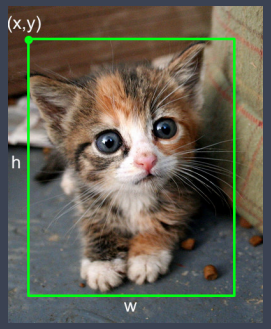
» Fun Ways To Use ConvNets

* Object Detection



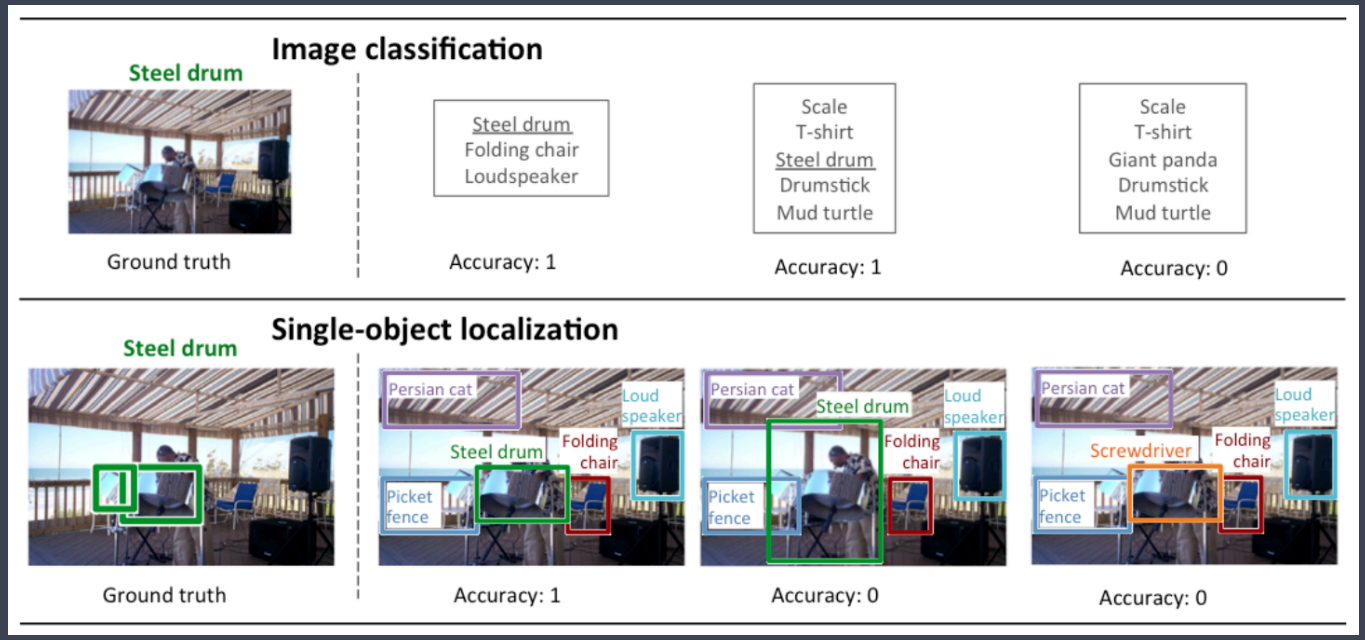
* Face Recognition

* Style Transfer

* Content of this lecture is optional - won't be examined

» Object Detection

* Many of the classic ConvNets were developed to win the ImageNet competition (now finished). ImageNet had two main challenges:

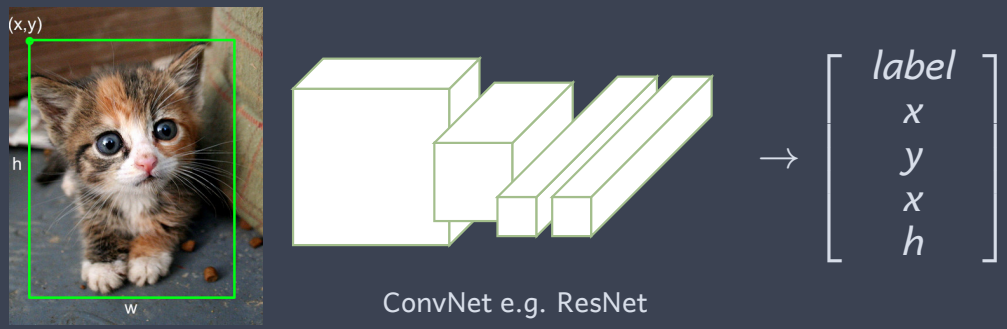


source: ImageNet Large Scale Visual Recognition Challenge https://arxiv.org/pdf/1409.0575.pdf

* In image classification the task is to identify a single object in an image.
* In single-object localization the task is to identify a single object plus its location/bounding box → how to carry out bounding box prediction?
* What about multiple objects?

» Single-object Localization

* Bounding box prediction $\rightarrow$ just extend the output vector!



* Need to modify cost function used for training:

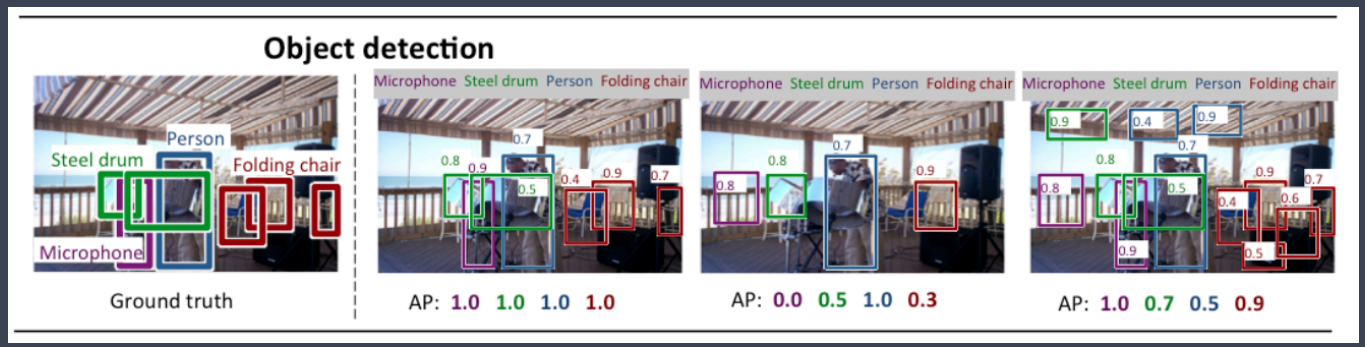
$$J(\theta) = J_{classification}(\theta) + \alpha J_{bounding\ box}(\theta)$$

e.g. use logistic loss for $J_{classification}(\theta)$ and square error for $J_{bounding\ box}(\theta)$, α is a hyperparameter balancing accuracy of classification vs accuracy of bounding box.

*

» Object Detection

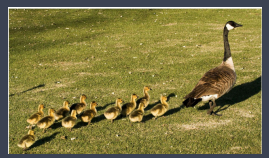
* Multiple objects → *object detection* task



source: ImageNet Large Scale Visual Recognition Challenge https://arxiv.org/pdf/1409.0575.pdf

* We could extend the ConvNet output to include multiple objects, but how many to include?

→ $\begin{bmatrix} dog(x, y, h, w) \\ dog(x, y, h, w) \\ cat(x, y, h, w) \end{bmatrix}$

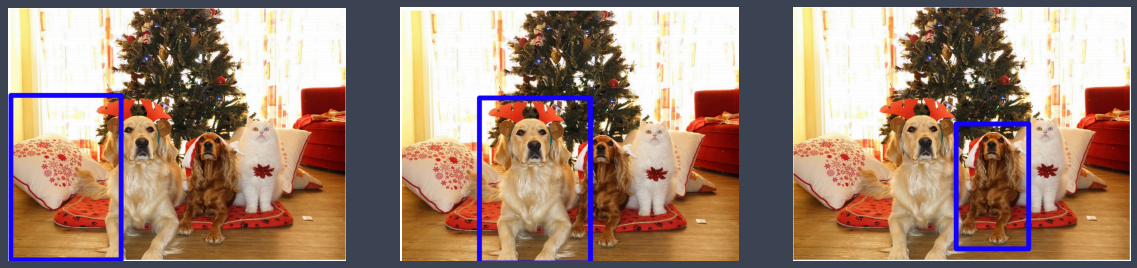


→ $\begin{bmatrix} goose(x, y, h, w) \\ goose(x, y, h, w) \\ \vdots \end{bmatrix}$

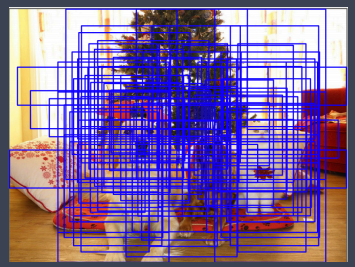
* Each image needs a different number of outputs

» Object Detection: Sliding Window

* Idea: define a box and gradually slide it across the image. At each position pass contents of box into ConvNet to classify any object in box



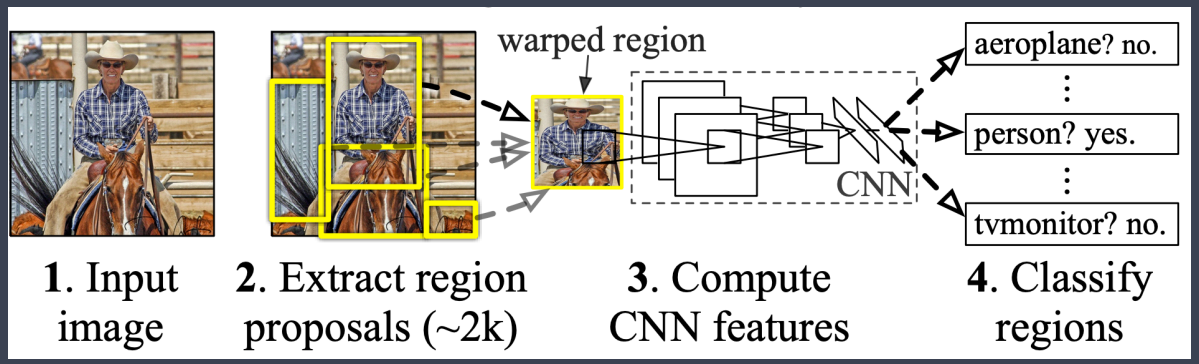
* Problems this *sliding window* approach:
  * Need different sizes and shapes of boxes (since objects can be big/small, wide/tall etc) and to scan across many positions $\rightarrow$ a *lot* of boxes to look at:



  * Need to run full ConvNet for each box shape and position $\rightarrow$ computationally expensive and v slow.

» Object Detection: Region Proposals

* Idea 1 *R-CNN*: Use some method to propose regions of interest in image, then feed each of these into a second ConvNet to classify any object

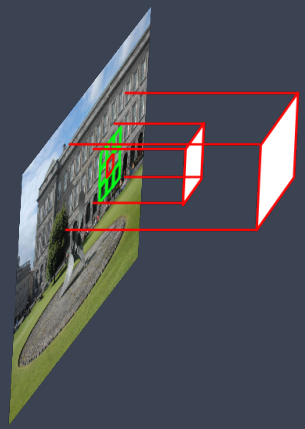


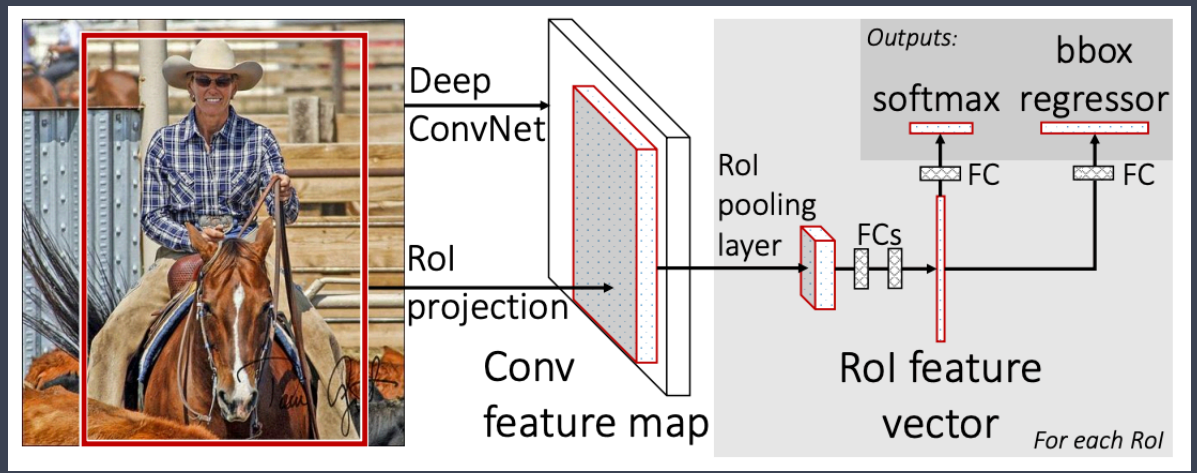
Rich feature hierarchies for accurate object detection and semantic segmentation https://arxiv.org/pdf/1311.2524v5.pdf

* But typically have 1-2K regions and so need to run classifier ConvNet 1-2K times, still v slow

» Object Detection: Region Proposals

* Idea 2 *Fast R-CNN*:
  1. Use some method to propose regions of interest in image
  2. Run whole image through a ConvNet to extract feature tensor after applying a fewer layers of processing.
  3. Map proposal regions onto these areas within feature tensor $\rightarrow$ remember each element of tendsor can be mapped back to a receptive field in input image
  4. Use max-pooling to downsample each area to a constant size.
  5. Flatten features for each region and estimate label and bounding box





Fast R-CNN https://arxiv.org/pdf/1504.08083.pdf

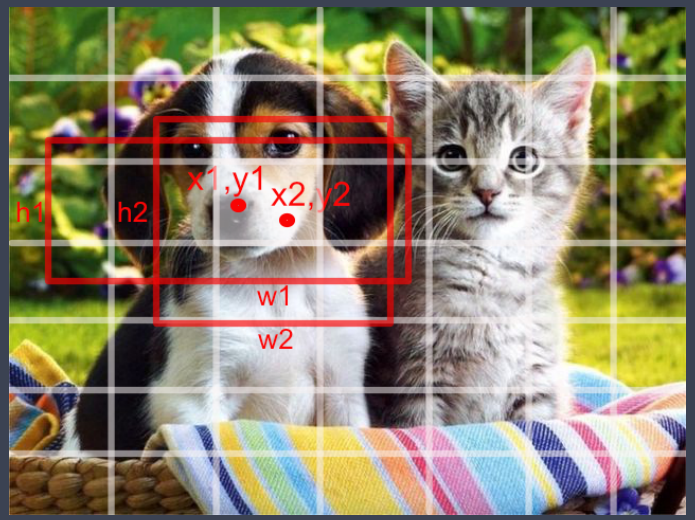
* Faster, but still pretty slow.

» One-Shot Object Detection

* Idea 3 *Faster R-CNN*: Towards Real-Time Object Detection with Region Proposal Networks https://arxiv.org/abs/1506.01497. Uses a ConvNet to make region proposals at feature layer - introduced idea of *anchor boxes*.
* All of these approaches are two-stage approaches: (i) run once per image to extract proposal regions, (ii) run once per region to classify object and predict bounding box
* Can we do this all in one pass?
* *YOLO* You Only Look Once: Unified, Real-Time Object Detection https://pjreddie.com/media/files/papers/yolo.pdf
* *SSD* SSD: Single Shot MultiBox Detector https://arxiv.org/abs/1512.02325

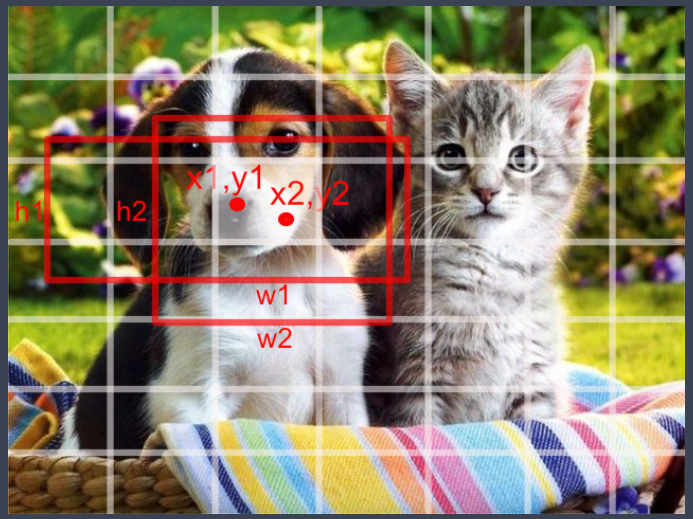
» One-Shot Object Detection

* Divide image up into e.g. a 7×7 grid of cells
* Associate a small set of B bounding boxes with each cell e.g. $B = 2$ or $B = 3$



* Bounding box i has (i) 4 location/size parameters x_i, y_i, h_i, w_i, (ii) vector c_i of confidences for C class labels i.e. if have $C = 80$ object labels then c_i is a vector of 80 values plus (iii) indicator o_i whether box contains an object or not
  * Bounding box centre point x_i, y_i is constrained to lie within grid cell
  * Height and width h_i, w_i can be bigger (perhaps much bigger) than grid cell
* Output from each grid cell is a $B \times (4 + C + 1)$ vector
* Overall output of ConvNet is a $7 \times 7 \times B(4 + C + 1)$ tensor

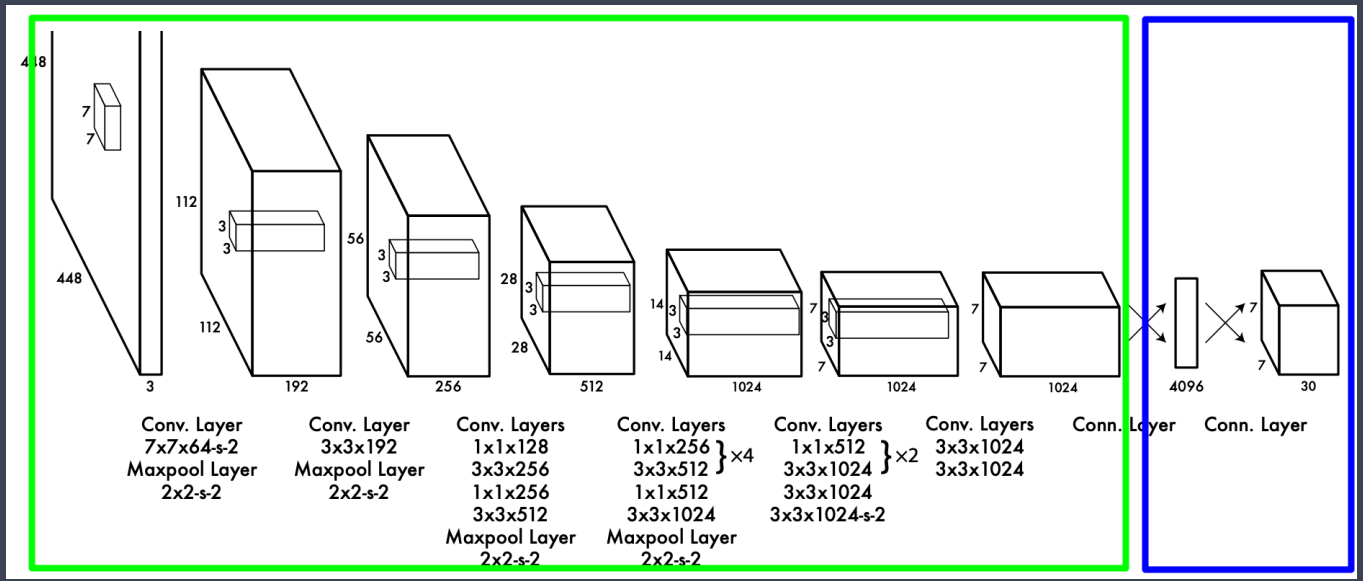
» One-Shot Object Detection



- Output from each grid cell is a $B \times (4 + C + 1)$ vector. Overall output of ConvNet is a $7 \times 7 \times B(4 + C + 1)$ tensor
- Now define cost function in terms of these variables:
  - For object in image pick grid cell closest to centre of object, say cell i.
  - Target objectness indicator $O_i = 1$ for this grid cell and $O_j = -1$ for other grid cells $j \neq i$.
  - Target class label C_i =object class, target bounding box parameters X_i, Y_i, H_i, W_i are equal to true bounding box.
  - Use logistic regression cost function for objectness indicator and square error for x_i, y_i, h_i, w_i, c_i i.e. $(X_i - x_i)^2 + (Y_i - y_i)^2 + \cdots$. Note: don't count square error for cells $j \neq i$.
  - Repeat for all objects in image and sum to get overall image cost function
- The effect is to "assign" one grid cell to be responsible for each object and then try to train ConvNet to reflect that.

» One-Shot Object Detection

* SDD uses VGG-16 to extract features (accurate, but big and slow)
* YOLO uses its own ConvNet (based on ResNet) to extract features: 24 layers in YOLOv1, 53 layers in YOLOv3.



YOLOv1 https://pjreddie.com/media/files/papers/yolo.pdf ($C = 20$ classes, $B = 2$ bounding boxes and uses same class vector for all boxes in a grid cell so $B \times (4 + 1) + C = 30$ output from each grid cell)

YOLO training is two-stage (i.e. uses transfer learning):

1. ConvNet trained for classification on ImageNet data. Add temporary output layer for this (pooling plus FC-layer)
2. FC-layers are used to map from ConvNet features to output tensor. Holding ConvNet parameters fixed train these FC-layers using object detection training data.

* Similarly SDD: VGG-16 ConvNet trained on ImageNet, then output layer trained for object detection

» One-Shot Object Detection

Extra details:

* *Non-max suppression*. Typically an object spans several grid cells and so several grid cells may propose bounding boxes for the same object:
  * Threshold grid cell/bounding box on confidence value – only keep predictions with confidence above a specified threshold.
  * Can also take overlap between predicted bounded boxes into account too (for overlapping bounding boxes keep just the box with highest confidence)

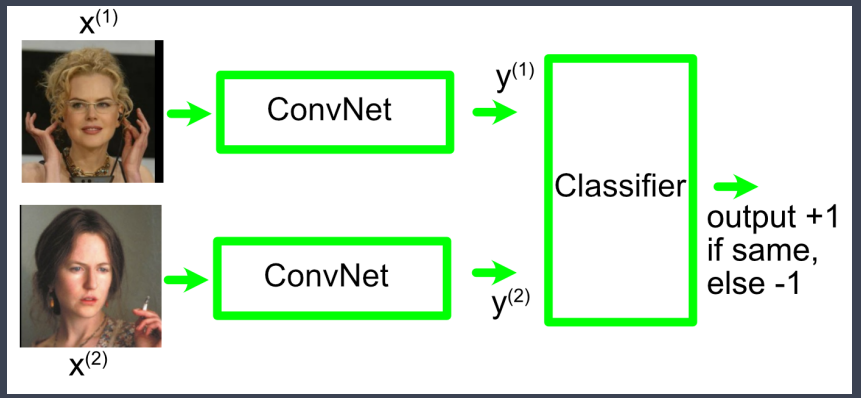
* Default bounding boxes (*anchor boxes*) $\rightarrow$ facilitates training, just learn offsets fromm default shapes/sizes of bounding box
* Tweaks (in SDD and YOLOv3) to operate at multiple length scales $\rightarrow$ better performance when have mix of large and small objects

» Face Recognition/Verification

* *Face Detection*.
  * Is there a face in the image e.g. putting bounding box around faces in camera viewfinder $\rightarrow$ special case of object detection.
* *Face Recognition*
  * Match face in image against a database of faces e.g. tagging faces in a photo with family/friends, CCTV surveillance
* *Face Verification*
  * Does presented face match a target face e.g. unlocking phone.
* Face Recognition/Verification: could train ConvNet using database of faces, but (i) classification problem with many labels and (ii) if add a new face then need to retrain ConvNet ... how to avoid this?

» Face Recognition/Verification

* Idea: Train ConvNet *once* to extract features from faces.



1. Use ConvNet to map one image $x^{(1)}$ to feature vector $y^{(1)}$.
2. Use ConvNet to map second image $x^{(2)}$ to feature vector $y^{(2)}$
3. Now formulate as a binary classification problem: given input feature vector $[y^{(1)}, y^{(2)}]$ predict whether the two images are of the same person or not.
    * E.g. use logistic regression or even just calc the difference between $y^{(1)}$ and $y^{(2)}$ and if this difference is small enough predict that images are of same person.

* Key question: how to train ConvNet?

» Face Recognition/Verification

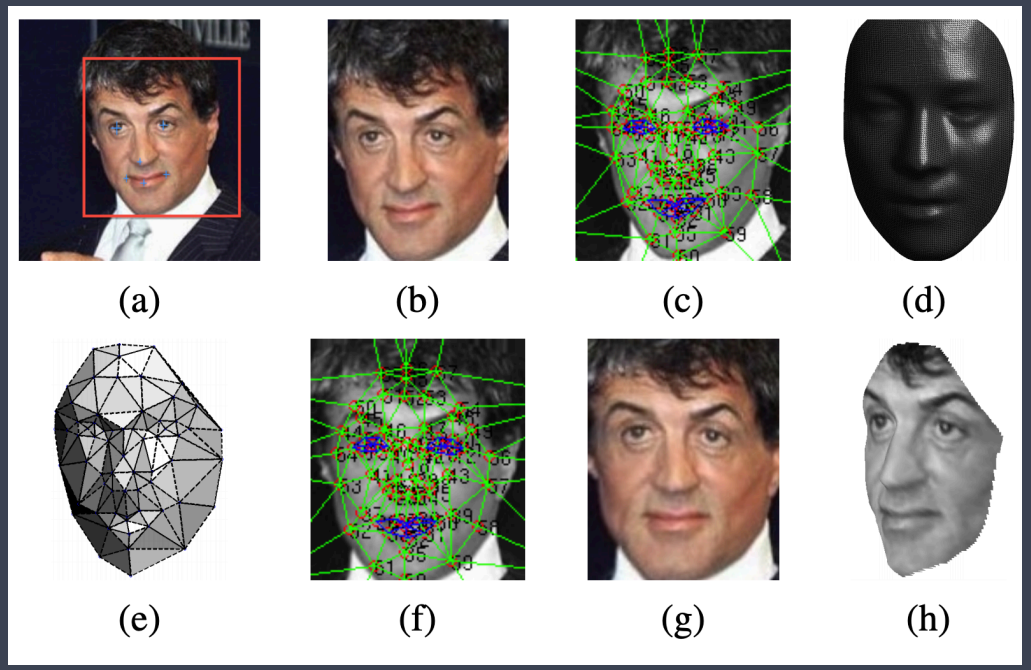
How to train ConvNet?

* *Triplet loss* FaceNet: A Unified Embedding for Face Recognition and Clustering 2015
  https://arxiv.org/abs/1503.03832v3
  * Input image x, ConvNet output is $f(x)$
  * Target image x, image of same person y, image of different person z
  * Want $\|f(x) - f(y)\|_2^2 + \alpha < \|f(x) - f(z)\|_2^2$ for all images, α a hyperparameter
  * Cost for image x is $\max\{\|f(x) - f(y)\|_2^2 - \|f(x) - f(z)\|_2^2 + \alpha, 0\}$ (cf SVM) and then sum overall all training images to get cost function.
* *Pairwise loss* DeepFace: Closing the Gap to Human-Level Performance in Face Verification 2014
  https://dl.acm.org/doi/10.1109/CVPR.2014.220
  * Logistic cost function on difference between pairs of images in training data.
* Use SGD to select ConvNet parameters that minimise cost function

» Face Recognition/Verification

Extra Details:

- Cropping → predict bounding box for face, then crop
- Alignment → improves FaceNet accuracy by about 1% and DeepFace accuracy by 2-3%

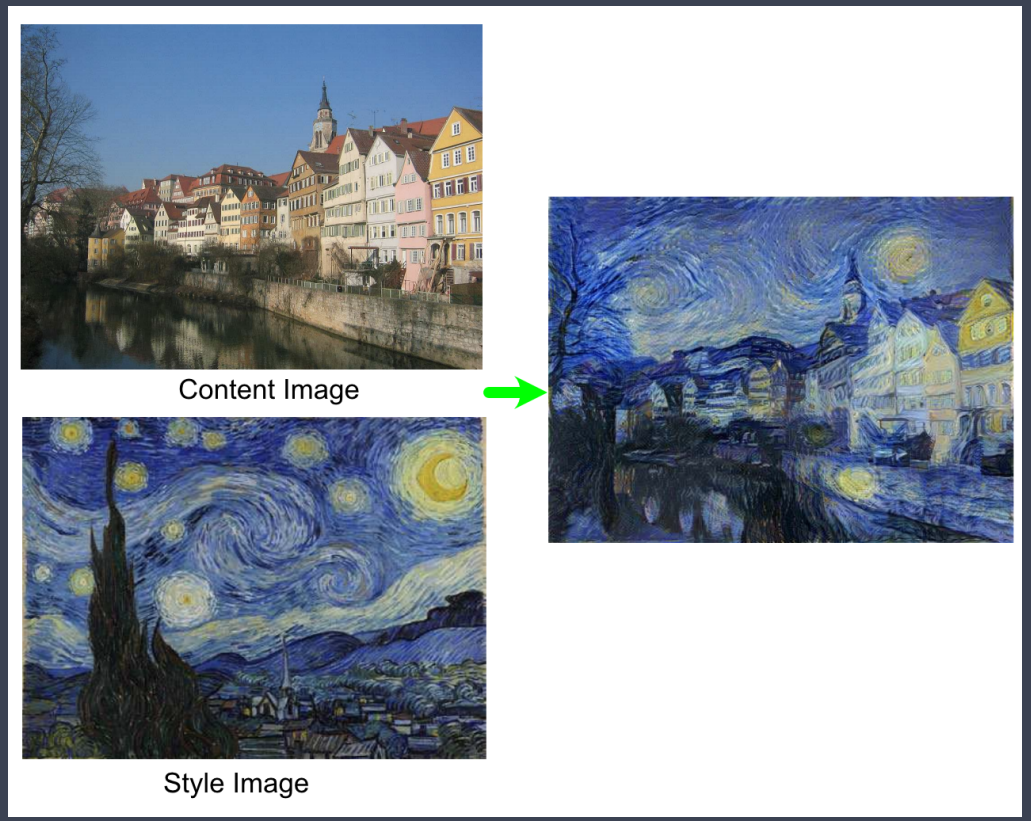


DeepFace https://dl.acm.org/doi/10.1109/CVPR.2014.220

(a) Identify landmarks on face (edges of eyes, mouth, front of nose)
(b) Crop
(c) Identify more detailed landmarks
(d)-(g) Map onto "standard" face and rotate/align

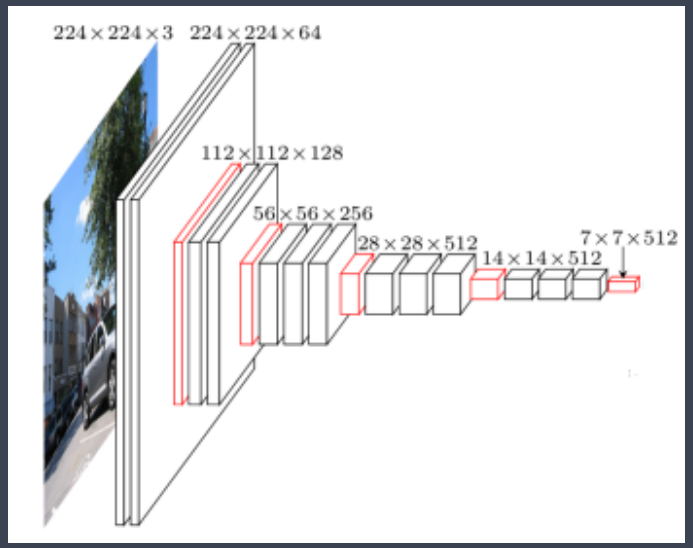
» Image Style Transfer

* Take content from one image and apply style from another. E.g.



* Image Style Transfer Using Convolutional Neural Networks 2016 https://www.cv-foundation.org/openaccess/content_cvpr_2016/papers/Gatys_Image_Style_Transfer_CVPR_2016_paper.pdf

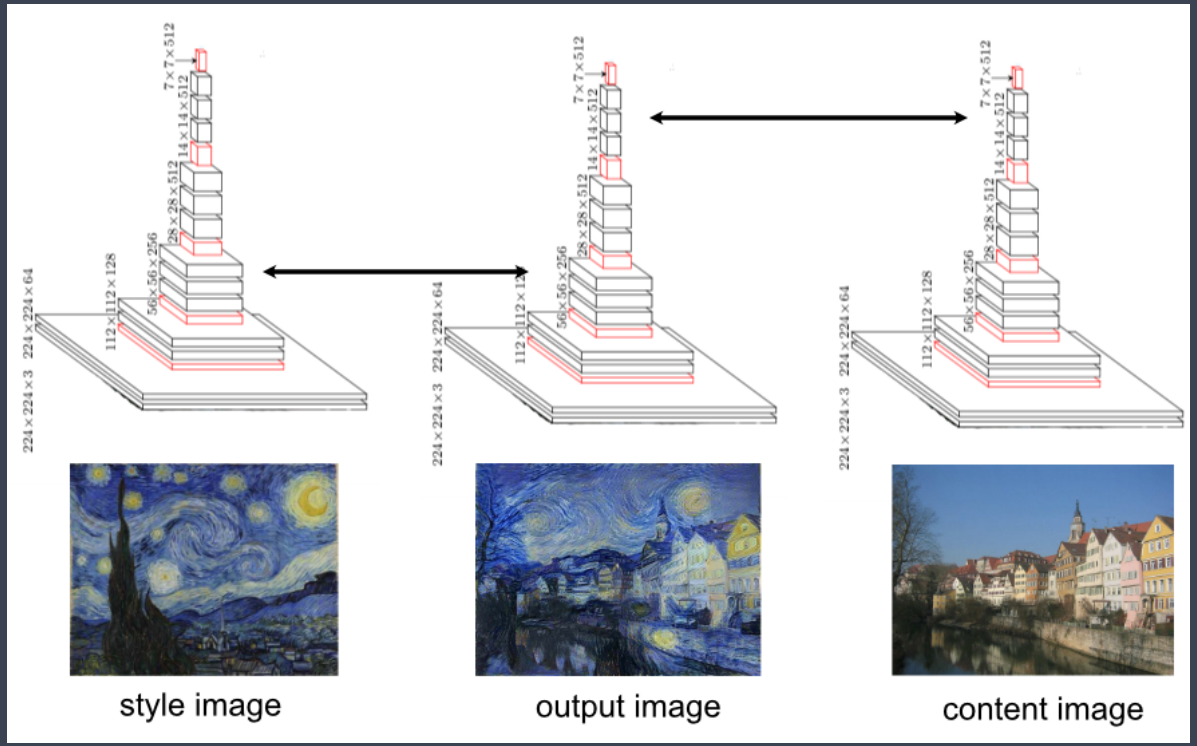
» Image Style Transfer



VGG19 minus output FC-layers

* Rough idea: earlier layers (closer to input image) capture smaller features of image such as texture, later layers capture larger features such as objects.
    1. Input style image into ConvNet and note output of early layers
    2. Input content image into ConvNet and note output of later layers
    3. Initialise image with random pixel values. Input into ConvNet.
    4. Cost function measures difference between (i) output of early layers and early layers of style image + (ii) later layers and later layers of content image.
    5. Using SGD to adjust image pixels to minimise cost function $\rightarrow$ hope is that texture of trained image matches style image and content matches content image

» Image Style Transfer



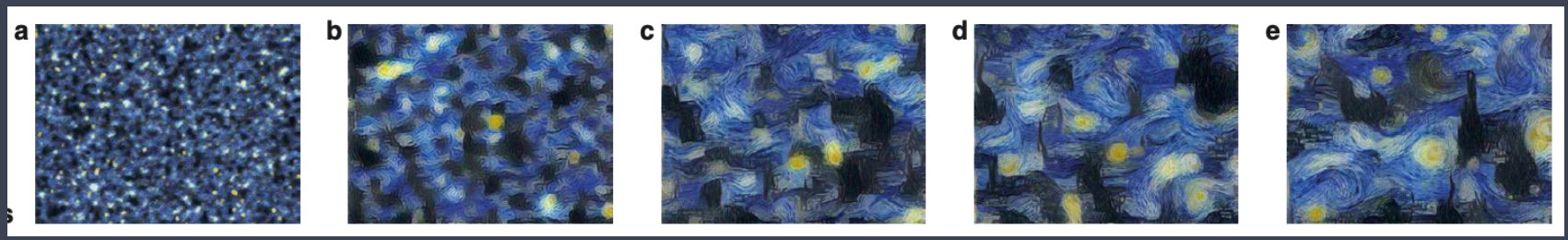
Train output image so that:

* Output of early layers are similar to those for style image
* Output of later layers are similar to those for content image
* Cost function: $J = (1 - \alpha)J_{style} + \alpha J_{content}$, α adjusts how much weight cost function gives to matching style image and to matching content image

» Image Style Transfer

Extra details:

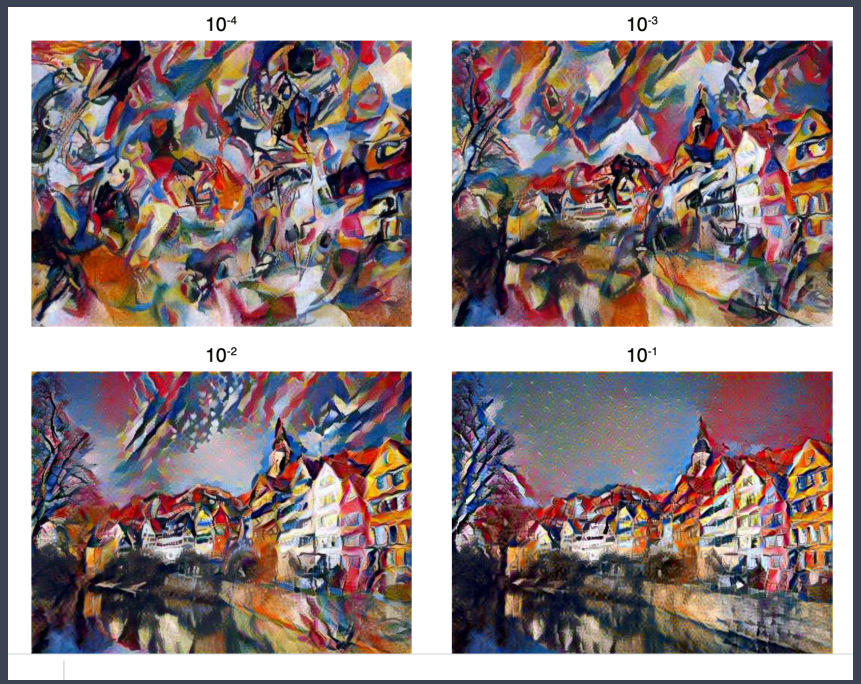
- "Early layers" idea isn't quite right. Instead:
  - Calculate cross-correlation amongst elements in kernel to keep texture but throw away spatial arrangement. Train output image so output of its layers are similar to these "smoothed" kernels
- E.g. Starry starry night smoothed kernels in early and later layers:



- Need to match output of multiple layers (not just early ones) of style image to capture larger features and nice looking images e.g. stars in starry starry night

» Image Style Transfer

Style image



Varying $\alpha/(1-\alpha)$

* Effect of adjusting weight given to content image in cost function $\rightarrow$ as increase weight the objects in content image become more prominent

» Summary

- Convolutional networks still not that well understood and architectures etc are still evolving:
  - Regularisation (reducing overfitting). E.g. see When Does Label Smoothing Help? 2019 https://arxiv.org/pdf/1906.02629v2.pdf
  - Width matters as well as depth. Also higher res inputs. E.g. see EfficientNet: Rethinking Model Scaling for Convolutional Neural Networks 2019 https://arxiv.org/pdf/1905.11946v3.pdf
  - Robustness in adversarial settings
- Transfer learning is important as takes a long time and lots of data to train a large ConvNet
- Insightful use of existing ConvNets within overall system is often a key step e.g. object detection, face recognition
- We've looked at image processing. Image = 2D sequence and ConvNets can be applied to other sequence data including 1D sequences:
  - Natural language processing: sentences = sequence. See BERT: Pre-training of Deep Bidirectional Transformers for Language Understanding 2019 https://arxiv.org/pdf/1810.04805.pdf
  - Time series data: time series = sequence.
  - DNA sequences