Computer Vision 3: Detection, Segmentation and Tracking

Lecture notes based on the lectures by Prof. Leal-Taixe

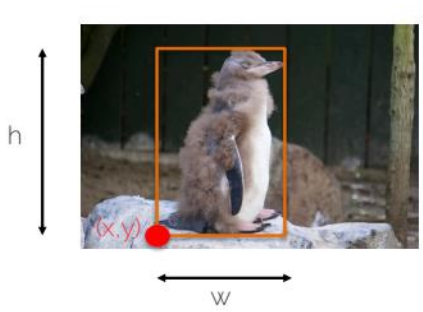
Vojtech Poriz, January 2022

v1.0

1. Object detection

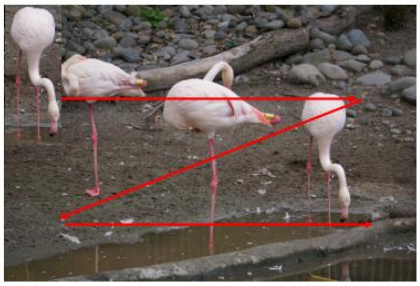
Task definition

- We want to find all the interested objects in the image
- Bounding box(x,y,w,h) + Class



Traditional object detection methods

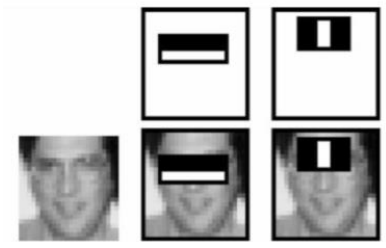
1. Template matching + sliding window



- We had image and a template
- We would start by putting the template on top of the image and sliding it along + compute correlation between the pixels

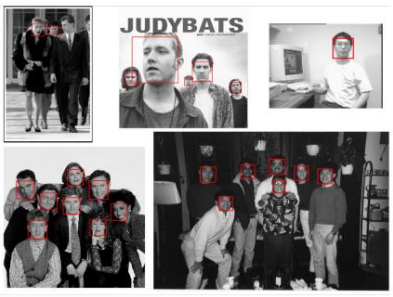
- Problems:
  - If the flamingo is **occluded**, we get low correlation and no match
  - we also need to have the exact instance - the template doesn't **include appearance/shape/pose changes**
  - we would have also to try different scales/aspect ratios

2. Feature extraction + classification



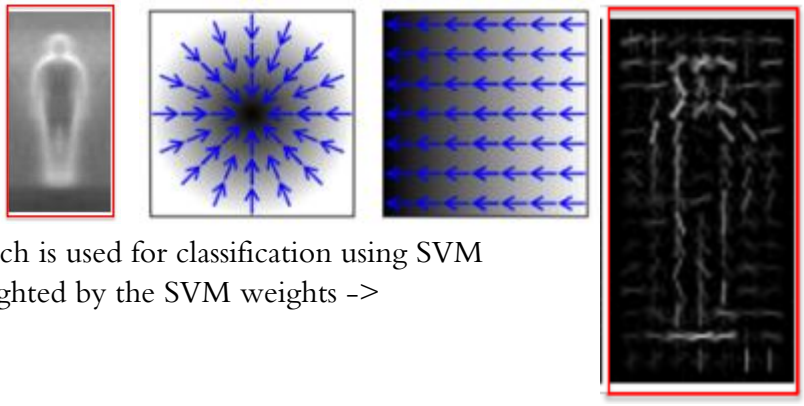
Viola-Jones detector (2001)

- First algorithm that worked well in the paradigm of Feature extraction + classification
- The method would multiple big learners to build a strong classifier = make many small decisions and combine them for a stronger final decision
- Select Haar-like features



- Use so-called **integral image** for fast feature evaluation - sliding feature over the image and find parts of the image with the highest correlation
- We do this with different weak learners – e.g. to detect mouth, nose, ...
- After that, we combine all the decisions from the weak learners to create a strong face classifier
- For frontal faces, the results were great

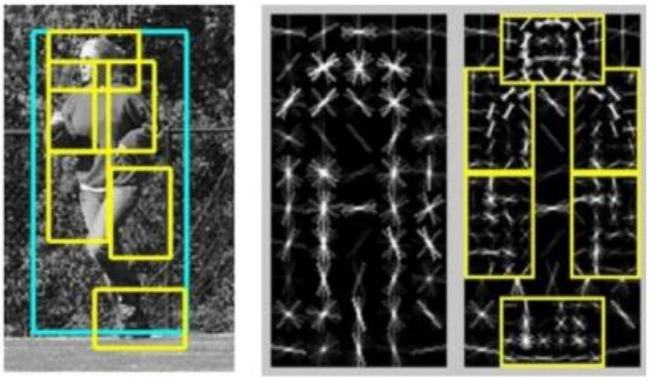
Histogram of oriented oriented gradients



- uses more advanced set of features
- they start with the gradient of the image
- then they do average gradient image over training samples, which provides very good shape model
- from this, they create a histogram, which is used for classification using SVM
- Final features for person detection weighted by the SVM weights ->

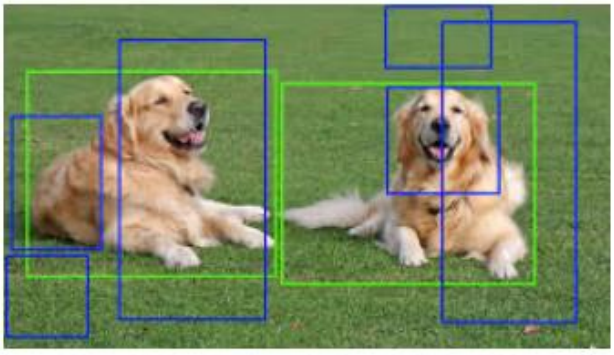
Deformable part model (2008)

- evolution of the HOG
- instead of detecting the full person, it is based on body part detection ⇒ that leads to bigger robustness to different body poses



Moving towards general object detection

- What defines an object?
  - we need a generic, class-agnostic objectness measure = how likely it is for an image region to contain an object
  - in another words - if we placed random bounding boxes, we need something to conclude that the two green bounding boxes contain an object
  - we want to focus all of our learning to differentiate, if a particular bounding box contains an object + differentiate classes of objects
  - these **object proposals** are also called **regions of interest (RoI)**



- Object proposal methods
  - **Selective search** - pixels with similar intensity/color/texture/composition (convex regions will)/shape will be grouped together
  - **Edge boxes** - similar objects are joined into contours ⇒ contour plots ⇒ we look for long contours ⇒ a good region proposal has high ratio between number of contours inside/outside.

Non-Maximum Suppression (NMS)

- there are many boxes that explain one object, but we only want to keep the "best" ones
- this method looks at the boxes and selects the one that best fit our measure
- it is being used even in the deep learning methods – at test run

Algorithm 1 Non-Max Suppression

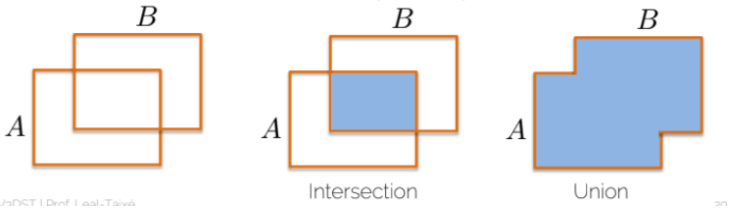
```
1: procedure NMS(B,c)
2:     B_nms ← ∅
3:     for b_i ∈ B do                                  ← Start with anchor box i
4:         discard ← False
5:         for b_j ∈ B do                              ← For another box j
6:             if same(b_i, b_j) > λ_nms then          ← If they overlap > PARAMETER OR THRESHOLD
7:                 if score(c, b_j) > score(c, b_i) then
8:                     discard ← True                  ← Discard box i if the score is lower than the score of j
9:         if not discard then
10:            B_nms ← B_nms ∪ b_i
11:    return B_nms
```

Overlap = to be defined Score = depends on the task

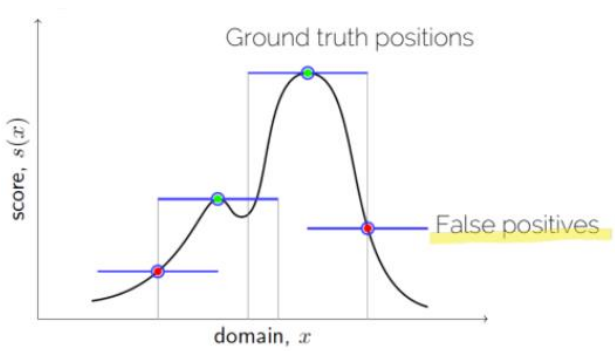
- we start with one box, then we look for another box that is overlapping
- if they overlap more than a threshold
- then we discard one of the boxes, that has lower score

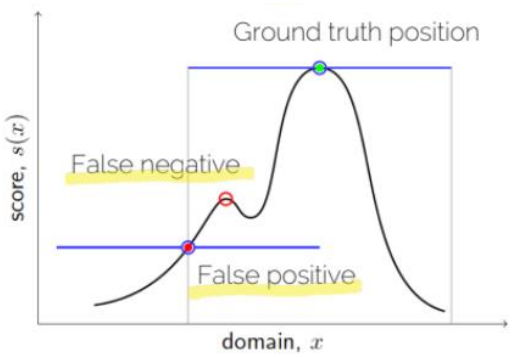
- Region overlap
  - **Intersection over Union (IoU) / Jackard Index**

$$J(A,B) = \frac{|A \cap B|}{|A \cup B|}$$



- Problem: how to define the overlap threshold for the NMS method?
  - narrow threshold (left) - low precision
  - wider threshold (right) - low recall

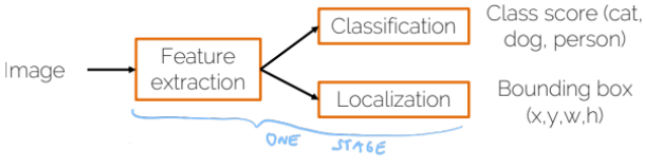




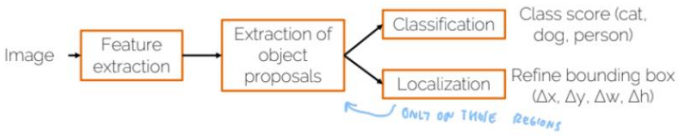
2. Two-stage object detectors

Learning-based detectors

- One stage detectors
  - still follow the paradigm of the feature extraction + classification and localization
  - uses CNN for feature extraction
  - examples:
    - YOLO, SSD, RetinaNet
    - CenterNet, CornerNet, ExtremeNet (point based)



- Two stage detectors
  - they separate these steps even more to perform better
  - they do feature extraction, then they extract object proposals and only on those they perform classification and localization
  - examples:
    - R-CNN, Fast R-CNN, Faster R-CNN
    - SPP-Net, R-FCN, FPN

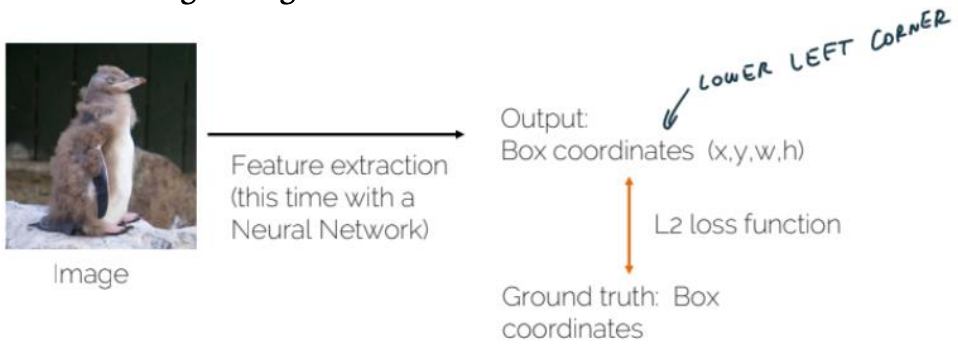


Two stage detectors

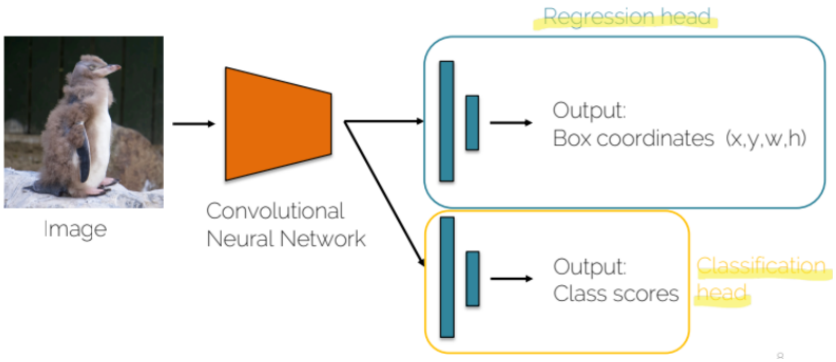
- we will start with the **localization and dassification** step

Localization and Classification

- let's assume that we have an image, that contains one object
- we want to find the exact bounding box that encloses the position of the penguin
- we do **bounding box regression**



- we use **CNN for feature extraction**
- then we **add fully connected layers** and train the CNN **for regression** with the ground truth bounding box coordinates using **L2 loss** function
- then we **add another set of fully connected layers** and train the CNN **for dassification** with the ground truth class using **softmax loss**

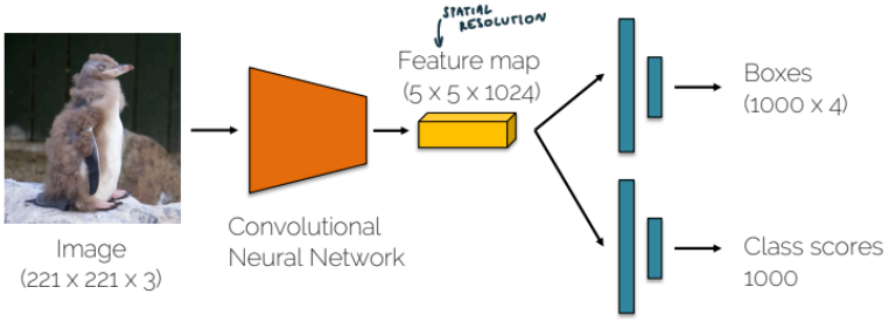


- in the past, people trained the classification head first, then freeze the layers and then train the regression head; at test time, used both
- nowadays, we train NN with multiple head at the same time and just balancing the losses

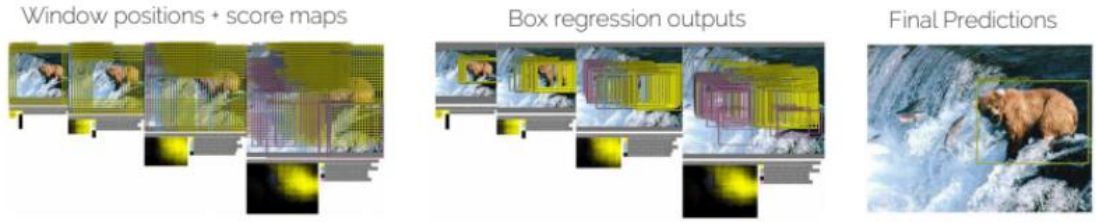
Different methods

Overfeat

- first algorithm that proposed to use a sliding window type approach for object detection plus a box regression and a classification head
- they wanted to do this in an end2end method
- Sliding window
  - we slide the box in the different location, and pass it along to the network
  - when we hit the right box, we get a high output for the classification head
  - off course there are other similar boxes that depict the penguin, so we have to use NMS
- Box regression + classification:



  - the feature map still maintains some spatial resolution
  - if there would be no penguin on the image, we might get something from the regression head, but the classification head output would be low
- Problems:
  - FCC layers expect fixed sized input
  - sliding window needs many locations and multiple scales

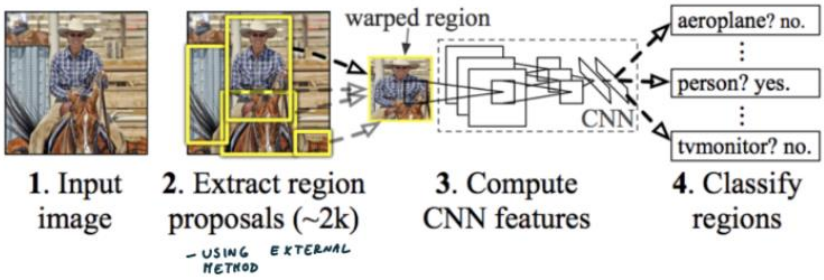


- What about multiple objects?
  - variable sized output is not optimal for neural networks
  - there are a couple of workarounds
    - Recurrent Instance Segmentation
    - Set prediction

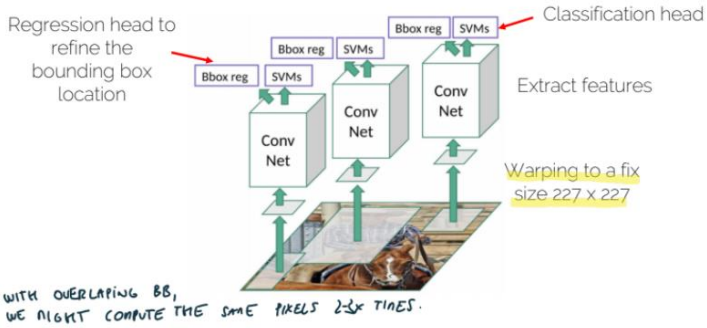
Detection as classification?

- it is expensive to try all the possible positions, scales and aspect ratios
- how about we preselect a subset of boxes with the most potential?
- **this is what two-stage detectors do**
  - find a subset of boxes, then classify what is on them and refine the bounding box a bit

R-CNN (region convolution neural networks)



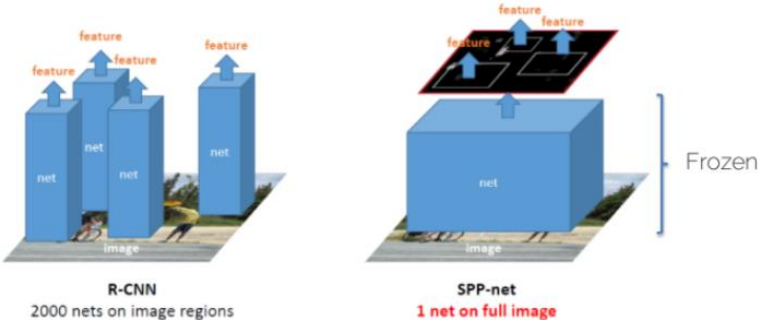
- we use external method to extract regions, warp them to a fixed size, pass them to a CNN and predict what we want



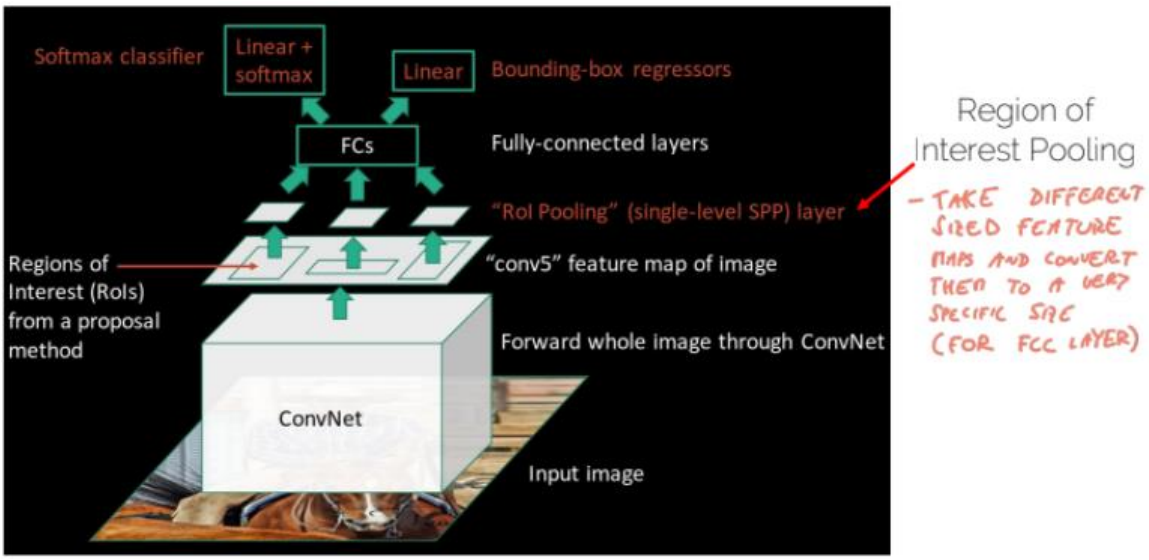
- training scheme:
  - pre-train the CNN on ImageNet
  - finetune the CNN on the number of classes the detector is aiming to classify (softmax loss)
  - train a linear SVM classifier to classify image regions – one per class (hinge loss)
  - train the bounding box regressor to refine the proposed bounding boxes (L2 loss)
- pros:
  - the pipeline of feature extraction, SVM, classification is well known. only the features have changed (CNN instead of HOG)
  - CNN summarizes each proposal into a 4096 vector
  - we use transfer learning
- cons:
  - slow – 47s/image with VGG16 backbone.
  - training is slow and complex
  - the object proposal algorithm is fixed -> not exploiting learning to its full potential

SPP-Net

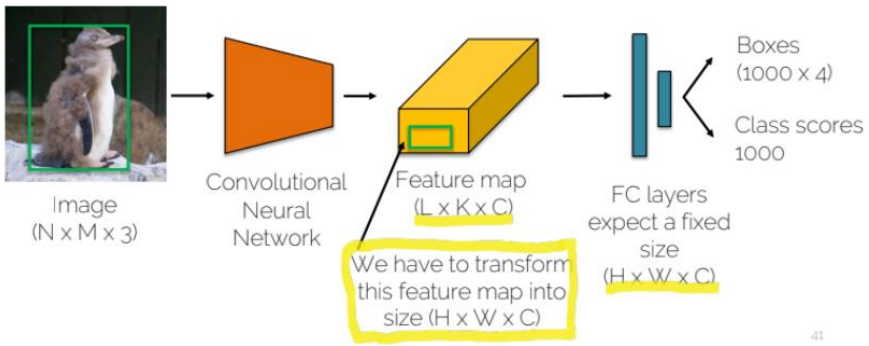
- tries to be **faster than R-CNN at test time**
- we pass the image only once through the net
- then we extract features from the feature map
- SPP-Net proposed a way to pool the features into a **common size**
- SPP-Net problems:
  - training is still slow
  - training scheme is still complex
  - still no end-to-end learning



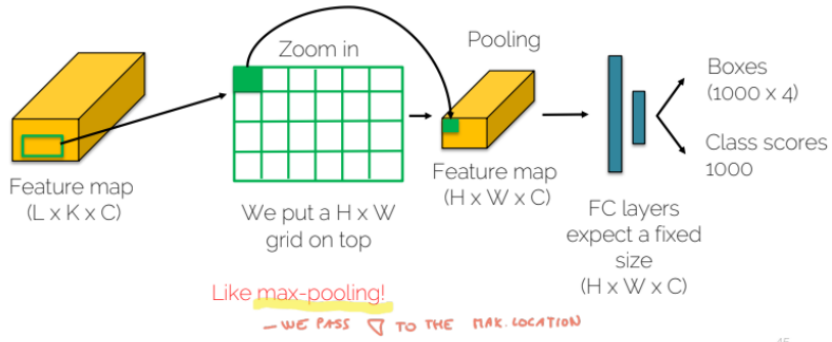
Fast R-CNN



- we also pass the whole image through the ConvNet at the start
- then we get our region proposals and we put them into a so called **RoI Pooling layer**
    - from all of the proposals that have different sizes/aspect ratios we have to convert their feature representations into a fixed size



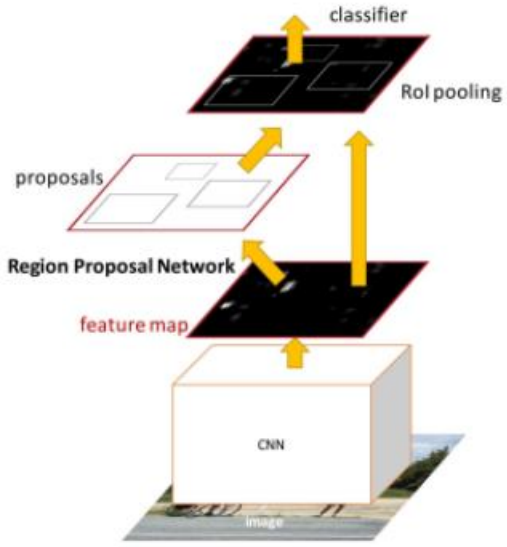
    - we put a **HxW grid on top** of the feature map



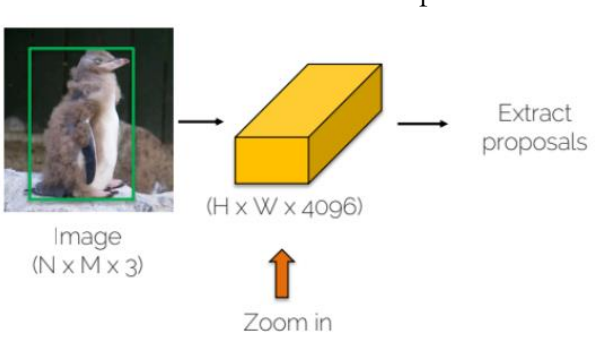
- then we pass them to the fully connected layers and further to the regression and classification head
- pros:
    - we have a speed up in both training and testing times
- cons:
    - the most time is spent on the external region proposal algorithm

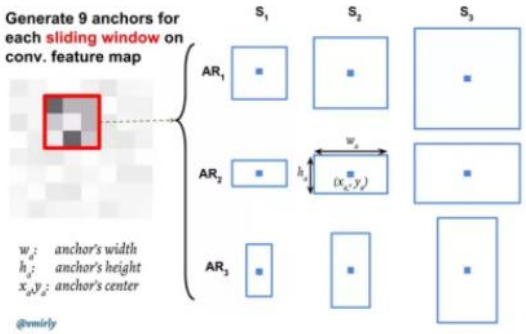
Faster R-CNN

- we upgrade the previous solution by integrating the generation to the rest of the pipeline = **Region Proposal Network (RPN)**

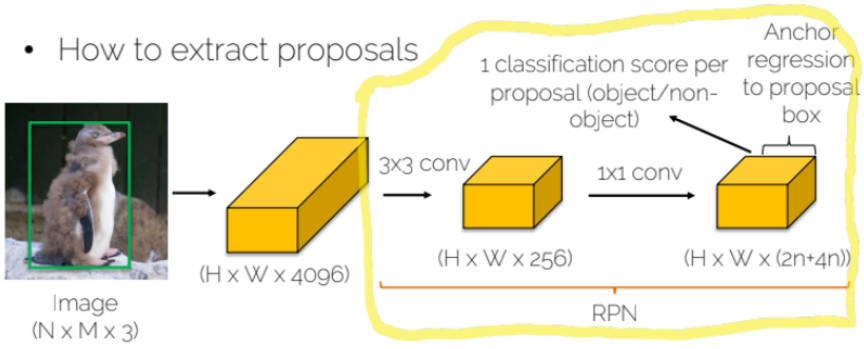


- Region Proposal Network:
  - we need a fixed number of proposals
  - we need to have them densely placed
  - **for each pixel**, we fix the number of proposals using a set of n=**9 anchors**
    - 9 anchors = 3 scales * 3 aspect ratios





    - from that, we get a 256-d descriptor for each location resulting in a HxWx256 feature map
    - we then perform 1x1 conv to decrease the dimensionality to the wanted HxWx(2n+4n) size
      - 2n for classification for each pixel
      - 4n for regression for each pixel



Per feature map location, I get a set of anchor correction and classification into object/non-object

- RPN Training and Losses:
  - we have ground truth boxes available
  - **dassification ground truth** – we compute p* which indicates how much anchor overlaps with a ground truth bounding box

    $$p^* = 1 \quad if \quad \text{IoU} > 0.7$$

    $$p^* = 0 \quad if \quad \text{IoU} < 0.3$$

    - 1 = anchor represents an object
    - 0 = anchor is just on the background
    - the rest do not contribute to the training
  - **dassification training**:
    - we randomly sample 256 anchors and for a mini-batch (balanced objects vs. non-objects)
    - we calculate the classification loss (binary cross-entropy)
    - those anchors that represent an object are going to be used to compute the regression loss
  - **regression:**
    - each of the anchors is described by the center position, width and height
    - but the network actually predicts relative values of all the values

      Normalized x $t_x = (x - x_a)/w_a, \qquad t_y = (y - y_a)/h_a,$ Normalized y

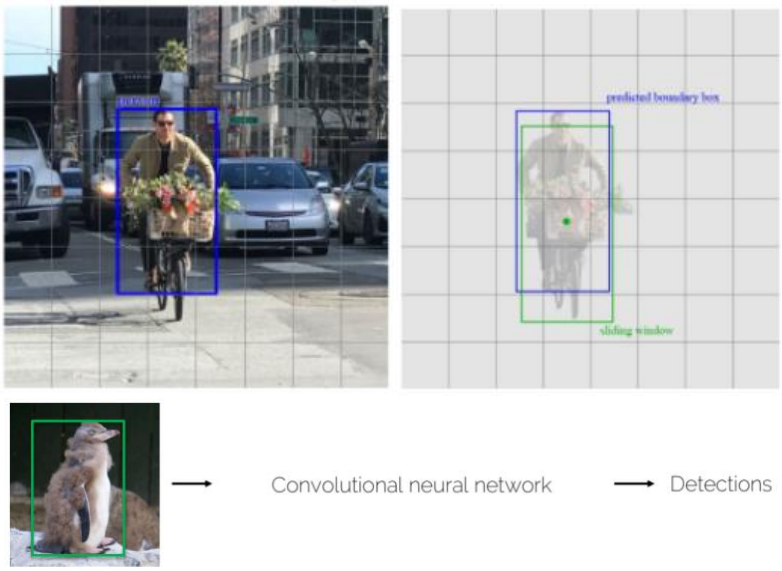
      Normalized width $t_w = log(w/w_a), \qquad t_h = log(h/h_a),$ Normalized height
    - we use Smooth L1 loss on regression targets
- Training of Faster R-CNN
  - used to be very complicated, first training of RPN separate from the rest
  - now we can train jointly
  - there are 4 losses in total
    - RPN classification (object/non-object)
    - RPN regression (anchor -> proposal)
    - Fast R-CNN classification (type of object)
    - Fast R-CNN regression (proposal -> box)
- Summary of Faster R-CNN:
  - 10x faster at test time than Fast R-CNN
  - trained end2end including feature extraction, region proposals, classifier and regressor
  - more accurate, since proposals are learned
  - RPN is fully convolutional

3. One-stage object detectors

- do we actually need to separate the detection process as in the two-stage detectors?
- one-stage detectors go directly from image to directly to classification+regression of the bounding boxes
- main advantage is that these detectors are incredibly fast

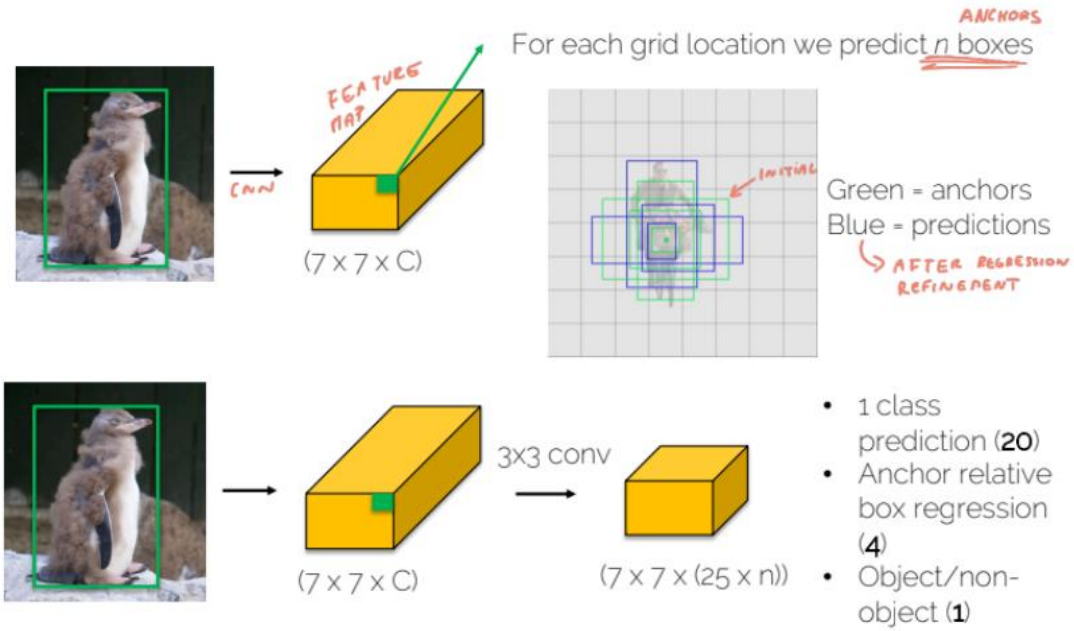
Yolo

- most famous one
- we make sliding window approach efficient by sliding only on certain locations along a grid
- we place a box at the center of each cell in the grid and use that as an initial box guess
- then we classify and regress the box



YoloV2

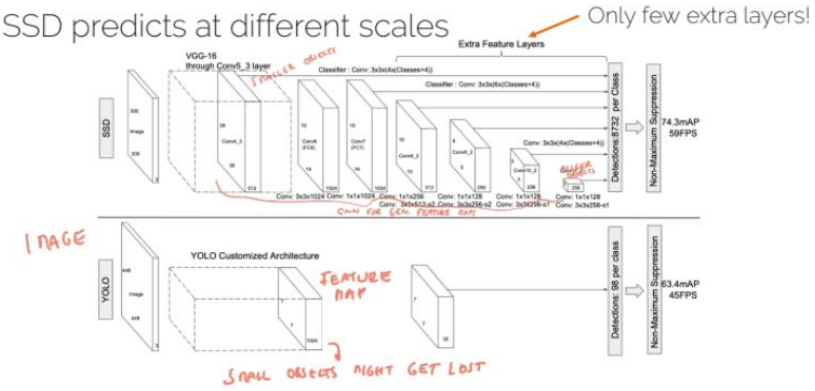
- very similar, but uses anchor boxes for leach grid location



- for each pixel, we get 25 numbers
- similar to RPN but we just do everything in one step

SSD: Single Shot multibox Detector

- similar to Yolo
- but it has multiple feature maps from which it performs classification -> improved detection of various sized objects

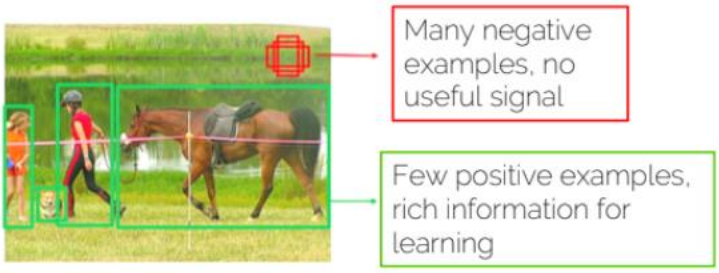


YOLO and SSD

- pros:
  - very fast
  - end2end trainable and fully convolutional
  - SSD detects more objects than Yolo
- cons:
  - performance is not as good as two-stage detectors
  - difficulty with small objects

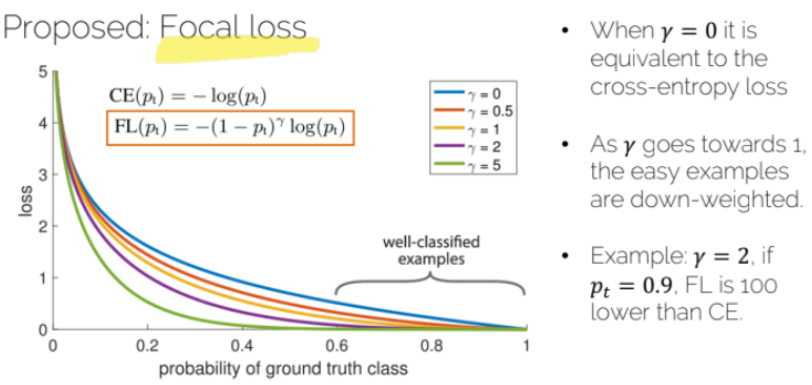
What is the problem with one-stage detectors?

- many locations need to be analyzed densely but there are only very few objects **-> foreground-background imbalance**
  - in two-step detectors, the proposal parts create balance for the following layers
  - in one-step detectors, we could use negative mining, but that is not sufficient
    - it's better to change the loss function



RetinaNet

- tries to address the class imbalance by changing the loss function to the **Focal loss**



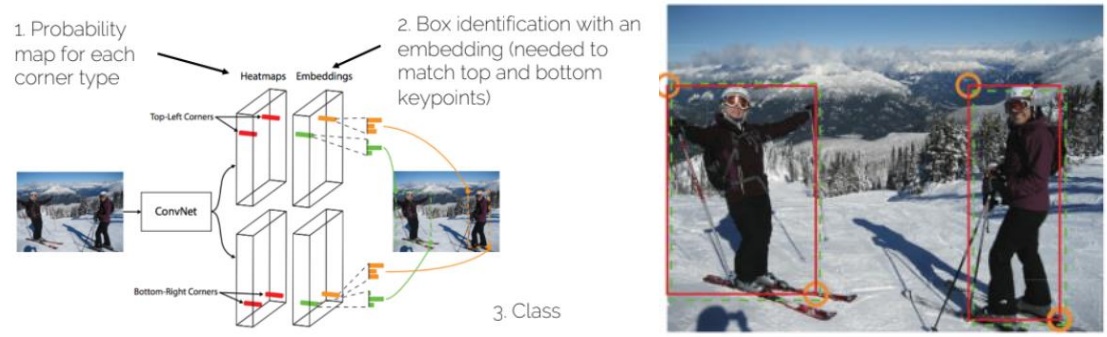
- on top of that, they
  - put a powerful feature extraction backbone – ResNet
  - they do multi-scale prediction like in the case of SSD
  - and they use 9 anchors per level – each with classification and regression head
- this leads to a very good performance

Point-based one stage detectors

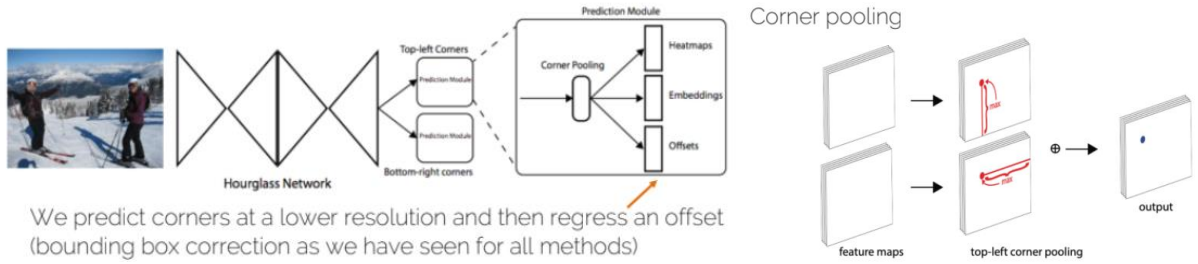
- limiting factor is the specific shape of the bounding box, which needs to be predicted by the different anchor boxes
- so can we get rid of the anchors?

CornerNet (2019)

- expresses bounding boxes with 2 points: top-left + bottom-right corners



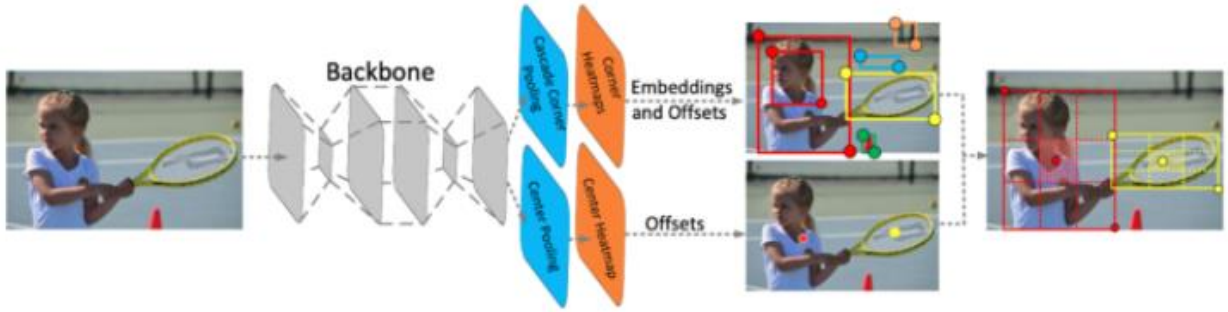
- we perform normal feature extraction
- then we have two heads
  - one predicts heatmaps for the top-left corners
  - the other one predicts heatmaps for the bottom-right corners
- once we have this is to match the top-left corners with the bottom-right corners
  - each location of probable corner in the heatmap has a signature, that pairs him with the corresponding other corner
- CornerNet also uses hourglass network, which is more detailed in the pose estimation part of these notes



- what is the problem with the CornerNet?
  - many incorrect bounding boxes (especially small) -> too many false positives

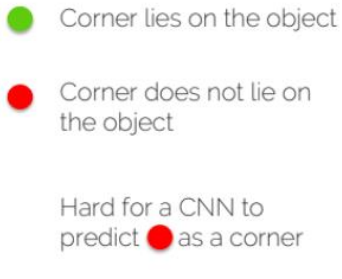
CenterNet

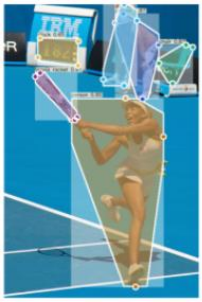
- focuses on the center of the object to infer its class
- uses the corners as proposals and the center to verify the class of the object and filter out outliers



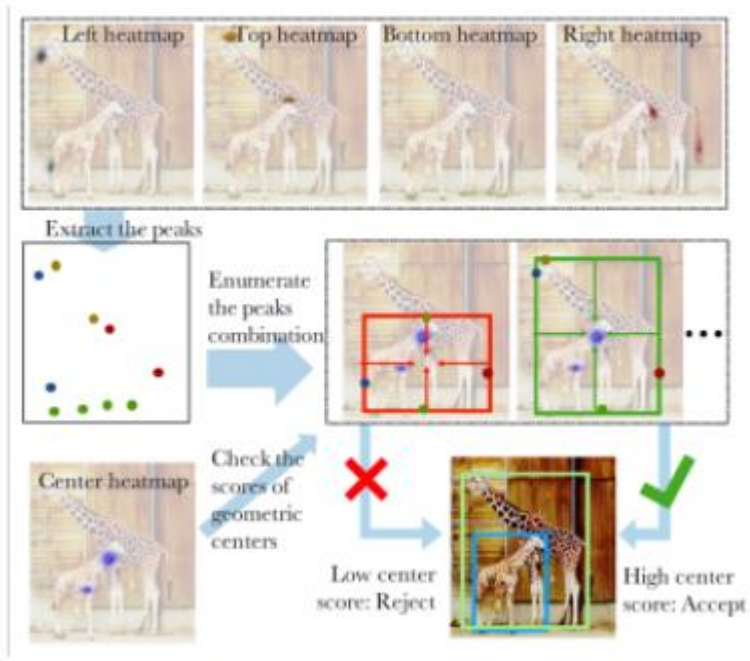
ExtremeNet

- completely leaves the idea of bounding boxes and represents objects by their extreme points



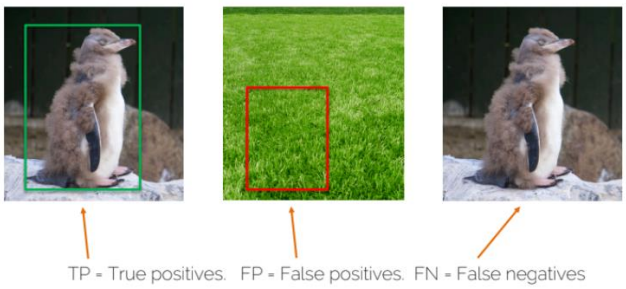


- it's easier for CNN to look for edges than an artificial representation corner (bounding box)
- ExtremeNet uses 5 heatmaps
    - 4 for the edge points that are touching the object
    - 1 for the center point location



- unlike in the CenterNet, which used the embeddings, here we use the center heatmap
    - we take all the corner combination and find the one, that matches its center to a center from the Center heatmap

Detection evaluation

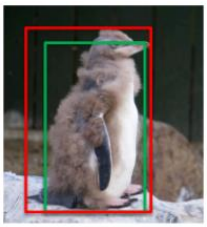


- **precision**: how accurate your predictions are

$$Precision = \frac{TP}{TP + FP}$$ → Good boxes detected; → All boxes that you detected

- **recall**: how good you are at finding all the positives

$$Recall = \frac{TP}{TP + FN}$$ → Good boxes detected; → All ground truth boxes



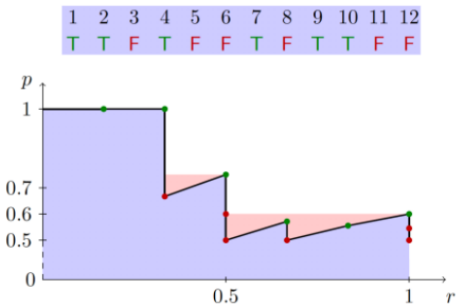
- Detection evaluation using **IoU**
  - if IoU > 0.5 -> positive match

- Detection evaluation using **Mean Average Precision** (mAP)
  - for each image and class independently, rank the predicted boxes by confidence score
  - we assign the boxes to the corresponding ground truth if IoU > 0.5
  - each ground truth box can only be matched to one predicted box
  - for each class independently, compute Average Precision
  - then compute mean over all classes to obtain mAP
    - Rank predictions by confidence

Rank	True/false	Precision	Recall
1	T	1.0	0.17
2	T	1.0	0.33
3	F	0.67	0.33
4	T	0.75	0.5
5	F	0.6	0.5
6	F	0.5	0.5
7	T	0.57	0.67
8	F	0.5	0.67
9	T	0.56	0.83
10	T	0.6	1.0
11	F	0.55	1.0
12	F	0.5	1.0

      - my method computes 12 boxes in this image, only 6 of them are true positives
      - from these results, we will iteratively plot a **graph of precision vs. recall**

    - Average precision - area under curve (filled)



Non-Maximum Suppression – additional info

- if our network predicts a lot of bounding boxes, we get a **high recall**, but **low recall and mAP**
- for the suppression of too many bounding boxes, we use NMS
- this is applied to all the two-stage, one-stage detectors or even the one-stage point-based detectors

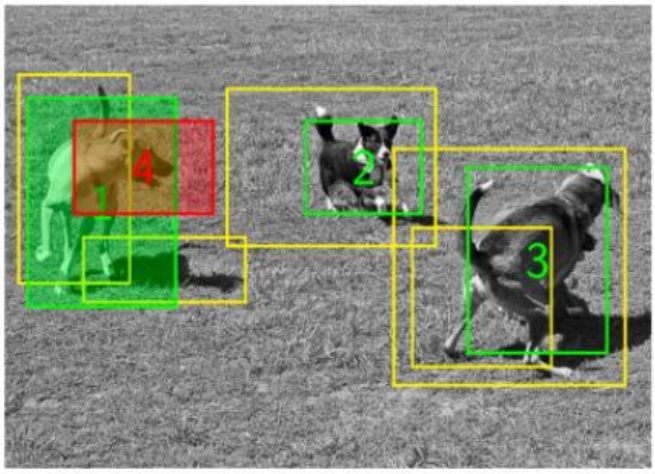
how does it work?

- we get many bounding boxes predictions by the CNN
- we rank the boxes by the confidence and take them one by one
- once a box overlaps an already cleared box (more confident one), we decide if the overlap is too big

$$p^* = 1 \quad if \quad \text{IoU} > 0.7$$

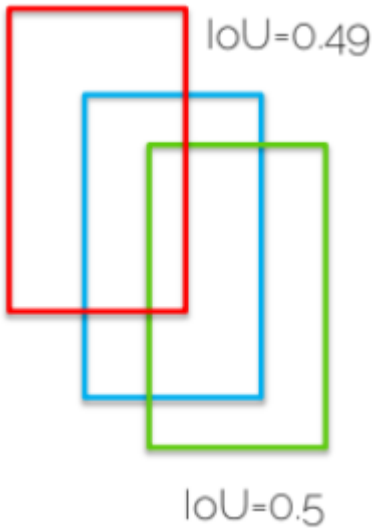
$$p^* = 0 \quad if \quad \text{IoU} < 0.3$$

  - if yes, we remove this box



what is the problem?

- highly overlapping object
  - for ex. with NMS threshold of 0.4, we remove both boxes -> false negative
  - for ex. with NMS threshold of 0.6, we keep both boxes -> false positive

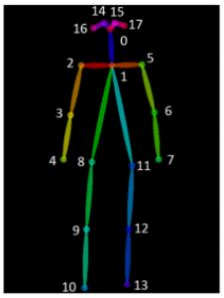


- **the choice of the threshold is very important**

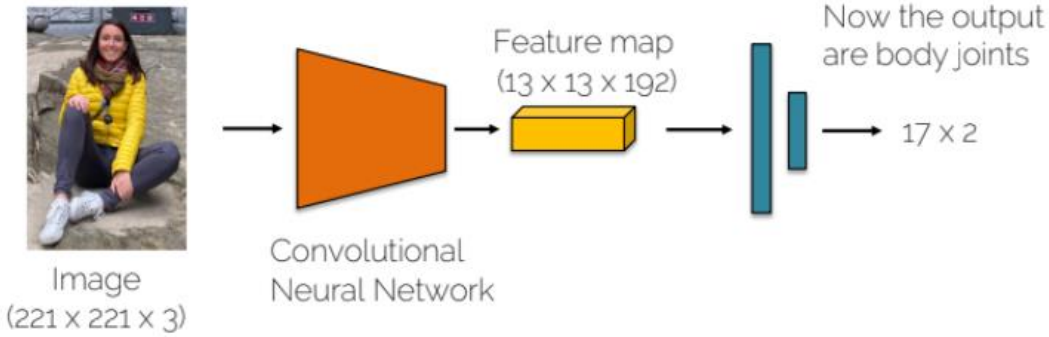
Detection beyond 2D boxes

- we can extend to **3D bounding boxes**
  - center + orientation + width + height + depth
- we can also predict other things like **facial landmarks**
- we can also do **human pose estimation**: body joints

Human pose estimation



- this is not so much different to detecting bounding boxes
- we only need a new parametrization
  - COCO parameterization = 17 joints to localize
- although similar to the BB, the whole problem is challenging
  - occlusions
  - clothing
  - extreme poses
  - viewpoint changes
- **Approach 1: Direct regression**



- **Approach 2: Heatmap prediction**
  - a better approach
  - for each joint, predict a full image with a heatmap of the joint location

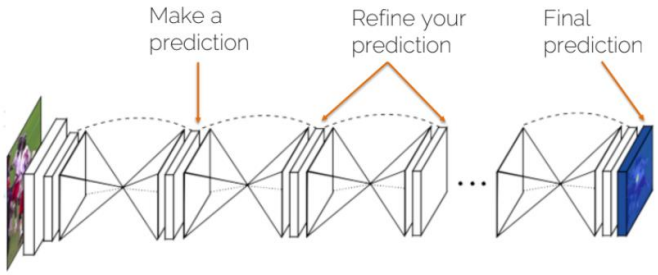
  - this creates a powerful representation <- we have a **confidence** per location
  - ground truth (GT) heatmap is constructed by placing a 2D gaussian around the joint position (e.g. variance 1.5 pixels)
  - Loss: MSE between the predicted and GT heatmap

- we can also make use of the **physical connectivity between the body parts**
  - using **graphical models** allows us to find the pose of several targets
    - we get all joint detections as provided directly by the CNN
    - then cluster them based on joint type and target ID
    - then we further solve the graphical model to get the final prediction

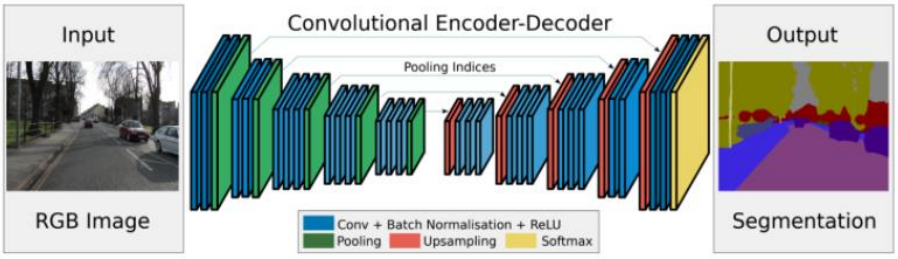
- **alternatively** we can further separate the problem into two subproblems
    1. object detection
    2. and then searching for the body joint positions only inside that bounding box pose estimation

Stacked hourglass networks

- another architecture that improved the joint position predictions
- we repeatedly go from high resolution to low resolution and back

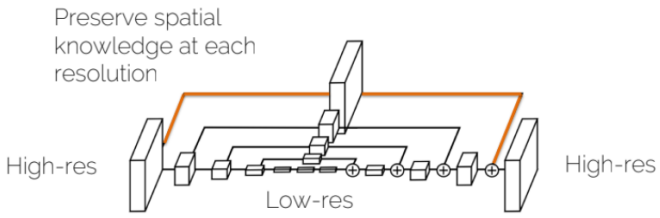


- Each level consists of two elements:
    - **Encoder**: normal convolutional filters + pooling
    - **Decoder**: Upsampling + convolutional filters

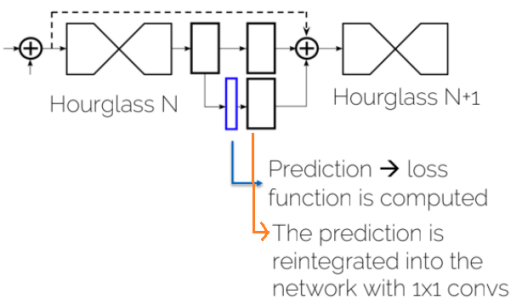


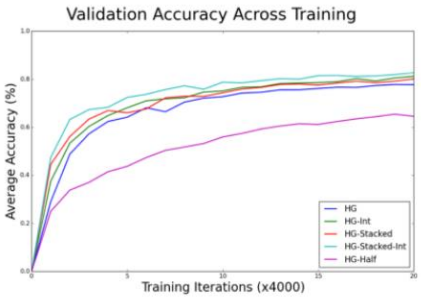
- The convolutional filters in the decoder are learned using backprop and their goal is to refine the upsampling
- The **upsampling** is done using nearest neighbour or interpolation

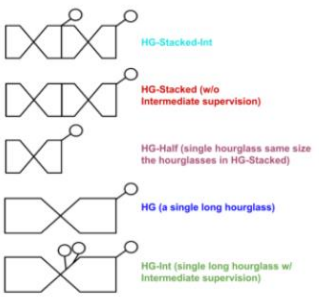
- This loss of information is then improved by **residual connections**, that preserve spatial knowledge at each resolution



- training is done using **intermediate supervision** (multiple predictions inside from inside the network), which further improves the results

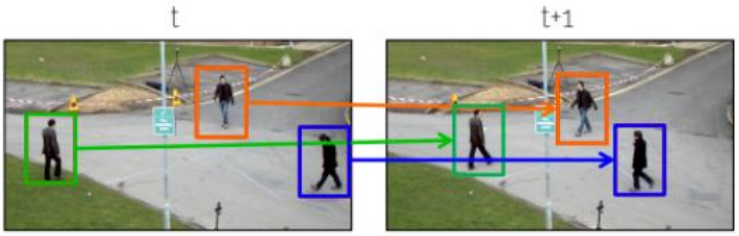






4. Object tracking

- Problem statement
  - given a video, find out which parts of the image depict the same object in different frames
  - often we run detectors at each frame



- Why do we need tracking?
  - occlusions
  - viewpoint/pose/blur/illumination variations (in a few frames of a sequence)
  - background clutter
  - we want to reason about the dynamic world = doing trajectory prediction
- Different tasks involved
  - similarity, correlation, correspondence, matching/retrieval, data association
- We have to learn different aspects of tracking
  - **appearance**: how our target looks like
    - single object tracking
    - re-identification
  - **motion**: make predictions of where the target goes
    - trajectory prediction

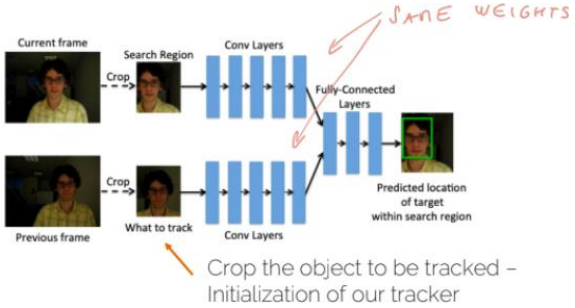
Single Target Tracking (STT)

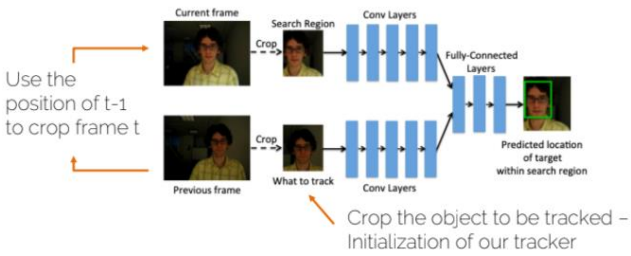
- task of tracking one object through a video sequence
- there are many challenges
- different approaches
  - STT as a **matching/correspondence problem** = GOTURN
  - STT as an **appearance learning problem** = MDNet
  - STT as **(temporal) prediction problem** = ROLO (CNN+LSTM)

Matching/correspondence problem

GOTURN architecture

- tries to find the same object in two consecutive frames
- uses siamese architecture = processes both frames in the same convolutional layers (same weights)
- then we have a series of FCC, that take concatenated both feature maps and compare these two frames



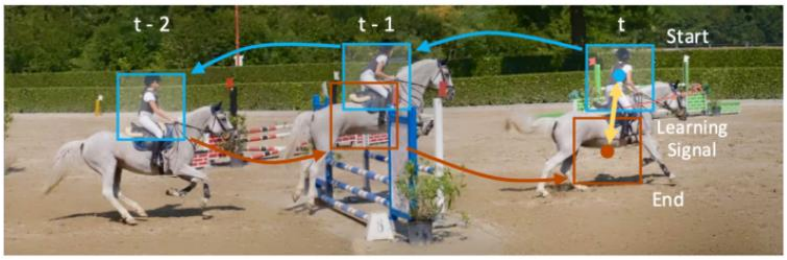


- we **assume very slow motion (we start with the very same position of the object in the following frame)**
- fully connected layers refine the location of the predicted bounding box

- pros:
  - no online training required – it is just refinement of bounding boxes
  - tracking is done by comparison, so we do not need to retrain or finetune our model for every new object
  - close to template matching approach
  - it is very fast
- cons:
  - we have a strict motion assumption = if the object moves fast and goes out of our search window, we cannot recover

Unsupervised (2019)

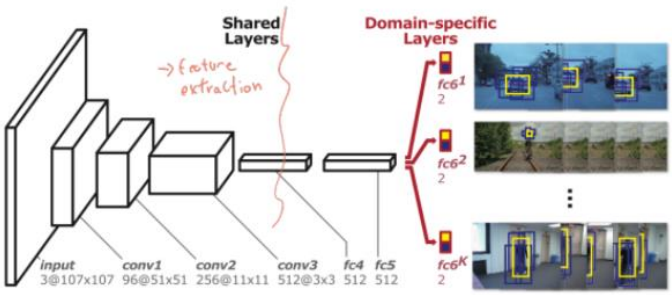
- another similar correspondence problem solution
- we assume, that if we track object in time, it should be the same if we track an object in the reverse time
- if we start at the first point, we want to track the object by comparison and then track back in time by for ex. regression
- we want to end up at exactly the same point we started



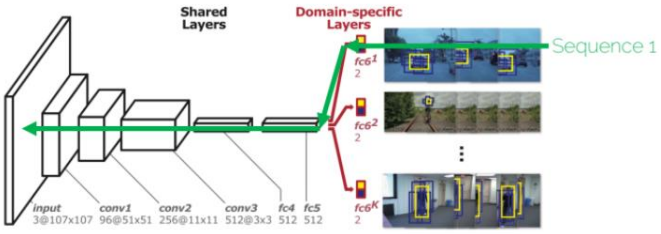
Appearance learning problem

MDNet

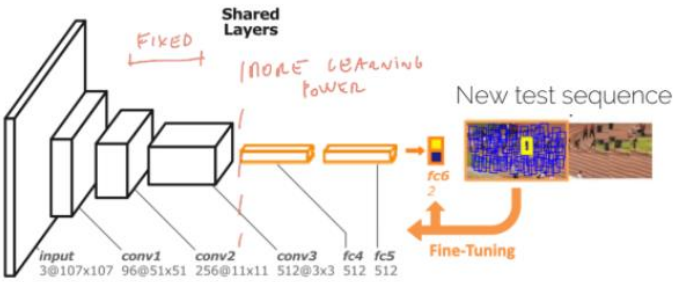
- we compute an appearance model of the object
- this entails training CNN at test time (we learn the appearances of the object in the test sequence)
  - slow: not suitable for real-time applications
  - solution: train as few layers as possible



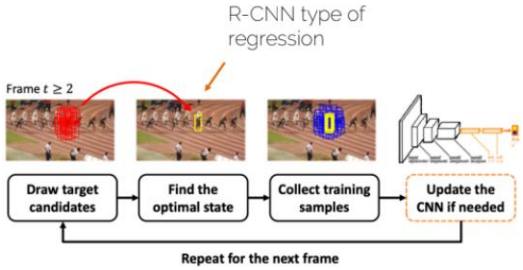
- in the MDNet, we have
  - series of shared layers that do feature extraction
  - domain specific layers that are specific for each target in the scene
- **training**:
  - we train with different sequences (on each there is only one object <- remember, we are solving STT)
  - each sequence has its own domain-specific layer
  - we perform the backpropagation independently per sequence



- **testing:**
  - we need to train only fc6 (or fc4 if we want to be more precise)
  - the extraction feature layers are already learnt and can be freezed



  - **online tracking approach of MDNet:**
    - we draw a series of target candidates

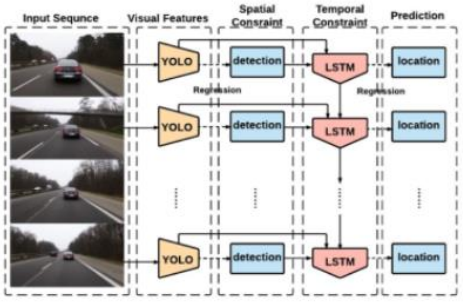


    - we want to regress the candidates to the correct bounding box
- pros:
  - no previous location assumption, the object can move anywhere
  - fine-tuning step is comparatively cheap
  - winner of VOT Challenge 2015
- cons:
  - not as fast as GOTURN

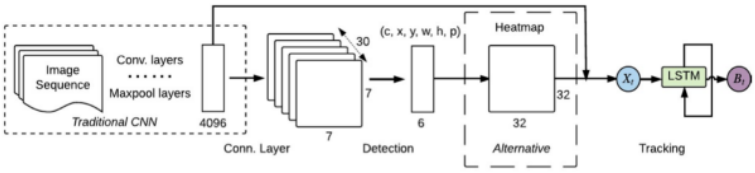
(Temporal) prediction problem

ROLO

- uses CNN (YOLO) to extract visual features from every frame of the sequence
- use LSTM to model the motion



- YOLO outputs a heatmap for the objects position and a 4096 descriptor

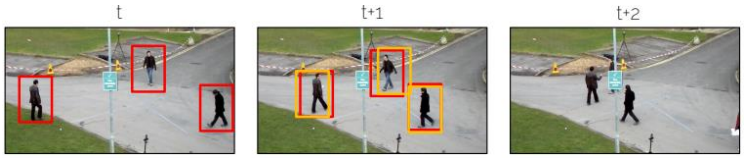


- LSTM outputs a new location for this object

5. Multi-object tracking


- different challenges
  - different objects of the same type
  - heavy occlusions
  - appearance is often similar
- **tracking-by-detection approach**
  - we will focus on algorithms where a set of detections (imperfect) is provided
  - we want to find detections that match and form a trajectory of the object
- Two possibilities
  - **Online tracking**
    - process two frames at a time
    - for real-time applications
    - prone to drifting -> hard to recover from errors or occlusions
  - **Offline tracking**
    - processes a batch of frames
    - good to recover from occlusions (at least short ones)
    - not suitable for real-time applications
    - suitable for video analysis

Online tracking



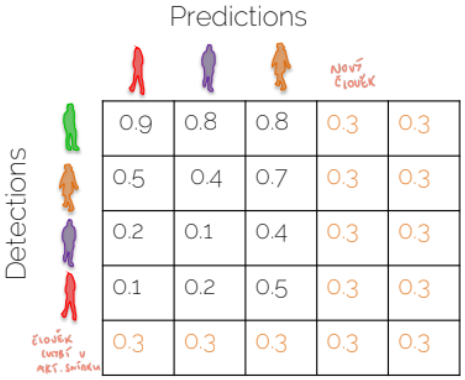
- our solution comprises of three steps
  - track initialization (initial prediction by a detector)
  - prediction of the next position (**motion model**)
  - matching predictions with detections (**appearance model**)

Motion model

- classic approach: Kalman filter
- nowadays: recurrent architecture
- for now: we will assume a constant velocity model (works well at high framerates & without occlusions)

Bipartite matching – Hungarian algorithm

- we want a measurement, that tells us how a set of current predictions are similar to the motion predictions (based on the previous frame)
- we can pick different measurements = IoU, pixel distance, 3D distance
- the smaller the distance, the better the match
- we want to **optimize the whole set at once**
- we construct a table of all the pairwise measurements
  - in columns there are predictions based on the previous frame
    - we add a few columns for new people (which were not in past frames)
    - with some predefined value
  - in rows there are current detections based on the current frame
    - we add a row for people from past, that might be occluded in the current frame
    - with some predefined value
- then we optimize this table to find the right correspondences

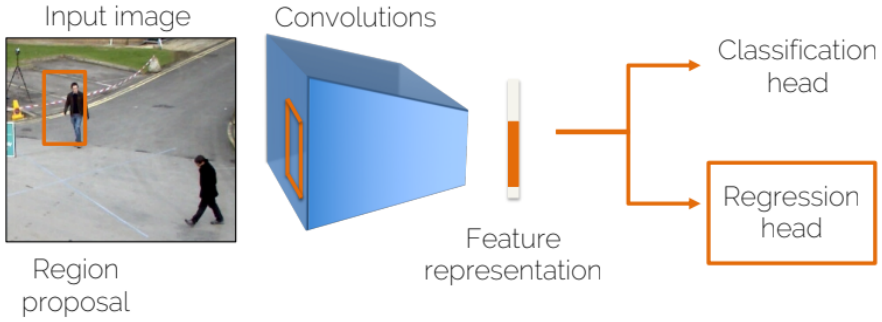


How can we improve?

- better track initialization = better detector
- adding temporal complexity = better motion models
- adding feature complexity = better appearance models
- adding computational complexity = using graph neural networks

Tracktor – baseline model

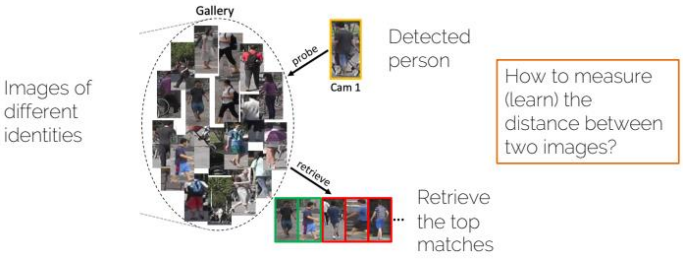
- break down tracking to make it as simple as possible
- similarly to the two-step-detectors



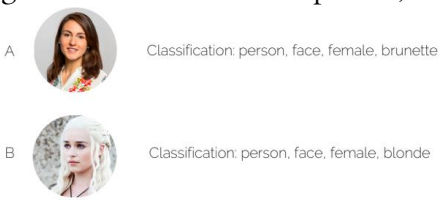
- in Tracktor we take the bounding box from the previous frame as a region proposal
- then we allow regression head to shift this bounding box to better fit the current frame
- Tracktor is trained as a detector but with tracking capabilities
- **pros**:
  - we can reuse an extremely well-trained regressor
  - we can train our model on still images -> easier annotation
  - Tracktor is online
- **cons**:
  - there is no notion of identity in the model -> causes confusion in crowded space
  - being an online tracker, the track is killed if the target becomes occluded
  - the regressor only shifts the box by a small quantity
    - we can't afford large camera motions or large displacement due to low framerate

Improving baseline by modeling appearance = Re-ID

- view change the problem statement and view tracking as **retrieval problem**
  - we have a gallery of different identities
  - if we have a new image of a person (probe), we look at the gallery and want to find the same person



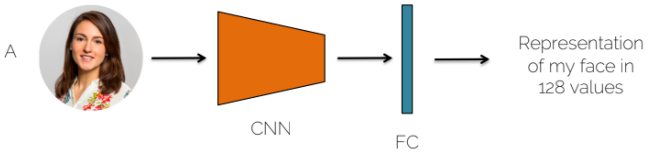
- **Approach 1**: Classification
  - get different attributes – person, face, female, brunette/blonde, … and then compare them



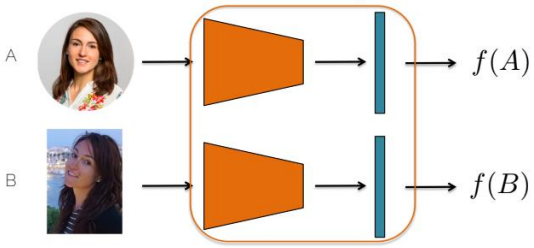
  - how many attributes we need? a lot

- **Approach 2**: Same person or not?
  - **Similarity learning**
    - learning a function that measures how similar two objects are
    - **dummy solution**

      - Testing: new foto – is this the person 1 – yes/no
      - <u>problem</u>: scalability – we need to retrain our model every time a new person enters the training set
    - **Better solution**
      - we will compare two images and say if they depict the same person
      - if the distance between the images is large enough – they don't depict the same person
      - if the distance between the images is bellow a threshold – they represent the same person
      - we gather the images of my gallery = database
      - then we take a new image = query image
      - we run all of these images through the CNN to get an embedding
      - we compute distance between query embedding and database images embeddings
      - we find the top images based on the distance (nearest neighbours, …)
    - <u>How do we force network to learn similarity?</u>



      - we use **siamese network** = shared weights!



    - <u>what is the **loss function**?</u>

$$d(A, B) = ||f(A) - f(B)||^2$$

      - if A, B are the same person, distance is small
      - if A, B are different people, distance is large
      - this will force the network to learn to pull embeddings together/apart

- problem: for A, B far apart, do not want to spend energy pulling them even further
- solution: hinge loss for the negative pair

Better use a Hinge loss: margin

$$\mathcal{L}(A,B) = \max(0, m^2 - ||f(A) - f(B)||^2)$$

- the complete loss for both positive and negative pair is then called **contrastive loss**:

$$\mathcal{L}(A,B) = y^*||f(A) - f(B)||^2 + (1-y^*)max(0, m^2 - ||f(A) - f(B)||^2)$$

Positive pair, reduce the distance between the elements

Negative pair, brings the elements further apart up to a margin

- alternatively, we can use a better way = **triplet loss**
  - we take three images
  - this allows us to learn a ranking


Anchor (A)


Positive (P)


Negative (N)

We want: $||f(A) - f(P)||^2 < ||f(A) - f(N)||^2$

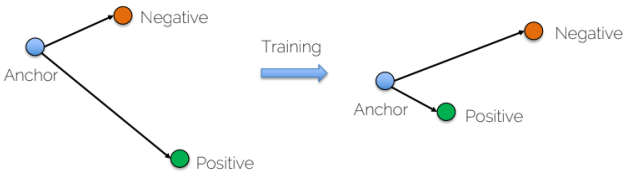
$$||f(A) - f(P)||^2 - ||f(A) - f(N)||^2 < 0$$

$$||f(A) - f(P)||^2 - ||f(A) - f(N)||^2 + m < 0$$

$$\mathcal{L}(A,P,N) = max(0, ||f(A) - f(P)||^2 - ||f(A) - f(N)||^2 + m)$$

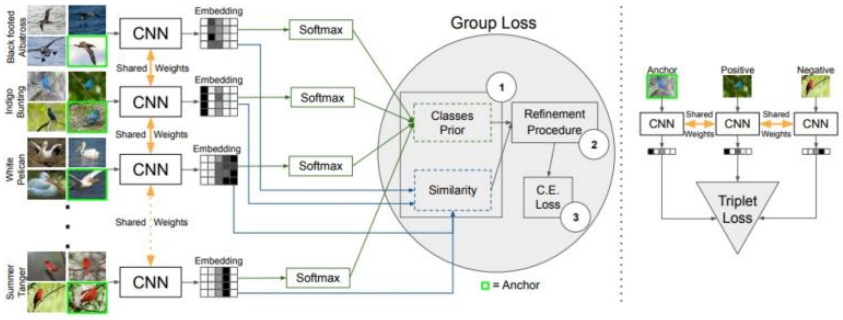
- If $d(A,P) > d(A,N)$ the loss is going to be positive
- If $d(A,P) < d(A,N)$ then I look at the margin

- with triplet loss, we can also use smart method of training = **training with hard cases**
  - we train for a few epochs
  - we choose hard cases where d(A, P) similar to d(A, N)
  - we train with those to refine the distance learned

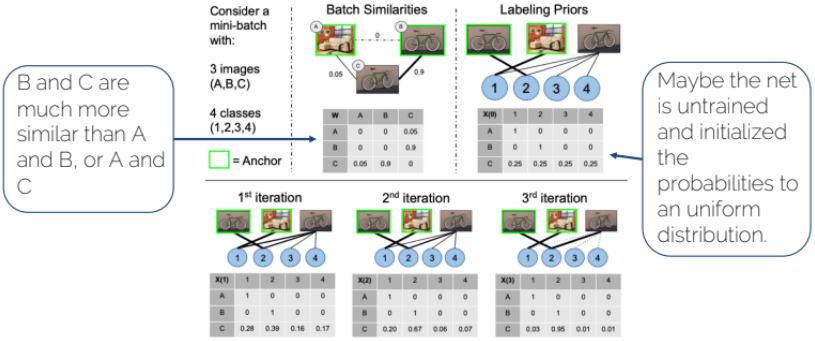


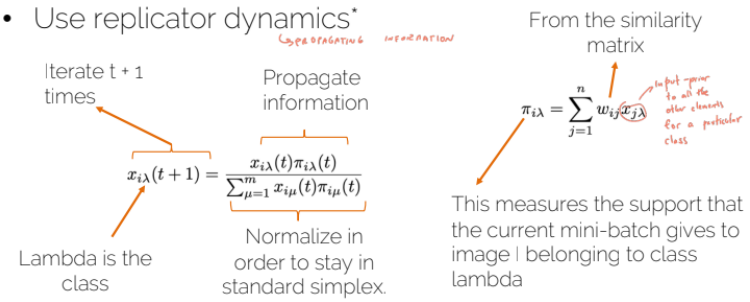
- problems with these losses:
  - the number of pairwise relations in a mini-batch is $O(n^2)$ but
    - contrastive loss considers only O(n/2) relations
    - triplet loss considers only O(2n/3) relations
  - we throw too much information away
  - we need many tricks like hard-negative mining, intelligent sampling and multi-task learning

- **Deep metric learning** – learning a distance function over objects
  - **Group Loss**
    - we take into account the similarity of all samples w.r.t. all other samples (in the minibatch)



    - training procedure consists of <u>three steps</u>
      - **initialization**: we will assign a label X to image. this step is used to get the embeddings well enough so that we can actually do similarity learning, after this, compute the *nxn* pairwise similarity matrix W using the neural network embeddings
      - **refinement (the key part)**: iteratively, refine X considering similarities between all the mini-batch images as encoded in W as well as their labeling preferences





      - **loss computation**: compute the cross-entropy loss of the refined probabilities and update the weights of the neural network using backpropagation

    - The refinement step has no parameters to learn, but it propagates the gradients over the network
    - Group loss achieves state-of-the-art results in retrieval

- <u>SUMMARY: How can we improve similarity learning?</u>
  - **loss**: contrastive vs. triplet
  - **sampling**: choose the best triplets to train with = sample the space wisely = diversity + hard cases
  - **ensembles**: use several networks, each of them trained with a subset of triplets
  - **use some special dassification loss**?

Offline tracking

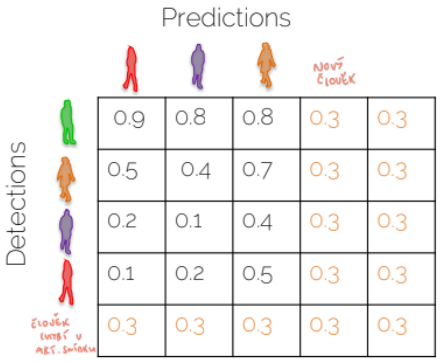
- we process a batch of frames
- there is a good chance to recover from occlusions (at least short ones)
- it is not suitable for real-time applications
- suitable for video analysis

Tracking by detection paradigm

- start with a set of detections
- perform data association

Bipartite matching – Hungarian algorithm (just a repeat)

- we want a measurement, that tells us how a set of current predictions are similar to the motion predictions (based on the previous frame)
- we can pick different measurements = IoU, pixel distance, 3D distance
- the smaller the distance, the better the match
- we want to **optimize the whole set at once**
- we construct a table of all the pairwise measurements
    - in columns there are predictions based on the previous frame
        - we add a few columns for new people (which were not in past frames)
        - with some predefined value
    - in rows there are current detections based on the current frame
        - we add a row for people from past, that might be occluded in the current frame
        - with some predefined value
- then we optimize this table to find the right correspondences



Problems with frame-by-frame tracking:

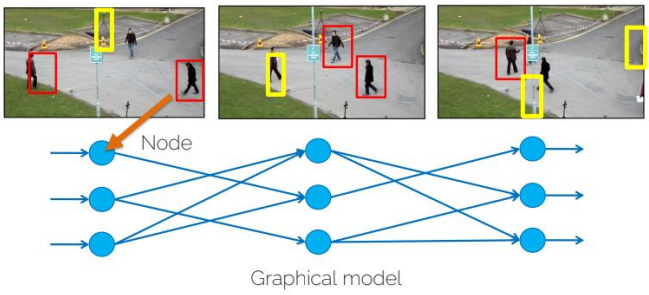
- we cannot recover from errors
- all decisions are essentially local
- hard to recover from errors in the detection step

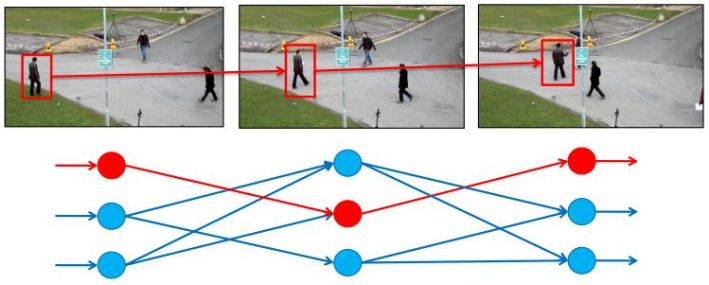
Solution: find the minimum cost solution for ALL FRAMES and ALL TRAJECTORIES -> Graph-based MOT

Graph-based MOT

- we start with a graphical model, where we have layers of nodes

- **node** = detection
- **edge** = flow = trajectory
    - 1 unit of flow = 1 pedestrian

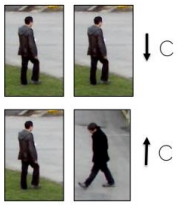




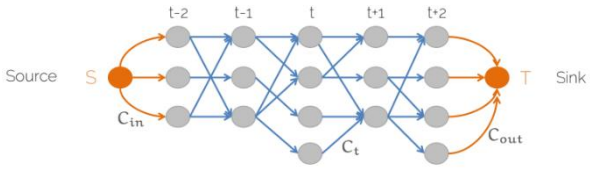
- With this, we want to solve the Minimum Cost Flow Problem
  - "Determine the minimum cost of shipment of a commodity through a network"
  - we define an objective function

    $$\mathcal{T}* = \arg\min_{\mathcal{T}} \sum_{i,j} C(i,j)f(i,j)$$

    where **C(i,j)** is the cost of transition between i,j and **f(i,j)** is an indicator [0,1] if the edge is being used or not

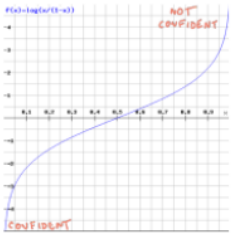


  - we want to minimize this function to get the optimal set of trajectories
  - to construct the whole graph,
    - we add 2 nodes: source S and sink T
    - we also add corresponding edges (to the first and from the last layer) with a cost to start/end a trajectory
    - for the edges between frames, the cost is proportional to the distance between detections

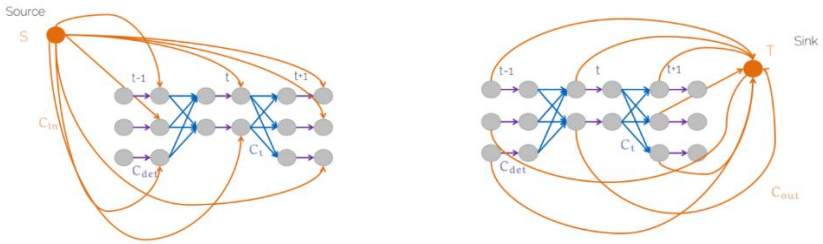


  - flow can only start at the source node and end at the sink node
  - problem: if all costs are positive, then there is a trivial solution = zero flow
  - solution: include somehow negative costs
    - we split nodes in each frame into two
    - in between, we create an edge, whose weight represents probability β_i of a false alarm
    - to do this, we use

$$C_{det} = \log \frac{\beta_i}{1-\beta_i}$$



  - new complete graph (twice the same graph, just with the different connections):



- we use this formulation for a linear program
  - using this approach, we use a common formulation in optimization for which there are very **fast solvers** (e.g. Simplex algorithm)
  - we are guaranteed to converge to the **global optimum**

- Objective function

$$\mathcal{T}* = \arg\min_{\mathcal{T}} \sum_{i,j} C(i,j)f(i,j)$$

- Subject to

Flow conservation at the nodes

$$f_{in}(i) + f_{det}(i) = \sum_j f_t(i,j) \qquad \sum_j f_t(j,i) = f_{out}(i) + f_{det}(i)$$

Edge capacities (either or) NP-hard!!

$$f_{in}(i) + f_{det}(i) \in \{0,1\} \qquad f_{out}(i) + f_{det}(i) \in \{0,1\} \qquad f \in \{0,1\}$$

- If we were to use discrete values for **f**, we would have a NP-hard problem
- If we do a LP-relaxation = replace them with a continuous variable between [0,1];
  - we have a non-NP-hard problem, which can be solved much better
  - also, given the shape of the constraints (total unimodularity) we are guaranteed to end up with an integer solution on the **f**

Edge capacities — LP-relaxation

$$0 \le f_{in}(i) + f_{det}(i) \le 1 \qquad 0 \le f_{out}(i) + f_{det}(i) \le 1 \qquad 0 \le f \le 1$$

- Our problem is also a bit harder – we have multiple terms

$$\mathcal{T}* = \arg\min_{\mathcal{T}} \sum_i C_{in}(i) f_{in}(i) + \sum_{i,j} C_t(i,j) f_t(i,j) + \sum_i C_{det}(i) f_{det}(i) + \sum_i C_{out}(i) f_{out}(i)$$

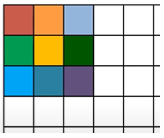
$$C(i) = -\log P(i)$$

- If we replace C(i), this formulation is equivalent to Maximum a-posteriori tracking formulation

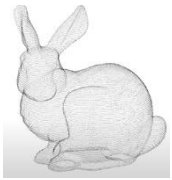
How to improve?

- put learning inside of this process
  - we can improve definition of the costs
  - we can make the graph more complex (include more connections – e.g. connections within frame)
  - end2end learning
    - we can learn features for object tracking (costs)
    - we can learn to do data association – to find a solution on the graph
    - let's exploit this!

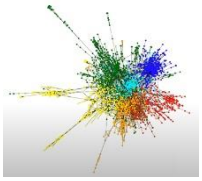
Deep learning on graphs



- let's examine the **domain of the images**, that we have been using so far
  - order of the pixels is important – convolution filter imposes a certain structure



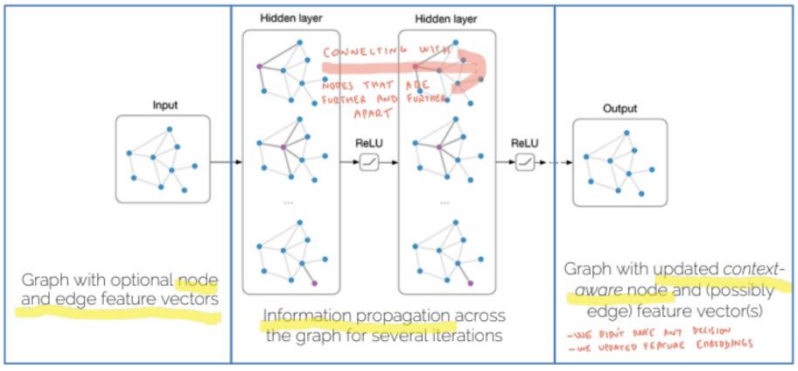
- what happens if we jump into a new domain (e.g. **point clouds**)
  - we have rich information – 3D point location, RGB intensity, semantic
  - problem is that point coulds have a lot of irregularity –> permutation + transformation invariance
  - the order is no more important



- another different domain is represented by **graphs**
  - we have nodes and edges
    - e.g. citation network where node is a paper and connection is a citation
    - e.g. social networks, recommender systems
  - graphs are irregular – we cannot apply convolution
  - if we want to perform any deep learning approach, we have to come up with **new tools**
    - **we need variable sized inputs (number of nodes and edges)**
    - **we need invariance to node permutations**

General idea

- we have nodes and edges embeddings that contains information that we are interested in
- we will perform a series of information propagation steps for several iterations
  - each update step is understood as a "layer" in common NNs
  - every iteration, we reach further with the information propagation



- we end up with a graph, that has updated context-aware nodes&edges feature vectors

Message passing networks

- Notation:
  - Graph: $G = (V, E)$
  - Initial embeddings: $h^{(0)}_{(i,j)}, (i,j) \in E \quad h^{(0)}_i, i \in V$
  - Embeddings after l steps: $h^{(l)}_{(i,j)}, (i,j) \in E \quad h^{(l)}_i, i \in V$

Embedding of edge that connects nodes i and j

Embedding of node i

- Goal
  - encode contextual graph information in node embeddings by iteratively combining neighboring node's features
- The idea is that we will iterate over several **message passing steps**
  - at every iteration, every node **receives features from neighboring nodes**
  - these features are aggregated with an order invariant operation and combined with the current features using a learnable function (that will be same for all the nodes)

$$h^{(l)}_{(i,j)} = \mathcal{N}_e\left([h^{(l-1)}_i, h^{(l-1)}_{(i,j)}, h^{(l-1)}_j]\right)$$

Learnable function

Embedding of node i in the precious message passing step

Embedding of edge (i,j) i in the precious message passing step

Embedding of node j in the precious message passing step

Combine node and edge embeddings

  - after that we **aggregate all these messages** from the neighbours using some order-invariant operation that also has to work with variable sized input (e.g. summation)

$$m^{(l+1)}_v = \sum_{u \in N_v} M^{(l)}(h^{(l)}_u, h^{(l)}_v, h^{(0)}_{(u,v)})$$

Message

Learnable function (e.g. MLP) with shared weights across the entire graph

Aggregation overall all neighbors

  - after that, we will **update our own embedding** by somehow combining my current feature vector and the aggregated message – the combination method has to be the same for all the nodes

$$h^{(l+1)}_v = U^{(l)}(h^{(l)}_v, m^{(l+1)}_v)$$

Embedding update

Learnable function (e.g. MLP) with shared weights across the entire graph

- Most graph Neural network Models can be seen as a specific example of this formulation
  - **example** of similar approach:
    - **creation of message**: node v looks at the neighbours u embeddings and performs an average of them

$$m^{(l+1)}_v = \sum_{u \in N_v} \frac{h^{(l)}_u}{|N_v|}$$

Average neighbors' feature embeddings

    - **update of the embedding** of the node v using learnable matrices W and B & non-linearity

$$h^{(l+1)}_v = \sigma\left(W^{(l+1)} m^{(l+1)}_v + B^{(l+1)} h^{(l)}_v\right)$$

New message

Previous embedding

Non-linearity

Learnable matrices, shared for all nodes

Combine node features with its neig

- another example of similar approach – graph convolution
  - we can take into account the node itself: creation of the message
    - unlike the normal image convolutional filter, the neighbours are not regular -> we have to do a permutation-invariant aggregation operation before the convolution
  - then we do the update step using only one matrix for self-loop & regular neighbours = convolution
    - matrix of weights is of size = #channels out x # channels in

$$m_v^{(l+1)} = \sum_{u \in N_v \cup \{v\}} \frac{h_u^{(l)}}{\sqrt{|N_v||N_u|}}$$

Self loop · Per neighbor degree normalization

$$h_v^{(l+1)} = \sigma\left(W^{(l+1)} m_v^{(l+1)}\right)$$

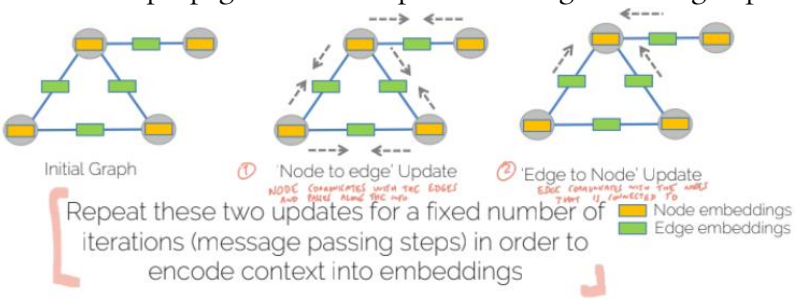
Same learnable matrix for self-loops and regular neighbors

So far, we have only updated node embeddings. What if we are interested in edge features?

- we can work on dual/line graph
- we can perform alternating node/edge updates

A more general framework

- we start with the initial graph
- we will perform the propagation in 2 steps: alternating node/edge updates



- Node2edge update
  - the same that we already discussed
  - combining the embedding of node *i* in the previous frame with embedding of the coinciding edges and nodes

$$h_{(i,j)}^{(l)} = \mathcal{N}_e\left([h_i^{(l-1)}, h_{(i,j)}^{(l-1)}, h_j^{(l-1)}]\right)$$

Embedding of node i in the precious message passing step · Embedding of edge (i,j) in the precious message passing step · Embedding of node j in the precious message passing step

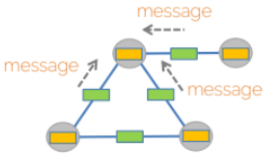
- Edge2node update
  - the opposite operation, the edge embeddings are used to update nodes

$$m_{(i,j)}^{(l)} = \mathcal{N}_v\left([h_i^{(l-1)}, h_{(i,j)}^{(l)}]\right)$$

Learnable function (e.g. MLP) with shared weights across the entire graph

$$h_i^{(l)} = \Phi\left(\{m_{(i,j)}^{(l)}\}_{j \in N_i}\right)$$

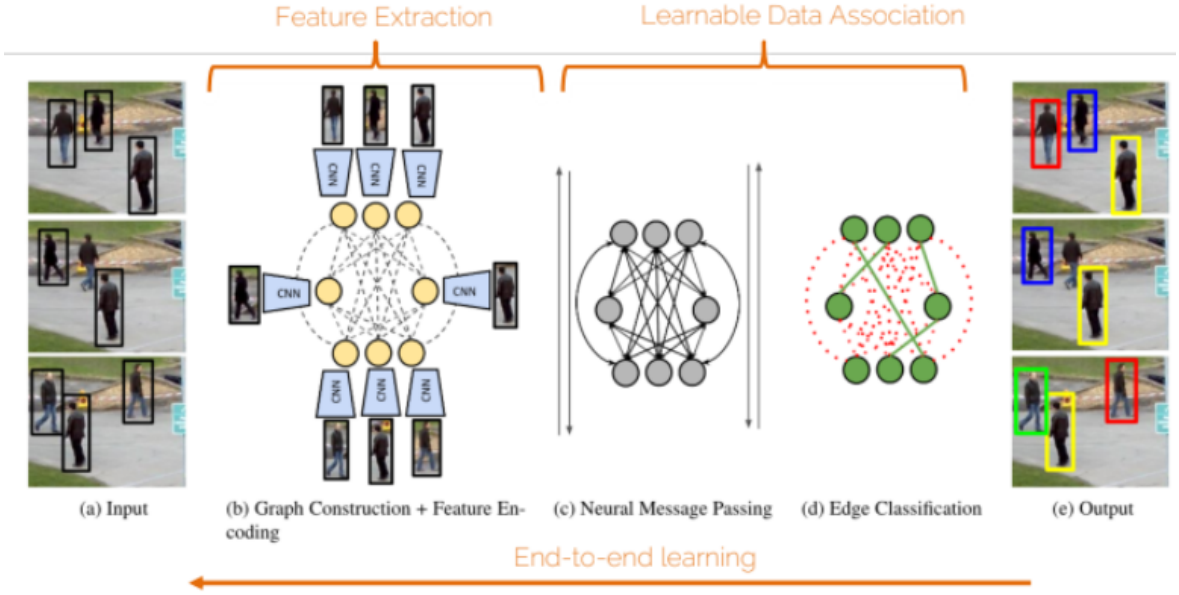
Permutation invariant operation (e.g. sum, mean, max) · Neighbors of node i



  - the aggregation provides each node embedding with contextual information about its neighbours
  - we don't know how many edges are connected to a node

- observe that all the operations used are differentiable -> can be used within end2end pipelines
- after repeating the updates for *l* steps, each node contains information about all nodes at distance *l* (*l*-1)

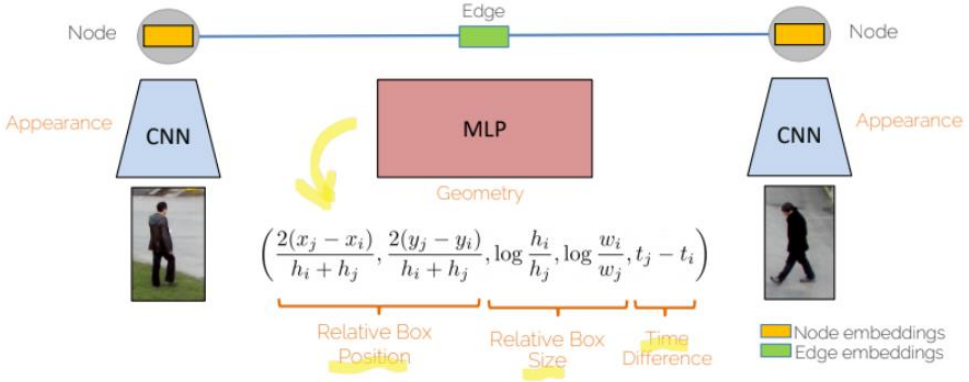
MOT with Message Passing Networks



a) we take our sequence
b) we construct a graph and encode appearance and scene geometry cues into node and edge embeddings
c) we do a series of message passing steps –> this propagates cues across the entire graph
d) we learn to directly predict solutions of the tracking problem by classifying edge embeddings
e) woila

<u>Feature encoding</u>

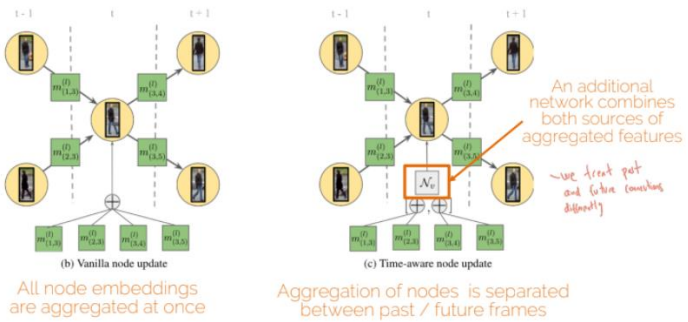
- first we want to encode appearance information – that is linked to a node
- then we want to encode geometry information – that is linked to an edge



- we then want to propagate these embeddings across the entire graph in order to obtain new embeddings encoding high-order information among detections

<u>Possible improvement = Time-aware Message Passing</u>

- we treat past and future connections differently
- aggregation of nodes is separated between past/future frames
- an additional network combines both sources of aggregated features



How to classify edges and perform end2end learning of our method?

- after several iterations of message passing, each edge embedding contains high-order information about other detections
- we feed the embeddings to an MLP that predicts whether an edge is active/inactive

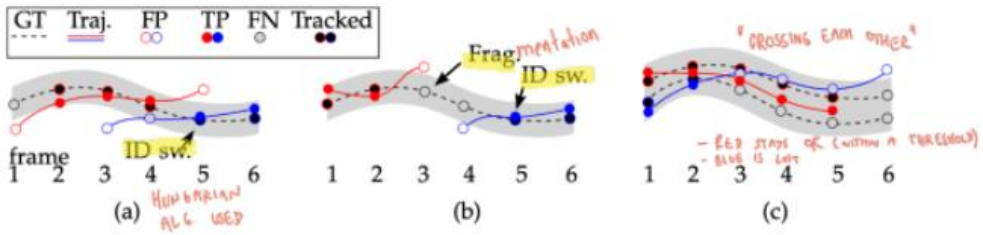
$$\mathcal{L} = \frac{-1}{|E|} \sum_{l=l_0}^{l=L} \sum_{(i,j)\in E} w \cdot y_{(i,j)} \log(\hat{y}^{(l)}_{(i,j)}) + (1 - y_{(i,j)}) \log(1 - \hat{y}^{(l)}_{(i,j)})$$

Sum over the last steps — Weight to balance active / inactive edges — Edge predictions (w. sigmoid) at iteration l — Binary cross-entropy

- after classifying edges, we get a prediction between [0, 1] for each edge in the graph
- directly thresholding solutions could yield infeasible solutions (not frequent in practice)
- we rather use a simple rounding scheme (greedy or LP based), that can be used to obtain the final trajectories
- in practice, with time-aware updates, around 98% of constraints are automatically satisfied and rounding takes negligible time
- the overall method is fast (~6 fps) and achieves SOTA in the MOT Challenge by a significant margin

MOT Evaluation

- there is a useful set of metrics to evaluate the results
- we have to perform matching between predictions and ground truth (Hungarian algorithm)
  - FP = False positives
  - FN = False negatives (missing detections)
  - IDsw = identity switches



a) IDsw is counted because the ground truth track is assigned first to red and then to blue
b) we count IDsw & fragmentation because the ground truth coverage was cut
c) two trajectories overlap each other and cross each other. the red trajectory starts being matched with the upper ground truth and then suddenly gets matched to a bottom one. we decided to keep identities consistent = as long as the detection is within a threshold to be matched, we will match it to a track that does not cause IDsw. So even though the blue is closer, we still match it to the red one. Of course, the blue trajectory is not matched to anything and we have a FN.

- we see from the IDsw cases, that the computation is a bit controversial

MOTA = multi-object tracking accuracy

$$\text{MOTA} = 1 - \frac{\sum_t (\text{FN}_t + \text{FP}_t + \text{IDSW}_t)}{\sum_t \text{GT}_t},$$

Ground truth — detections

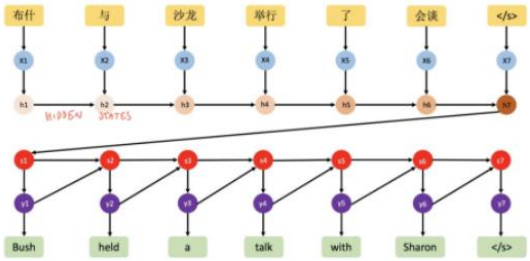
Datasets

- People = MOTChallenge
- Vehicle tracking = KITTI benchmark, UA-Detrac

6. Transformers and DETR

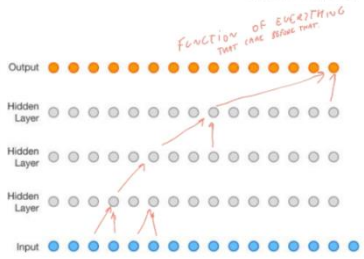
Motivation

- in the same way there was a revolution introduced by **convolutional neural networks** in 2012, there was a new revolution introduced in 2017 introduced by the **attention mechanism**
- it all started with the problems in language translation
    - people were using recurrent neural networks (RNN), where each word depended on the words coming before it – parametrized by the hidden states
    - there was a problem with vanishing gradients as well as insufficient long-short term memory dependencies (in for ex. 20-word sequence)



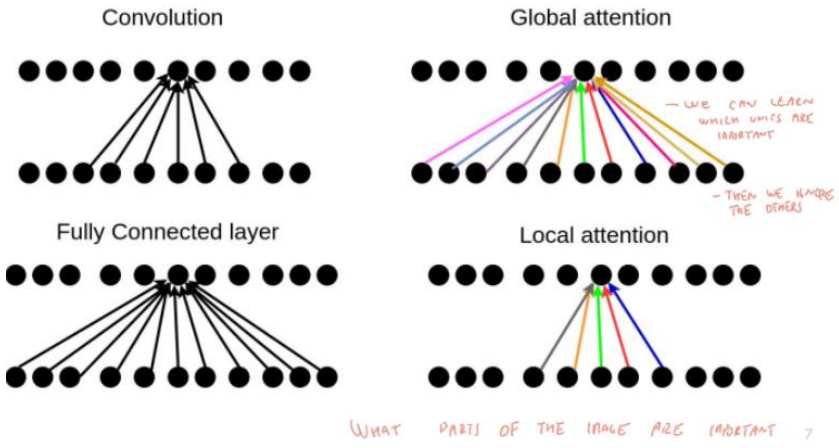
(Sutskever et al., 2014)

    - Google tried to solve this problem using WaveNet - convolution, where each output was a function of everything that came before that. but with this solution, the position of the word was maybe too significant



    - the best solution came in a paper called "Attention is All you Need" that introduced Attention

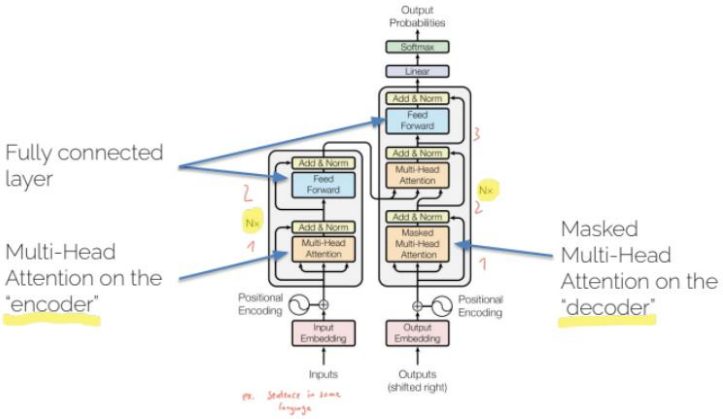
Attention



- **convolution** – rectangular structure – one neuron is dependent only on some particular inputs that come before it
    - in images we have for ex. these 3x3 strides, where every unit is connected to the nine units in the previous layer
- **fully connected layer** – connected to everything before it
- **attention** – every unit learns to know, which other units to attend to
    - instead of everything like fully connected layer, it can choose what is important
    - the units learn what other units it should be connected to = what parts of the image are important for the prediction

Transformers

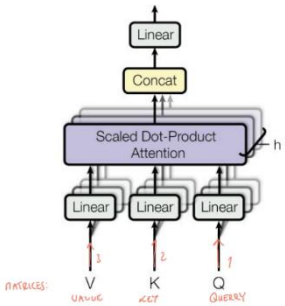
- architecture, that heavily uses attention
- we have some input – like a sentence in some language
- this **input** goes to some layer, which contains multi-head attention (1 left) and a fully connected neural network (2 left)
- **output** we pass through the masked multi-head attention layer (1 right) and then joins it with the output of the input part, pass it through a multi-head attention layer (2 right) and finally through a fully connected part (3 right)

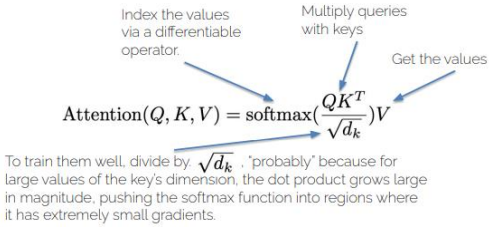


- both left and right modules (called decoder and encoder) are typically **stacked of N** consecutive layers, like in the CNN convolution layers
- final layers of the right module comprise of linear layers and softmax, that predicts probabilities for every word, that we want to translate
- we don't feed transformers one word at a time, but we give it an entire sentence/text
- let's go into more detail

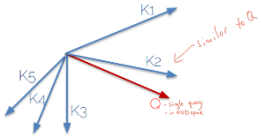
<u>Multi-head attention</u>

- instead of having one matrix of weights to learn, we have 3 weight matrices:
  - value V – here is a bunch of interesting things
  - key K – here is how we can index some things
  - query Q – I would like to know this interesting thing
- **intuition**: take the query Q, find the most similar key K and then find the value V that corresponds to the key
- loosely connected to neural Turing machines
- **computation**:

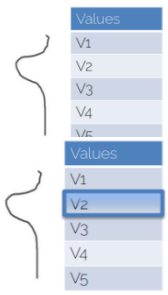




  - the typical size of keys would be 512
- **computation example**:
  - we start with a query Q
  - we start with QK^T = dot product with the keys vectors
    - in real 512D space, most of the other keys will be perpendicular to our query



  - then we compute the $\text{softmax}(\frac{QK^T}{\sqrt{d_k}})$ to induce a distribution over the resulting values
    - the larger the value, the higher its softmax
    - this can be interpreted as differentiable soft indexing

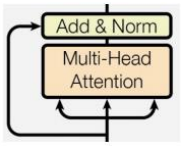


  - then we select the value V where the network needs to attend

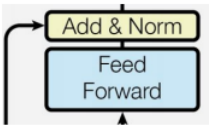
Back to the whole picture

Input path:

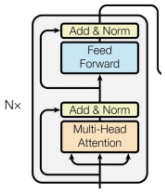
- we have several (like 6-8) parallel attention heads
- they act similarly to how multiple convolutional filters work



- after that, we pass the input into good old fully connected network

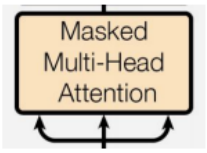


- we repeat this pattern several times (N times – also like 6-8 times)

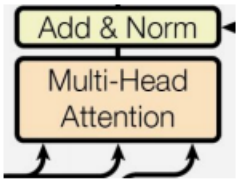


Output path:

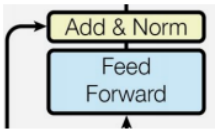
- we feed the output into a similar multi-head attention head, but this time masked
- this ensures that the predictions for the position *i* can only depend on the known outputs at position less then *i*



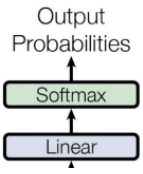
- after that, we have classical multi-head attention, where we merge input and output
  - query Q comes from the **decoder** (left module)
  - keys K and values V come from the **encoder** (right module)
  - it tries to do the following: for this word in English, to what words from Chinese it depends and how to translate them



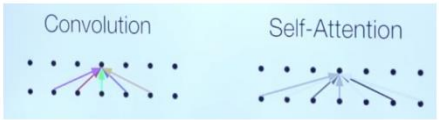
- next there are once again fully connected layers



- finally, we have the linear projection followed by softmax to get the probabilities



What is missing?



- convolution allows to distinguish what information came from where
- attention is more like a weighted average

- in transformers we use positional encoders based on trigonometric series in order for the model to make use of the order of the sequence

- and that is all

Self-attention: complexity

Layer Type	Complexity per Layer	Sequential Operations	Maximum Path Length
Self-Attention	$O(n^2 \cdot d)$	$O(1)$	$O(1)$
Recurrent	$O(n \cdot d^2)$	$O(n)$	$O(n)$
Convolutional	$O(k \cdot n \cdot d^2)$	$O(1)$	$O(log_k(n))$
Self-Attention (restricted)	$O(r \cdot n \cdot d)$	$O(1)$	$O(n/r)$

- where n is the sequence length, d is the representation dimension and k is the convolutional kernel size
- considering that most sentences have a smaller dimension that representation dimension (512), self-attention is very efficient

Training tricks

- we use **Adam optimizer** with **proportional learning rate**

$$lrate = d_{\text{model}}^{-0.5} \cdot \min(step\_num^{-0.5}, step\_num \cdot warmup\_steps^{-1.5})$$

- **residual dropout**
- **label smoothing** – instead of using labels {1, 0}, we smooth them to be something like {0.9, 0.1}
- **checkpoint averaging** – instead of taking the final model, we take different models from various training checkpoints and then do an ensemble between them

Transformers results

- significantly improves SOTA in machine translation

Table 2: The Transformer achieves better BLEU scores than previous state-of-the-art models on the English-to-German and English-to-French newstest2014 tests at a fraction of the training cost.

Model	BLEU		Training Cost (FLOPs)	
	EN-DE	EN-FR	EN-DE	EN-FR
ByteNet [15]	23.75			
Deep-Att + PosUnk [32]		39.2		$1.0 \cdot 10^{20}$
GNMT + RL [31]	24.6	39.92	$2.3 \cdot 10^{19}$	$1.4 \cdot 10^{20}$
ConvS2S [8]	25.16	40.46	$9.6 \cdot 10^{18}$	$1.5 \cdot 10^{20}$
MoE [26]	26.03	40.56	$2.0 \cdot 10^{19}$	$1.2 \cdot 10^{20}$
Deep-Att + PosUnk Ensemble [32]		40.4		$8.0 \cdot 10^{20}$
GNMT + RL Ensemble [31]	26.30	41.16	$1.8 \cdot 10^{20}$	$1.1 \cdot 10^{21}$
ConvS2S Ensemble [8]	26.36	**41.29**	$7.7 \cdot 10^{19}$	$1.2 \cdot 10^{21}$
Transformer (base model)	27.3	38.1	$3.3 \cdot 10^{18}$	
Transformer (big)	**28.4**	**41.0**	$2.3 \cdot 10^{19}$	

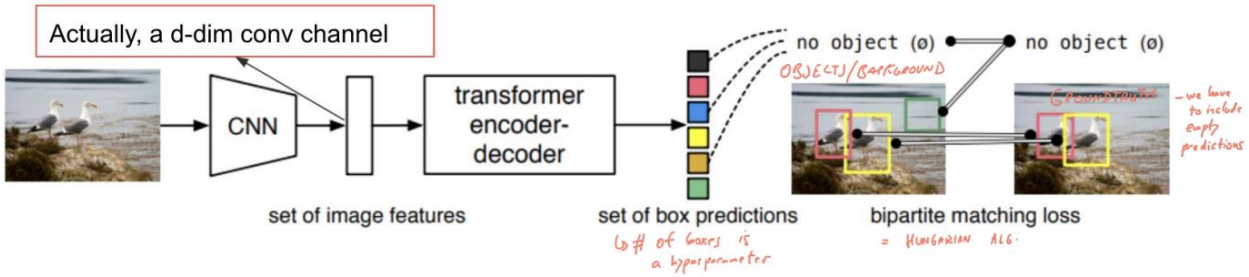
Transformers summary

- very good performance
- launched a new deep-learning revolution in MLP
- building block of NLP models like BERT (Google) or GPT-3 (OpenAI)
- BERT has been heavily used in Google Search
- eventually transformers also made their way to computer vision (and other related field)

Transformers usage in Computer Vision

DETR – end2end object detection with transformers (Facebook, 2020)

- task: given an image, find all the objects in it
- it is similar to one-stage object detectors
- it works very simply
  - we take an image
  - we pass it to a CNN that tries to do feature learning and outputs D-dimensional convolutional feature map
  - this is fed into an encoder of the transformer
  - the decoder predicts a set of box predictions (like 50) – even if there is nothing on the image
  - then we have the ground truth bounding boxes and we do bipartite matching loss (Hungarian alg.)
- DETR predicts the final set of predictions in parallel



- CNN is used for feature learning
- transformer is used to make predictions
  - thanks to this, we don't need NMS and therefore we don't lose valid detections that intersects with each other
- loss function:
  - given a set of prediction and a set of ground truth
    - the network is forced to give us K predictions every time even if there are less objects
    - we have to have the same number of ground truths

$$\hat{\sigma} = \arg\min_{\sigma \in \mathfrak{S}_N} \sum_{i}^{N} \mathcal{L}_{\text{match}}(y_i, \hat{y}_{\sigma(i)}),$$

It also contain empty ground-truth

  - the loss comprises of two parts
    - loss for the class – softmax cross-entropy
    - loss for the bounding box
    - we match the predicted and ground truth boxes using Hungarian alg.

Loss for the class (softmax-like entropy) — Loss for the bounding-box

$$-\mathbb{1}_{\{c_i \neq \varnothing\}} \hat{p}_{\sigma(i)}(c_i) + \mathbb{1}_{\{c_i \neq \varnothing\}} \mathcal{L}_{\text{box}}(b_i, \hat{b}_{\sigma(i)}).$$

$$\mathcal{L}_{\text{Hungarian}}(y, \hat{y}) = \sum_{i=1}^{N} \left[-\log \hat{p}_{\hat{\sigma}(i)}(c_i) + \mathbb{1}_{\{c_i \neq \varnothing\}} \mathcal{L}_{\text{box}}(b_i, \hat{b}_{\hat{\sigma}}(i)) \right]$$

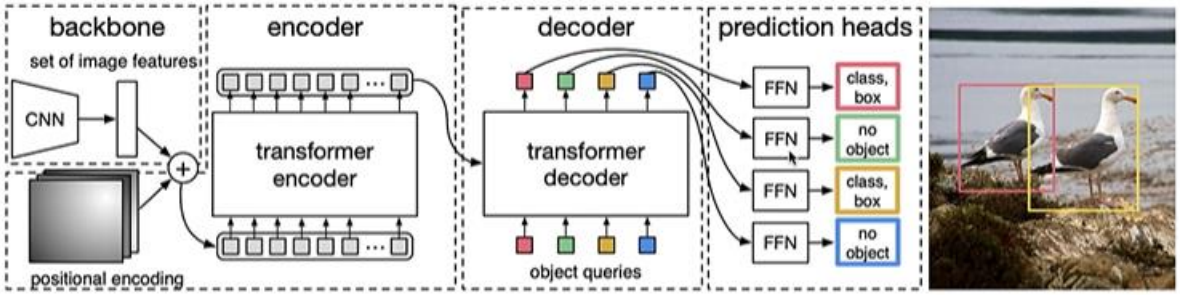
Optimal assignment computed in the first step

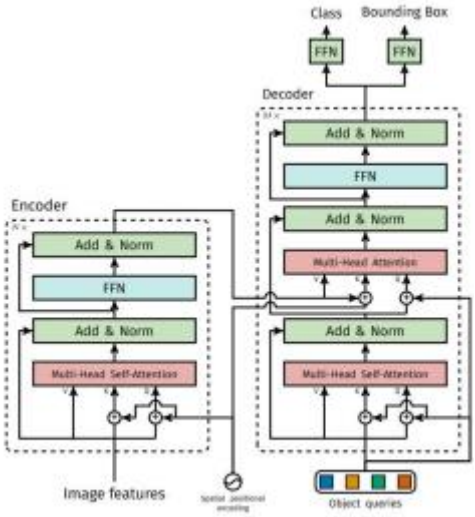
  - there is a problem
    - as we give directly box predictions, we have an issue with relative scaling of the loss = the typical l1 loss will have different scales for small and large boxes even if their relative errors are similar
  - solution:
    - we use a linear combination of the l1 loss and the generalized IoU loss

$L_{\text{box}} = \lambda_{\text{iou}} \mathcal{L}_{\text{iou}}(b_i, \hat{b}_{\sigma(i)}) + \lambda_{\text{L1}} ||b_i - \hat{b}_{\sigma(i)}||_1$

Closer look on DETR

- first we have the backbone, which is a CNN (for example ResNet 50) to which we pass our image
- this produces image features which we combine with positional encodings and give it to the encoder of the transformer
- after the encoder there is the decoder, which takes as input a small, fixed number of learned positional embeddings, which we call object queries and additionally attends to the encoder output
- then we have a fully connected layer which gives us the class probability and the bounding box
    - there is a special class for "no object"
    - the number of predictions needs to match the number of object queries

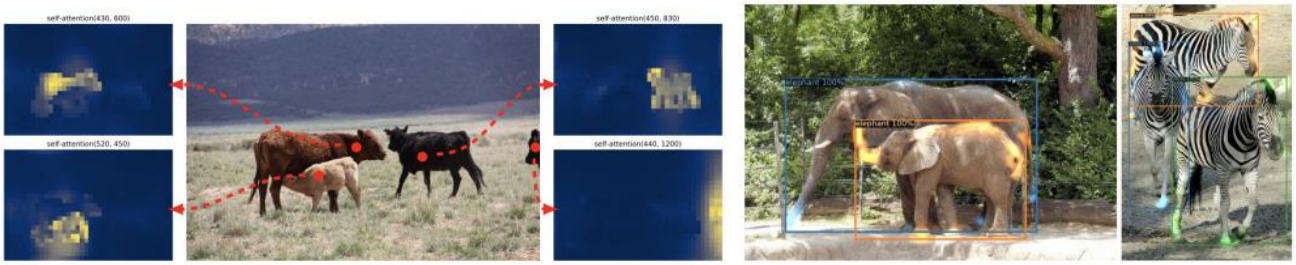




- results:

Model	GFLOPS/FPS	#params	AP	AP_{50}	AP_{75}	AP_S	AP_M	AP_L
Faster RCNN-DC5	320/16	166M	39.0	60.5	42.3	21.4	43.5	52.5
Faster RCNN-FPN	180/26	42M	40.2	61.0	43.8	24.2	43.5	52.0
Faster RCNN-R101-FPN	246/20	60M	42.0	62.5	45.9	25.2	45.6	54.6
Faster RCNN-DC5+	320/16	166M	41.1	61.4	44.3	22.9	45.9	55.0
Faster RCNN-FPN+	180/26	42M	42.0	62.1	45.5	26.6	45.4	53.4
Faster RCNN-R101-FPN+	246/20	60M	44.0	63.9	**47.8**	**27.2**	48.1	56.0
DETR	86/28	41M	42.0	62.4	44.2	20.5	45.8	61.1
DETR-DC5	187/12	41M	43.3	63.1	45.9	22.5	47.3	61.1
DETR-R101	152/20	60M	43.5	63.8	46.4	21.9	48.0	61.8
DETR-DC5-R101	253/10	60M	**44.9**	**64.7**	47.7	23.7	**49.5**	**62.3**

- sample images:



- DETR can also be used for **panoptic semantic segmentation**
  - we only need a special head
  - a binary mask is generated in parallel for each detected object, then the masks are merged using pixel-wise argmax

Model	Backbone	PQ	SQ	RQ	PQ^{th}	SQ^{th}	RQ^{th}	PQ^{st}	SQ^{st}	RQ^{st}	AP
PanopticFPN++	R50	42.4	79.3	51.6	49.2	82.4	58.8	32.3	74.8	40.6	37.7
UPSnet	R50	42.5	78.0	52.5	48.6	79.4	59.6	33.4	75.9	41.7	34.3
UPSnet-M	R50	43.0	79.1	52.8	48.9	79.7	59.7	34.1	78.2	42.3	34.3
PanopticFPN++	R101	44.1	79.5	53.3	**51.0**	**83.2**	60.6	33.6	74.0	42.1	**39.7**
DETR	R50	43.4	79.3	53.8	48.2	79.8	59.5	36.3	78.5	45.3	31.1
DETR-DC5	R50	44.6	79.8	55.0	49.4	80.5	60.6	**37.3**	**78.7**	**46.5**	31.9
DETR-R101	R101	**45.1**	**79.9**	**55.5**	50.5	80.9	**61.7**	37.0	78.5	46.0	33.0

- **training** of DETR takes 3 days using 16 V100 GPUs
  - recently, there are more efficient modifications
- **testing** is faster and convenient

DETR summary

- reaches good results in object detection without any bells and whistles
- gets rid of NMS
- with a minor modification can be used for panoptic segmentation
- shows that transformers can be used in Vision

7. Pedestrian Trajectory Prediction

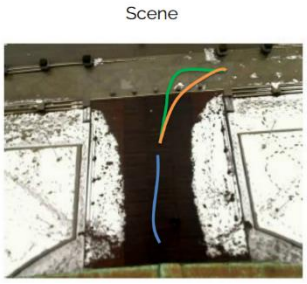
Motivation

- autonomous vehicles must predict the future movements of pedestrians in order to avoid fatal collisions
- social robots move autonomously through crowded scenes and interact with moving humans
    - people can anticipate movement of other people and adjust our trajectory accordingly
- motion models improve the performance of multi-object trackers

Task definition

Introduction

- human motion behavior is influenced by a variety of different factors
    1. **Destination** – we don't know
    2. **Personal Preferences** – slow/fast movement
    3. **Human-Space Interactions** – house, path
    4. **Human-Human Interactions** - avoidance



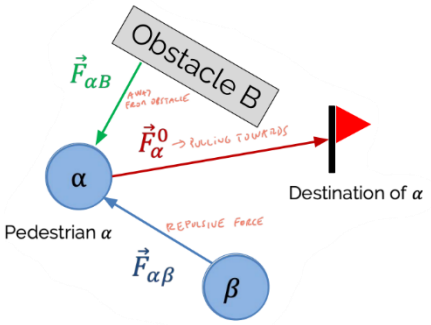
usualy 8 timestamps

- Sequence of observations: $X_i^t = (x_i^t, y_i^t)$ for $t = 1, \dots, T^{Obs}$
- Ground Truth: $Y_i^t = (x_i^t, y_i^t)$ for $t = T^{Obs+1}, \dots, T^{Pred}$
- Prediction: $\hat{Y}_i^t = f_W(X_i) = (x_i^t, y_i^t)$ for $t = T^{Obs+1}, \dots, T^{Pred}$

Social Force Model

- first model that tried to model the interactions (1998)
- mathematical model of dynamics
- pedestrian act in force field F like particles in e.g. electric field
- second Newton's law
- trajectory x(t) is then solution to a differential equation
- pedestrian alpha wants to reach the destination
    - there is a force that is pulling him towards the destination

$$\ddot{x} = \frac{dx^2}{dt^2} = \frac{F(t)}{m}$$



desired velocity, direction of destination

$$\vec{F}_\alpha^0 = \vec{F}_\alpha^0(\vec{v}_\alpha(t), v_\alpha^0\ \vec{e}_\alpha)$$

    - there are also other pedestrians that effect the trajectory by creating expulsive force that is proportional to the distance

distance betw. peds

$$\sum_\beta \vec{F}_{\alpha\beta}(\vec{e}_\alpha, \vec{r}_{\alpha\beta})$$

    - if there is an obstacle, it also creates a repulsive force

$$\sum_B \vec{F}_{\alpha B}(\vec{e}_\alpha, \vec{r}_B^\alpha)$$

    - the total solution than has a form

$$\vec{F}_\alpha(t) := \underbrace{\vec{F}_\alpha^0(\vec{v}_\alpha, v_\alpha^0 \vec{e}_\alpha)}_{\substack{\text{acceleration term} \\ \text{towards dest.}}} + \underbrace{\sum_\beta \vec{F}_{\alpha\beta}(\vec{e}_\alpha, \vec{r}_{\alpha\beta})}_{\substack{\text{force} \\ \text{between pedestrians}}} + \underbrace{\sum_B \vec{F}_{\alpha B}(\vec{e}_\alpha, \vec{r}_B^\alpha)}_{\substack{\text{force} \\ \text{between ped. and obstacles}}}$$

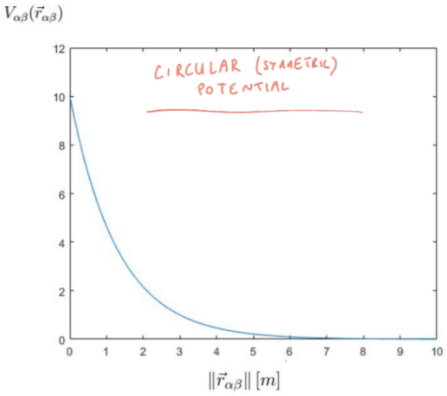
- we can simulate this model

How do the forces look like?

- they are negative gradients of a potential
- objects try to minimize their energy

$$\vec{F}_{\alpha\beta}(\vec{r}_{\alpha\beta}) = -\nabla_{\vec{r}_{\alpha\beta}} V_{\alpha\beta}(\vec{r}_{\alpha\beta}) \text{ with } V_{\alpha\beta}(\vec{r}_{\alpha\beta}) = V^0 \, e^{-\frac{\|\vec{r}_{\alpha\beta}\|}{\sigma}}$$

- parameters V_0 and σ shape the potential -> they effect strength of interaction between the pedestrians
- **example**: interaction potential



$$V_{\alpha\beta}(\vec{r}_{\alpha\beta}) = V^0 \, e^{-\frac{\|\vec{r}_{\alpha\beta}\|}{\sigma}}$$

- once we start increasing the parameters, the motion is more and more non-linear

Drawbacks of social force model

- for each pair of agents, we have to get the right parameters
- shapes of interaction functions are handcrafted

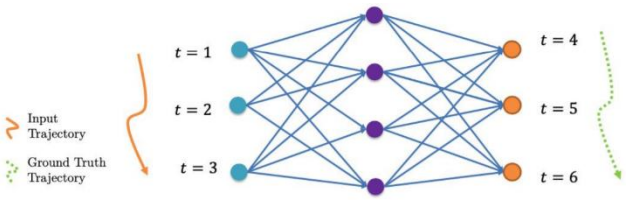
Solution:

- let's try to use learnable functions

Neural networks for Trajectory Prediction

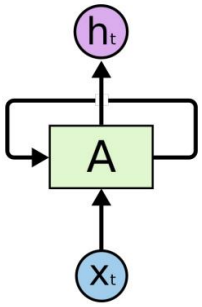
Naïve prediction model with FC layers

- not optimal <- FC layers do not account for sequential and temporal behavior of trajectories
- we have to move towards Recurrent Neural Networks

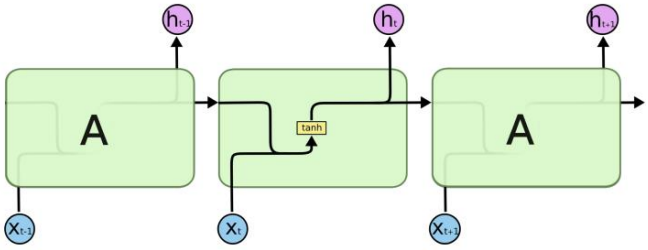


Recurrent neural networks

- function, that takes input x and uses a previous output of the previous timestamp



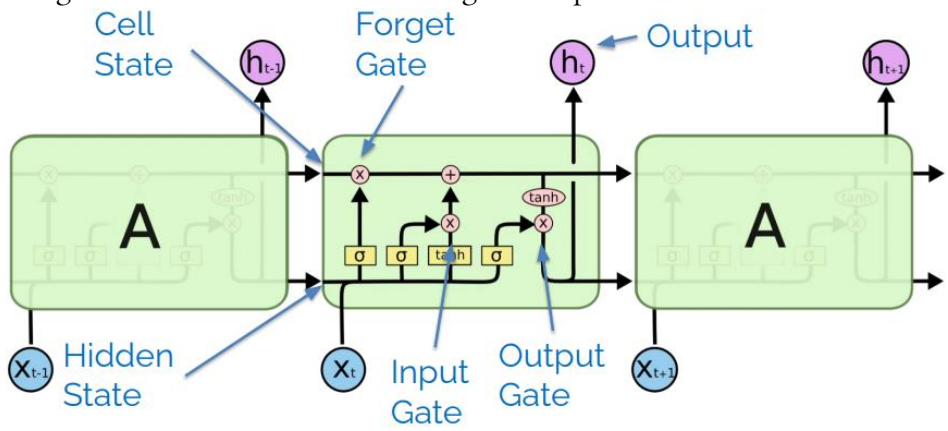
$$h_t = \tanh\left(W \cdot h_{t-1} + V \cdot x_t + b\right)$$



- this allows us to link data points across different timestamps
- problem: vanishing gradients – forgetting what was in past (<- derivative of tanh is very small)
- solution: use LSTM

Long Short-term Memory (member of RNN)

- uses gated structure to deal with the gradient problem



- forget gate – how much of the old cell state we pass through
  - if it is 1, then it is almost not forgetting at all, gradient is directly propagating
- input gate – how much of the current information we save into the cell state
- output gate – what information we output at that timestamp

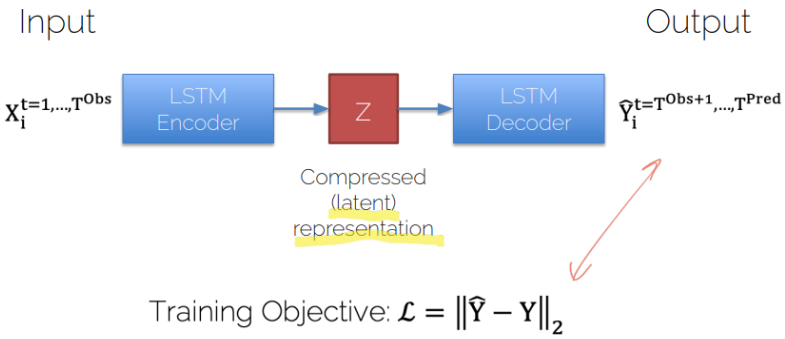
<u>Simple model for trajectory prediction</u>

- we could just take an input, pass it into the LSTM and produce output

- but it has been shown that a single LSTM does not work well
  - because we use the same for encoding & decoding
  - we have to use **encoder-decoder architecture**

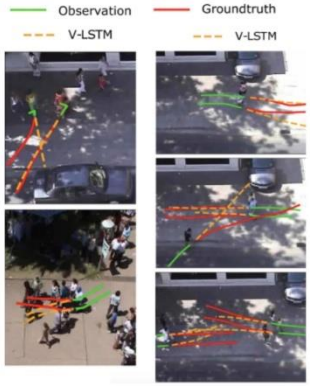
LSTM Encoder-Decoder Architecture

- we pass the input through LSTM encoder to get a latent representation, that stores the information of the past trajectory
- this latent representation is passed into LSTM decoder, to predict future trajectory
- the encoder and decoder are two different models

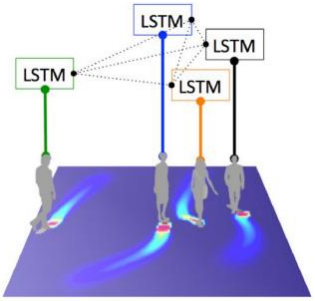


<u>Vanilla LSTM</u>

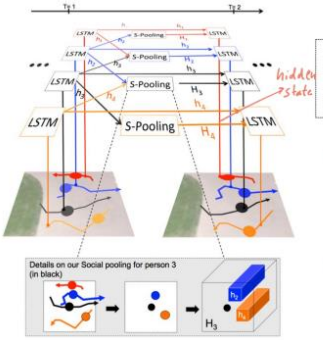
- <u>problem</u>: vanilla LSTM is not able to predict trajectories of interacting pedestrians
- this is because we predict trajectory for each agent independently
- <u>solution</u>: we have to include the social part of the problem

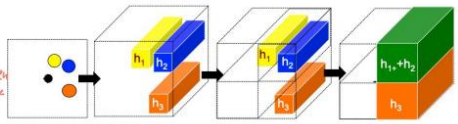


Social LSTM (2016)



- modelling social interactions inside LSTM
- applies social pooling between neighboring pedestrians in each time step
- on the left we have four pedestrians, and we have their hidden states
- we pool that along with the neighboring pedestrians
- we make a discrete grid where we discretize our neighbourhood and we use the hidden states of the neighbouring pedestrians
    - if two pedestrians are in the same grid cell, we just add up their hidden states





- Interaction Module pools hidden states of LSTM of pedestrian in vicinity
- Pooled hidden states are passed to decoder for next step prediction

- this information is passed into decoder to create trajectory prediction
- results:
    - social pooling can resolve social interactions

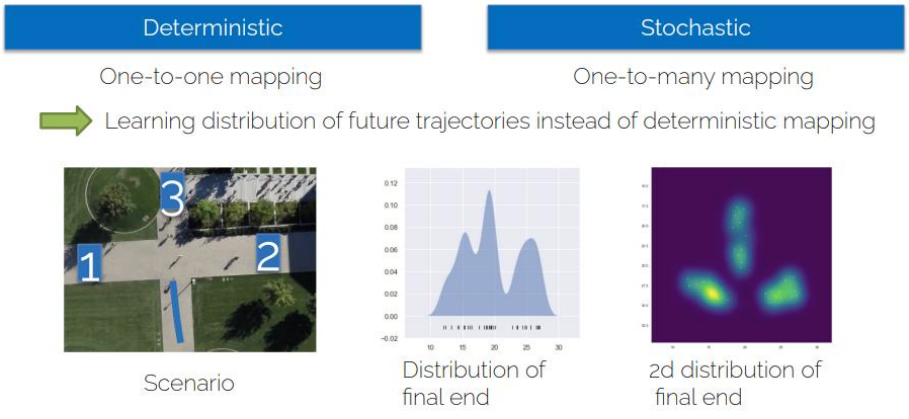
Generative models

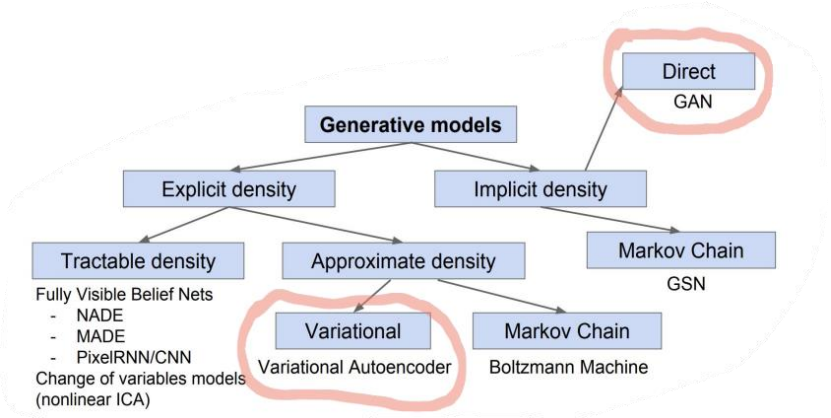
Problem of all the past models:

- all of our results so far are unimodal – every past trajectory has one future trajectory = DETERMINISTIC

Solution:

- use stochastic/generative models



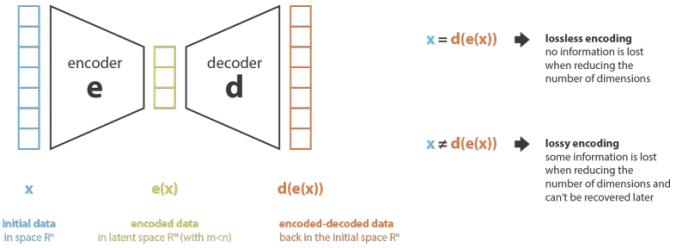


Variational Autoencoder

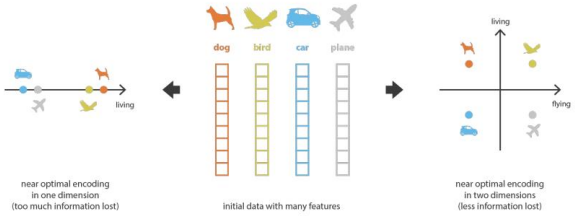
- assumes Gaussian distribution and approximates it with lower bound

Autoencoder:

- takes an input, represents the input in a lower representation latent space and uses this representation to represent this code
- it extracts the important features that are necessary to reconstruct the same input



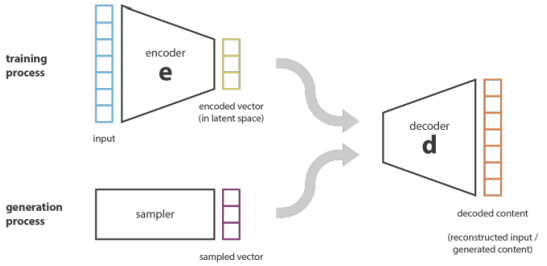
- we have to find the right dimensionality of latent space, otherwise we might lose too much features or overfit



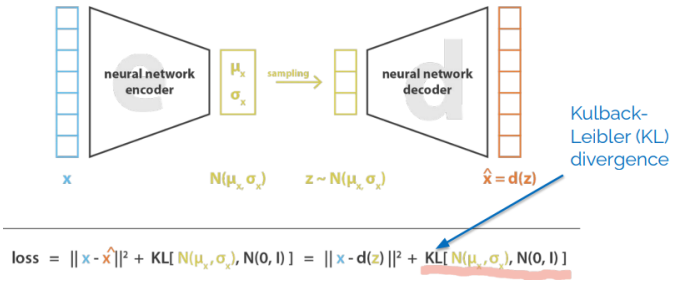
- problem: latent space has no structure and is meaningless – two points next to each other might represent totally different objects
- solution: variational autoencoder

Variational autoencoder

- method that allows to sample new samples
- instead of predicting a single variable, we predict a Gaussian distribution
  - and then we sample from this distribution and get the output
- training: we encode our input into latent vector and decode the content
- sampling: we use sampling to get new instances



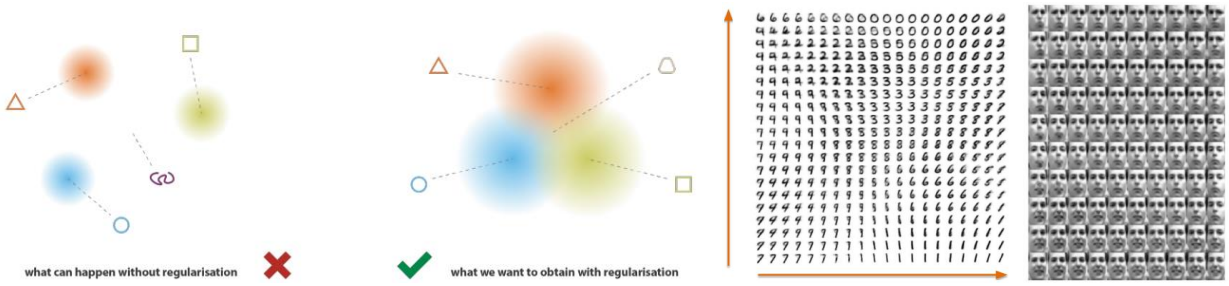
- to do this, we have to introduce regularization in the loss = **Kullback-Leibler (KL) divergence**



  - KL measures similarity between two distributions

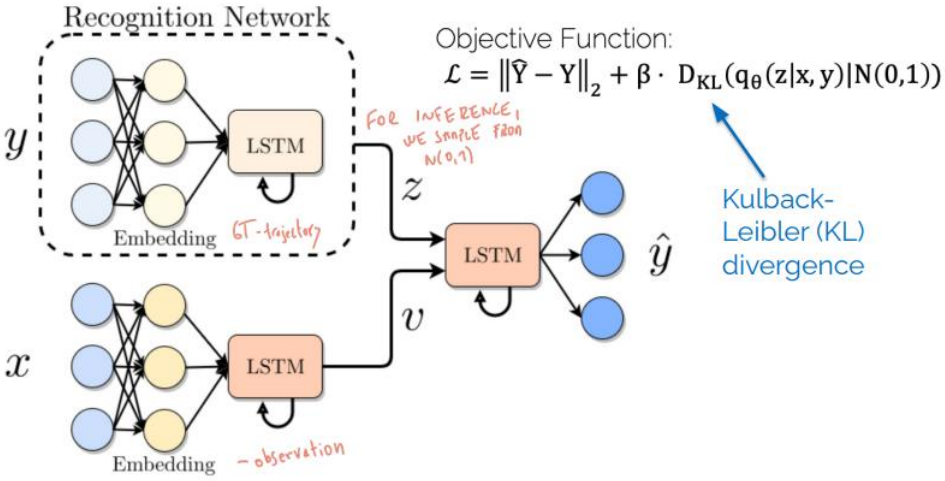
$$D_{\mathrm{KL}}(P \parallel Q) = \sum_{x \in \mathcal{X}} P(x) \log\left(\frac{P(x)}{Q(x)}\right) \qquad D_{\mathrm{KL}}(P \parallel Q) = \int_{-\infty}^{\infty} p(x) \log\left(\frac{p(x)}{q(x)}\right) dx$$

- examples:



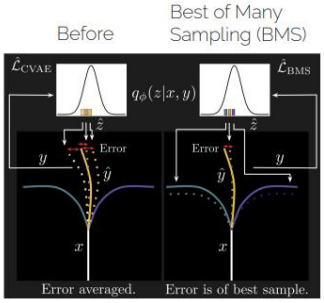
Conditional Variational Autoencoder

- we encode our ground truth trajectory as **y** to get latent representation **z**
- in addition, we can condition the generation with our observation **x**, that has latent representation **v**
- we concatenate **z**, **v** and produce the final output
- during inference, we don't have recognition network (requires the ground truth), but we sample from N(0,1)
    - that is because we train with KL divergence, which ensures that **z** is normally distributed

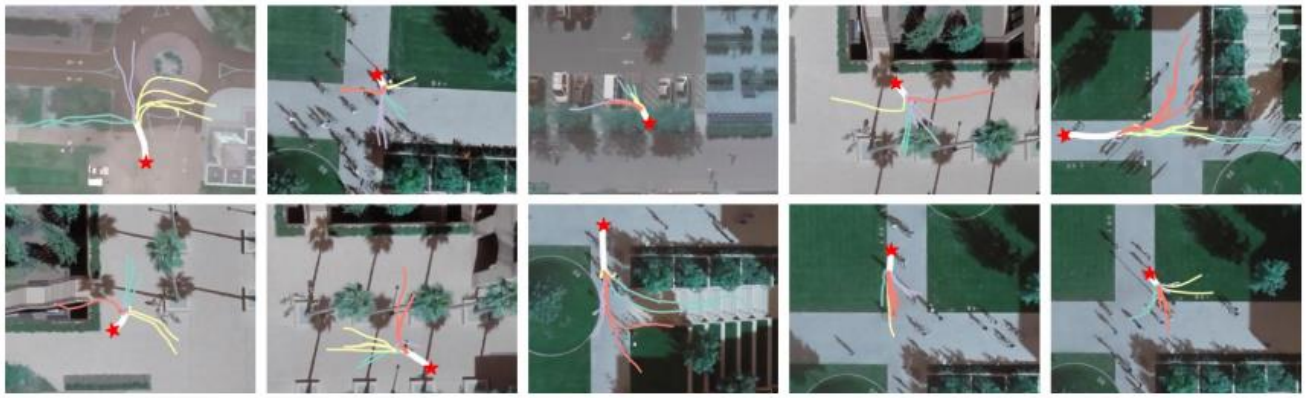


- suggested improvement: **Best-of-many sampling**
    - instead of sampling one trajectory, we sample multiple trajectories
    - but we only sample back the one with the minimal error
    - objective function

$$\mathcal{L} = \min_j \|\hat{Y} - Y\|_2 + \beta \cdot D_{KL}(q_\theta(z|x,y)|N(0,1))$$



- results:

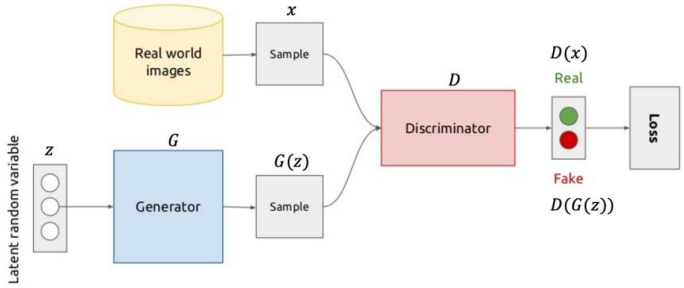


K-mean clustered trajectories

    - we sample multiple times

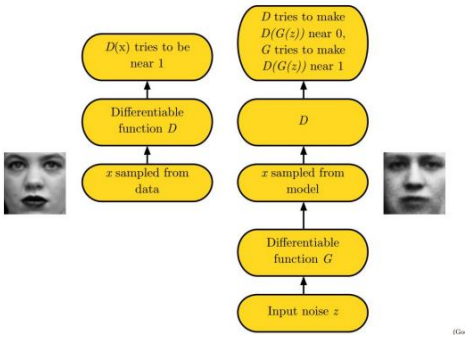
Generative Adversarial Networks (GANs)

- really popular
- used for mostly image generation, now also for trajectory predictions
- basic idea:
  - we have a generator G and a discriminator D
  - generator learns the ground truth distribution, that is given by the samples
  - discriminator takes the samples and learns to compare the ground truth distribution and generators distributions
  - we want generator to generate instances with the same distribution as the ground truth



  solutions
    - **z** can be uniform or gaussian (could be any)
    - **G** takes this noise and creates samples
    - **D** discriminates between real/fake

- example – face generation:
  - if we take the real samples and feed it into discriminator, D tries to say 1 – it's real
  - if we take input noise z, feed it to the generator, sample from the model an image and feed that to the discriminator, which tries to say 0 – it's fake



- loss functions:
  - discriminator loss:

$$J^{(D)} = -\frac{1}{2}\mathbb{E}_{\boldsymbol{x}\sim p_{\text{data}}} \log D(\boldsymbol{x}) - \frac{1}{2}\mathbb{E}_{\boldsymbol{z}} \log\left(1 - D\left(G(\boldsymbol{z})\right)\right)$$

  - generator loss:

$$J^{(G)} = -J^{(D)}$$

  - equilibrium is a saddle point of discriminator los
  - D provides supervision (i.e. gradients) for G
  - in practice, we often use **heuristic method**
    - discriminator loss is the same
    - generator loss is

$$J^{(G)} = -\frac{1}{2}\mathbb{E}_{\boldsymbol{z}} \log D\left(G(\boldsymbol{z})\right)$$

    - generator maximizes the log-probability of D being mistaken
    - G can still learn even when D rejects all generator samples
- training loop of the vanilla GAN:
  - learning is not very stable
  - most of the time the discriminator wins
  - pure GAN also does not make sense for trajectory prediction

for number of training iterations **do**
  for k steps **do**
    - Sample minibatch of m noise samples $\{\boldsymbol{z}^{(1)}, \ldots, \boldsymbol{z}^{(m)}\}$ from noise prior $p_g(\boldsymbol{z})$.
    - Sample minibatch of m examples $\{\boldsymbol{x}^{(1)}, \ldots, \boldsymbol{x}^{(m)}\}$ from data generating distribution $p_{\text{data}}(\boldsymbol{x})$.
    - Update the discriminator by ascending its stochastic gradient:

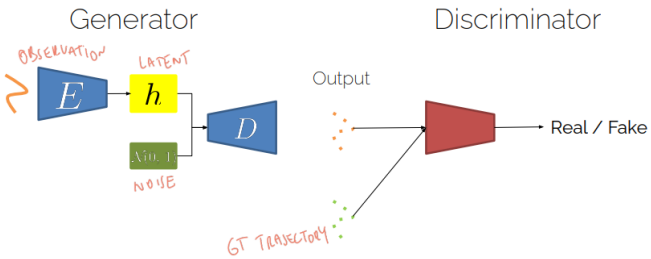
$$\nabla_{\theta_d} \frac{1}{m} \sum_{i=1}^{m} \left[\log D\left(\boldsymbol{x}^{(i)}\right) + \log\left(1 - D\left(G\left(\boldsymbol{z}^{(i)}\right)\right)\right)\right].$$

  end for
  - Sample minibatch of m noise samples $\{\boldsymbol{z}^{(1)}, \ldots, \boldsymbol{z}^{(m)}\}$ from noise prior $p_g(\boldsymbol{z})$.
  - Update the generator by descending its stochastic gradient:

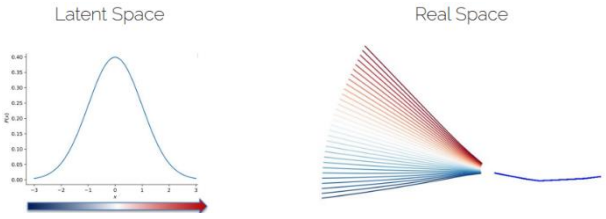
$$\nabla_{\theta_g} \frac{1}{m} \sum_{i=1}^{m} \log\left(1 - D\left(G\left(\boldsymbol{z}^{(i)}\right)\right)\right).$$

end for

GANs for Trajectory prediction:

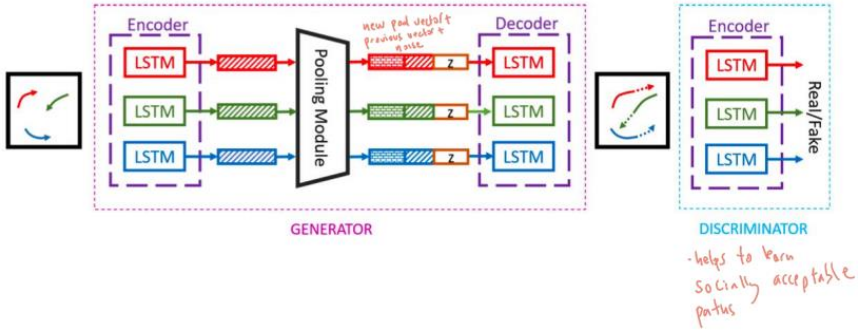


- we condition our discriminator on the noise and observation
- different latent space samples result into different real space output
    - that is really nice, structures the latent space in a meaningful way



Social GAN (SGAN, 2018)

- combines GAN with social interaction
- before, we had social LSTM, but that was a social pooling that required grid and only the pedestrians inside this grid would be considered

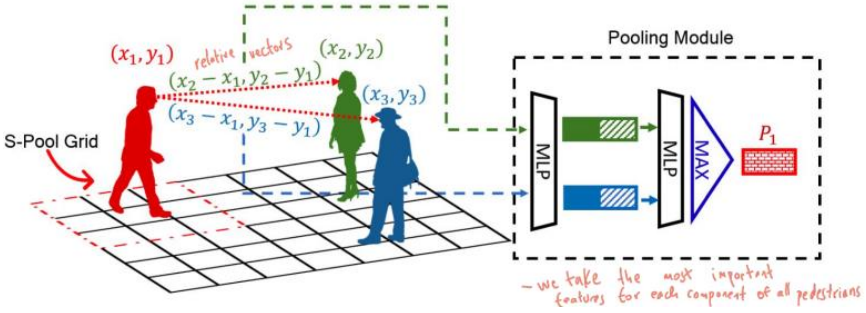


- similarly to social LSTM, we have a pooling module, that provides with a pooling vector
- as the decoder input, we combine the pooling vector with previous vector and noise **z**
- the discriminator network helps us to learn socially reasonable trajectories
- pooling module:
    - max operation is symmetric (initial order does not matter)

$$\max_j \text{MLP}([x_j - x_i, y_j - y_j, h_j^{t-1}])$$

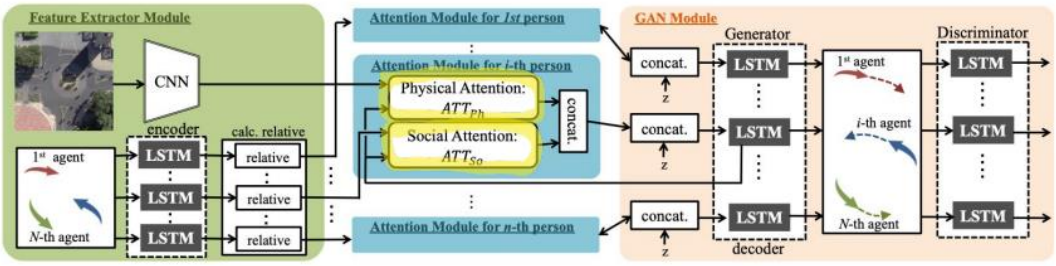
hidden state of the other person

    - pooling is not restricted to grid

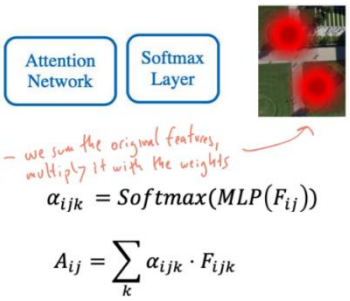


        - for each pedestrian, we get one tensor containing the most important features
- results:
    - pros: quite nice
    - cons: as we only take the max in the pooling, we kind of lose the information how many pedestrians are in there and we also cannot propagate the gradients back correctly

SoPhie



- more advanced model
- combining not only social but also physical interactions
  - we use an input image, that is passed through CNN
- instead of max-pooling we use soft-attention
- attention = weighting of the visual features $F_{i,j,k}$



- results:

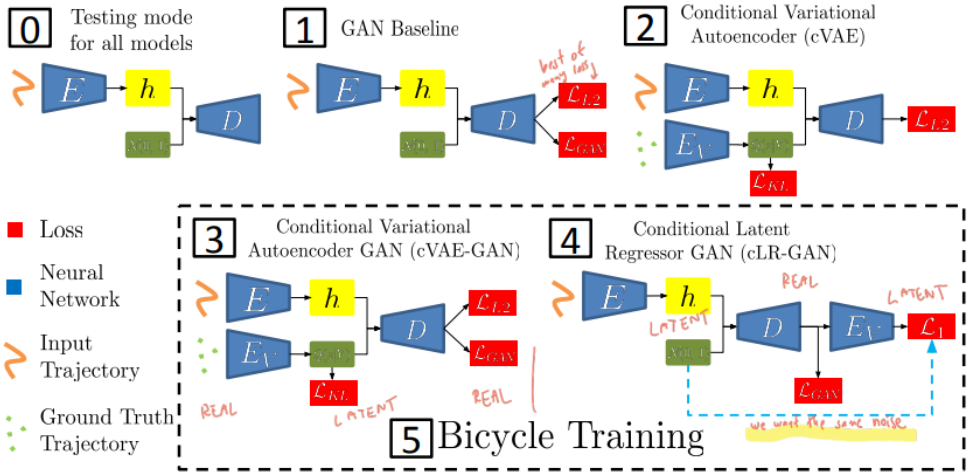
  - very good
  - but can we do even better?

Multimodal Image-to-Image translation

- concept from computer vision, where we want to have for ex. a realistic image by daylight given a night image

- this is kind of similar to the problem of trajectory prediction
  - we have one observation and there is a set of realistic and possible outputs
  - so, in 2017, a new concept of GANs arised

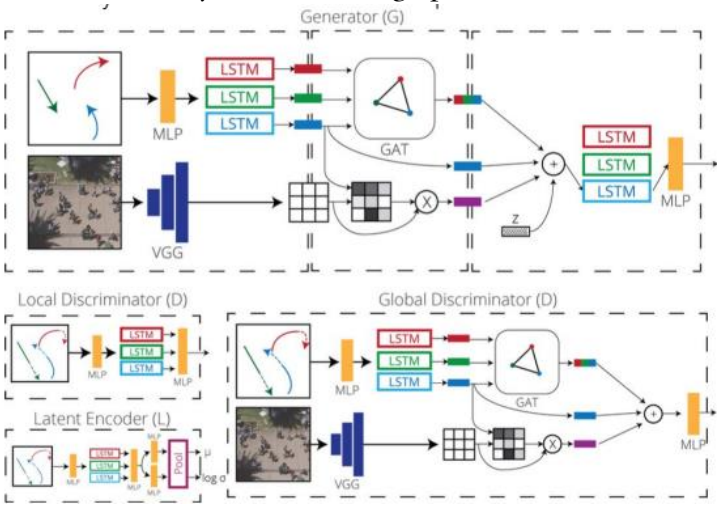
Bicycle GAN



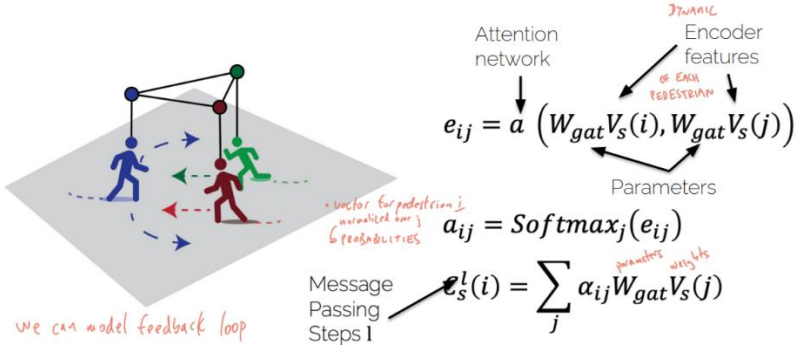
- we combine a GAN (scheme 1) together with conditional variational autoencoder (scheme 2) to create **Conditional Variational Autoencoder GAN** (scheme 3)
- scheme 3 takes the real space (ground truth trajectory), encodes it into the latent space and goes back to the real space
    - but it still does not ensure that we are efficiently using the latent space
- that's why we use a **Conditional latent regressor GAN** (scheme 4)
    - here we draw a latent code, generate a trajectory and use our encoder to go back to the latent space
    - we want the sampled value to be the same as the value the value that is encoded
- bicycle GAN training combines schemes 3 and 4
    - scheme 3 constrains the real space to latent and back to real
    - scheme 4 constrains the latent space to real and back to latent
    - we train these schemes iteratively
    - we use the same modules, we just plug them differently

Social BiGAT

- first use of bicycle GAN in trajectory prediction
- we combine bicycle GAN and graph attention network

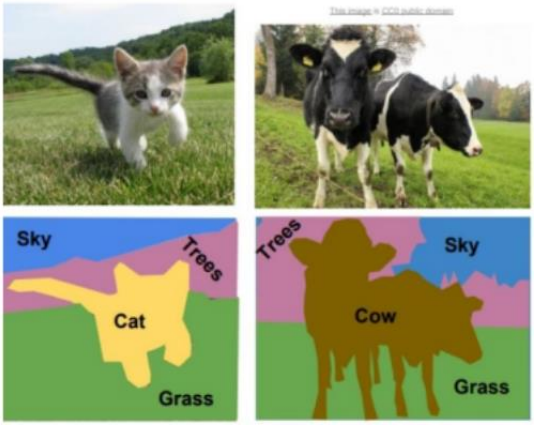


    - social pooling is replaced by graph attention network
    - we then combine this social vector with the motion features, visual features and the noise
    - there are also two discriminators
        - local: evaluates each trajectory independently
        - global: takes a scene does the social interaction
- graph attention network:
    - creates a graph of all pedestrians in one scene
    - we use the dynamic features of each pedestrian
    - we compute attention of each pair of pedestrians
    - from that, we compute a normalized vector for each pedestrian
    - now the features are multiplied with these weights and parameters to create a message



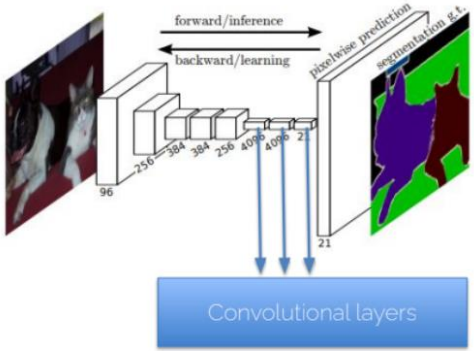
8. Semantic Segmentation

- we move from bounding box representation to pixelwise representation
- every pixel in the image needs to be labelled with a category label
- we do not differentiate between the instances
- it is very common fully convolutional networks for this



Fully convolutional networks

- we use only convolutions
- networks that are able to deal with any input/output size
  - although there is always a limitation in the lower end
- how to do it?
  - replace FC layers with convolutional layers
  - convert the last layer output to the original resolution
  - do softmax-cross entropy between the pixelwise predictions and segmentation ground truth
  - backprop (loss per pixel) and SGD



- how to convert fully connected layers into convolutional?
  - we use 1x1 convolution = keeps the dimensions and scales input
  - and of course we use multiple kernels to get a deeper output
  - example: kernel [2]

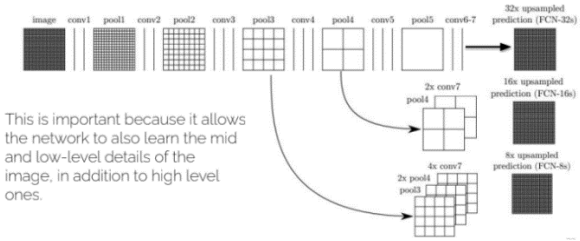
Image 5x5

-5	3	2	-5	3
4	3	2	1	-3
1	0	3	3	5
-2	0	1	4	4
5	6	7	9	-1

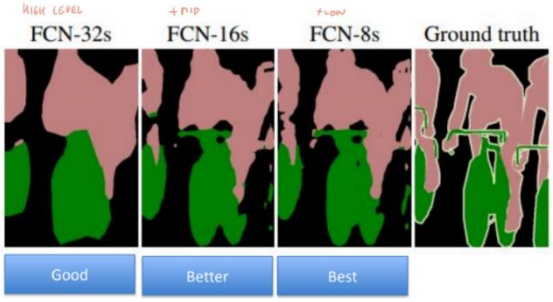
→

-10	6	4	-10	6
8	6	4	2	-6
2	0	6	6	10
-4	0	2	8	8
10	12	14	18	-2

  - in practice we use 1x1 convs to shrink the number of channels
  - often we further add a non-linearity
- how to upsample?
  - authors of the FCN paper proposed to predict mask from different levels of the networks



- results:
  - FCN outperforms significantly pre-deep learning and quasi-deep learning methods and is recognized as the AlexNet of semantic segmentation
    - FCN outperforms even SDS (a R-CNN based method that uses proposals)

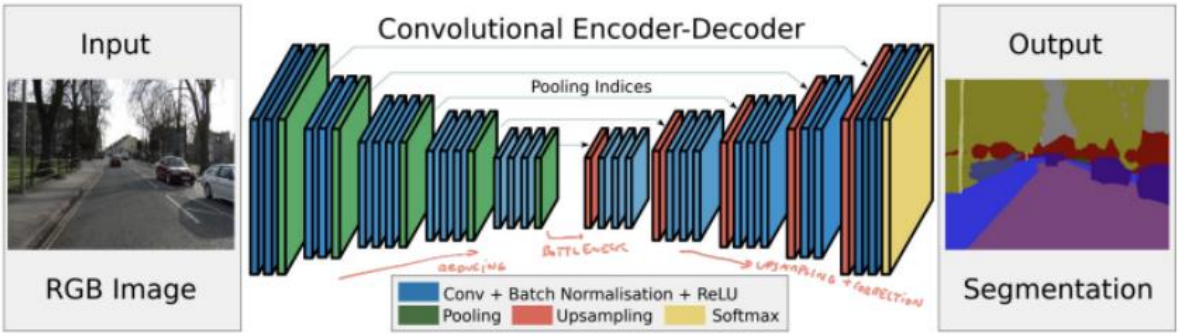


Autoencoder-style architecture

- architectures that learn not just the downsampling but also the upsampling

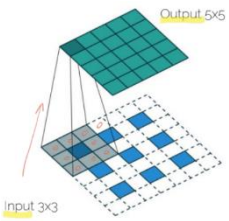
SegNet

- uses stepwise upsampling



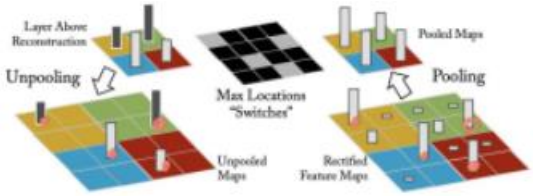
- encoder: normal convolutional filters + pooling
- decoder: upsampling + convolutional filters
    - the convolutional filters in the decoder are learned using backprop and their goal is to refine the upsampling
- types of upsampling:
    1. **interpolation**
        - more soft output, we will have fewer artifacts

    2. **fixed unpooling**



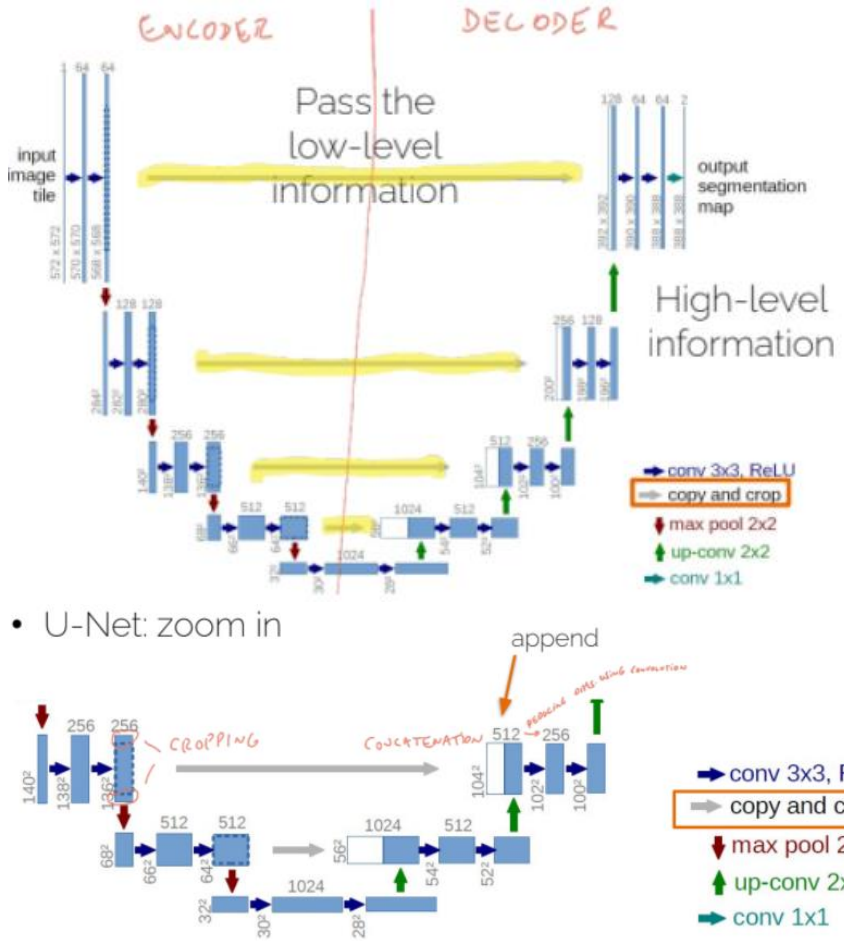
        transposed convolution (also called up-convolution):
        - first do unpooling with zeros
        - then apply learned convolution filter
        - but can we do better?

    3. **unpooling ala DeconvNet**
        - we are keeping the location where the maximum came from
        - this allows to keep the details of the structures

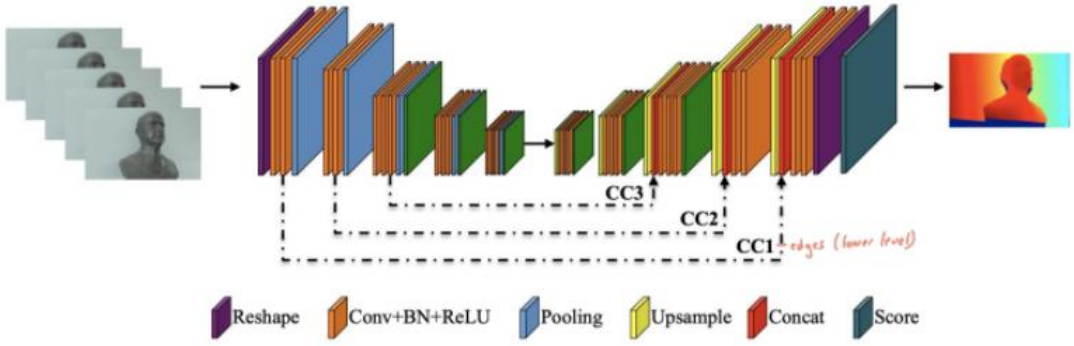


U-Net

- introduced for medical applications
- encoder-decoder architecture that has skip connections
- skip connections:
    - important element to get good pixelwise predictions
    - we want to keep low-level information (e.g. where are the edges) and combine it with the high-level information from the bottleneck representation

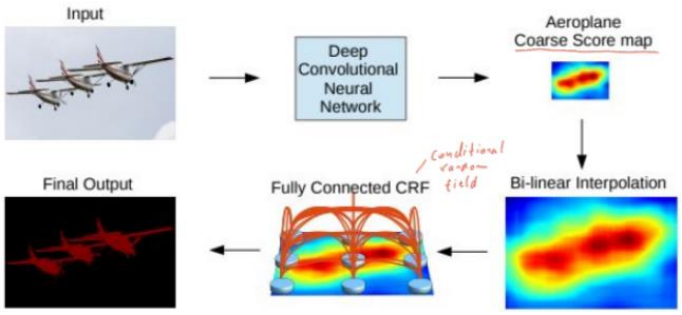


    - by concatenating, we get a bigger depth, which we then reduce using convolution
    - the idea of skip connections is similar to ResNet
    - in practice, each problem requires skip connections at different levels

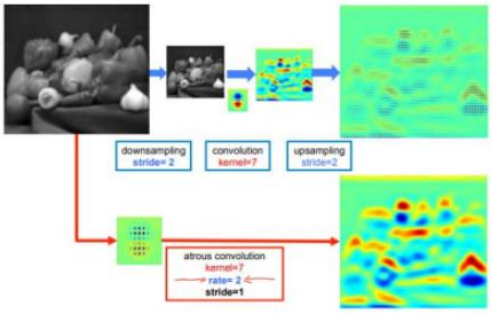


DeepLab

- one of the most famous and well performing architectures
- it takes an input image and processes it through deep convolutional neural network to get a coarse map of the semantic classes
- then it does a bi-linear interpolation and refines this map using fully connected conditional random field



- there are three main challenges:
  - reduced feature resolution <-> proposes **atrous convolution**
  - objects exist at multiple scales (half of the image/a few pixels) <-> proposes **pyramid pooling**
  - poor location of the edges <-> proposes refinement with **conditional random field (CRF)**
- Dilated/atrous convolution
  - ideally, we would like to keep the feature resolution and not reduce it
  - problem is that this would be super expensive
  - Dilated convolution proposes a new approach:

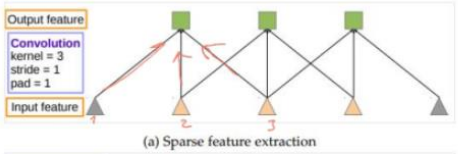


Sparse feature extraction with standard convolution on a low resolution input feature map.

Dense feature extraction with atrous convolution with rate r=2, applied on a high resolution input feature map.

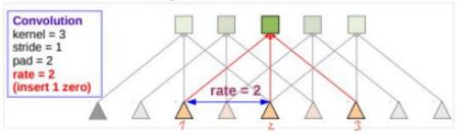
↳ without increasing operational expensiveness

  - 1D example:



(a) Sparse feature extraction

(a) Sparse feature extraction with standard convolution on a low resolution input feature map.

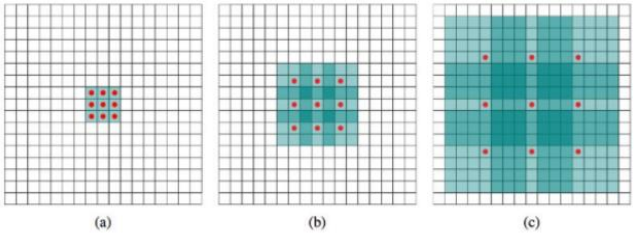


(b) Dense feature extraction

(b) Dense feature extraction with atrous convolution with rate r = 2, applied on a high resolution input feature map.

- reduces output feature map without reducing the input resolution

  - 2D example:



(a) the dilation parameter is 1, and each element produced by this filter has receptive field of 3x3.

(b) the dilation parameter is 2, and each element produced by it has receptive field of 7x7.

(c) the dilation parameter is 4, and each element produced by it has receptive field of 15x15.

- we don't use all the pixels in the receptive field

class torch.nn.Conv2d (*in_channels, out_channels, kernel_size, stride=1, padding=0, dilation=2*)

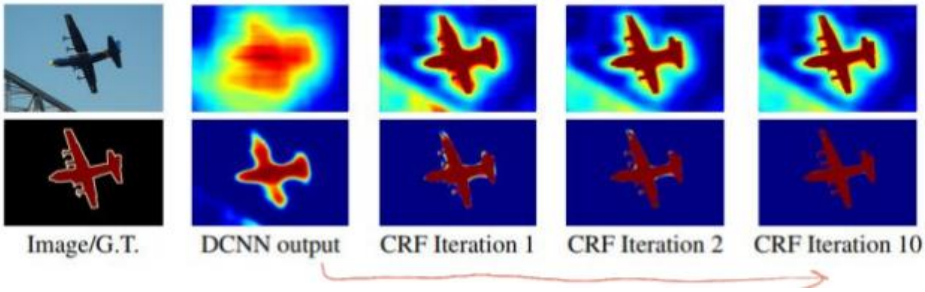
class torch.nn.ConvTranspose2d (*in_channels, out_channels, kernel_size, stride=1, padding=0, dilation=2*)

- Conditional Random Fields (CRF):
  - it is also used in interactive image segmentation (helps with the annotation)
    - Boykov and Jolly (2001)

$$E(x,y) = \sum_i \varphi(x_i, y_i) + \sum_{ij} \psi(x_i, x_j)$$

    - Variables
      - x_i: Binary variable
        - foreground/background
      - y_i: Annotation
        - foreground/background/empty
    - Unary term
      - $\varphi(x_i, y_i) = K[x_i \neq y_i]$
      - Pay a penalty for disregarding the annotation
    - Pairwise term
      - $\psi(x_i, x_j) = [x_i \neq x_j] w_{ij}$
      - Encourage smooth annotations
      - w_{ij} affinity between pixels i and j

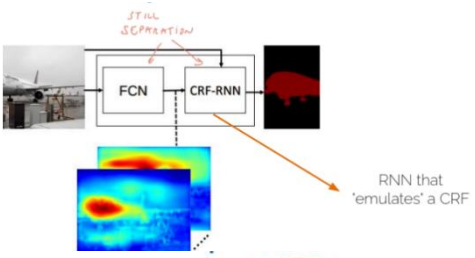
  - we want to obtain binary labels foreground/background
  - unary term = price we have to pay if we don't respect the annotation
  - pairwise term = looking at neighbouring pixels and want smooth annotation
  - there is no deep learning involved

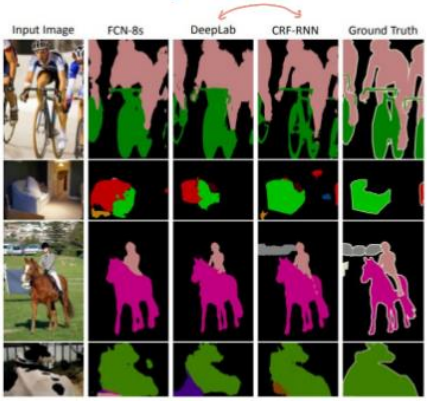


  - problem:
    - network is not trained end2end -> we have to train FCN and CRF independently -> training is slow and arguably suboptimal
  - solution:
    - formulate CRF as a recurrent neural network

CRF-RNN (2015)

- FCN gives a raw coarse map
- CRF-RNN refines the map
- results improve quiet a lot even compared to the DeepLab performance and in some cases are more precise than the ground truth



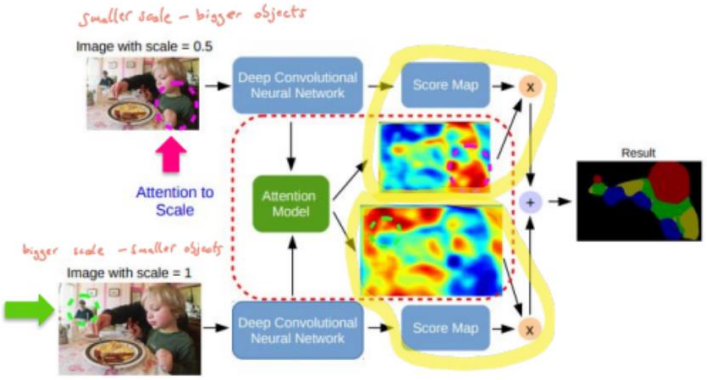


- do we actually need CRF?
  - CRF is conditioned on the RGB image
  - **what if we use attention?**

Attention for semantic segmentation

- "Not all the pixels are equally important"
  - *for more detail about attention, see lectures "Pedestrian trajectory prediction" and "Transformers and DETR"*

- the attention model learns to put different weights on objects of different scales
  - for example, the model learns to



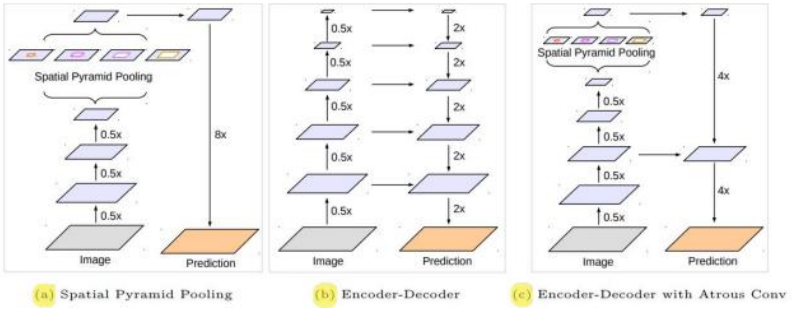
    - put large weights on the small-scale person (green dashed circle) for features from scale = 1
    - put large weights on the large-scale child (magenta dashed circle) for features from scale = 0
- we jointly train the network component and the attention model

- <u>do we even need these blocks which include the global information (CRF, RNN, attention)?</u>
  - not necessarily

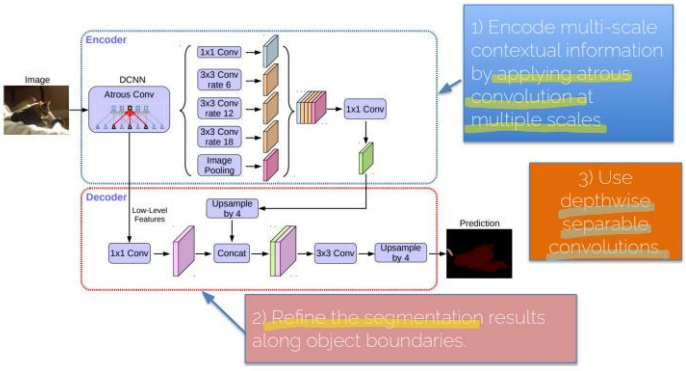
More advanced approaches

<u>DeepLabV3+</u>

- much much better version of the previous DeepLab
- combines not only the dilated convolution with spatial pyramids pooling, but it also combines together with encoder decoder module (type of U-Net architecture)



- <u>principles in detail:</u>

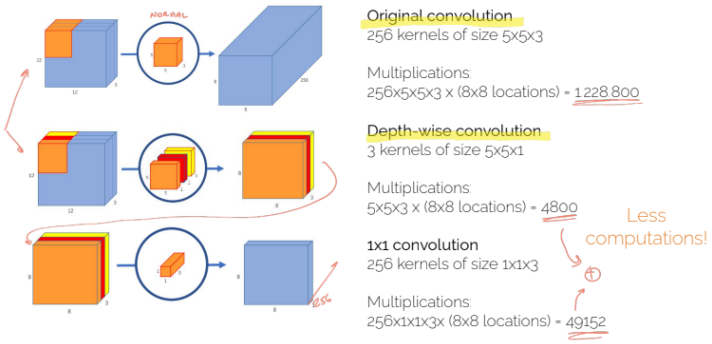


  - <u>step 1</u>: encoding information from different scales with different receptive fields
  - <u>step 2</u>: gather all of the information and refine the segmentation results
  - what makes it all work are depthwise separable convolutions

- depthwise separable convolutions:
  - different filter for each colour
  - filters are applied only at certain depths of the features

    `class torch.nn.Conv2d(in_channels, out_channels, kernel_size, stride=1, padding=0, groups=3)`

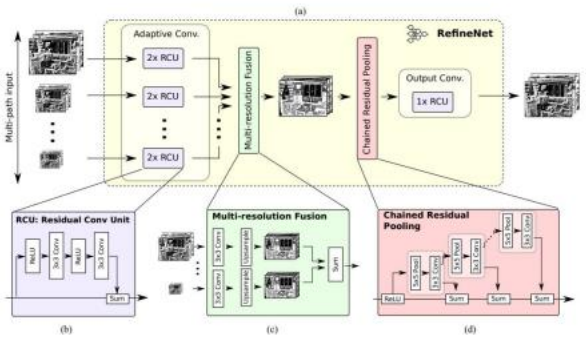
    `class torch.nn.ConvTranspose2d(in_channels, out_channels, kernel_size, stride=1, padding=0, groups=3)`
  - why do we do this?



  - DeepLabV3+ has SOTA performance.

RefineNet (2017)

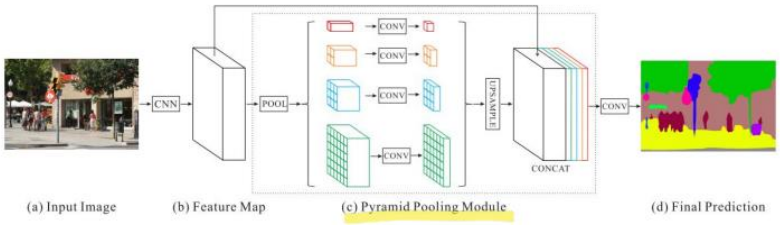
- fairly complex architecture



Many building blocks but the goal is the same: use convolutional layers to refine the information coming from different scales.

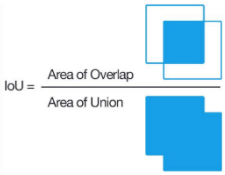
PSPNet

- similar to RefineNet
- but here the features are shared and multiscaling comes from a pyramid pooling module
- performs slightly better than RefineNet



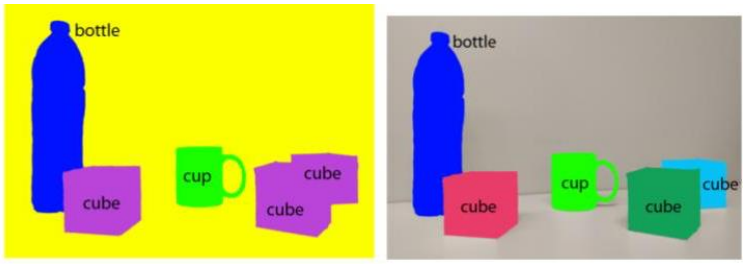
Datasets and metrics

- Datasets
  - models are often pre-trained in the large MS-COCO dataset (ImageNet of semantic segmentation) before finetuning to the specific dataset
  - Pascal VOC, Cityscapes, ADE20K, Mapillary Vistas
- Metrics
  - **mIoU** = Mean intersection over union
    - simple compute IoU for each class and then compute the mean of those values
  - **pixel accuracy**



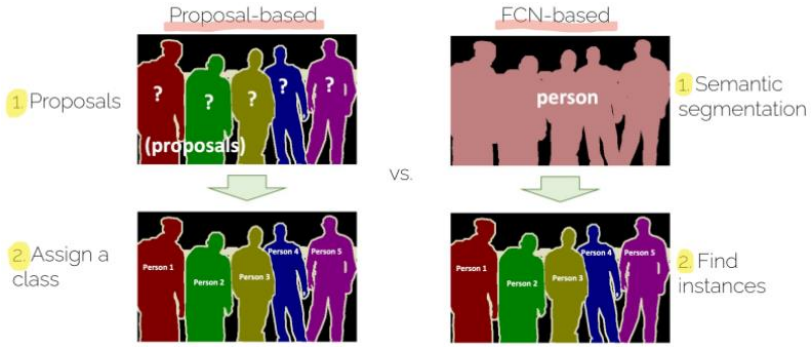
9. Instance and Panoptic Segmentation

- in **semantic segmentation**, we want to label every pixel including the background into a semantic class
  - we don't differentiate between the pixels
- in **instance segmentation**, we focus only segmenting the pixels that are coming from instances of classes of interest (cars, people, …)
  - we do want to differentiate the individual instances
  - we don't focus on uncountable objects (sky, road, …)



Instance segmentation methods

1. **proposal-based**: use knowledge we have from detection and start with a series of proposals, then do segmentation and assign a semantic class to each of these proposals
2. **FNC-based**: starts from semantic segmentation map, then separate individual instances

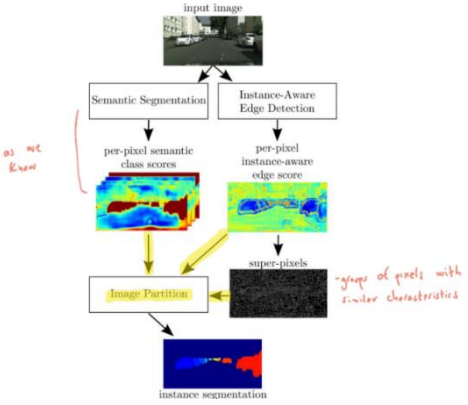


FCN-based methods

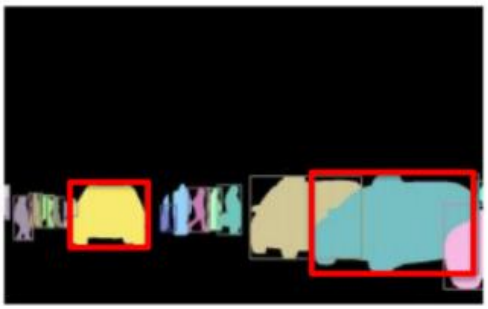
- great advantage is that they start from a semantic map, which we already know how to obtain
  - done mostly using fully convolutional networks
- there are three important methods:
  1. Proposal-free network for instance-level object segmentation
  2. InstanceCut: from Edges to Instances with a MultiCut
  3. Deep Watershed Transform for Instance Segmentation
- we will focus on the second one

<u>InstanceCut: from Edges to Instances with a MultiCut</u>

- uses the concept of clustering
- we have a set of pixels that represent the class person and within this cluster, we want to recognize individual instances
- we start from the input image, perform semantic segmentation and then try to perform image partition, that would separate the image into two small sets (like super-pixels = groups of pixels that show certain characteristic – e.g. smooth transition in the colour space) and then put it together into instances
  - left branch performs semantic segmentation
  - right branch performs separation of the image (using very low-level features like edges) and forms super-pixels representation



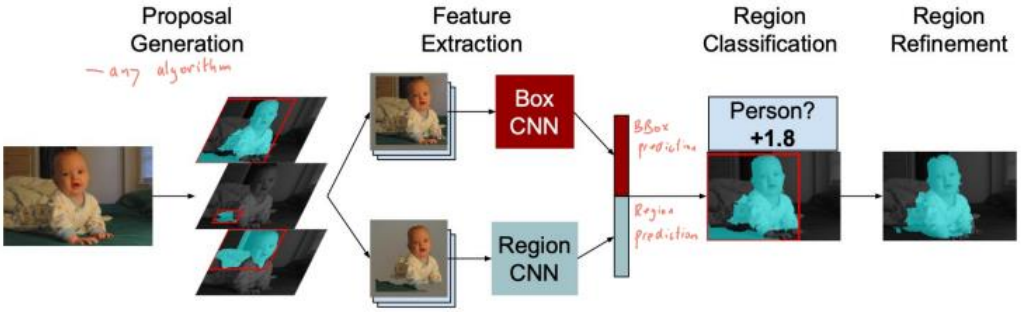
Proposal-based methods



- main focus of this lecture
- they have much better performance
- if we already know how to separate different instances with object detection, why not use it as an input and then try to find the segmentation mask within this bounding box
- there are two proposal based methods
  1. Simultaneous Detection and Segmentation
  2. Instance-aware Semantic Segmentation via Multi-task Network Cascades

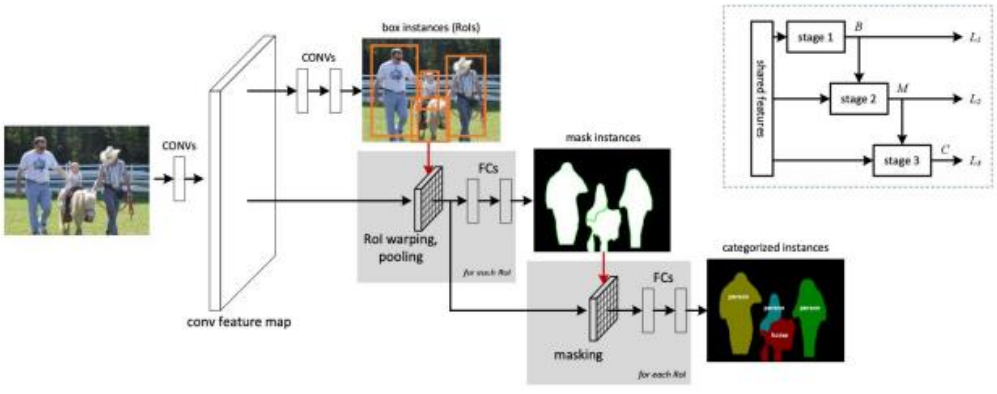
Simultaneous Detection and Segmentation (SDS)

- presents a very simple concept
- proposals are used as a starting point for bounding box prediction and also directly for region (mask) prediction
- idea is to start with a set of proposals (from any algorithm you like) and then perform feature extraction for BB prediction and region prediction
  - we have 2 separate CNNs for that
- combining these outputs leads to region classification and region refinement



Multi-task Network Cascades

- idea is to start always with region of proposals
- we convert them to mask instances
- later refine them to categorized instances

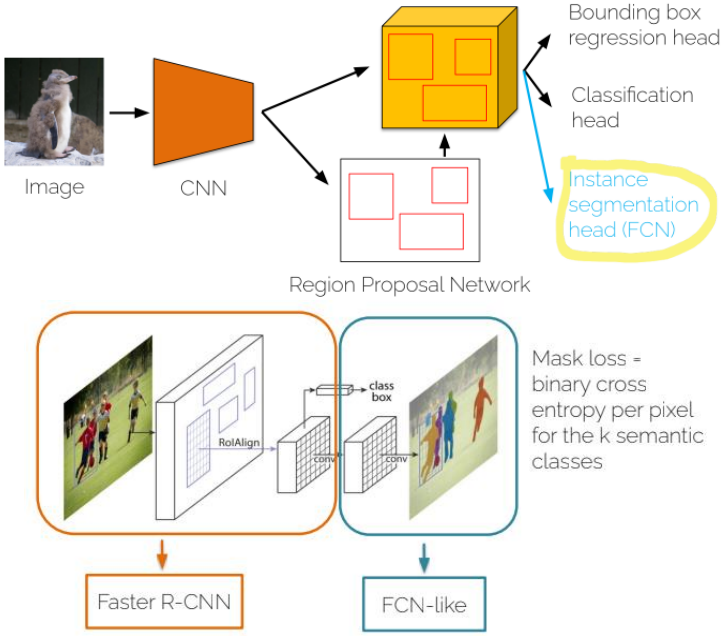


IS: The best of both worlds

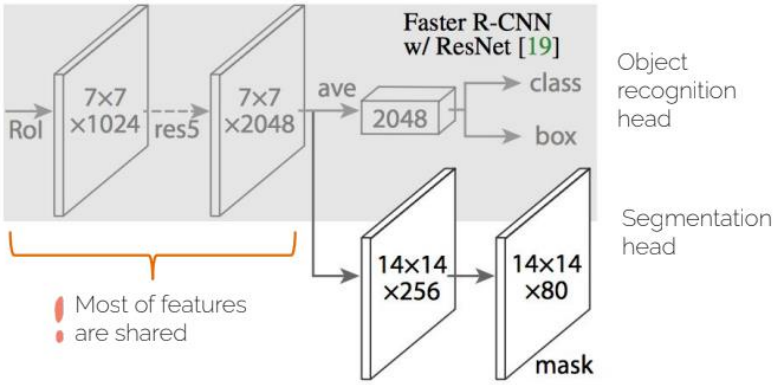
- why should we constrain our method to a fixed set of proposals that might be incorrect?
- why should we constrain to semantic segmentation?
- ideally, we should leverage from both world

Mask R-CNN

- derives from Faster R-CNN + FCN
- we start with Faster R-CNN to perform feature extraction and propose region proposal network
- then we have the Bounding box regression head, Classification head and Instance segmentation head

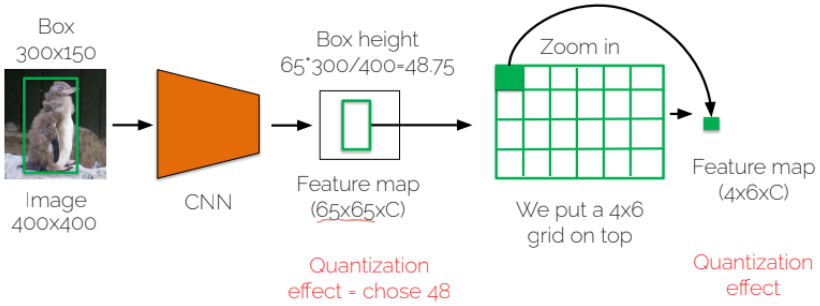


- we will directly try to predict semantic class for the particular class
- the instance problem is solved by Faster R-CNN
- the mask head can focus entirely on finding semantic class of this instance
    - we take the feature representation and we essentially do series of convolution operations

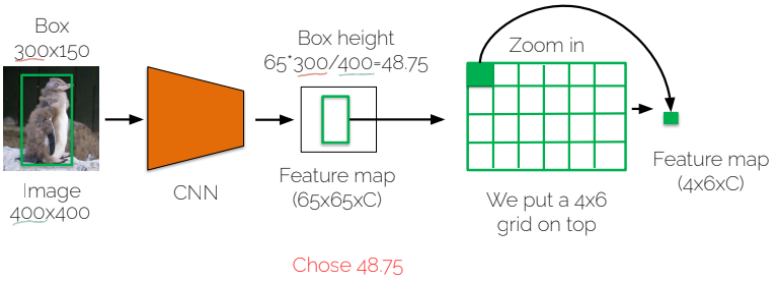


- we are really adding only a couple of operation on top of Faster R-CNN
- Representations:
    - in detection, we require invariant representations (translational, rotational, …)
    - in segmentation, we require **equivariant representations**
        - translated object -> translated mask
        - scaled object -> scaled mask
        - for semantic segmentation, small objects are less important (less pixels = lower loss), but for instance segmentation, all objects (no matter the size) are equally important -> we need some changes to the Faster R-CNN

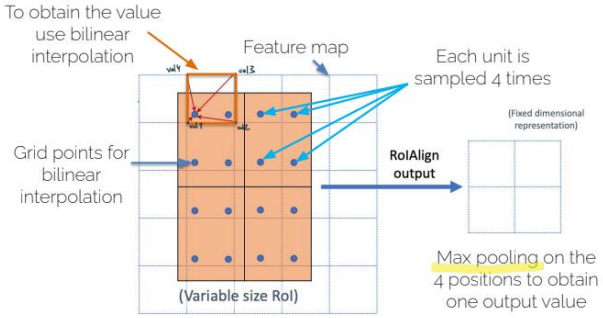
- let's look at different Faster-RCNN operations:
  - what operations are equivariant?
    - features extraction = convolutional layers = EQUIVARIANT
    - segmentation head = fully convolutional network = EQUIVARIANT
  - what operations are not equivariant?
    - fully connected layers & global pooling
- Recall RoI pooling:
  - we had this proposal that is on top of the penguin
  - we perform feature extraction with the feature map
  - from the feature map we are only interested in region of proposals
  - we put *HxW* grid on top of the feature map and perform pooling



  - this means that the bounding box is scaled down and we have to choose, how many pixels we take into our grid (we can't take 48.75 pixels)
  - the resulting bounding box has not the correct height
  - how to improve this?
- ROIAlign
  - aims to erase quantization effects
  - now we are going to be able to choose exactly the right scale and still perform region of interest pooling operation



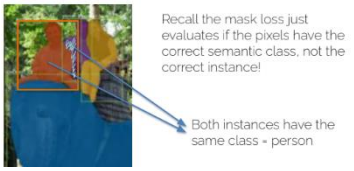
  - example:
    - salmon color represents the bounding box
    - from this bounding box RoIAlign gets us fixed size representation of 2x2
    - we need to fill these 4 positions
    - for that, we use create grid points and do bilinear interpolation



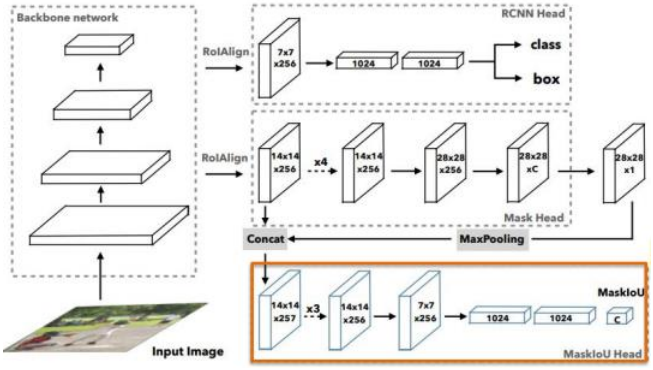
- results: pretty good

- Mask R-CNN are also very flexible, we can even extend it for **joints detection**
  - model a keypoint's location as a one-hot mask and adopt Mask R-CNN to predict K masks (which form 1 pixel) – one for each of K keypoint types (e.g. left shoulder, right elbow)

- can we do even better?
  - problem:
    - mask quality score is computed as a confidence score for the bounding box = if the bounding box has low confidence, then the mask quality is also going to suffer
    - remember that the mask loss only evaluates if the pixels have the correct semantic class, not the correct instance
      - if all of the pixels inside the orange bounding box were classified as one person, it does not really matter



  - solution:
    - other authors propose Mask Scoring R-CNN
    - we have Mask IoU head



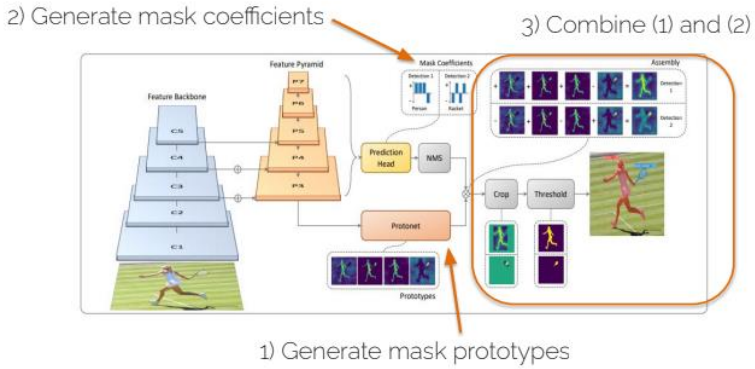
      - we want to measure IoU between the predicted mask and the ground truth mask
        -> loss is acting on the instance level
      - this modification causes to give lower confidence scores than Mask R-CNN, but the performance is SOTA

One stage detectors for masks

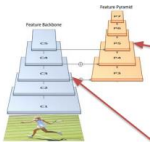
- recall that one-stage detectors (like YOLO) were faster than two-stage detectors like (Faster R-CNN)

YOLACT (You Only Look At CoefficienTs)

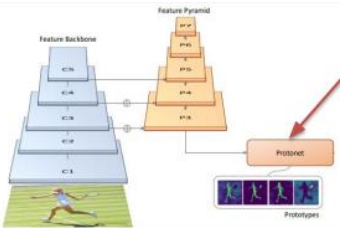
- we don't need the boxes
- idea is not so straightforward
- we need to design the network carefully
- we have this pattern



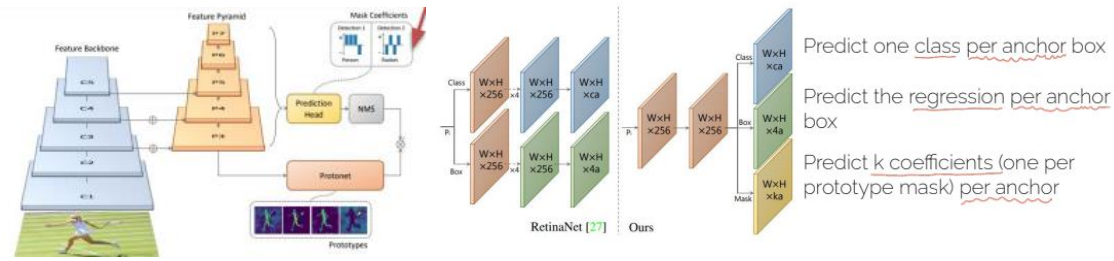
  - firstly, we need a network that generates mask prototypes – possible segmentations
  - then we use prediction head (another network) to evaluate each mask prototype and generate the mask coefficients
  - finally, we combine the mask prototypes and mask coefficients to generate the final segmentation
- Backbone – ResNet 101
  - we do feature pyramid



- Protonet
  - responsible for generating *k* prototype mask (*k* has no relationship with the number of classes)
  - the architecture is fully convolutional network consisting of two 3x3 convs and a final 1x1 conv that reduces the depth to *k*
  - this is similar to Mark R-CNN but there is no loss



- Mask coefficients
  - another network predicts coefficients for the predicted masks
  - judges how reliable the mask is
  - the mask coefficients are intertwined with the box predictions



  - this is similar to RetinaNet

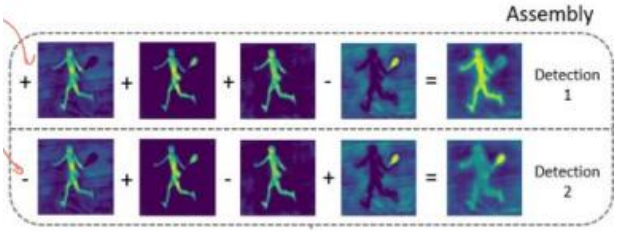
- Mask assembly
  - we do a linear combination between the mask coefficients and mask prototypes
  - we predict the mask as

    $M = \sigma(PC^T)$

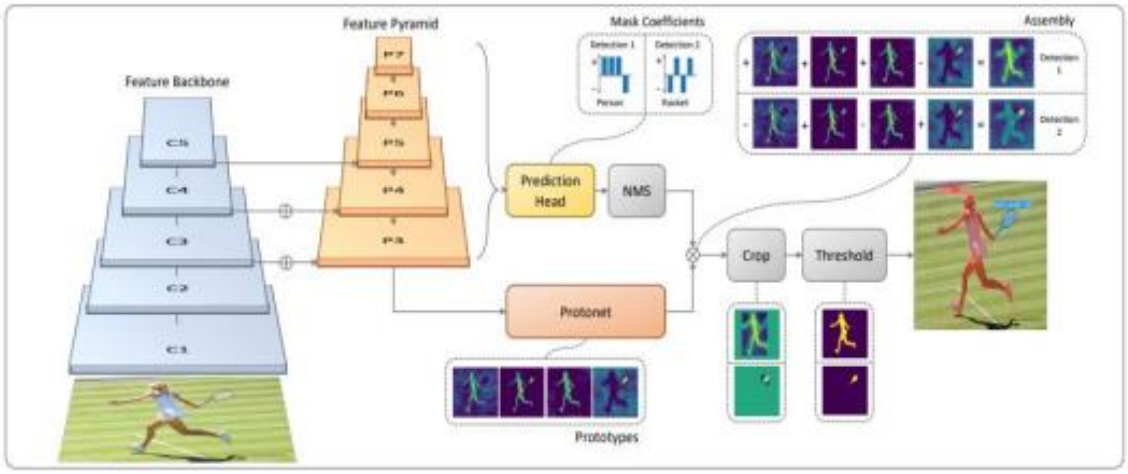
    P is a matrix of prototype masks ($HxWxK$)

    C is matrix of mask coefficients surviving NMS (both positive and negative)

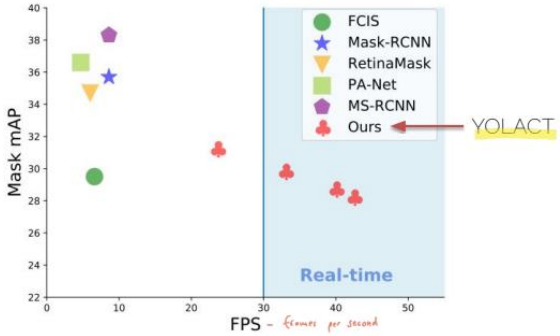
    σ is a nonlinearity



- Full picture



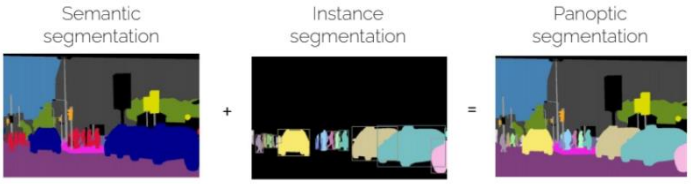
- Loss function:
  - standard losses (regression for bounding box & classification for the class of the object/mask)
  - AND cross entropy between the assembled masks and the ground truth
- Results:
  - for large objects, the quality of the masks is even better than those of two-stage detectors



- Improvements:
  - use specially designed version of NMS (faster)
  - auxiliary semantic segmentation loss function performed on the final features of the FPN (not used in inference)
  - Yolact++: better real-time instance segmentation

Panoptic segmentation

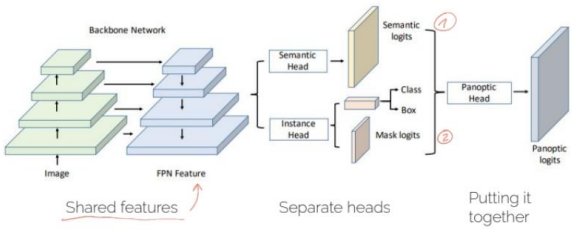
- combines semantic segmentation and instance segmentation
- becoming more important, for ex. in in mobile robot vision



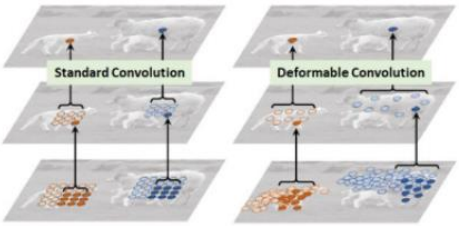
- we give label to uncountable objects called "stuff" (sky, road, etc.) similar to FCN-like networks
- we differentiate between pixels coming from different instances of the same class of countable objects called "things" (cars, pedestrians, etc.
- <u>problem</u>:
    - if we tackle this task individually, some pixels might get classified as "stuff" from the FCN network while at the same time as instances of some class "things" from Mask R-CNN = conflicting results
- <u>solution:</u>
    - parametric-free panoptic head which combines the information from the FCN and Mask R-CNN, giving final predictions

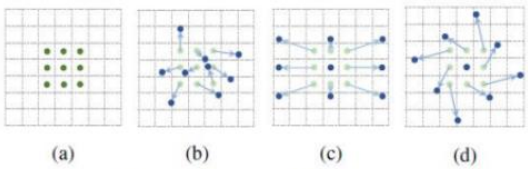
<u>UPSNet</u>:

- we have a set of shared features with feature pyramids
- then the semantic head + instance head on top
- in the end, we combine all the information together to create the panoptic logits

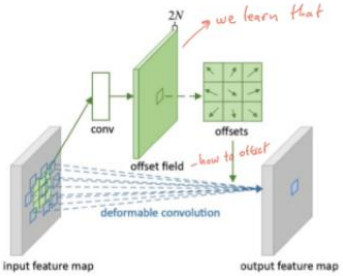


- <u>Semantic head</u>
    - fully convolutional network
    - uses a new element = **deformable convolutions**
        - generalization of dilated convolutions where we can learn the offset
        - we will pick the values at the different locations for convolutions conditioned on the input image of the feature maps





(a) Conventional Convolution, (b) Deformable Convolution, (c) Special Case of Deformable Convolution with Scaling, (d) Special Case of Deformable Convolution with Rotation



Regular convolution

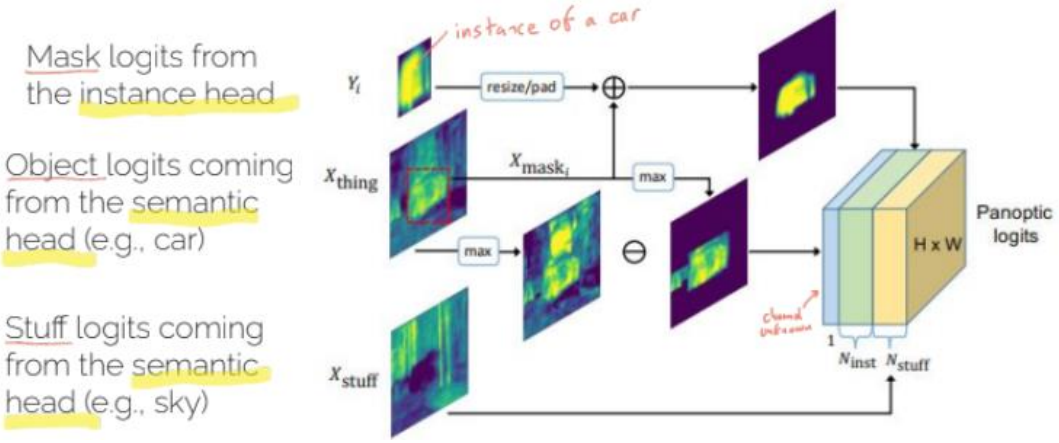
$$y(p_0) = \sum_{p_n \in \mathcal{R}} w(p_n) \cdot x(p_0 + p_n)$$

Deformable convolution

$$y(p_0) = \sum_{p_n \in \mathcal{R}} w(p_n) \cdot x(p_0 + p_n + \Delta p_n)$$

where Δp_n is generated by a sibling branch of regular convolution

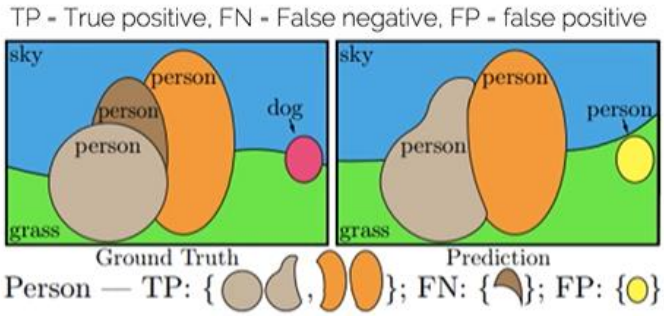
- Panoptic head
  - puts together mask output and the semantic information output



    - on top there are mask logits: come from Mask R-CNN, where are the instances located
    - object logits: probability that pixel belongs to class x
      - need to be masked by the instances
    - stuff logits: same as object logits but used for uncountable objects
  - we perform softmax over panoptic logits
    - if the maximum value falls into the first stuff channels, then it belongs to one of the "stuff"
    - otherwise, the index of the maximum value tells us the instance ID the pixel belongs to

Metrics

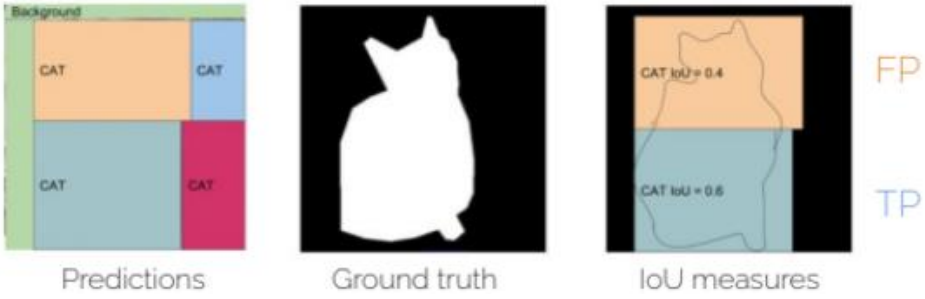
$$PQ = \underbrace{\frac{\sum_{(p,g)\in TP} \text{IoU}(p,g)}{|TP|}}_{SQ} \underbrace{\frac{|TP|}{|TP| + \frac{1}{2}|FP| + \frac{1}{2}|FN|}}_{RQ}$$



- **SQ** – segmentation quality – how close the predicted segments are to the ground truth segments
  - using IoU
- **RQ** – recognition quality – just like in detection, are we missing any instances? are we predicting more instances?

- as in detection, we have to match ground truth and predictions
  - we use segment matching
  - segment is patched if IoU > 0.5

- no pixel can belong to two predicted segments

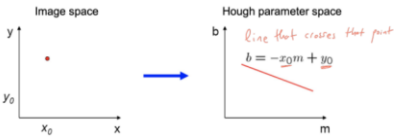


Object Instance Segmentation as Voting

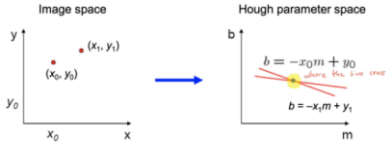
- prof particularly likes this method
- inspired by what we used before CNNs came = **detection-as-voting**

Hough Voting

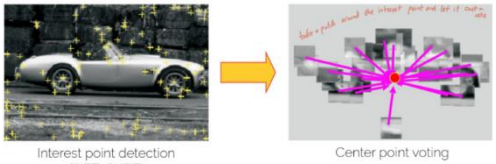
- we detect analytical shapes (e.g. lines) as peaks in the dual parametric space
- each pixel casts a vote in this dual space
- we detect peaks and 'back-project' them to the image space
- example: **line detection**
  - all we have are different points that are placed on top of the line
  - we want to fit a line though these points
  - each point casts a vote into the Hough parameter space
  - that vote takes a form of a line that crosses that point



  - we accumulate votes from different points in (discretized) parameter space
  - then we read out the maximum, which is the yellow point in this case
    - this point is a point represented by a value of m and b

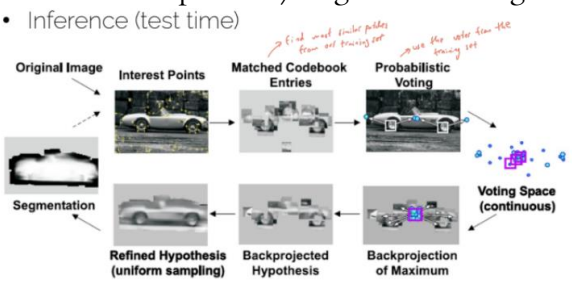


- example: **object detection in a car**
  - objects are detected as consistent configurations of the observed parts (visual words)
  - we know that car has two wheels and these two wheels are always in roughly the same position with respect to the center of the car
  - idea: whenever wheel/window/… patch is detected, we cast a vote for the center of the car
    - we know the center of the car is always going to be in the almost same position with respect of the car parts
    - by detecting the peak, we can decide if there is a car
  - training:
    - extract features using SIFT, SURF
    - cast a vote



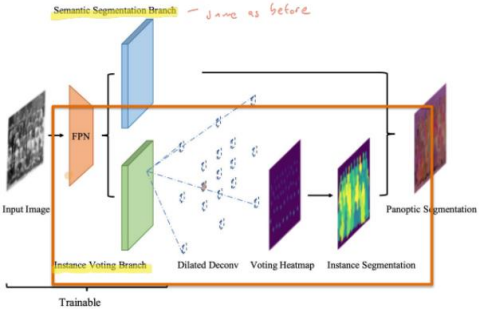
  - inference:
    - get original image and compute interest point
    - find similarity between the interest points and the codebook entries (find most similar patches from our training set)
    - use the voter from the training set to estimate the center (we need to find peak of the vote locations in the parametric space)

- then we can go back to the image space (look at all the patches that voted for the chosen center position) to get the raw segmentation of the object

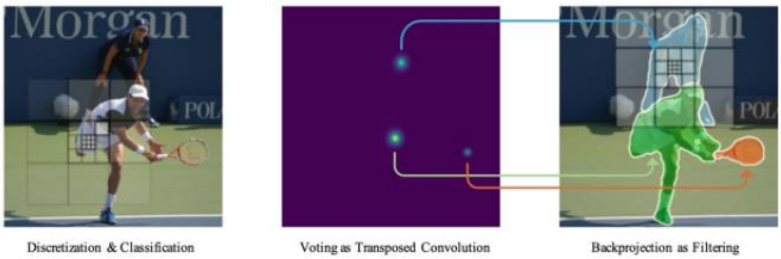


Pixel consensus voting for panoptic segmentation

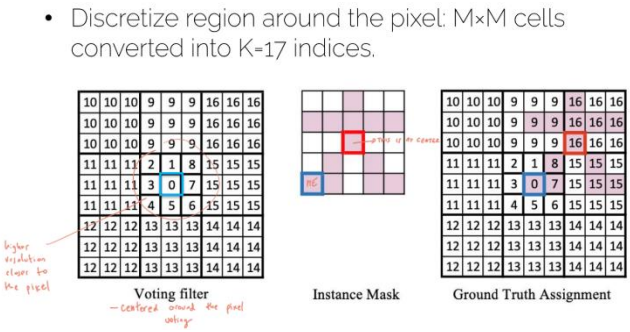
- merges the concept of voting with the modern CNNs
- we will use FPN backbone to extract features and semantic segmentation head
- apart from that, there is a second branch = **Instance Voting Branch**
    - predicts for every pixel if it is part of an instance mask
    - and if so, we get the relative location of the instance mask center



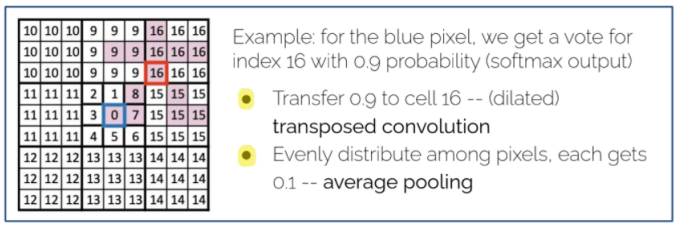
- the authors proposed to design the Instance Voting Branch in a way, that is "fully backpropagatiable"
- nutshell:
    - discretize regions around each pixel
    - let every pixel vote for a centroid (of the object it belongs to) over a set of grid cells
        - there is no centroid for the "stuff" class (e.g. sky)
    - vote aggregation probabilities at each pixel are cast to accumulator space via (dilated) transposed convolution
    - detect object as "peaks" in the accumulator space
    - do a backprojection of the "peaks" back to the image to get an instance mask (all the pixels that voted for that center)
    - get category information that is provided by the parallel semantic segmentation head



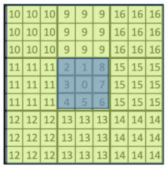
- how to implement this into a neural network? use voting lookup table:
    - we use the following filter
    - we place it center around the pixel that has to cast a vote
        - this filter converts the *NxM* cells into 17 indices
        - we can cast a vote for 17 positions
        - there is more resolution closer to the pixel
        - less resolution as we go further away
        - in this example, the blue pixel votes for 16 (indice of the center)



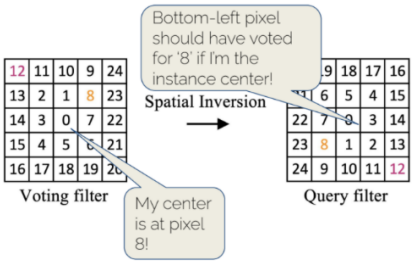
- at inference
  - the instance voting branch provides a tensor of size (HxWxK+1 (all the positions I can vote for + one for "stuff" class)
  - softly accumulates votes in the voting accumulator



- transposed convolutions
  - used to perform the voting operation
  - we take a single value in the input
  - then multiply with a kernel and distribute in the output map
    - kernel defines the amount of the input value that is being distributed to each of the output cells
  - for the purpose of vote aggregation, however, we fix the kernel parameters to 1-hot across each channel that marks the target location.
- example:
  - Output tensor: [H,W,K+1]
  - Example: 9 inner, 8 outer bins, K=17
  - Split the output tensor to two tensors: [H,W,9],[H,W,8]
    - Apply two transposed convolutions, with kernel of size [3,3,9], stride-1 and [3,3,8], stride-3
    - Pre-fixed kernel parameters; 1-hot across each channel that marks the target location
    - Dilation -> spread votes to the outer ring
  - Smooth votes evenly via average pooling



- how to do object detection?
  - we cast votes and detect peaks in the heatmaps
  - the peaks determine the consensus between different pixels in the instance
  - by simple thresholding and doing connected component analysis, we can detect center for all of these objects
  - then we do backprojection
    - for every peak, we need to determine, which pixels voted in favor of this regions above all others
    - for that we use a query filter (= spatial inversion of the voting filter), followed by vote argmax and equality test to find the consensus



  - this produces quite a good segmentation

10. Video Object Segmentation

- we use object masks (more precise representations) for object tracking
- the goal is to generate accurate and temporally consistent pixel masks for objects in a video sequence
- challenges:
    - object's appearance
        - strong viewpoint/appearance changes
        - scale changes
        - illumination
        - shape
    - object's motion
        - occlusions

- two possible approaches:
    - **unsupervised (zero-shot) video object segmentation**
        - we have to find all the objects as well as their masks
        - "motion segmentation", "salient object detection"
    - **semi-supervised (one-shot) video object segmentation**
        - we have the first frame and its ground truth
        - this lecture

Supervised Video Object Segmentation

- given first-frame ground truth
- goal: complete pixel-accurate video segmentation
- currently a major testing ground for segmentation-based tracking

Datasets

- we need large-scale datasets
- DAVIS 2016
    - 30/20 train/test sequences of single object tracking
- DAVIS 2017
    - 60/90, multiple objects
- Youtube-VOS 2018
    - 3471/982, multiple objects, first frame where object appears

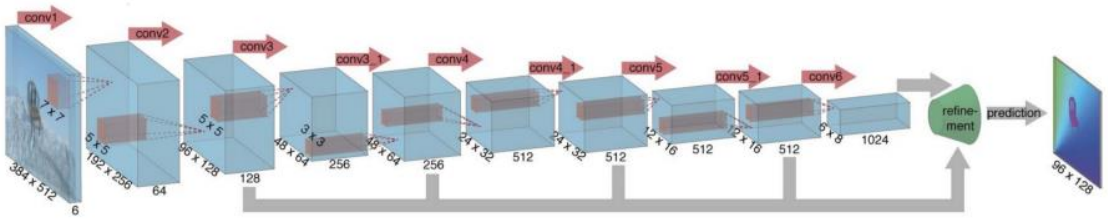
Optical flow

- concept in computer vision
- if we get 2 consecutive images
- we want to have the perceived 2D motion of the objects
- by looking at this, we might get a rough segmentation results
    - works for the cyclist, doesn't work for the two pedestrians (they are marked as 1 instance)

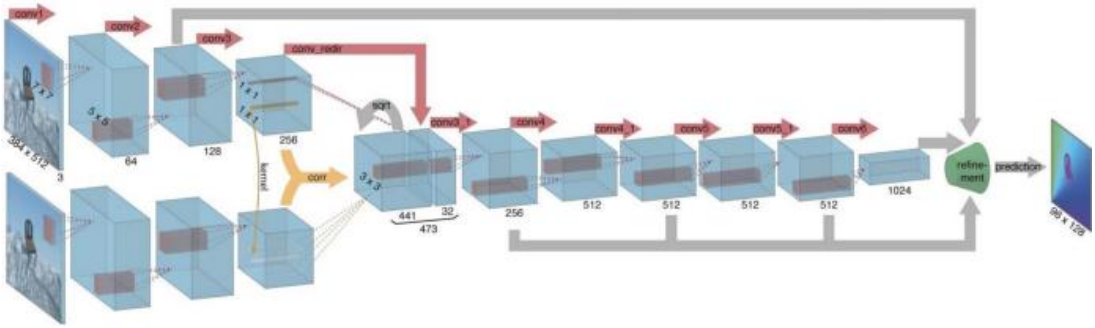
FlowNet (2015)

- end2end supervised learning of optical flow
- results were actually quite good

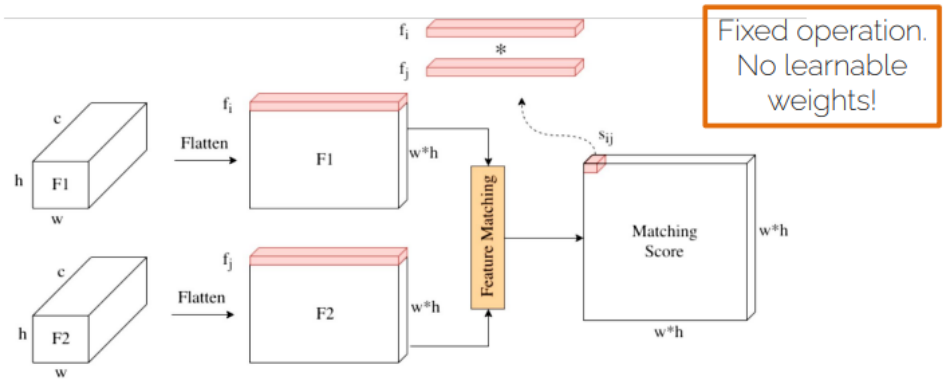
- architecture1:
  - uses stacked images (2x RGB)



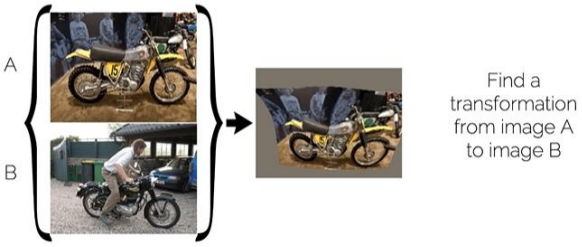
- architecture2:
  - uses siamese architecture



  - **how to combine the information from both images?**
    - using correlation layer = multiplying two feature vectors, using no learnable weights



    - we flatten the feature maps *CxWxH* to *CxW*H* -> one feature is going to represent one spatial location
    - now we want to compare all the features from first image with the second image
    - matching score corresponds to how the region's feature vectors are correlated
    - correlation layer can also be used for finding semantic correspondences (2017)

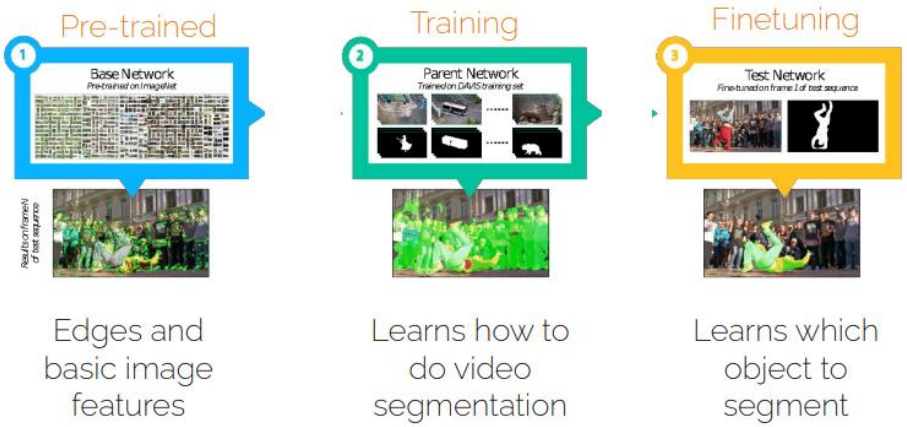


  - **how to upsample?**
    - similar to U-Net: skip-connections, up-convolutions
- <u>can we use optical flow for video object segmentation (VOS)?</u>
  - yes, we can improve and optical flow iteratively (no DL yet) – 2016
- <u>how about other methods?</u>

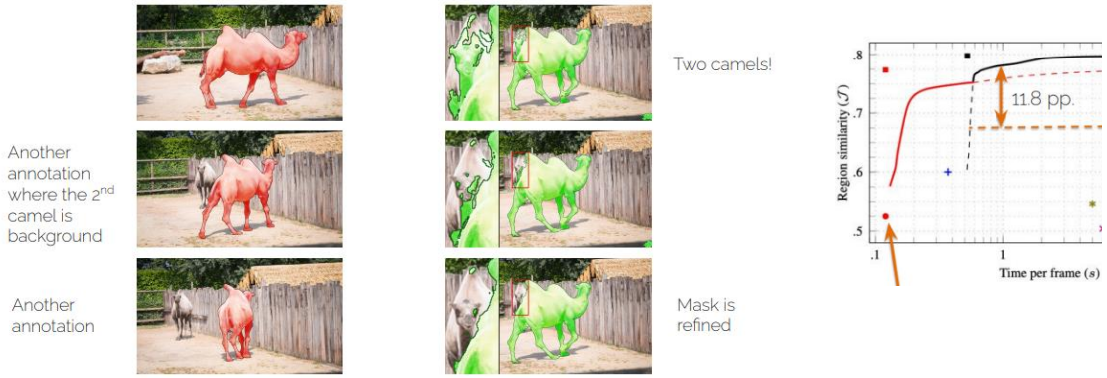
Different approach

OSVOS

- uses first-frame fine tuning
  - goal is to learn the appearance of the object to track
- there are separate training steps:
  - pre-training of "objectness"
  - first frame adaptation to specific object-of-interest using fine-tuning – finetuning the network to the specific object to track
- steps:
  - we start with a pre-trained network -> we get activation on edges and basic image features
  - then we train parent network training: how to do VS -> what it means to do VOS
  - then we do finetuning of the parent network on the first frame ground truth
    - we technically overfit to the test sequence



  - pros: we can recover from occlusions (unlike optical flow-based methods), this method is very robust
  - con: we need the finetuning at test time, each frame is processed independently = no temporal information = temporally inconsistent
    - problem: there is a problem if there comes second camel
    - solution: provide more frames
      (left annotations, right segmentations)

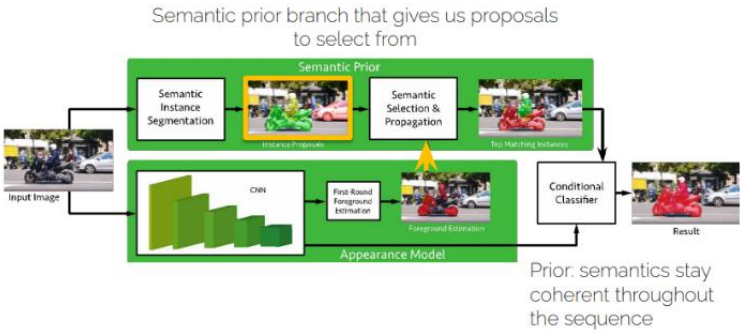


    - problem: OSVOS does not have object shape – it's pure appearance-based method

    - solution: we need object prior -> use instance segmentation methods!

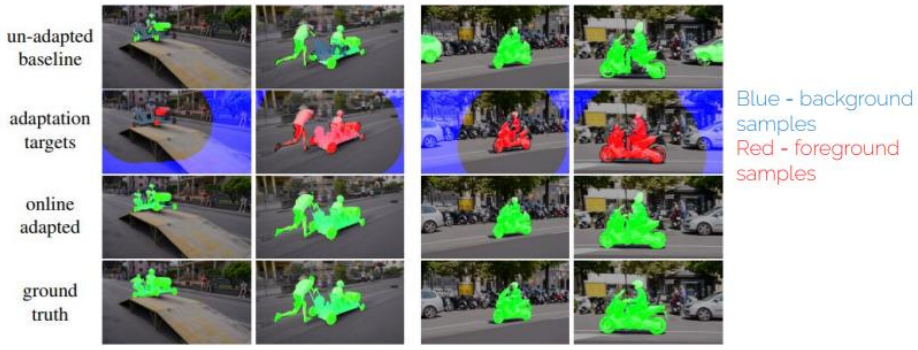
OSVOS-S

- first extension
- the idea is to perform semantic propagation
    - we take input image
    - down branch is classical OSVOS branch -> we get foreground estimation
    - upper branch gives instance proposals, then select the proposals with the most overlap -> we get the top-matching instances
    - then another classifier outputs the final classification results
- as we move forward in the sequence, the OSVOS images are breaking down (motorbike shifts and we see another shape)
- but this is still good enough prior to select the correct proposals -> as long as we have some mask overlap, it still works



OnAVOS: Online Adaptation

- when model cannot adapt due to for ex. pose changes, camera changes
- online adaptation: adapt model to appearance changes every frame – not just the first frame
    - we iteratively fine-tune the model on previous prediction every frame
- problem: extremely slow-
- steps:



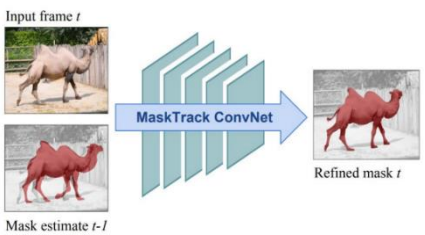
    - the most confident pixels are confident as foreground pixels
    - pixels away from the foreground are classified as background

Mask refinement

- another way to deal with drifting
- if object does not move a lot from frame to frame
- we can start with an approximate mask, then use refinement network to refine it for the current frame

MaskTrack ConvNet

- uses crop-and-zoom to do segmentation at higher resolution
- input is frame t and mask from t-1
- originally used for static images
    - we use similar technique as in the regressor for Faster R-CNN (we take ground truth bounding box and simulate small displacement) or Tracktor



(a) Annotated image (b) Example training masks

- we simulate the slight changes using deformations of the image and then train the network on deformed masks

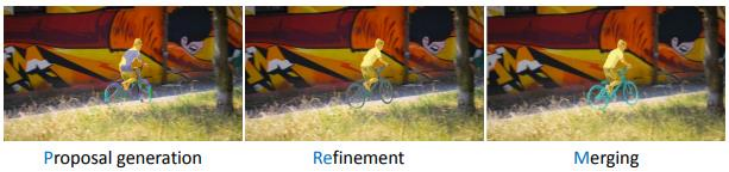
There are other papers in this field

- Fusionseg: Learning to combine motion and appearance for fully automatic segmentation of generic objects in videos." CVPR 2017 = combines optical flow with the above mentioned methods
- Lucid Data Dreaming for Video Object Segmentation, 2019 = clever data augmentation technique
- Video object segmentation with re-identification, 2017 = uses ReID techniques to recover from occlusions

Proposal-based approaches

- until now, we used whole images as inputs and proposals just to refine (in OSWOS-S)
- now we will explore methods that start with proposals and want to link them (as we did in tracking-by-detection)
- we will use instance segmentation networks (like Mask-RCNN) to obtain object instance segmentation proposals
- then we perform some data association to link these in time

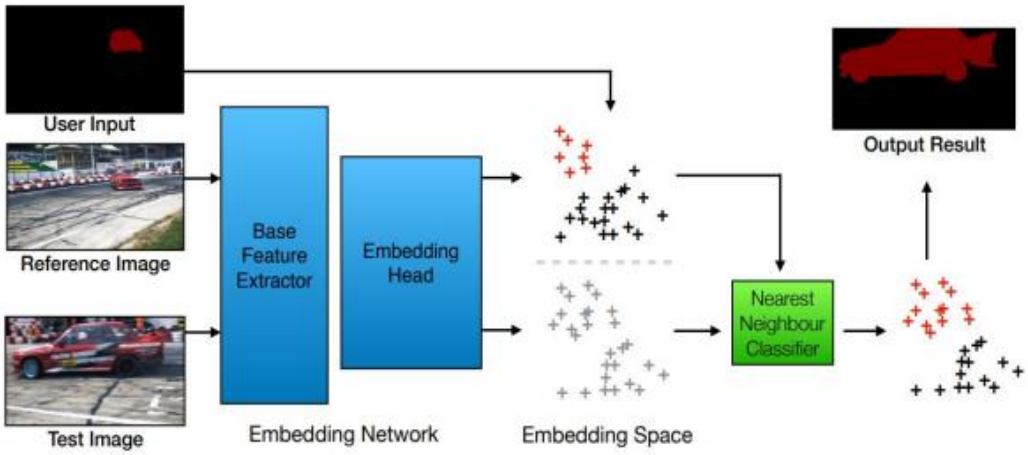
PReMVOS



- combines all of previous VOS principles and gives SOTA results
- combines first-frame fine-tuning, mask refinement, optical flow mask propagation, data augmentation, object appearance re-identification, proposal generation
- steps:
  - proposal generation: we get category-agnostic Mask-RCNN proposals
  - refinement: uses fully convolutional segmentation network trained to refine the segmentation given a proposal bounding box
  - merging: greedy decision process, chooses proposals with the best score
    - we can optionally expand this through optical flow propagation
    - proposal score is a combination of: objectness score, mask propagation IoU score (with Optical Flow propagation), ReID score, Object-Object interaction scores
- very complex but very good results and winner of challenges in 2018
- lessons learned:
  - how to generate proposals?
    - deep learning based region proposal generators are good
  - how to track region proposals?
    - region overlap works as a consistency measure
    - optical flow based propagation really helps
    - ReID score helpful
- open issues:
  - we have no notion of 3D objects moving through 3D space
  - the track initialization/termination logic is really hard and heuristic based
  - problem is how to obtain the initial segmentation

Retrieval approaches

- rely on pixel-wise retrieval
  - we have seen that ReID networks for bounding-box region proposals work really well
  - this idea can be extended for embedding for every pixel = we know if they are closer to foreground or background pixels
- steps:
  - user can give any type of input (first frame ground truth mask, scribbles in case of interactive segmentation)
  - in training we use triplet loss to bring foreground pixels together and separate them from background pixels
  - at test time, we need to embed pixels from annotated frame and test frame + perform nearest neighbor search for the test pixels



  - we do not need to retrain the model for each sequence, nor finetune (like we had to do in OSVOS)

Spatio-temporal approaches

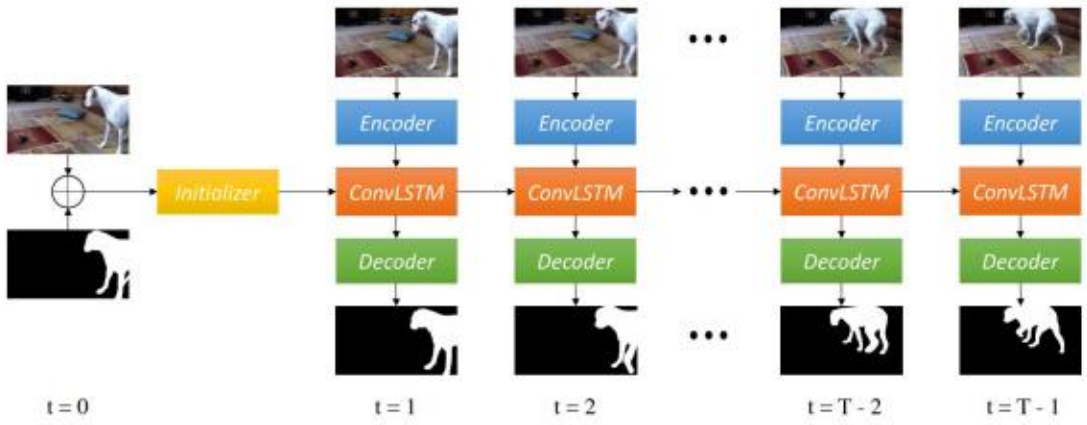
We are still ignoring temporal information of the video!

- how about we think about Recurrent Neural Network?

Recurrent Neural Networks

Temporal LSTM = Youtube-VOS

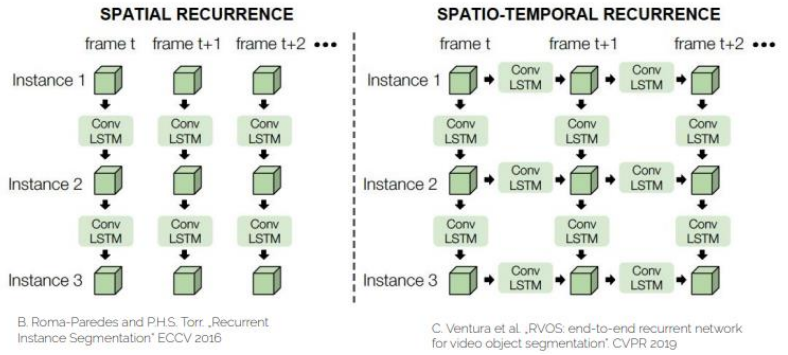
- does one-shot video object segmentation
- this info is fed into the initializer
- this is then processed by ConvLSTM
- then we use at each time step the CNN encoder, ConvLSTM and then decoder

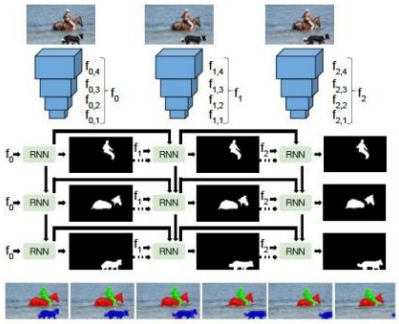


- if we have multiple objects, each of them is predicted independently

R-VOS

- deals with multiple objects in temporal domain
- uses spatio-temporal recurrence (**right part**)
    - use instances in a list as items that need to be returned in a recurrent fashion

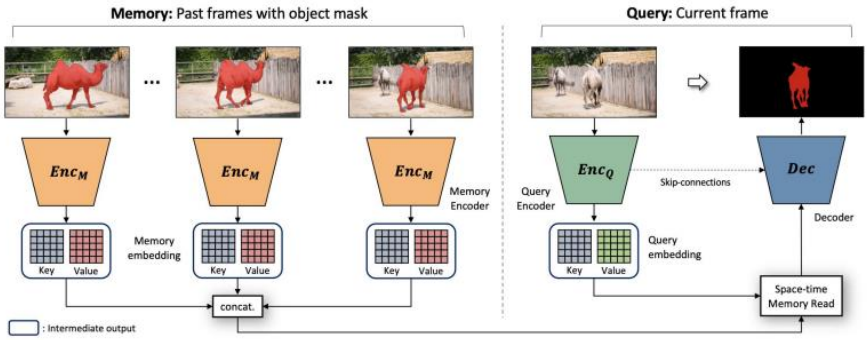




- instance generation and temporal coherence are both trained end2end
- image just needs to be processed once (unlike ConvLSTM before) -> more efficient

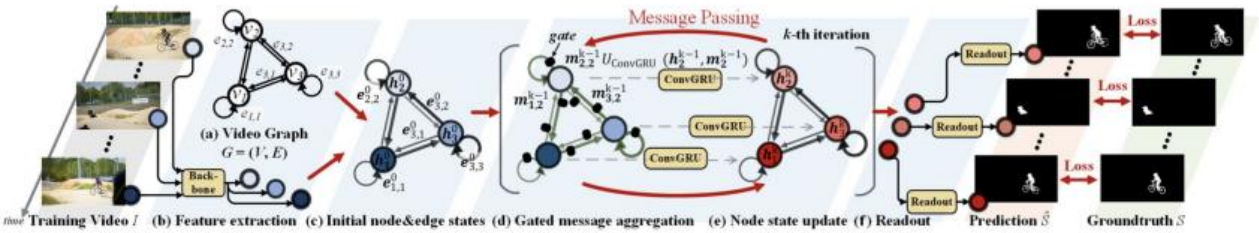
Transformers

- current state of the art
- we use attention to different spatial locations and different space-time memory



Graph attention Networks

- used for zero-shot segmentation, but could be similarly used for one-shot VOS



Conclusion

- OSVOS: First-frame fine-tuning (appearance model)
- OSVOS-S: semantic guidance through proposals (shape)
- OnAVOS: Online adaptation (stronger appearance model)
- MaskTrack: Mask refinement (motion)
- Lucid: clever data augmentation
- ReID-VOS: Object appearance Re-Identification (matching)
- PReMVOS: putting it all together (appearance + motion + shape + matching)
- Seq2seq and RVOS: recurrent architectures (motion)

Evaluation and metrics

- **Region similarity**: Jaccard index (IoU) of ground truth mask and predicted mask.
- **Contour Accuracy**: measures the precision and recall of the boundary pixels. This is put together in the F-measure.
    - on pixel level!

$$Precision = \frac{TP}{TP + FP} \qquad Recall = \frac{TP}{TP + FN}$$

$$F = \frac{2 * Prec * Rec}{Prec + Rec}$$

- **Temporal stability**:
    - measures the evolution of the object shapes = how stable are the boundaries in time
    - we measure the stability by estimating the deformation of the mask from *t* to *t+1*
        - if the transformation is smooth and precise, the result is considered stable
        - if the mask is jittery, we have a bad result
    - problem: occlusions = mask should actually be jittery
    - solution: this measure has been dropped
    - **Region similarity:**
        - used instead of the temporal stability
        - we use error measure statistics = compute IoU of ground truth mask and predicted mask
            - mean: average for the dataset
            - decay: quantifies the performance loss (or gain) over time (this is currently used to judge temporal stability)
            - recall: fraction of sequences scoring higher than a threshold

Merging tracking and segmentation

- this leads to transition from video object segmentation (VOS) to Multiple object tracking and segmentation (MOTS)
- VOS
    - first frame mask given (mostly)
    - short video clips with object present in almost all frames
    - objects are mostly of different categories and there are only few of them (max around 7 per video)
- MOTS
    - we will have a large number of objects (20-40), mostly of the same category (pedestrians)
    - long sequences
    - no first frame annotation provided, one has to deal with appearing and disappearing objects

11. 3D Tracking and Segmentation

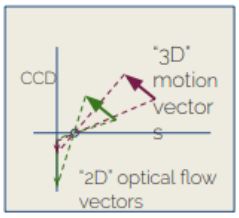
- robots need to detect objects to navigate
- it is beneficial to find the position in space for improving robotic estimation tasks

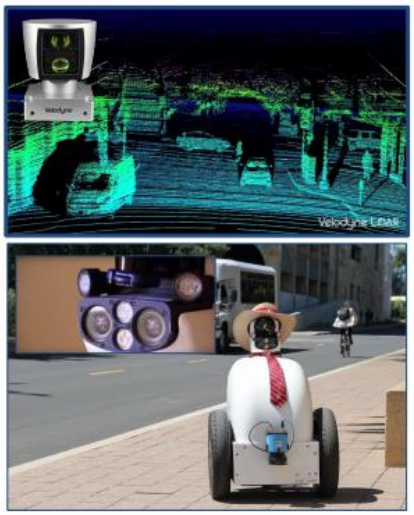
Reminder: Vision-based MOT

- we do tracking-by-detection and then do track associations using bipartite matching
- the matching is done using different costs
    - can we include some 3D information?

3D information sources

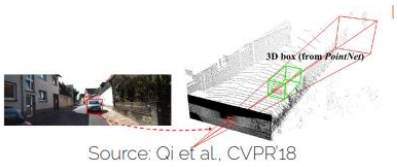
- we have variety of sensor available
    - stereo cameras, RGB-D cameras, LiDAR
- we can use this rich geometry
    - we can exploit it to devise stronger motion models
        - "apparent velocity" = real velocity gets distorted due to the camera perspective
    - we can also use geometric constrains
        - cars don't fly





3D tracking challenges:

- depth sensors limitations
    - limited scan range
    - non-cooperative materials
    - sparse and unstructured signal
- mobile platform, combining different sensor data
- object localization in 3D



Source: Qi et al., CVPR'18

Historical perspective

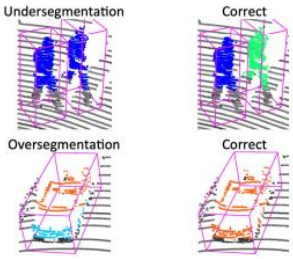
- origin in aeronautical and naval navigation (radars)
- then in robotics
    - at first 2D line laser scanners (we have a slice of the visual world), it is hard to detect anything
    - in 2005 there was DARPA autonomous driving challenge winner Stanley

Tracking-before-detection (ConvNets)

- we had full 360° LiDAR scans
- we would take each scan and then segment it
- we would project object onto ground and use low level spatial cues to find objects (using connected components), then use point projection classification (using HoGs = histograms of oriented gradients)
- message: tracking should be agnostic to recognition (if an unknown animal jumps in front of our car)

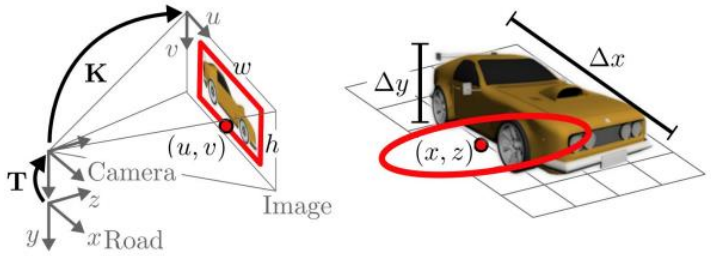
Segmentation

- is difficult
- we have interacting objects, crowded scenes
- sensor resolution decreasing with distance from the sensor, there are holes due to reflective/low-albedo surfaces

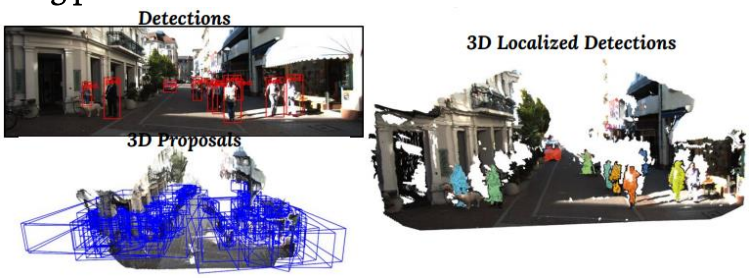


Stereo-vision based MOT

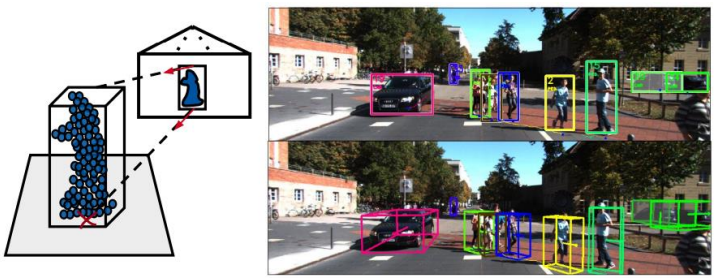
- uses the success of tracking by detection paradigm
- how to localize objects in 3D space?
  - **just using camera**



    - prone to errors: one pixel error can lead to big 3D error
  - **using point clouds = CIWT**



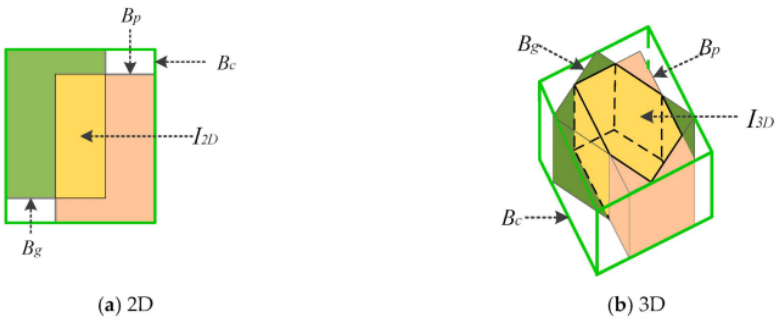
    - oversegmentation of stereo point clouds using several clustering parameters
    - we are jointly keeping track of both 2D and 3D spaces
      - we can update track states more smartly in severely occluded scenarios



    - this method actually works very well in online tracking even today

3D evaluation

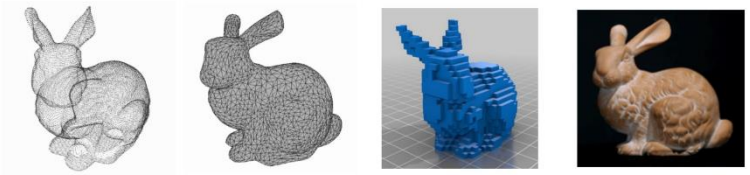
- as before: mAP, MOTA
- 3D IoU



(a) 2D (b) 3D

3D Object Detection Intro

- we will be using LiDAR point clouds



Deep learning on point clouds

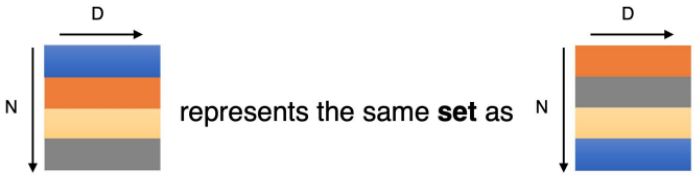
- which signal representation?
  - sensors provide us with point clouds
  - point clouds have irregular format that forced researchers to transform the data into volumetric
    - this process requires transformation that may bring errors
    - could we actually use the point cloud?

PointNet (2017)

- game-changer
- uses directly point clouds
- uses end2end learning for scattered, unordered point data

- challenges that PointNet has solved:
  - model needs to be invariant to N! permutations and geometric transformations

- permutation invariance:



$$f(x_1,x_2,\ldots,x_n) \equiv f(x_{\pi_1},x_{\pi_2},\ldots,x_{\pi_n}), \quad x_i \in \mathbb{R}^D$$

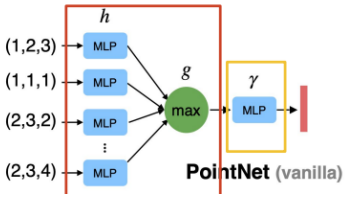
Examples:

$$f(x_1,x_2,\ldots,x_n) = \max\{x_1,x_2,\ldots,x_n\}$$
$$f(x_1,x_2,\ldots,x_n) = x_1 + x_2 + \ldots + x_n$$

  - we need a family of symmetric functions
  - PointNet solution:
    - Observe:

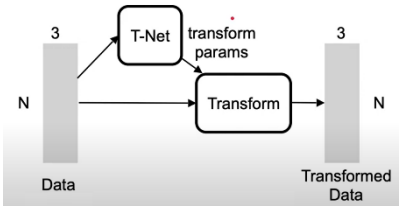
$$f(x_1,x_2,\ldots,x_n) = \gamma \circ g(h(x_1),\ldots,h(x_n)) \text{ is symmetric if } g \text{ is symmetric}$$

    - PointNet: MLP + max pooling



      - we add multi-layer-perceptron on top of each point
      - we learn spatial encoding of each point and then aggregate all individual point features to create global point cloud signature
      - on top we add our softmax classifier
  - this approach would work if we had always only canonical
  - but we would actually like to be invariant to transformations (such as rigid transformation)

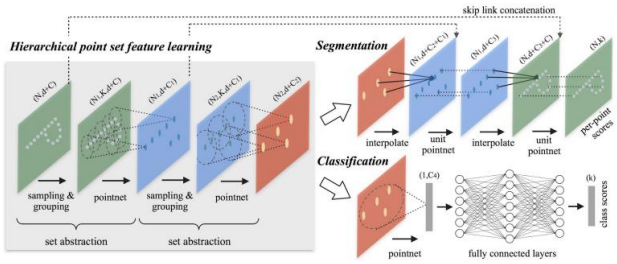
- invariance to transformation
  - PointNet solution:
    - we have a mini network (T-Net), that predicts the transformation
    - after transformation, we can apply the point extraction procedure from previous chapter



- PointNet problems:
  - does not capture local structures
  - global representation depends on absolute coordinates = poor generalization

PointNet++

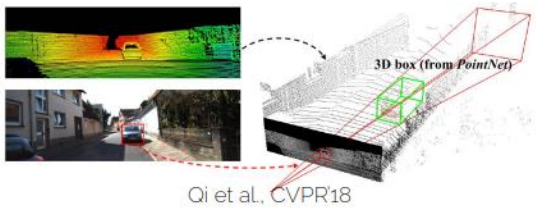
- applies PointNet recursively on a nested partitioning of the input point set
- learns local features with increasing contextual scales
- "Multi-scale point-net"



3D Object Detection Landscape

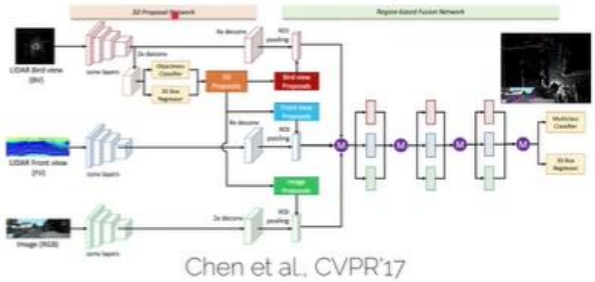
There are three leading paradigms

1. use image based detectors to detect objects in images and then do 3D object detection as a task of localizing objects in the point cloud. uses PointNet to find foreground/background pixels and then regress the bounding box

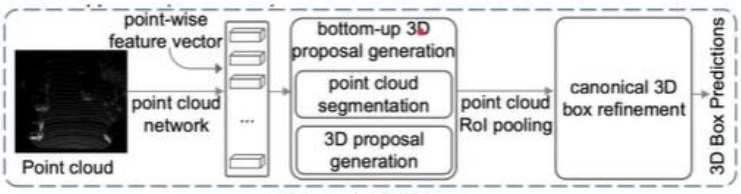


Qi et al., CVPR'18

2. based on 3D regions proposals and sensor fusion


Chen et al., CVPR'17

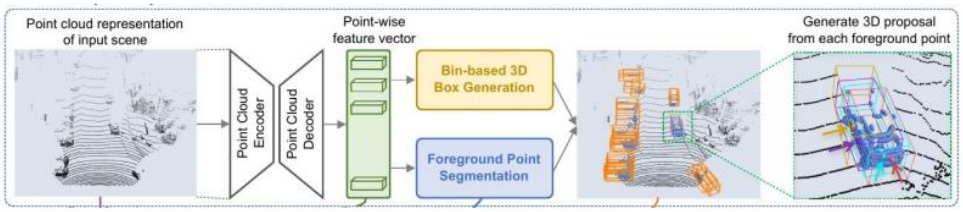
3. we follow idea of Faster R-CNN object detection = we get point cloud embedding, generate 3D object proposals and then classify



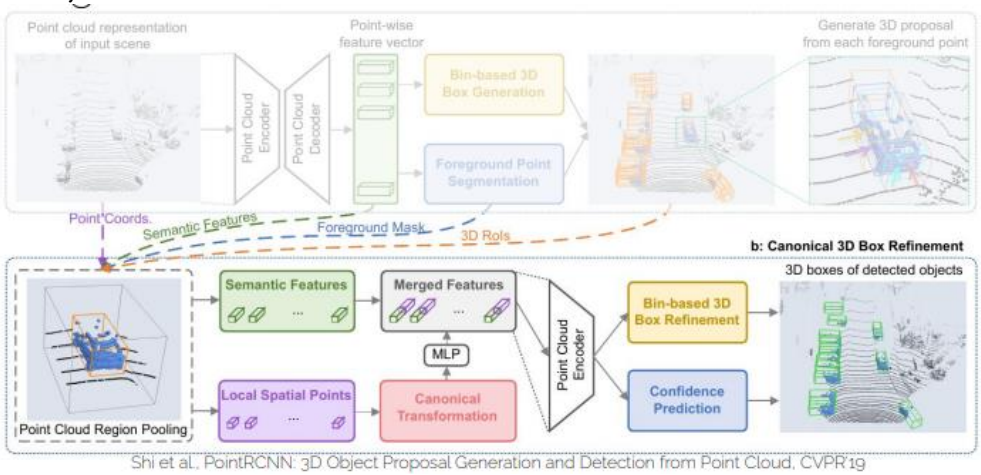
Shi et al., CVPR'19

Point R-CNN

- based on the 3rd approach
- 2 stage 3D detection network
    - foreground/background segmentation + proposal generation

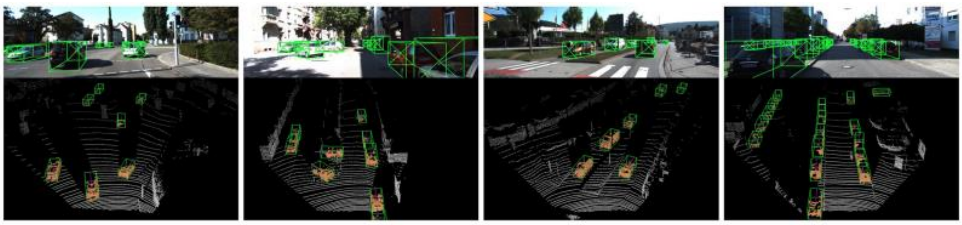


    - then slightly enlarge proposal, transform to canonical and then refine the bounding box + classify



Shi et al., PointRCNN: 3D Object Proposal Generation and Detection from Point Cloud, CVPR'19

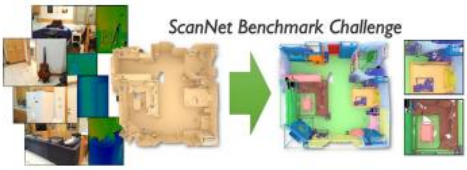
    - outperforms all prior work
        - including the ones that use LiDAR + Camera

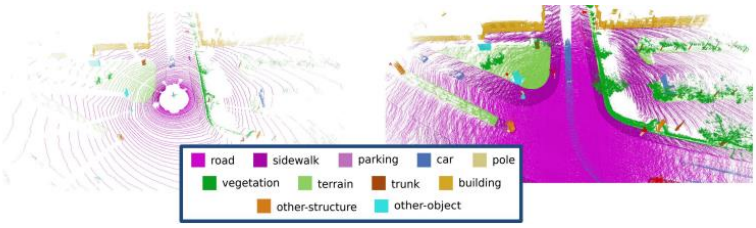


Method	Modality	Car (IoU=0.7)			Pedestrian (IoU=0.5)			Cyclist (IoU=0.5)		
		Easy	Moderate	Hard	Easy	Moderate	Hard	Easy	Moderate	Hard
MV3D [4]	RGB + LiDAR	71.09	62.35	55.12	-	-	-	-	-	-
UberATG-ContFuse [17]	RGB + LiDAR	82.54	66.22	64.04	-	-	-	-	-	-
AVOD-FPN [14]	RGB + LiDAR	81.94	71.88	66.38	50.80	42.81	**40.88**	64.00	52.18	46.61
F-PointNet [25]	RGB + LiDAR	81.20	70.39	62.19	**51.21**	**44.89**	40.23	71.96	56.77	50.39
VoxelNet [43]	LiDAR	77.47	65.11	57.73	39.48	33.69	31.51	61.22	48.36	44.37
SECOND [40]	LiDAR	83.13	73.66	66.20	51.07	42.56	37.29	70.51	53.85	46.90
Ours	LiDAR	**85.94**	**75.76**	**68.32**	49.43	41.78	38.63	**73.93**	**59.60**	**53.59**

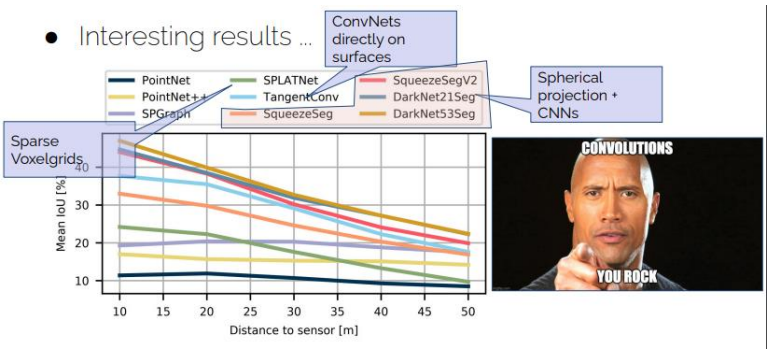
3D Segmentation

- classifying each 3D point
- the most used dataset is ScanNet
- however this is not suitable for autonomous driving applications (they are sparse, contain different distances) -> there is SemanticKITTI

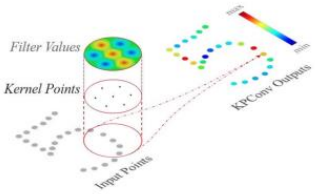




- results of different 3D networks decrease with distance from sensor



  - projective method performs better than point based methods
  - however, the best results are achieved using point convolutions!
    - Kernel Point Convolution



- General point convolution

$$(\mathcal{F} * g)(x) = \sum_{x_i \in \mathcal{N}} g(x_i - x) f_i$$

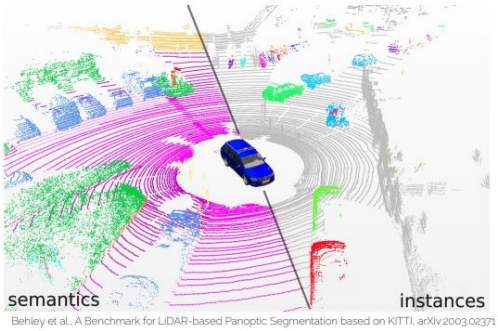
Kernel function (domain: r-Ball

3D Panoptic Segmentation

- simple baseline:
  - compute semantic segmentation and object detections
  - fuse the results (using heuristic postprocessing)

Method	mIoU	PQ	$PQ^{\dagger}$	RQ	SQ	PQ^{Th}	RQ^{Th}	SQ^{Th}	PQ^{St}	RQ^{St}	SQ^{St}
KPConv [21] + PointPillars [13]	58.8	44.5	52.5	54.4	80.0	32.7	38.7	81.5	53.1	65.9	79.0
RangeNet++ [16] + PointPillars [13]	52.4	37.1	45.9	47.0	75.9	20.2	25.2	75.2	49.3	62.8	76.5

TABLE II: Comparison of test set results on SemanticKITTI using *stuff*(St) and *thing*(Th) *classes*. All results in [%].

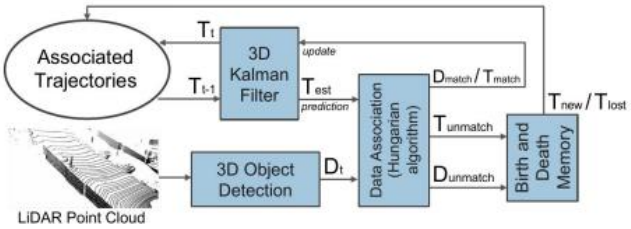


Behley et al.. A Benchmark for LiDAR-based Panoptic Segmentation based on KITTI. arXiv:2003.02371

3D MOT

AB3D-MOT

- even a simple tracker yields great performance
- bi-partite matching, 3D IoU
- dynamics model: const-velocity Kalman filter



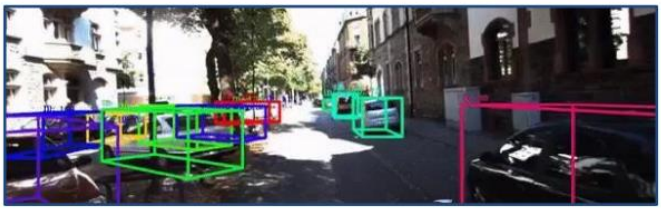
$$\mathbf{x}^k = [x, y, z, \theta, w, h, l, s, v_x, v_y, v_z]$$

$$f(\mathbf{x}):$$

$$\hat{x}^{k+1} = x + v_x,$$

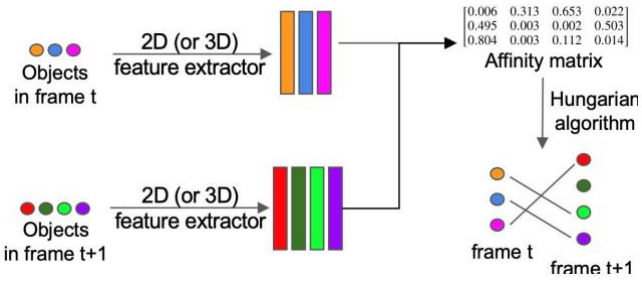
$$\hat{y}^{k+1} = y + v_y,$$

$$\hat{z}^{k+1} = z + v_z.$$

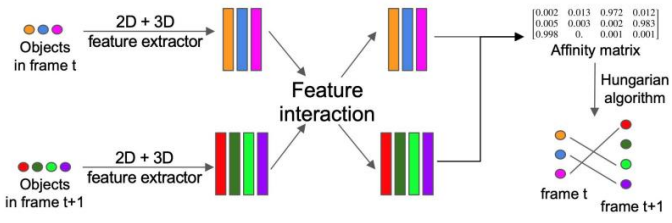


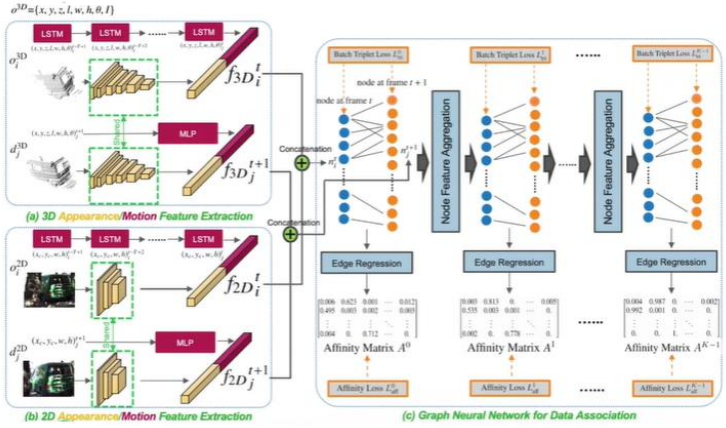
Method	Type	MOTA (%) ↑	MOTP (%) ↑	MT (%) ↑	ML (%) ↓	IDS ↓	FRAG ↓	FPS ↑
Complexer-YOLO [26]	3D	75.70	78.46	58.00	**5.08**	1186	2092	**100.0**
DSM [22]	3D	76.15	83.42	60.00	8.31	296	868	10.0 (GPU)
MDP [46]	2D	76.59	82.10	52.15	13.38	130	387	1.1
LP-SSVM [27]	2D	77.63	77.80	56.31	8.46	62	539	50.9
FANTrack [21]	3D	77.72	82.32	62.61	8.76	150	812	25.0 (GPU)
NOMT [38]	2D	78.15	79.46	57.23	13.23	**31**	**207**	10.3
MCMOT-CPD [28]	2D	78.90	82.13	52.31	11.69	228	536	**100.0**
extraCK [24]	2D	79.99	82.46	62.15	5.54	343	938	33.9
3D-CNN/PMBM [23]	2.5D	80.39	81.26	62.77	6.15	121	613	71.4
JCSTD [25]	2D	80.57	81.81	56.77	7.38	61	643	14.3
BeyondPixels [20]	2D	**84.24**	**85.73**	**73.23**	**2.77**	468	944	3.3
Ours	3D	**83.84**	**85.24**	**66.92**	11.38	**9**	**224**	**214.7**

- AB3DMOT (and existing):



GNN3DMOT





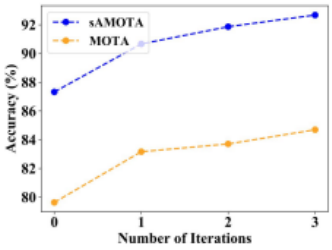
- How to learn representations using Graph Neural networks?

$$A_{ij} = \text{Sigmoid}(\sigma_2(\text{ReLU}(\sigma_1(n_i^t - n_j^{t+1}))))$$

Linear layers — Features at time $t, t+1$

  - Trained using triplet loss, cross-entropy ("affinity") loss
    - MOTA/AMOTA/sAMOTA improves (+ 1.35 MOTA)

Method	Input Data	sAMOTA (%) ↑	AMOTA (%) ↑	AMOTP (%) ↑	MOTA (%) ↑	MOTP (%) ↑	IDS ↓	FRAG ↓
mmMOT [59] (ICCV'19)	2D + 3D	70.61	33.08	72.45	74.07	78.16	10	125
FANTrack [2] (IV'19)	2D + 3D	82.97	40.03	75.01	74.30	75.24	35	202
AB3DMOT[48] (arXiv'19)	3D	91.78	44.26	77.41	83.35	78.43	**0**	15
Ours	2D + 3D	**93.68**	**45.27**	**78.10**	**84.70**	**79.03**	**0**	**10**



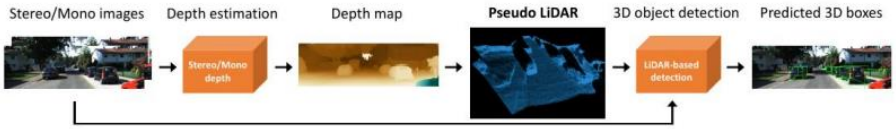
- Ablation
  - Large gap between 2D and 3D motion model
  - 3D motion > 2D appearance > 3D appearance
  - Performance gain when combining 2D+3D

Feature Extractor	sAMOTA (%) ↑	AMOTA (%) ↑	AMOTP (%) ↑	MOTA (%) ↑
2D A	88.31	41.62	76.22	79.42
2D M	64.24	23.95	61.13	54.88
3D A	88.27	41.55	76.29	77.38
3D M	88.57	41.62	76.22	81.84
2D+3D A	89.39	42.55	76.24	83.02
2D+3D M	91.75	44.75	78.05	84.54
2D M+A	90.56	44.39	**78.20**	83.15
3D M+A	91.30	44.31	78.16	84.06
2D+3D M+A (**Ours**)	**93.68**	**45.27**	78.10	**84.70**

  - Motion cues are super useful!

Stereo-cameras

- we use depth estimation using stereo/mono images and then convert it to Pseudo LiDAR, we can actually use all the above-mentioned approaches for LiDAR



Detection algorithm	Input signal	IoU = 0.5 Easy	IoU = 0.5 Moderate	IoU = 0.5 Hard	IoU = 0.7 Easy	IoU = 0.7 Moderate	IoU = 0.7 Hard
MONO3D [4]	Mono	30.5 / 25.2	22.4 / 18.2	19.2 / 15.5	5.2 / 2.5	5.2 / 2.3	4.1 / 2.3
MLF-MONO [33]	Mono	55.0 / 47.9	36.7 / 29.5	31.3 / 26.4	22.0 / 10.5	13.6 / 5.7	11.6 / 5.4
AVOD	Mono	61.2 / 57.0	45.4 / 42.8	38.3 / 36.3	33.7 / 19.5	24.6 / 17.2	20.1 / 16.2
F-POINTNET	Mono	70.8 / 66.3	49.4 / 42.3	42.7 / 38.5	40.6 / 28.2	26.3 / 18.5	22.9 / 16.4
3DOP [5]	Stereo	55.0 / 46.0	41.3 / 34.6	34.6 / 30.1	12.6 / 6.6	9.5 / 5.1	7.6 / 4.1
MLF-STEREO [33]	Stereo	-	53.7 / 47.4	-	-	19.5 / 9.8	-
AVOD	Stereo	89.0 / 88.5	77.5 / 76.4	68.7 / 61.2	74.9 / 61.9	56.8 / 45.3	49.0 / 39.0
F-POINTNET	Stereo	89.8 / 89.5	77.6 / 75.5	68.2 / 66.3	72.8 / 59.4	51.8 / 39.8	44.0 / 33.5
AVOD [17]	LiDAR + Mono	90.5 / 90.5	89.4 / 89.2	88.5 / 88.2	89.4 / 82.8	86.5 / 73.5	79.3 / 67.1
F-POINTNET [25]	LiDAR + Mono	96.2 / 96.1	89.7 / 89.3	86.8 / 86.2	88.1 / 82.6	82.2 / 68.8	74.0 / 62.0

Wang et al., Pseudo-LiDAR from Visual Depth Estimation, CVPR'19

3D TAKEAWAYS

- Nowadays, we know how to learn representations from unstructured point clouds, yay!
- 3D object detection, semantic/instance segmentation
- 3D detection/tracking/segmentation vibrant and exciting area of research!
- Surprisingly, we can turn any depth map to a point cloud and apply techniques we learned about -- unifying framework!