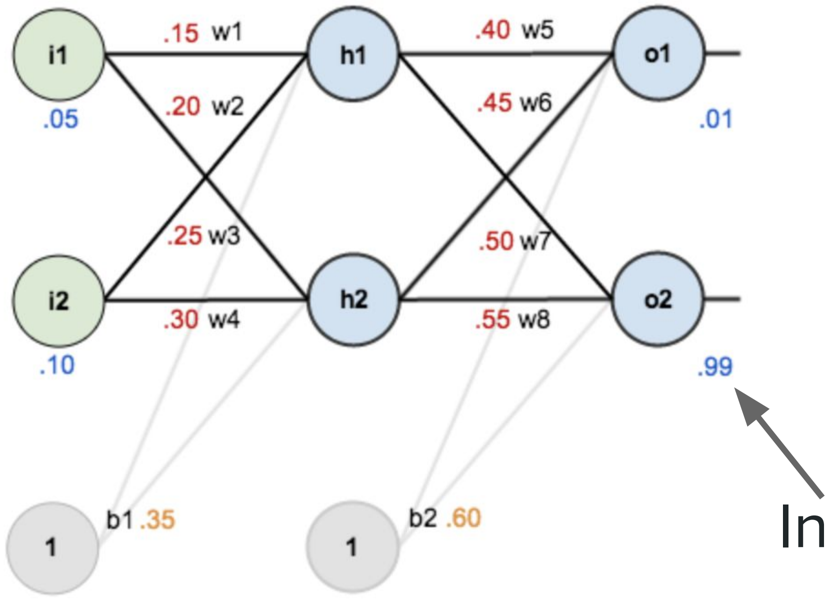


- Input features (predictors): X1, X2
- We want to find weights **W**, activation function **f**, and bias term **b** to model Y (expected output)


Input Layer
Hidden Layer
Output Layer
Bias
1
Bias
1
X1
X2
w0
w1
w2
f(summation) calculated below
f(summation)
f(summation)
f(summation)
Y1
Y2
Output from the highlighted neuron = f(summation) = f(w0 . 1 + w1 . X1 + w2 . X2)

Feedforward Network

Gradient Descent



Step 1. randomly guess initial values for all of the weights

Step 2. for each training instance, make an initial prediction and compute the **sum of squared errors** between the predictions and ground truth values (i.e. the cost function)

Incorrect output

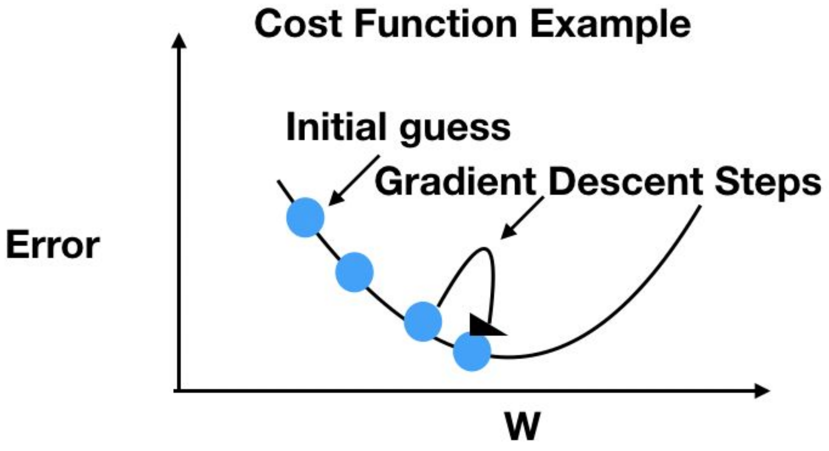
Forward Propagation

source:https://mattmazur.com/2015/03/17/a-step-by-step-backpropagation-example/

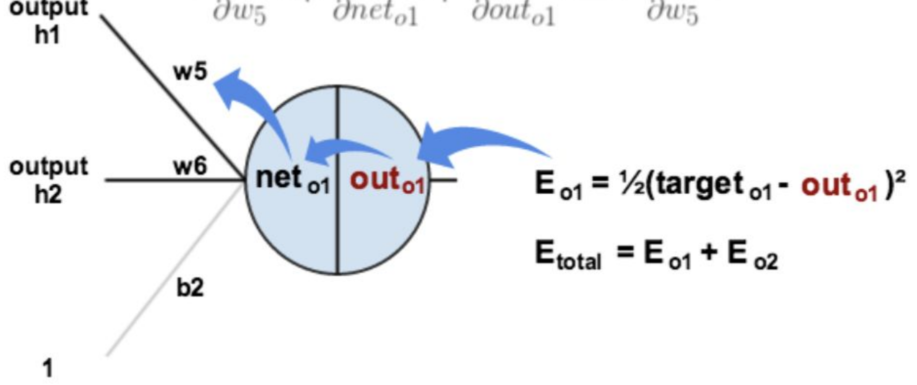
Batch size: # training samples per iteration
Epoch: time to go through all training samples in data

source:https://mattmazur.com/2015/03/17/a-step-by-step-backpropagation-example/

Step 3. In each iteration, compute the gradient of the cost function wrt to each weight and adjust it accordingly



$$\frac{\partial net_{o1}}{\partial w_5} * \frac{\partial out_{o1}}{\partial net_{o1}} * \frac{\partial E_{total}}{\partial out_{o1}} = \frac{\partial E_{total}}{\partial w_5}$$



Back Propagation

$$w_5^+ = w_5 - \eta * \frac{\partial E_{total}}{\partial w_5}$$

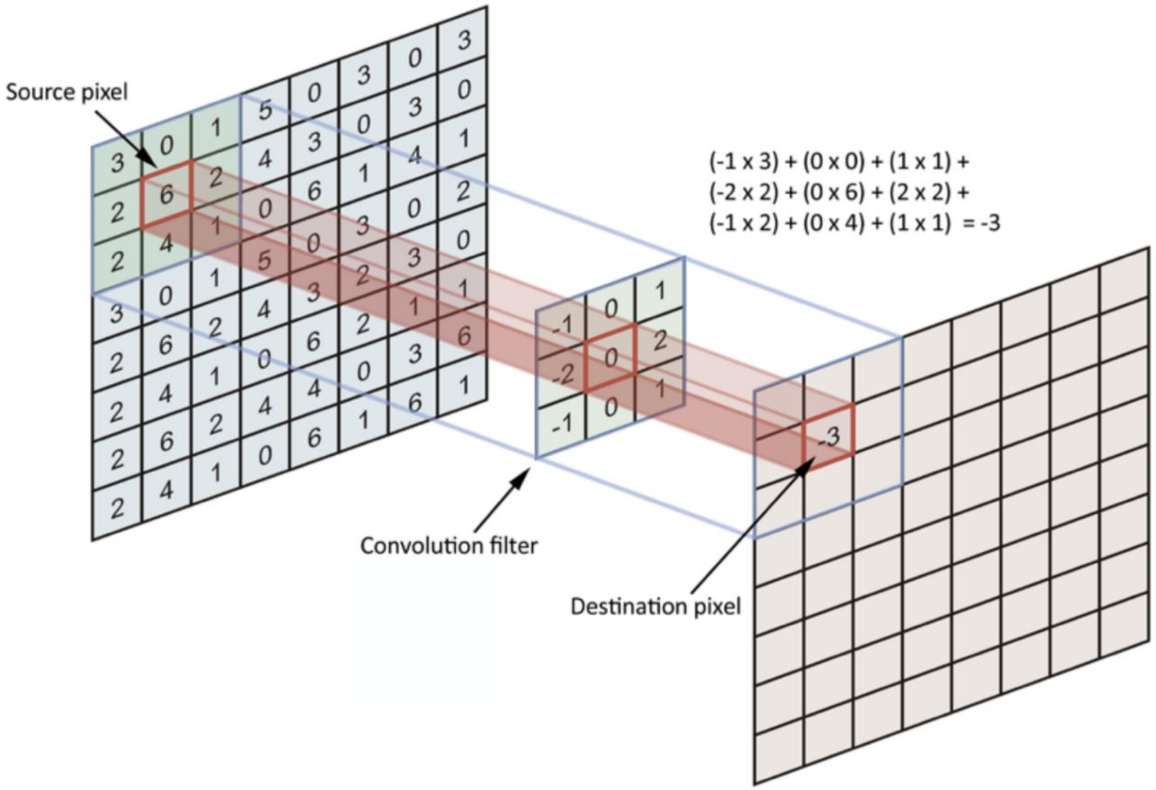
Weights Update

Each *baby step* “downhill” (down the cost function), progresses toward the minimum error

Learning rate (eta): size of the increment downhill

Super Quick Intro to *Convolutional* NNs

Source:https://towardsdatascience.com/simple-introduction-to-convolutional-neural-networks-cdf8d3077bac


Source pixel
(-1 x 3) + (0 x 0) + (1 x 1) +
(-2 x 2) + (0 x 6) + (2 x 2) +
(-1 x 2) + (0 x 4) + (1 x 1) = -3
Convolution filter
Destination pixel

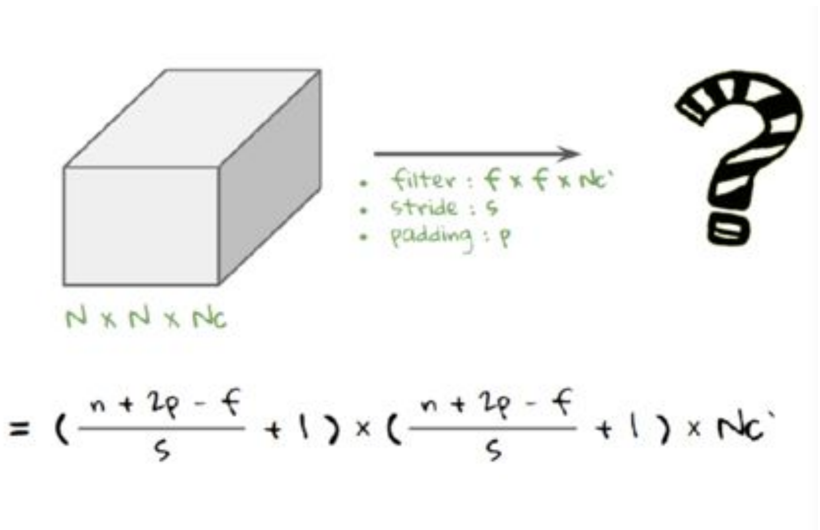
Convolution Operation

Convolutional Filters

- each filter learns patterns used to detect a specific feature in the image
- Each creates abstractions based on the previous layers input
- Beginning layers just look for lines and simple shapes, but as the feature map becomes more and more abstract/complex it captures more detail in the image

CNN Layer Types

- process of building a CNN always involves four major steps:
  - Step - 1 : Convolution
  - Step - 2 : Pooling (eg. Max-pooling, avg pooling)
  - Step - 3 : Flattening
  - Step - 4 : Full connection


filter : f x f x Nc'
stride : s
padding : p
N x N x Nc
$= \left(\frac{n+2p-f}{s}+1\right) \times \left(\frac{n+2p-f}{s}+1\right) \times N_c'$

Size of Output

- **Padding (in convolution)**: giving additional pixels to the input data to preserve edge information (when sliding filter across edges of image map)
- **Stride (in convolution)**: number of rows/cols skipped per shift of convo filter (Higher stride = smaller output)
- **ReLU (activation):** outputs the input if it is positive otherwise it outputs 0
  i. linear transformations would make the neural network simpler, but less powerful and unable

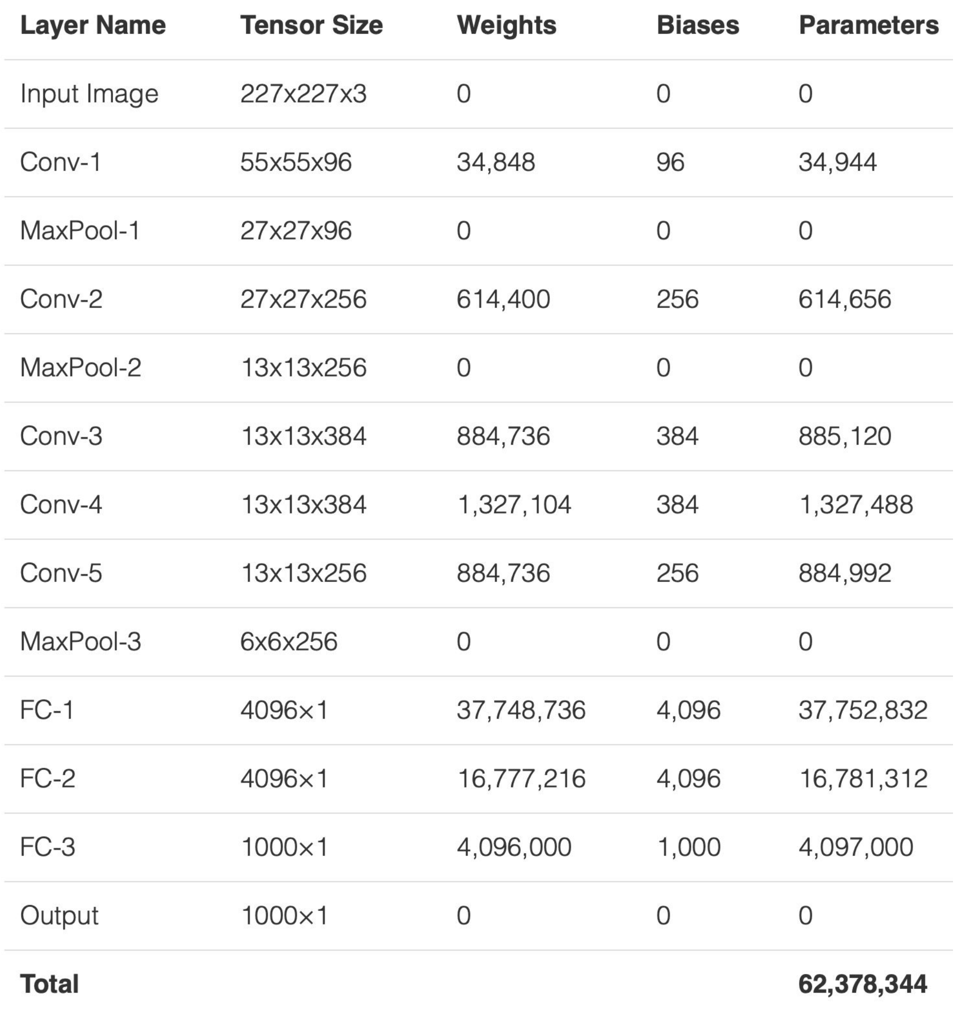
Total CNN Parameters

- # of **parameters** in a layer is the # **learnable** elements for a filter

W_c = Number of weights of the Conv Layer.
B_c = Number of biases of the Conv Layer.
P_c = Number of parameters of the Conv Layer.
K = Size (width) of kernels used in the Conv Layer.
N = Number of kernels.
C = Number of channels of the input image.

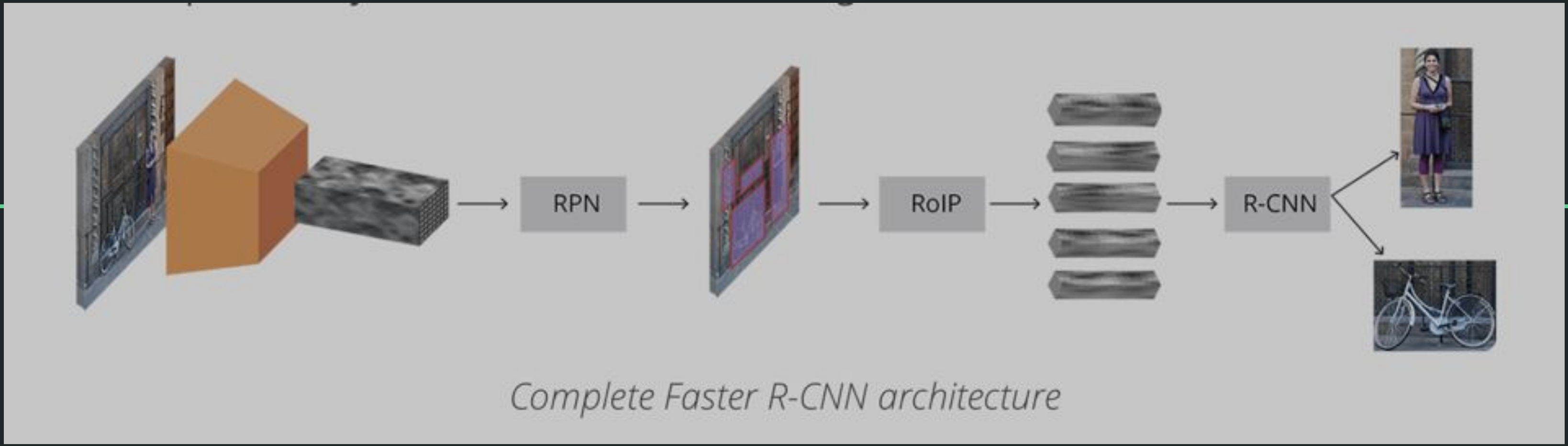
$$W_c = K^2 \times C \times N$$
$$B_c = N$$
$$P_c = W_c + B_c$$

Source: https://www.learnopencv.com/number-of-parameters-and-tensor-sizes-in-convolutional-neural-network/



Layer Name	**Tensor Size**	**Weights**	**Biases**	**Parameters**
Input Image	227x227x3	0	0	0
Conv-1	55x55x96	34,848	96	34,944
MaxPool-1	27x27x96	0	0	0
Conv-2	27x27x256	614,400	256	614,656
MaxPool-2	13x13x256	0	0	0
Conv-3	13x13x384	884,736	384	885,120
Conv-4	13x13x384	1,327,104	384	1,327,488
Conv-5	13x13x256	884,736	256	884,992
MaxPool-3	6x6x256	0	0	0
FC-1	4096×1	37,748,736	4,096	37,752,832
FC-2	4096×1	16,777,216	4,096	16,781,312
FC-3	1000×1	4,096,000	1,000	4,097,000
Output	1000×1	0	0	0
Total				**62,378,344**

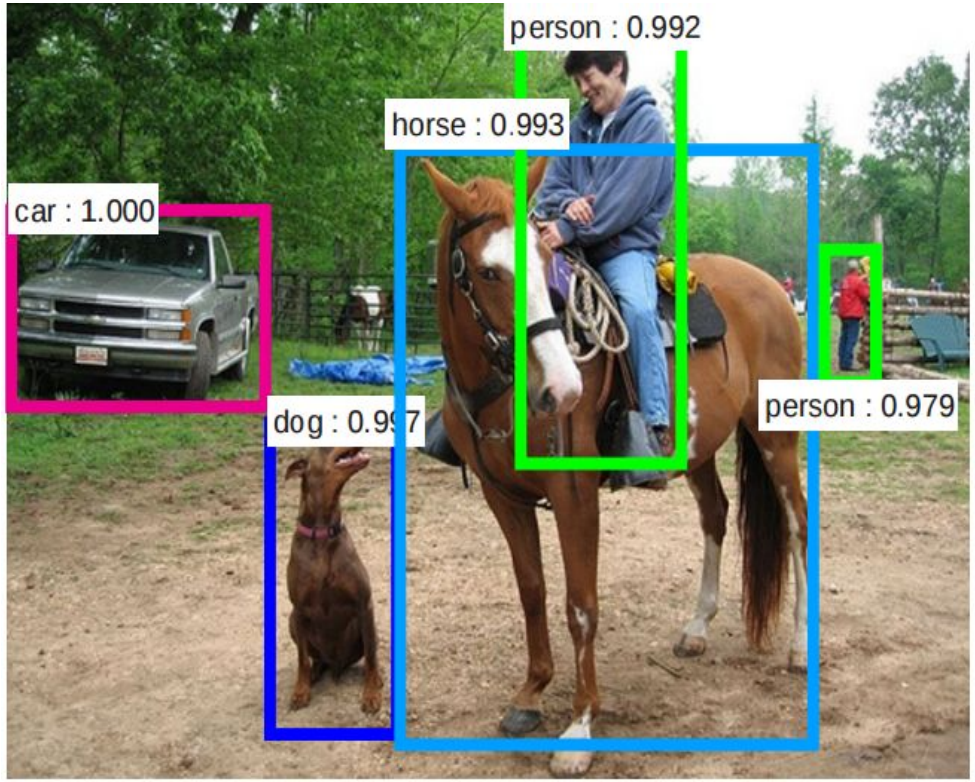
Faster R-CNN (Intro to Deep Learning)



Complete Faster R-CNN architecture

source:https://tryolabs.com/blog/2018/01/18/faster-r-cnn-down-the-rabbit-hole-of-modern-object-detection/

Faster RCNN

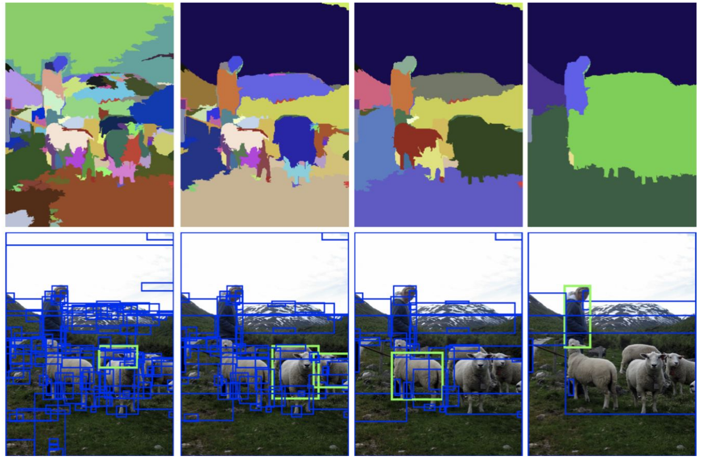


- FRCNN is a popular *object detection network*
  - Other popular methods include RetinaNet, YOLOv3, SSD
- for each image input it returns a list of **bounding boxes** (x1, y1, x2,y2) which correspond with **probabilities** assigned to each **class label**
- It is the third iteration of RCNN (NIPS 2015) which had **Ross Girshick** as author ...

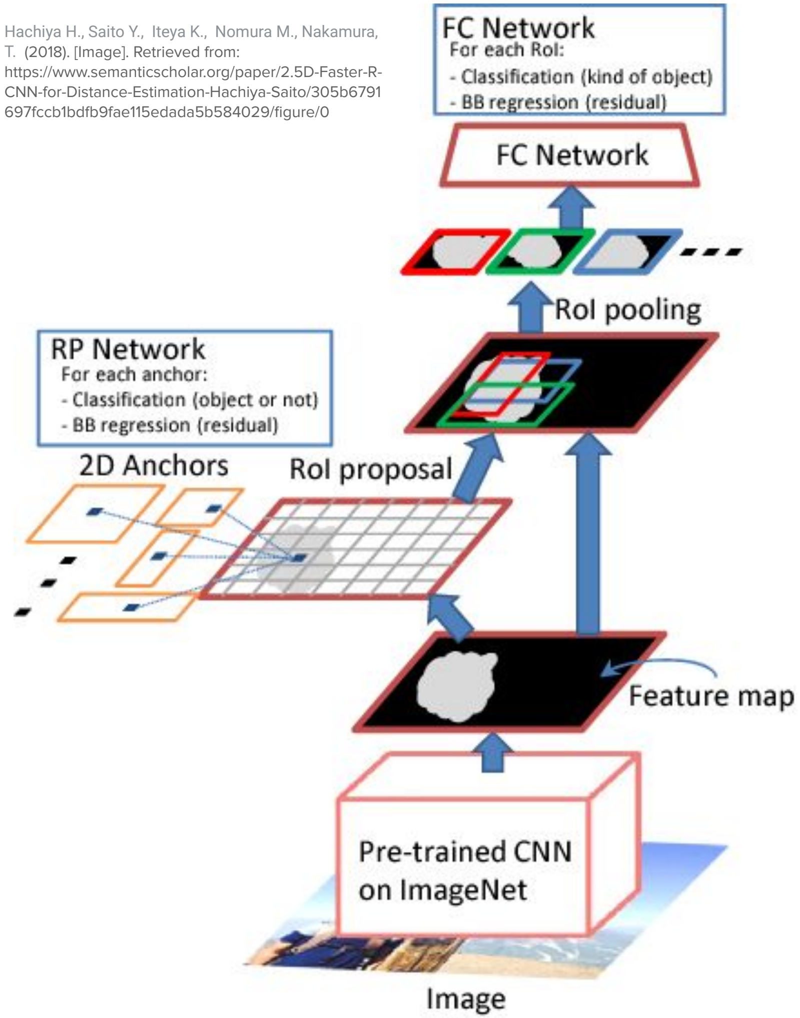
Patil, R. (2018). [Image]. Retrieved from: https://medium.com/@rohitrpatil/how-to-use-tensorflow-object-detection-api-on-windows-102ec8097699

Iterations: R-CNN, Fast - RCNN, and Faster RCNN

- **RCNN** used *selective search* to find regions of interest and then takes the 2000 top proposals and passes each of them to a pretrained **CNN**, then feature maps are passed to **SVM** for classification
    - **Selective search** is a search algorithm that identifies object in an image using region proposals made by grouping together regions in the image with similar colors, textures, and shapes
- **Fast RCNN** improves upon this by passing the input directly into a **CNN** and using selective search on the output feature map. **ROI pooling** is then used to maintain the predefined box size before being passed to a fully connected layers
- **Faster RCNN** replaces selective search with an **Region Proposal Network (RPN)**


hierarchical segmentation process of search selective.
Source: J.R.R. Uijlings and al. (2012)

Hachiya H., Saito Y., Iteya K., Nomura M., Nakamura, T. (2018). [Image]. Retrieved from: https://www.semanticscholar.org/paper/2.5D-Faster-R-CNN-for-Distance-Estimation-Hachiya-Saito/305b6791697fccb1bdfb9fae115edada5b584029/figure/0

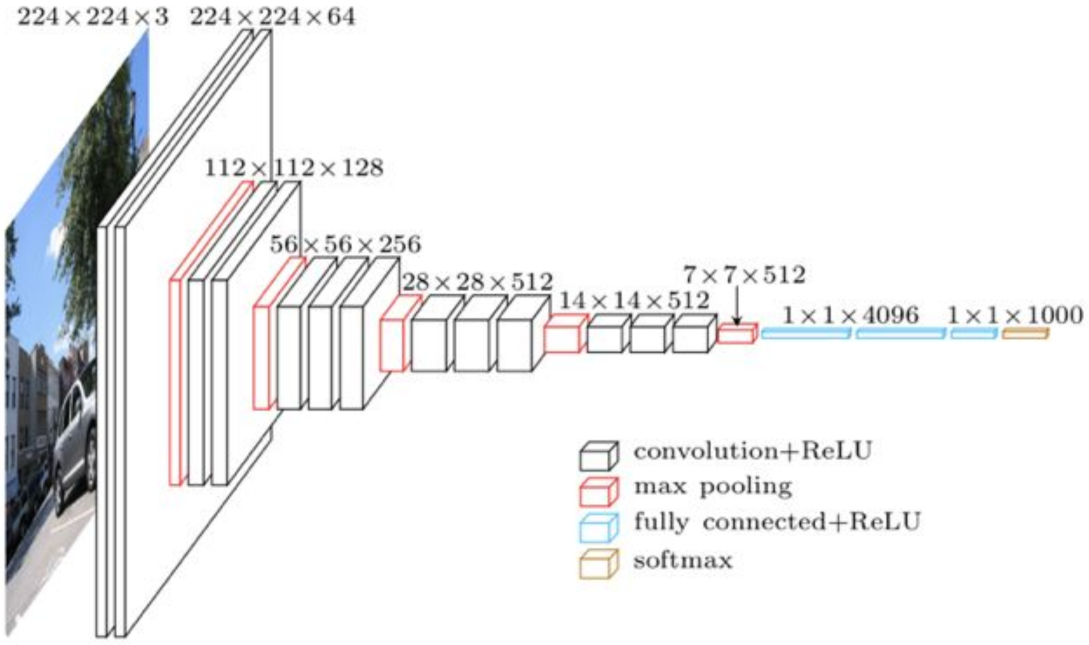

FC Network
For each RoI:
- Classification (kind of object)
- BB regression (residual)
FC Network
RoI pooling
RP Network
For each anchor:
- Classification (object or not)
- BB regression (residual)
2D Anchors
RoI proposal
Feature map
Pre-trained CNN on ImageNet
Image

Basic Components

- First, images are input into a pre-trained base **CNN** as a **H x W x D** tensor
    - We extract the convolutional feature map from an intermediate layer
- Next, we use a **Region Proposal Network (RPN)**
    - BB regression and classification are used to predict the probability that each anchor contains an object
- **RoI Pooling** is then used to obtain only the fixed sized feature maps based the RPNs proposals extracted from the CNN layer
- **R-CNN** is used to better adjust the bounding box based on proposed object

Base CNN

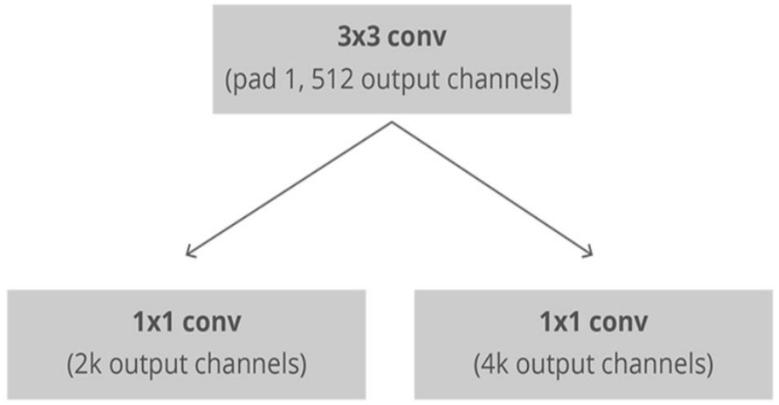
- base CNN is (often) a **VGG-16** network pre-trained on **ImageNet**
    - Various weights,parameters, layers have been tested
- ~9 boxes with different scales and aspect ratios are generated from each position (anchor) of the output feature map produced in the last layer of the CNN
- This produces thousands of boxes that can be used for region proposals generated by the Region Proposal Network (RPN) during training
    - Recycles feature map



VGG architecture

source:https://tryolabs.com/blog/2018/01/18/faster-r-cnn-down-the-rabbit-hole-of-modern-object-detection/

Region Proposal Network (RPN)



RPN Architecture (where k is the number of anchors)

Source:https://towardsdatascience.com/faster-r-cnn-object-detection-implemented-by-keras-for-custom-data-from-googles-open-images-125f62b9141a

- Use of anchors saves much more time than the sliding window approach (where we have to search all pixel locations and at all scales)
- From these it selects a fixed number of boxes before sending them to RCNN (to further reduce and refine)
- simple 3 layer CNN: 1 layer that is fed into 2 sibling fully connected layers used for classification and BB regression (right)
- uses all the anchors selected for the **mini batch** to calculate the classification loss using **binary cross entropy**

RPN (Continued)

- (During training) proposals that overlap the **ground-truth** (GT) objects with an IoU greater than 0.5 are considered foreground, while those with a 0.1 IoU or less are considered background objects (the rest are ignored)
- BB regression is done using the distance vector to the nearest foreground object using **smooth L1 loss**
  - The difference of smooth L1 from L2 and normal L1 is that when the magnitude of the L1 error is less than 1, it is considered almost correct, so it diminishes at a faster rate
- proposals are output as this class probability representing the objectness score paired with a more refined estimation of it's rectangular coordinates

$$L_{RPN}(g_i, f_i) = \frac{1}{N_{cls}} \sum_i L_{cls}(g_i, g_i^*) + \lambda \frac{1}{N_{reg}} \sum_i g_i^* L_{reg}(f_i, f_i^*)$$

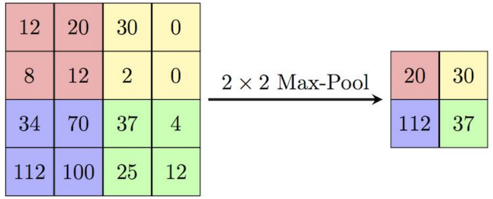
Total RPN Loss Equation

$$L_{1;smooth} = \begin{cases} |x| & \text{if } |x| > \alpha; \\ \frac{1}{|\alpha|}x^2 & \text{if } |x| \leq \alpha \end{cases}$$

Smooth L1 Loss

NMS, ROI Pooling

- **Non-Maximum Suppression (NMS)** ensures there is not redundancy in the proposed regions, retaining only the overlapping boxes with the highest probabilities
- each RoI is defined by a four-tuple (r, c, h, w) sorted by score
    - to balance computation time we retain the top N
        - large enough so plenty will still be classified as background
- **Region of Interest (RoI) pooling** layer takes all of the coordinates provided by the RPN and extracts a fixed sized feature map from the the CNN using **max-pooling**
    - saves computation time (would be very slow doing all N)
    - this fixed sized feature map can be used as input to the classification stage of the detection network to output a probability for fixed number of object classes

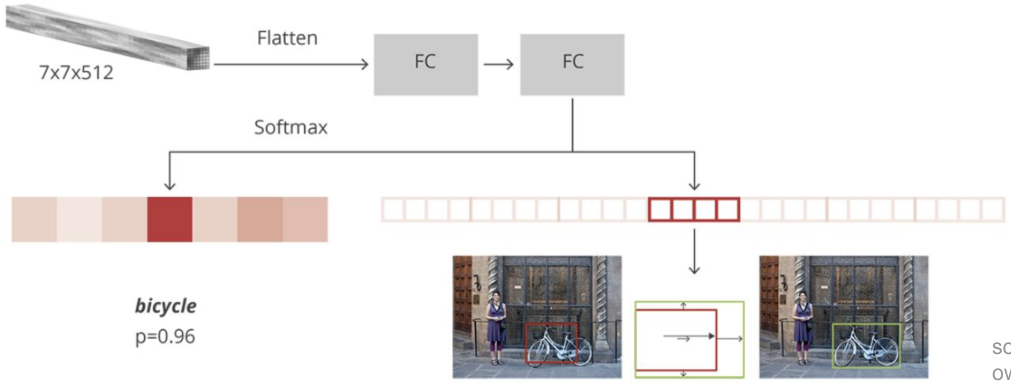


Example of max-pooling

source:https://computersciencewiki.org/index.php/File:MaxpoolSample2.png

RCNN Detection Network

- RCNN uses two fully connected layers to do the final classification of objects and regress bounding boxes based on object class, respectively
- Regions that do not overlap the GT at all are ignored during training, while those that have an **IoU** greater than 0.5 are assigned to a specific object class (not just object or not)
- The offset of the bounding box to that GT is used to better adjust the bounding boxes, using **smooth L1** loss again
- Multi-class **sparse cross entropy** loss is used for classification
- final predicted classes are output from the network's **fully connected** and **soft-max** layers



Architecture

source:https://tryolabs.com/blog/2018/01/18/faster-r-cnn-down-the-rabbit-hole-of-modern-object-detection/

Multi-stream Faster R-CNN

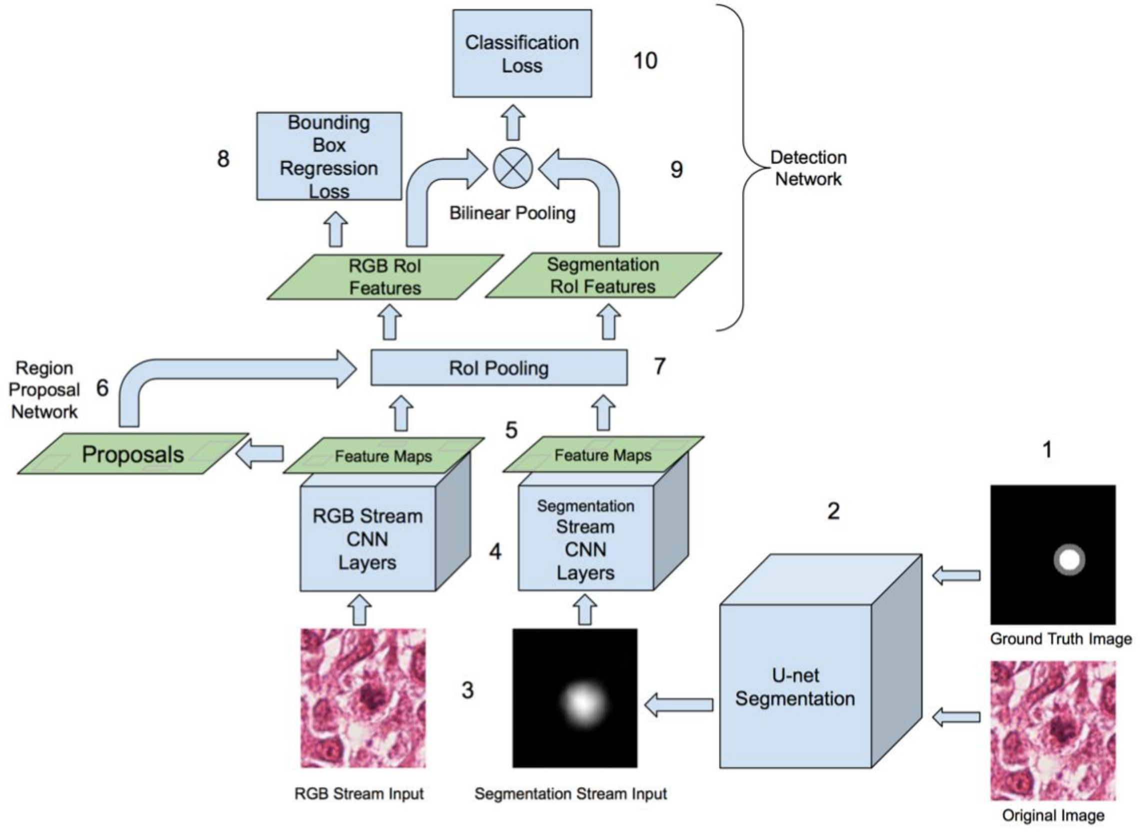
2. Segmented images produced by testing the complete set of training images on **U-net** (half used for training)

6. Only the last layer of the RGB stream CNN is used as input to the RPN, so both streams share these region proposals

7. ROI pooling layer selects corresponding spatial features from each stream and outputs a fixed size feature vector for each proposal.

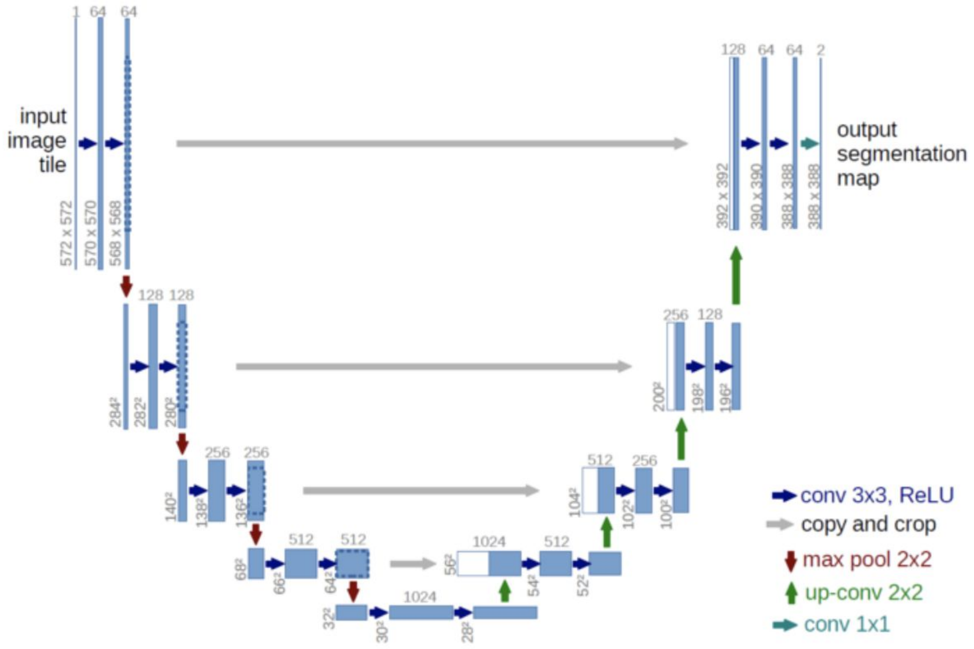
8. RoI features from the RGB stream alone are used for the final mitosis bounding box location prediction

9. Bilinear Pooling used to obtain spatial co-occurrence features from both streams



Multi-Stream Faster RCNN

U-net Segmentation



https://towardsdatascience.com/understanding-semantic-segmentation-with-unet-6be4f42d4b47

- 23-layer network is based on the work of (Ronneberger et al., 2015) winner of ISBI Challenges in biomedical image segmentation and cell tracking
- downsampling side of ‘U’ doubles the number of features in each step
  - uses repeated pairs of 3x3 un-padded convolutions to capture image context before a rectified linear unit (ReLU), followed by a 2x2 max pooling operation with stride two
- upsampling side halves features in each step
  - symmetric expanding path helps to achieve localization
  - uses a 2x2 convolution and two 3x3 convolutions each followed by a ReLU.

The Code :)
(The really fun part)
TBC...

- **Remember** that this week 0 material is all **just for fun** to get you interested in what you can do with machine learning (I won't even test on basic NNs until we do more of them)
  - We will get into the coding for neural networks later in the quarter
- Then, (now that we have a basic understanding of them) we can code some CNNs and/or implement Faster-RCNN after we finish NNs (if we have time/you guys are ready)