CNNs by Andrew Ng NOTES

Link: https://www.youtube.com/playlist?list=PLBAGcD3siRDjBU8sKRk0zX9pMz9qeVxud

GENERAL CNNS

1. What is Computer Vision?

- Computer vision problems
  - Image classification
  - Object detection
  - Neural style transfer
- Large input images
  - If we use standard fully connected matrix, scales horribly
  - Hard to not overfit, hard to train

2. Edge Detection Example

- Standard CNN structure
  - Edges to parts of objects to faces
- First, maybe detect horizontal and vertical edges
- How to detect?
  - Start with 6x6 image
  - Construct 3x3 matrix (filter)
    - [[1 0 -1], [1 0 -1], [1 0 -1]]
  - Convolve image with filter
    - Take matrix and paste at top left edge
    - Take element-wise product
    - Keep sliding filter across image
  - Results in a 4x4 matrix
  - Python: conv-forward
  - TF: tf.nn.conv2d
  - Keras: Conv2D
- Large values mean edge

3. More Edge Detection

- What if light and dark parts are flipped?
  - Can take abs val if it doesn't matter
- Horizontal filter
  - [[1 1 1], [0 0 0], [-1 -1 -1]]
- Possible other filters
  - [[1 0 -1], [2 0 -2], [1 0 -1]]: Sobel filter
  - [[3 0 -3], [10 0 -10], [3 0 -3]]: Scharr filter
  - Could also learn the values by treating them like weights
    - Neural networks can learn higher-level features

4. Padding

- Convolution usually ends up with smaller output matrices
  - Image shrinks on every layer
- (nxn) * (fxf) → (n-f+1 x n-f+1)
- Pixels on corners/edges are used a lot less in convolution
  - Throwing away lots of information
- Pad the image with 1 pixel border of 0s
  - (n+2p-f+1 x n+2p-f+1)
- Can pad with wider borders too
- Valid and Same convolutions
  - Valid: no padding
    - nxn * fxf → n-f+1 x n-f+1
  - Same: pad so output size is the same as input size
    - 2p = f-1
    - f usually odd

5. Strided Convolution

- Convolve with a stride of 2
  - Instead of stepping over 1 step, step 2 steps as filter is slided
  - 7x7 * 3x3 = 3x3
- nxn * fxf with padding p and stride s = (n+2p-f)/s+1 x (n+2p-f)/s+1
- What if fraction not an integer?
  - Take floor
- If stride causes part of filter to go out of image, floor applies
- Cross-correlation vs. convolution
  - Some conventions flip the filter on the horizontal and vertical axes
  - Then, convolve with the flipped matrix
  - By convention, we do not flip
    - This is technically cross-correlation, but is called convolution
  - The mirroring operation makes the convolution op. associative

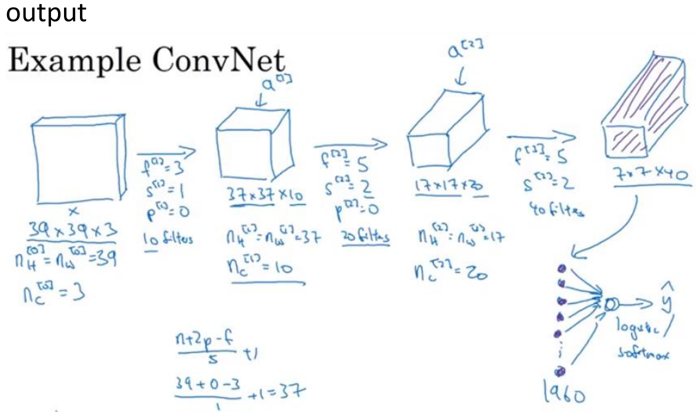
6. Convolutions Over Volume

- Convolutions on RGB images (nxnx3 volumes)
  - Height x width x channels
  - Filter is 3x3x3
    - Results in 4x4 matrix
- Again, take element-wise product, just now across multiple layers
- If we only want to detect edges on one channel, just make other channels have 0 matrices
- What if we want to use multiple filters at the same time?
  - Have more filters (e.g. 2); each outputs a 4x4 output
  - Stack outputs together into 4x4x2 output volume
- Summary: (n x n x n_c) * (f x f x n_c) = n-f+1 x n-f+1 x n_c’
  - n_c and n_c’ are called depth or channels
- Can detect large number of features with this technique

7. One Layer of CNN

- Add a bias to each output matrix of the filters and apply nonlinearity to get final output
  - Nonlinearities like Relu
- Stack up the final output matrices; result in 1 layer of CNN
- Analogy to 1 layer forward propogation
  - z1 = W1*a0+b1
  - a1 = g(z1)
  - Input image = a0, each filter = W1, bias = b1, nonlinearity = g
- 10 filters that are 3x3x3 in one layer of a neural network: how many parameters does that layer have?
  - 27 parameters in each filter
  - +1 for bias gives 28 parameters
  - 10 of these makes 28*10 = 280 parameters
  - Number of parameters fixed with changing input size
    - Less prone to overfitting
- Summary of notation; If layer l is a convolution layer:
  - $f^{[l]}$ = filter size
  - $p^{[l]}$ = padding
  - $s^{[l]}$ = stride
  - $n_c^{[l]}$ = number of filters
  - Each filter is: $f^{[l]} \times f^{[l]} \times n_c^{[l-1]}$
  - Activations: $a^{[l]} = n_H^{[l]} \times n_W^{[l]} \times n_c^{[l]}$, $A^{[l]} = m \times n_H^{[l]} \times n_W^{[l]} \times n_c^{[l]}$
  - Weights: $f^{[l]} \times f^{[l]} \times n_c^{[l-1]} \times n_c^{[l]}$
  - Bias: $1 \times 1 \times 1 \times n_c^{[l]}$
  - Input: $n_H^{[l-1]} \times n_W^{[l-1]} \times n_c^{[l-1]}$
  - Output: $n_H^{[l]} \times n_W^{[l]} \times n_c^{[l]}$
  - $n_H^{[l]} = \text{floor}((n_H^{[l-1]} + 2p^{[l]} – f^{[l]})/s^{[l]} + 1)$; same for width

8. Simple Convolutional Network Example

- Example ConvNet
  - Flatten last activation layer into 1D vector and feed to logistic/softmax unit to get final output
  - 

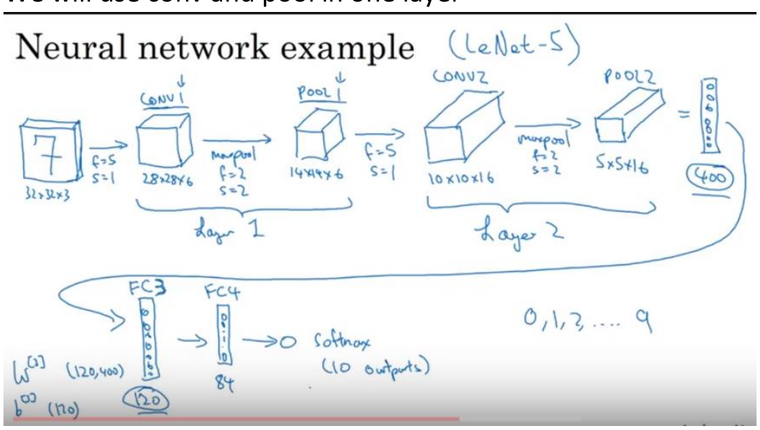
- Types of layer in a CNN

  - Convolution (Conv)
  - Pooling (Pool)
  - Fully connected (FC)

9. Pooling Layers

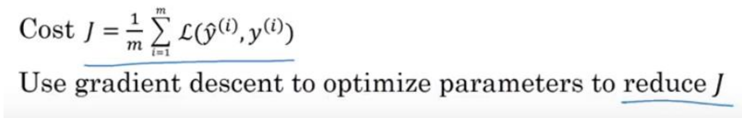
- Max pooling
  - 4x4 → 2x2 by taking max of each quadrant
  - Hyperparameters: f = 2, s = 2
  - Indicates presence of feature within a quadrant
  - No parameters!
- Input 5x5, f = 3, s = 1 → Output 3x3
- If 3D, same as conv layers
  - Perform max pooling on each layer independently
- Another type: average pooling
  - Take mean instead of max
  - Sometimes, used in deep CNN layers to collapse into 1x1xN
- Could use padding hyperparameters, but almost never used!

10. CNN Example

- Some conventions have conv and pool in layer
- Some have them as separate layers
- We will use conv and pool in one layer
- 

11. Why Convolutions?

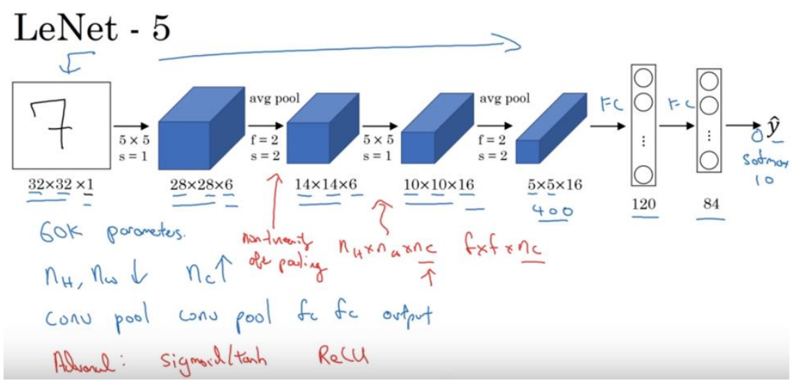
- Parameter sharing and sparsity of connections
- Number of parameters in CNN is much fewer than normal FC layers
- Parameter sharing: A feature detector that's useful in one part of the image is probably useful in another part of the image
- Sparsity of connections: In each layer, each output value depends only on a small number of inputs
- Cost function

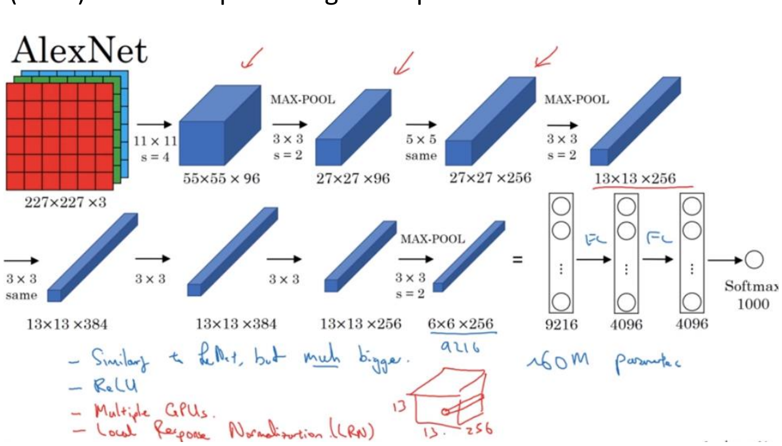
- 

12. Why look at case studies?

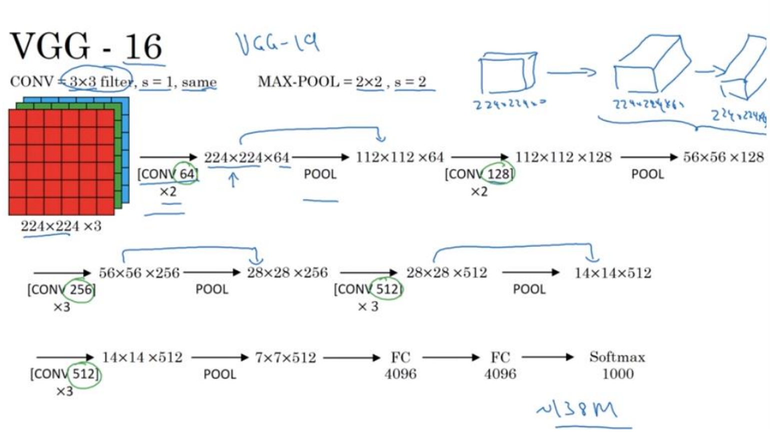
- Classic networks: LeNet-5, AlexNet, VGG
- ResNet (152 layer)
- Inception

13. Classic Networks

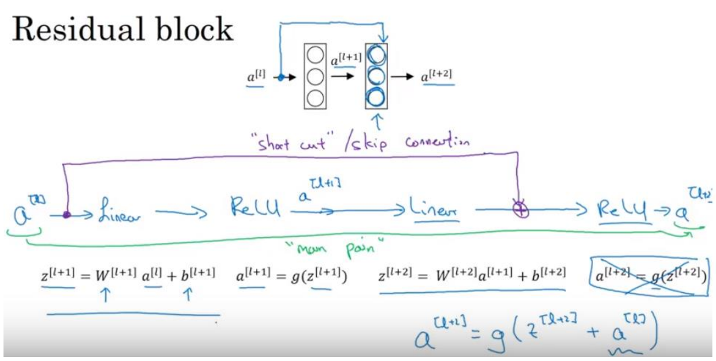
- LeNet-5 (1998)
  - 

- AlexNet (2012): made deep learning seem plausible
  - 

- VGG-16 (2015): simple but large, relatively uniform

- 

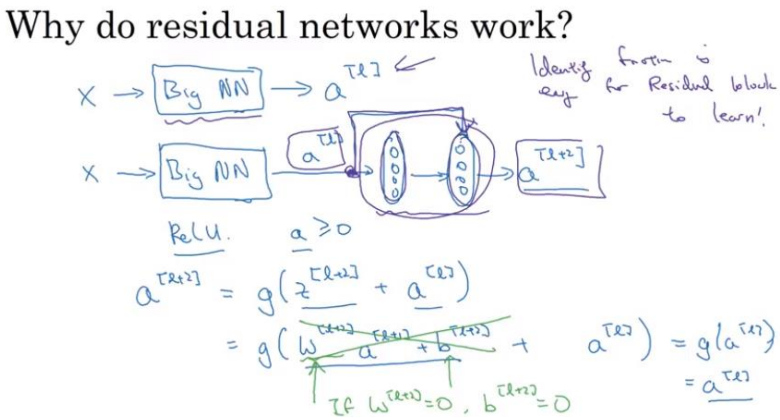
14. ResNet

- Allow very deep CNNs to be trained
- Residual block
  - Short-cut/skip before ReLU part
  - 

- Stack many residual blocks
  - In practice, training plain network that's is very deep causes training error to increase
  - For ResNets, performance keeps on going down as layers increase
    - Helps with vanishing/exploding gradients problem

15. Why ResNets Work

- X → Big NN → $a^{[l]}$
- Add more layers to above with residual
- Identity function is easy for residual block to learn!
- Adding more layers does not hurt NN's ability to learn

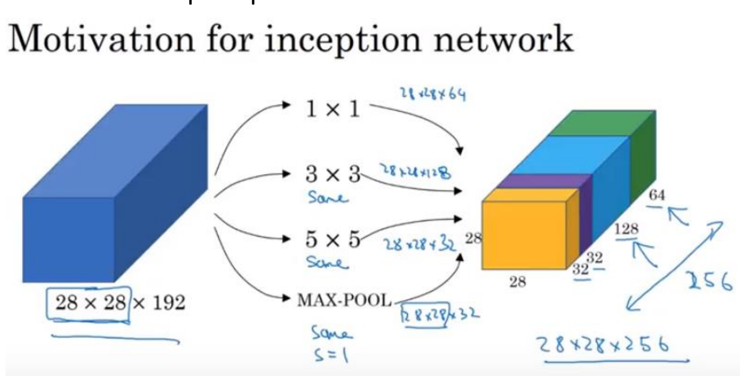
- 

- However, we still have potential to improve performance
- Have to use "same" convolutions to do this technique
    - Or, add a matrix to convert dimensions

16. Network In Network

- 1x1 filter convolutions just scalar multiplies the input with one channel
- If multiple channels, though, then it makes more sense since we end up with a single number for each position with same height and width; also applies a ReLU nonlinearity
- Kind of like having a FC network for each height, width position
- Can shrink number of channels without changing height and width
    - Or, maintain number of channels but add nonlinearity

17. Inception Network Motivation

- Want to try multiple filter dimensions or pooling with padding
    - Stack up outputs from each dimension
- 

- Instead of having to pick filter size/layer type, we can just let network pick
    - However, this adds computational cost
- Can first reduce channels using 1x1 conv and apply filter afterwards to save on computational cost
    - Bottleneck layer
    - Saves cost by around 10x
- Does shrinking channels hurt?
    - Not significantly

18. Inception Network

- Previous activation → (1x1 conv → 5x5 conv), (1x1 conv → 3x3 conv), (1x1 conv), (max pool 3x3, s=1 same → 1x1 conv) → channel concat
- A few additional side branches
  - Takes some hidden layer and tries to make a prediction from there
  - Helps ensure that features even in intermediate layers are not too bad
  - Prevents overfitting
- Name inspired by a meme lol
  - Actually cited by paper

19. Using Open Source Implementation

- Search resnets github on Google
- Click Clone or Download and copy URL
- Git clone URL

20. Transfer Learning

- Can download weights that someone else has already pre-trained
- Can use these as initial weights
- Change the softmax layer from previous implementation to work with your own custom classes
  - Then, freeze all weights except softmax layer!
  - Most frameworks can do this
  - Could also save output of last frozen layer in disk to train softmax
    - Faster computation
- If we have a larger labeled dataset
  - Freeze fewer layers and train the rest
  - Can change the non-frozen layers too
- If we have a lot of data
  - Just use the weights as initialization and train entire network (with modified softmax)

21. Data Augmentation

- Used to improve performance of CV systems
- Common augmentation methods: transform but maintain label
  - Mirroring: flip horizontally
  - Random cropping: take random crops as new images
    - May of may not work, but in practice works well
  - Rotation, shearing, local warping also okay but used less
  - Color shifting: add some (random) constant value to each RGB channel
    - PCA color augmentation: keeps 'overall tint' of the picture the same
- Implementing distortions during training
  - Use CPU thread to implement distortions on loaded pictures to create mini-batch
  - Then, passed on to training
  - Distortions and training can run in parallel

22. State of Computer Vision

- Data vs. hand-engineering
  - Little data to lots of data spectrum
    - Object detection → Image recognition → Speech recognition
  - Simpler algorithms and less hand-engineering for problems with large datasets
  - More hand-engineering ("hacks") for problems with small datasets
  - Two sources of knowledge
    - Labeled data (x, y)
    - Hand engineered features/network architecture/other components
      - Heavier reliance on this because of lack of data due to complexity of problem
      - Transfer learning helps
- Tips for doing well on benchmarks/winning competitions
  - Ensembling: train several networks independently and average their outputs
    - Not weights!
    - Slows down as you add more networks (and takes more memory)
    - Hard to use in production
  - Multi-crop at test time: run classifier on multiple versions of test images and average results
    - 10-crop method (center + 4 corners and mirrored versions) and average results
    - Might help for production systems
    - Also slows down, but does not take too much more memory
- Use open source code
  - Use architectures of networks published in the literature
  - Use open source implementations if possible
  - Use pretrained models and fine-tune on your dataset

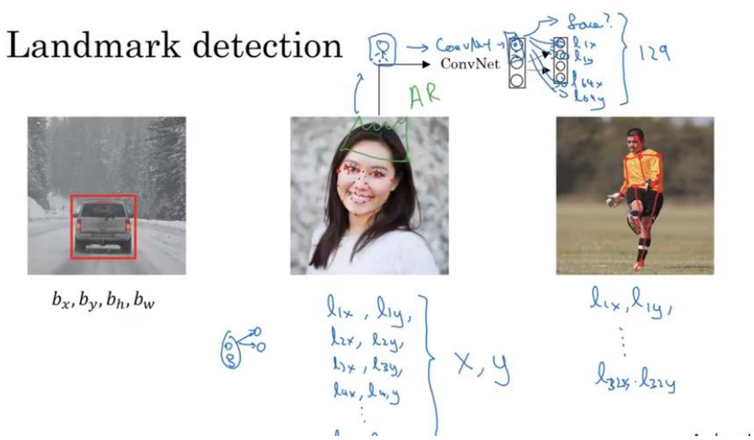
OBJECT DETECTION

22. Object Localization

- Problem types
  - Image classification: algorithm looks at picture and outputs class
  - Classification with localization: algorithm must also bound the object within image in addition to giving output
  - Detection: Deal with multiple objects and localize them all
- Can have network output 4 more numbers bx, by, bh, bw to parameterize bounding box of object (center, height, and width)
  - (0,0) at top left of image, (1,1) at bottom right
- Defining the target label y
  - 1: pedestrian, 2: car, 3: motorcycle, 4: background
  - Need to output bx, by, bh, bw, class label (1-4)
  - $y = [p_c, b_x, b_y, b_h, b_w, c_1, c_2, c_3]$ where p_c is the probability of there being an object
  - If $p_c = 0$, then rest of output is 'don't care's
- Loss function

  - $L(\hat{y}, y)$ = sum of square error between y^ and y if $y_1 = 1$
  - $L(\hat{y}, y)$ = only square error of y_1 if $y_1 = 0$
    - Don't care about remaining outputs
  - Could use other loss (like likelihood loss)

23. Landmark Detection

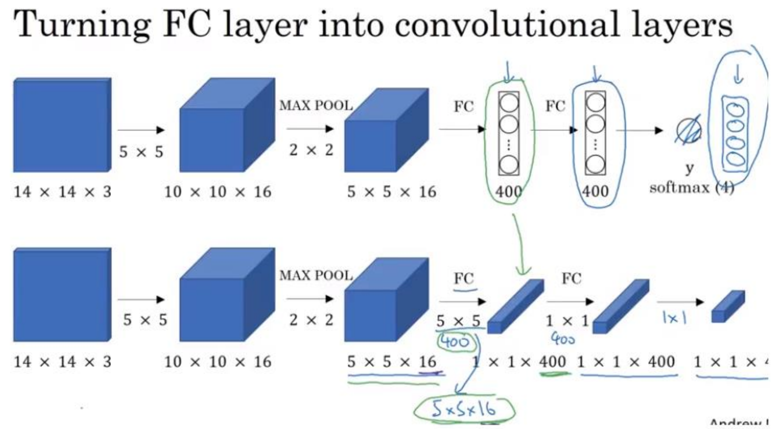
- Could also just find landmarks on the image (points of interest)
- Conv net to output presence of face and locations of all landmarks
- Need labeled training set with annotated landmarks
- Pose prediction: could also annotate key positions on person
- 

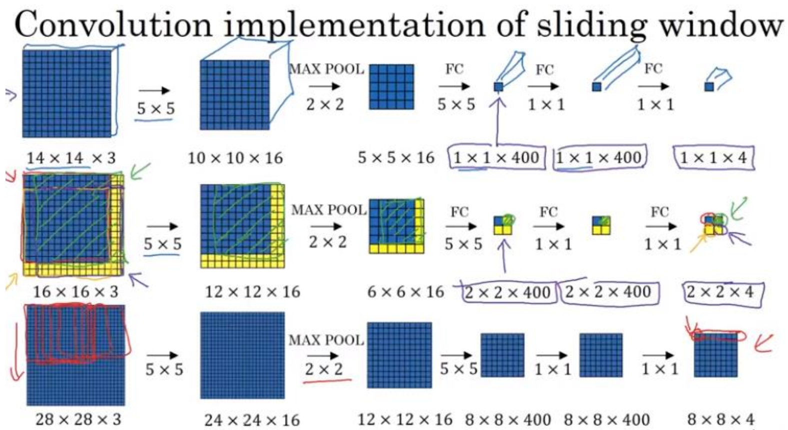
25. Object Detection

- Train ConvNet to identify cropped images to use in sliding windows detection
- Pick a window size and input into the ConvNet cropped images of the same size, sliding across the image
- Repeat with slightly larger window
- And again
- Hope that if we do this, then the car will be detected by some window
- Huge disadvantage: computational cost
  - Running so many cropped images independently through CNNs

26. Convolutional Implementation Sliding Windows

- Turning FC layer into convolutional layers
  - Implement as 400 5x5 filters to get 1x1x400 instead of FC layer

  - 

- Convolution implementation of sliding windows
  - If input image is larger than expected, then we can rerun the CNN on each corner of image
  - Or, we could just run the larger image through original network and get left with a slightly larger output, where each value of output gives what you would get from previous method
  - Combines all FC computations into one computation
  - 

  - Stride of 2 from max pool size

27. Bounding Box Predictions

- Sliding windows are more efficient, but bounding boxes not very accurate
- Ground truth might have non-square bounding boxes
- YOLO algorithm (you only look once)
  - Place grid on top of image
  - Apply image localization to each grid cells
  - For each grid cell: y = [p_c, b_x, b_y, b_h, b_w, c_1, c_2, c_3]
  - Only grid cell containing midpoint contains the object
  - Target output 3x3x8 for example (3x3 grid, 8d y vector)
- Precise bounding box outputs
- Multiple objects in grid cells will interfere with accuracy
- Very fast due to convolutional implementation

- Specify the bounding boxes
  - Specified relative to grid cell size
    - b_x and b_y must be between 0 and 1
    - b_h and b_w could be more than 1

28. Intersection Over Union

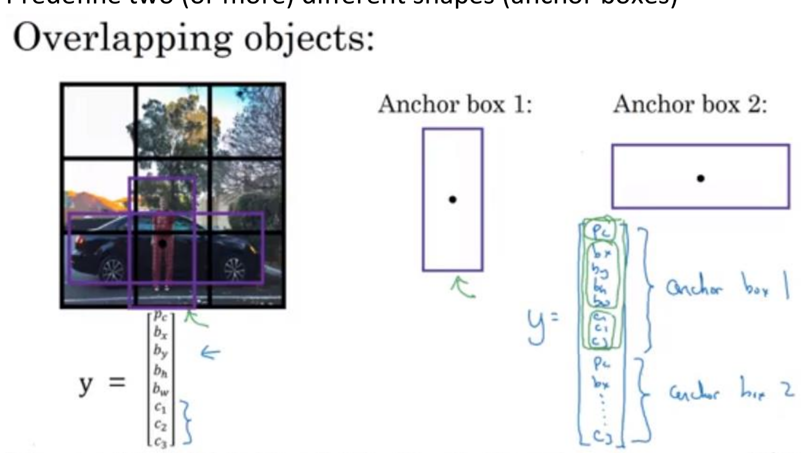
- Evaluating object localization
  - Intersection over union (IoU)
    - Literally take intersection of output and target bounding box over the union
    - If IoU >= 0.5, should be okay
  - More generally, IoU is a measure of the overlap between two bounding boxes

29. Non-max Suppression

- Could detect object more than once
- Looks at probabilities associated with each detection
  - Discard all boxes with p_c <= 0.6 (low probability boxes)
  - Takes largest one first
  - Suppress all rectangles with high overlap (IoU >= 0.5)
  - Repeat with remaining rectangles

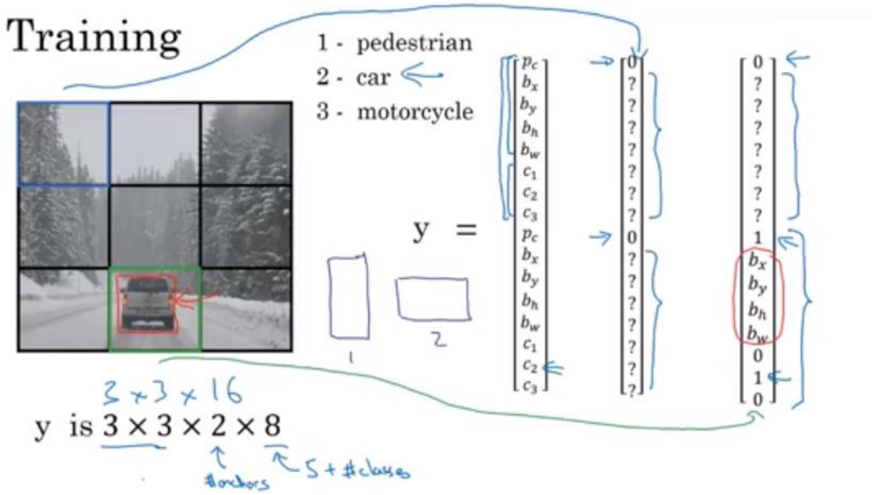
30. Anchor Boxes

- Deal with multiple objects in one grid cell
  - Overlapping objects
- Predefine two (or more) different shapes (anchor boxes)



-
- Previously, each object in training image is assigned to grid cell that contains that object's midpoint
- With two anchor boxes, each object in training is assigned to grid cell that contains object's midpoint and anchor box for the grid cell with highest IoU
- Allows algorithm to specialize better to detect certain types of anchor box shapes
- Could use k-means algorithm to choose anchor box shapes

31. YOLO Algorithm

- Training
  - 

- Making predictions
- Output non-max suppressed outputs
  - For each grid cell, get 2 predicted bounding boxes
  - Get rid of low probability predictions
  - For each class, use non-max suppression to generate final predictions

32. Region Proposal

- R-CNN (regions with CNNs)
  - For sliding windows, only select a few windows
  - Segmentation algorithm to figure out what could be objects
  - Find maybe 2000 blobs
  - Classify each region once at a time; output label and bounding boxes
  - Still quite slow
- Fast R-CNN
  - Propose regions (bottleneck step)
  - Then, use convolution implementation of sliding windows to classify all the proposed regions
- Faster R-CNN
  - Use CNN to propose regions

FACE RECOGNITION

33. What is Face Recognition?

- Liveness detection in conjunction with face recognition
- Face verification vs. face recognition
  - Verification
    - Input image, name/ID
    - Output whether the input image is that of the claimed person
  - Recognition
    - Has a database of K people

        - Get an input image
        - Output ID if the image is any of the K persons (or “not recognized”)

34. One-shot Learning

- Need to recognize image given just one example of person’s face
- Learning from one example to recognize the person again
- Retrain each time new person joins? Not feasible.
- Learn a “similarity” function
    - d(img1, img2) = degree of difference between images
    - If d <= tau (a threshold), then output “same”
    - Otherwise, output “different”
- For recognition, do this for every face in database
- Adding new people to database does not require retraining

35. Siamese Network

- Feed pictures to same network to get output vector of n parameters
- Define $d(x1, x2) = |f(x1)-f(x2)|_2^2$
- How to train?
    - Parameters of NN define an encoding $f(x^{(i)})$
    - Learn parameters so that:
        - If $x^{(i)}$, $x^{(j)}$ are the same person, $|f(x^{(i)})-f(x^{(j)})|^2$ is small
        - Otherwise, it should be large

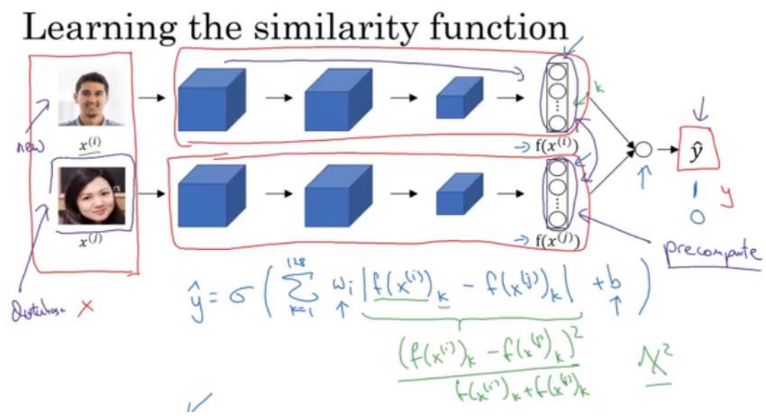
36. Triplet Loss

- Want anchor image to be similar to positive images and different from negative image
    - A, P, N
- Want: $|f(A)-f(P)|^2 <= |f(A)-f(N)|^2$
    - $|f(A)-f(P)|^2 - |f(A)-f(N)|^2 <= 0$
    - Could just output f = 0 to trivially solve this
    - Or f = k for every image
- Modify objective:
    - $|f(A)-f(P)|^2 - |f(A)-f(N)|^2 + alpha <= 0$
    - Alpha is the margin
- Loss function
    - Given 3 images A, P, N:
        - $L(A, P, N) = max(|f(A)-f(P)|^2 - |f(A)-f(N)|^2 + alpha, 0)$
    - Overall loss J = sum{i=1 to m}[$L(A^{(i)}, P^{(i)}, N^{(i)})$]
    - Training set: 10k pictures of 1k persons
        - Need multiple pictures of the same person
- How to choose triplets?
    - During training, if A, P, N are chosen randomly, d(A, P) + alpha <= d(A, N) is easily satisfied
    - Choose triplets that’re “hard” to train on

  - Maybe choose d(A, P) approx. d(A, N)
- Use gradient descent to minimize J
  - Will have effect of backpropagating
- Lots of pre-trained models online using very large data sets

37. Face Verification and Binary Classification

- Instead of triplet loss, input embeddings into logistic regression to output 1s and 0s



- Precompute for all images in database to save memory and computation time
- Train using supervised learning and pairs of images as inputs

NEURAL STYLE TRANSFER

38. What is it?

- Recreate image in style of another input image
  - C = content image, S = style image, G = generated image

39. What are deep ConvNets learning?

- Pick a unit in layer 1; find the nine image patches that maximize the unit's activation
- Repeat for other units
- Do the same thing for later layers
- Features get more complicated as we get deeper in network

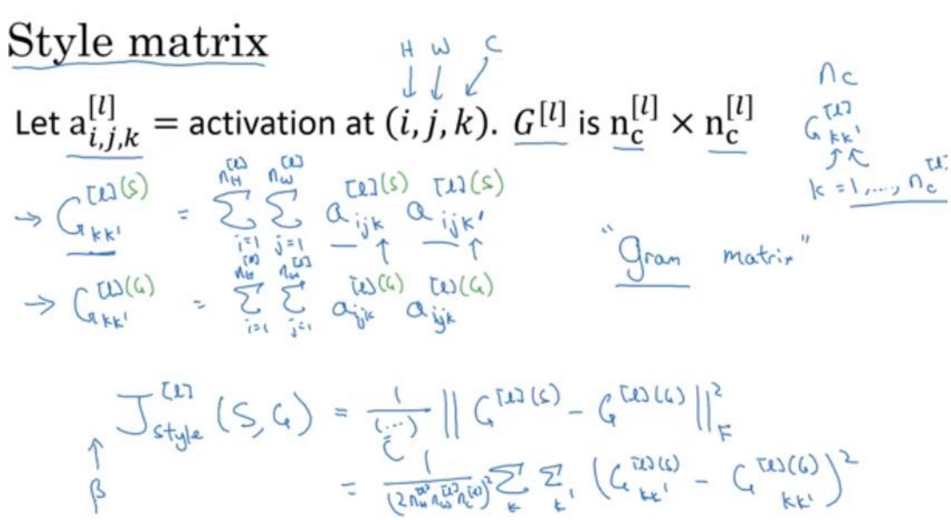
40. Cost Function

- Given C and S generate G
- Minimize a loss J(G) using gradient descent
  - $J(G) = alpha*J_{content}(C, G) + beta*J_{style}(S, G)$
- Find the generated G
  - 1. Initialize G randomly
  - 2. Use gradient descent to minimize J(G)
    - $G = G – partial_G J(G)$

41. Content Cost Function

- Use hidden layer l to compute content cost
- Usually choose a intermediate layer
    - Too early on will make images too similar
    - Too deep will make images too different
- Use pre-trained ConvNet (e.g. VGG network)
- If activations of C and G at layer l are similar, both images have similar content
- $J_{content}$ (C, G) = ½* | $a^{[l](C)} - a^{[l](G)}$ |^2

42. Style Cost Function

- Say we are using layer l's activation to measure "style"
- Define style as correlation between activations across channels
    - Whenever a particular feature appears, other features may tend to appear with it
    - Thus, using degree of correlation between channels will allow us to get certain types of features occurring at the same time
- Style matrix
    - Actually using unnormalized co-covariance, not correlation
    - Calculate correlation matrix for both S and G
    - Style cost is therefore difference between these two style matrices
    - 

- Better if we use from multiple layers!
    - Sum up loss from every layer with some weight $\lambda^{[l]}$

FINAL REMARKS

43. 1D and 3D Generalization of Models

- We learned about 2D convolution with multiple channels
- Similar idea can be applied to 1D data
    - Convolve with a 1D filter by sliding across data
    - Usually use RNNs
- Same with 3D data
    - Just use a 3D filter and slide across everything