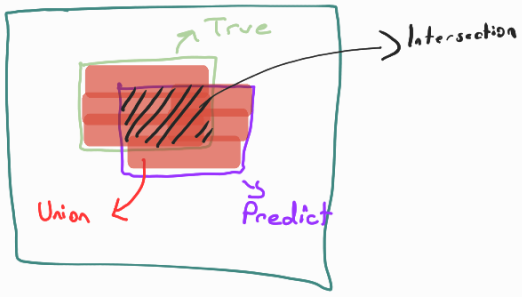
Object Evaluteion

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

$TP+FP$: Number of predicted boxes

$TP+FN$: Number of ground-truth boxes

IoU = Intersection over Union



Computing Average Precision

1. Find predicted boxes by using confidence scores.
2. For each boxes find ground trut box. Ground truth should be unassigned and pass IoU threshod. → if 2 prediction goes the some ground truth select the most confident.
3. Find Presicion and recall
4. Find max Presicion such that is at least $R \in \{0, 0.1 .. 1\}$

- Plot Precision vs Recall

1	2	3	4	5	6	7	8	9	10	11	12
T	T	F	T	F	F	T	F	T	T	F	F

1) $P = \frac{1}{1} = 1$ $r = \frac{1}{6} = 0.17$

2) $P = \frac{2}{2} = 1$ $r = \frac{2}{6} = 0.33$

3) $= \frac{2}{3} = 0.66$ $r = \frac{2}{6} = 0.33$

4) $= \frac{3}{4} = 0.75$ $r = \frac{3}{6} = 0.5$

5) $= \frac{3}{5} = 0.6$ $r = \frac{3}{6} = 0.5$

6) $= \frac{3}{6} = 0.5$ $r = \frac{3}{6} = 0.5$

7) $= \frac{4}{7} = 0.57$ $r = \frac{4}{6} = 0.66$

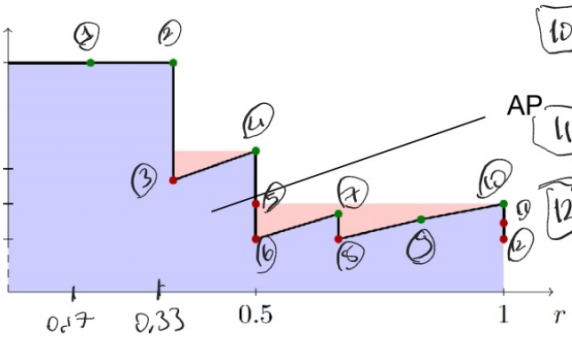
8) $P = \frac{4}{8} = 0.5$ $r = \frac{4}{6} = 0.66$

9) $P = \frac{5}{9} = 0.55$ $r = \frac{5}{6} = 0.83$

10) $P = \frac{6}{10} = 0.6$ $r = \frac{6}{6} = 1$

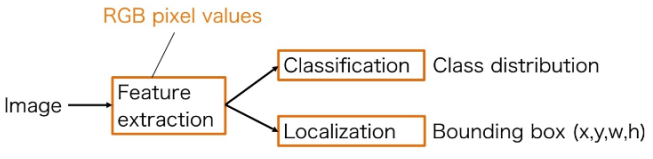
11) $P = \frac{6}{11} = 0.54$ $r = \frac{6}{6} = 1$

12) $= P = \frac{6}{12} = 0.5$ $r = \frac{6}{6} = 1$

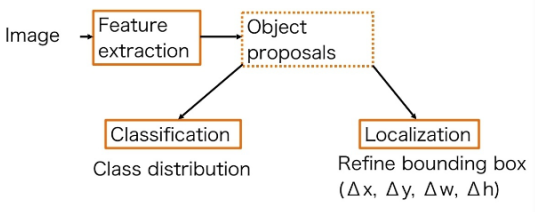


Types Of Object Detectors

- One-stage detectors:



Two-stage detectors

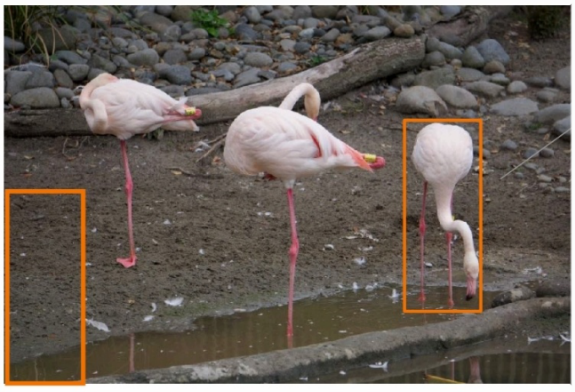


First lets start with one stage detectors.

For feature extraction we need methods

1- Template matching

You have template and searching in image



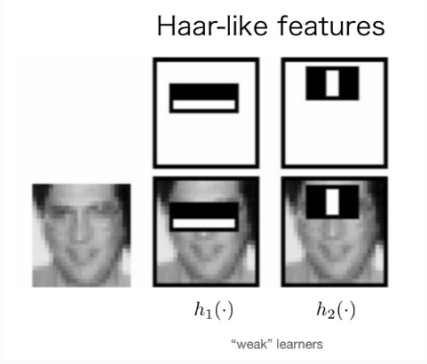
Disadvantages

1. Self-occulusions
   (Resmenin kendi parcasinin kendini kapamasi)

Exp: Look cube from upper side you will see the square

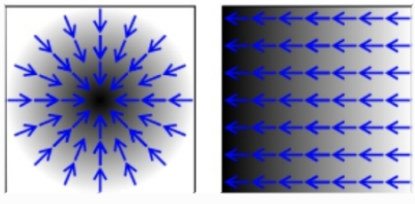
2. Changes in appearance

2. Viola-Jones Detector



Apply some Haar like features

3. Histogram of Orient (HOG)



We can use gradients

So we need something working data

5. YOLO

In next pages

4. Overfeat

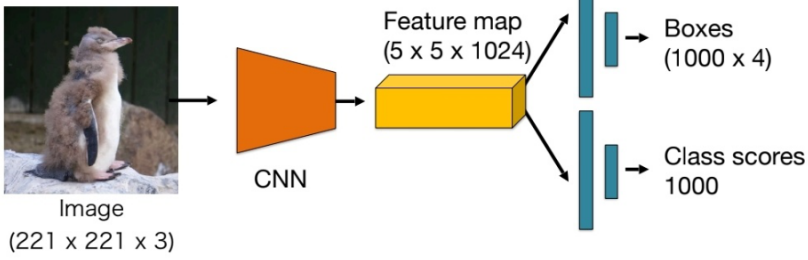
- Train the classification head, first then regression head, freeze layers.

- Sliding Window used here

Disadvantages

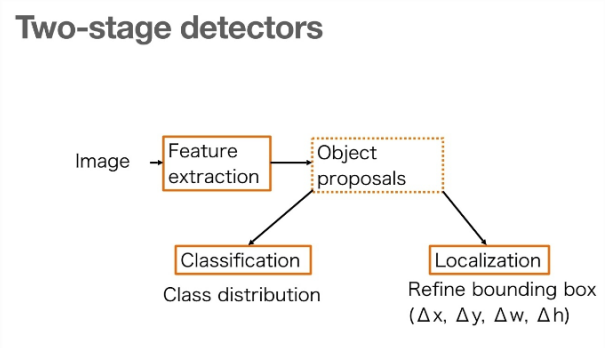
1- Expensive to try all possible positions, scales and aspect ratios.

2- Network works on a fixed input



Two-stage detectors

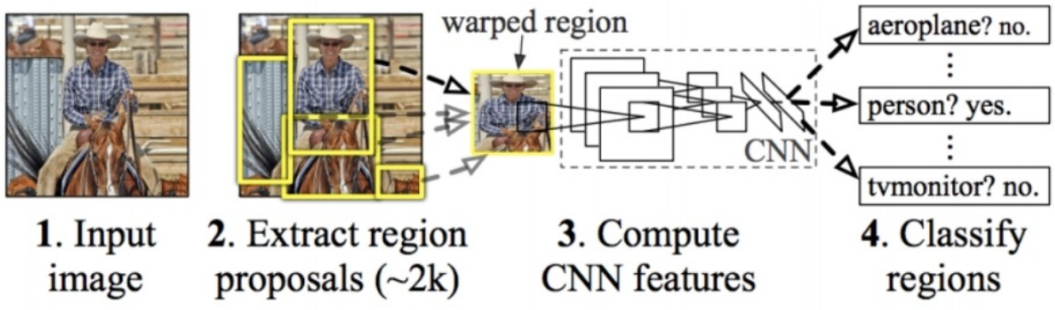
We can use region proposal as a stage in here.



1-) RCNN

R-CNN

Warping to 227x227



or extract locations

1-) Pretrain on CNN ImageNet

2-) Finetune CNN to your dataset for classification

3-) Train your SVM to classify image regions

4-) Train the bounding box regressor

Pros

- CNN Features
- More compact vectors than HOG
- Tranfer Learning

Cons

- 1 image 2000 proposal and they warped
- Training also slow
- Object proposal algo is fixed
- Not end-to-end, Feature extractor and SVM classifier trained seperately

2-) Fast R-CNN

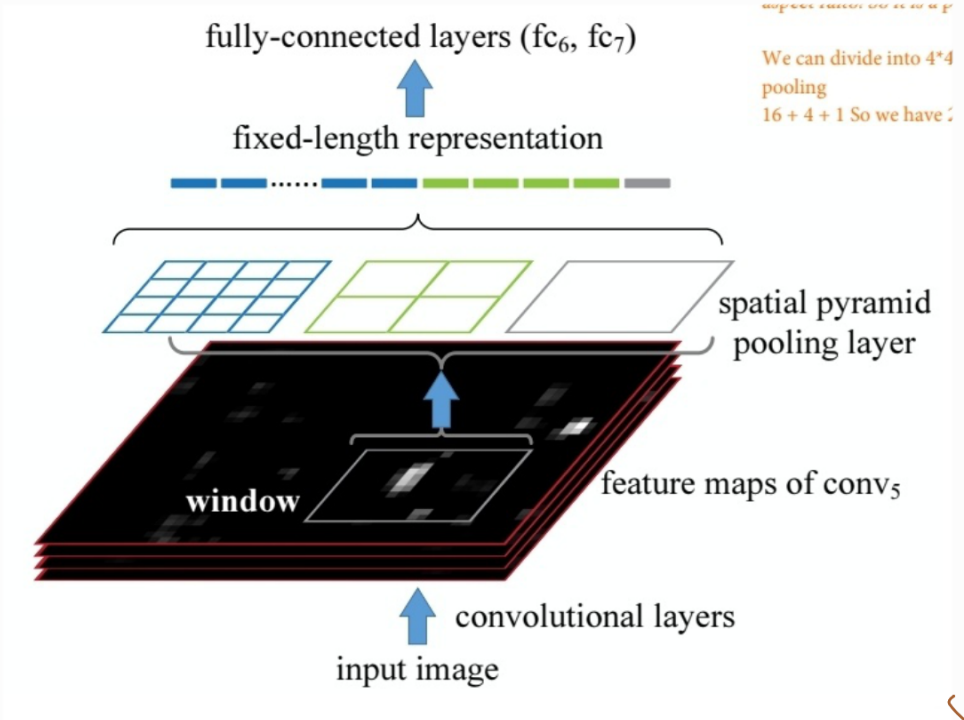
- We need to deal with any input size each object can be different size and aspect ratio

2.1-) SPP-NET

Note: SPP-NET not in the Fast R-CNN But, introduced before it.

Problem / Pooling Layers had fixed size in R-CNN so output remains proportional to input aspect ratio.

Solution



-> Spatial Pyramid Pooling is part of SPP-NET

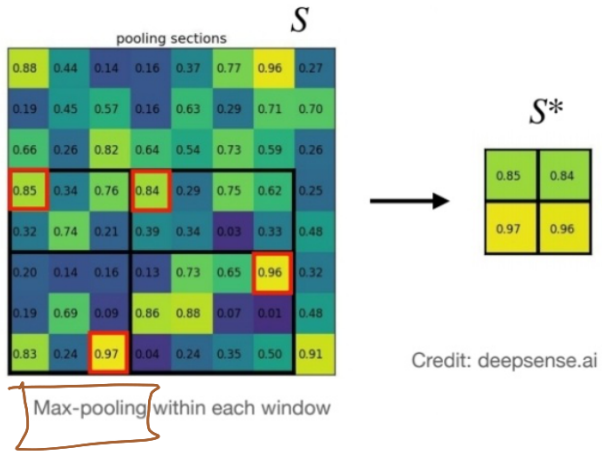
It creates 4x4, 2x2, 1x1

Lets say we have 13x13 feature map

Input -> CNN -> 13x13 feature map

window = ceil (13/4) = 4
stride = floor (13/4) = 3
} if you apply this you will get 4x4

- Is it differentiable?
- S^* is differentiable w.r.t. S ?
  - yes and no – depends on pooling
- S^* is differentiable w.r.t. (x, y, h, w) ?
  - no

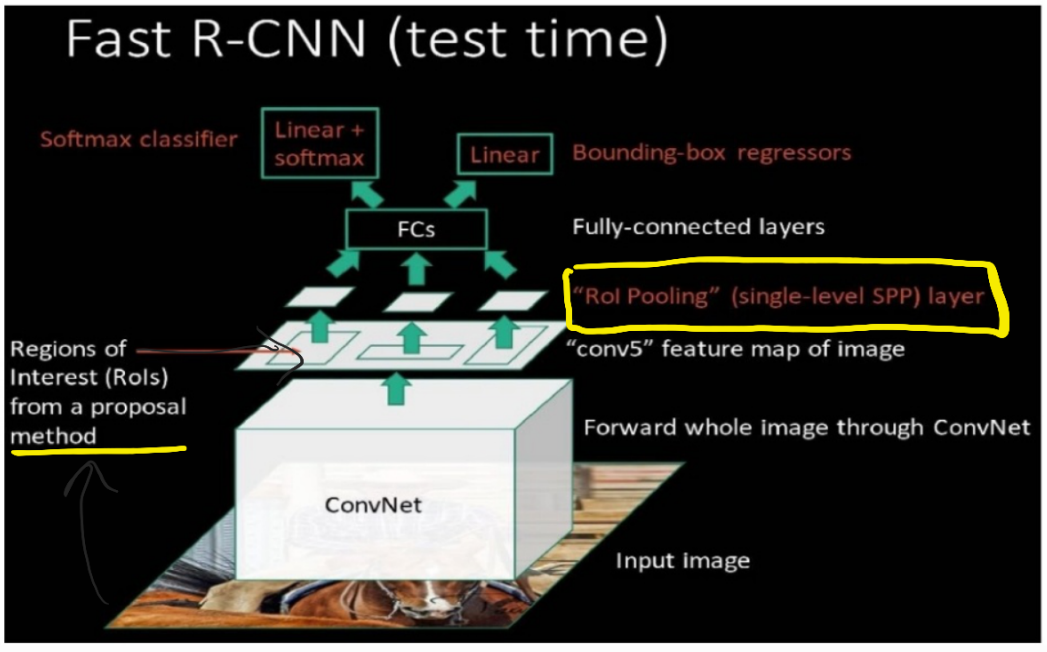


Example of Spitial Pooling

Advantages
- Faster then RCNN

Disadvantages
- Traing is complex
- No end-to-end train (Fixed CNN Layers)

Fast RCNN continues...



Shared computation at test time (like SPP)

1. We extract regions as RCNN does. N proposal
2. We have feature map from Convnet. We get each proposal feature from Conv net
3. Each proposal feature map goes RoI pooling Layer to get fixed size. (Single-Level SPP)
4. Soft max and regresson.

Fast R-CNN

Advantages

1-) Can deal with arbitary aspect ratios

2-) Faster than R-CNN (We only use Convnet Once)

3-) More Accurate

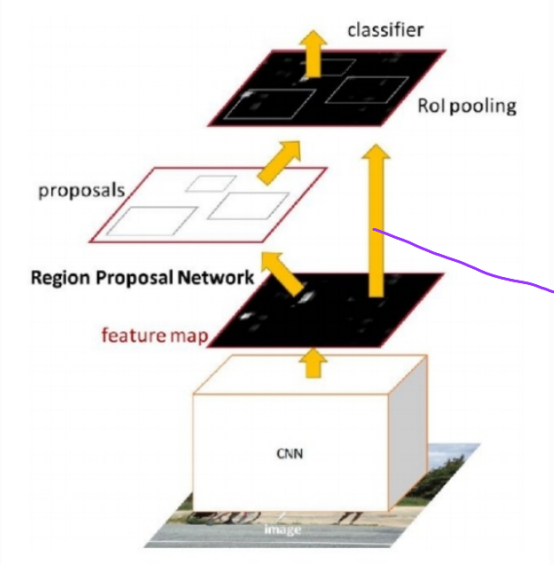
4-) Doesn't need to cache feature vector. (storage)

Disadvantages

- Standalone Object Proposal
- Still slow

---

3. Faster R-CNN



The only difference is

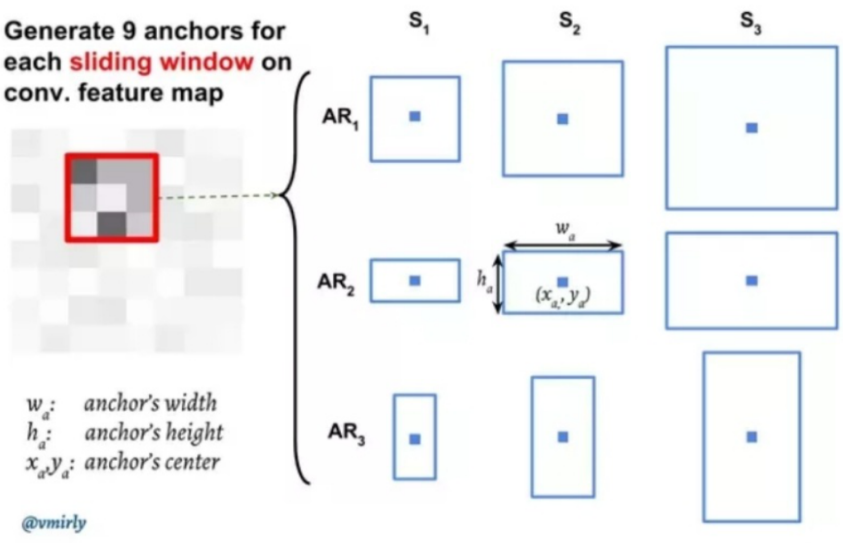
- new Region Proposal Network (RPN)

Also we are using CNN features in RoI pooling

3.1 RPN

We have feature map from CNN net.

Each anchor is described by the center position, width and height



For each location, we present 9 anchor. 3 diffart size, 3 different aspect ratio.

p^* ->

Anchor represent an object

$$p^* = 1 \quad if \quad \text{IoU} > 0.7$$
$$p^* = 0 \quad if \quad \text{IoU} < 0.3$$

->We dont get $0.3 < x < 0.7$ since we are not sure.

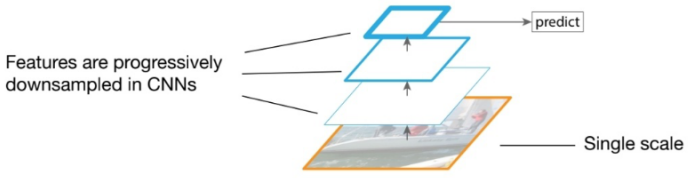
-> Anchor is background.

Faster R-CNN Advantages

->We can define arbitrary size in anchor

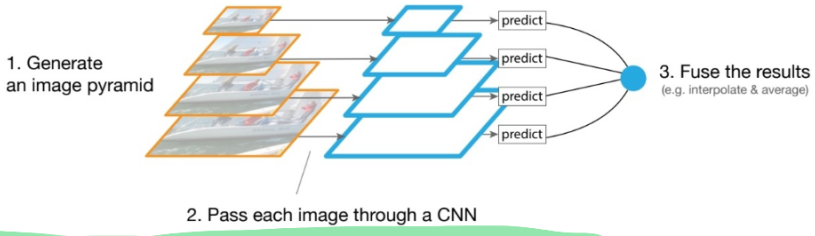
Addressing scale variance

- CNNs are not scale-invariant → Small objects can Lost
- Our approach so far:



➡ Make CNNs for object detection more robust to scale changes

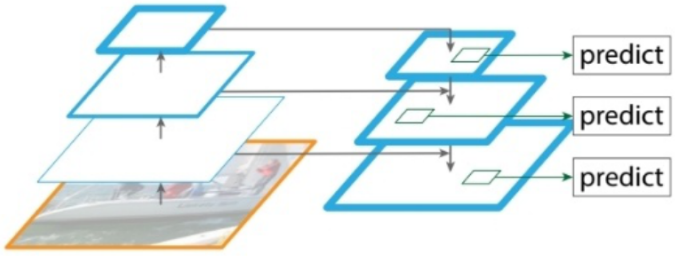
- Idea A: featurised image hierarchy



We can try interpolate results at each Layer

- Pros: Typically boosts accuracy (esp. at test time)
- Cons: Computationally inefficient

Feature Pyramid Network (FPN)



- Define RPN at each layer
- Merge predictions from all Levels at test time

Low scale → About big picture Overall structure Layout

High Scale → Details of individual Objects

Pro

- Better Recall
- Good at small objects
- More Accurate
- Also usable in one stage

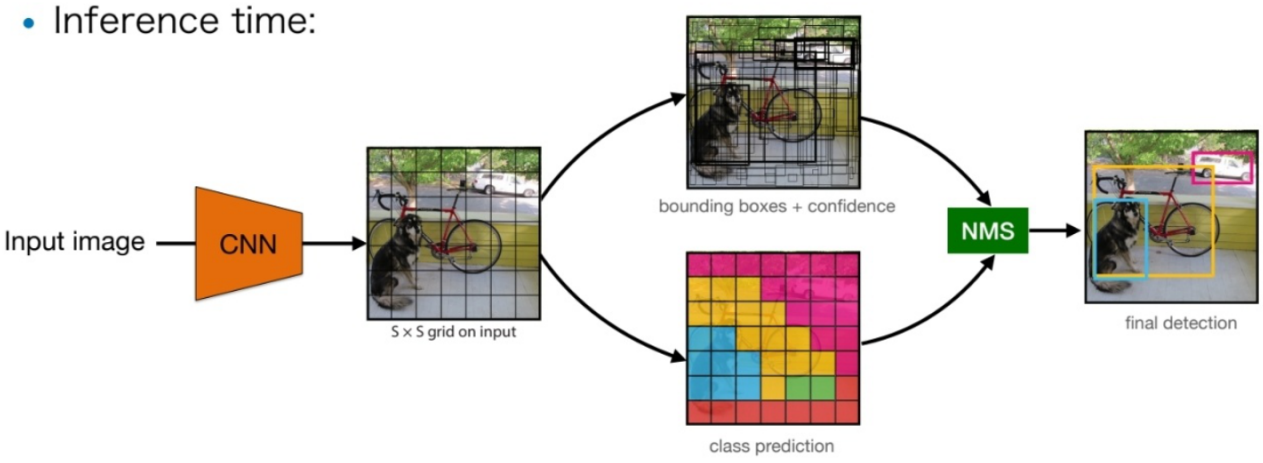
Cons

- Increased Model Complexity

One Stage Detectors - Continue

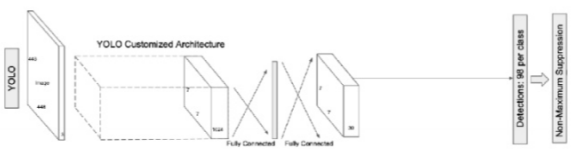
5-) YOLO - You Look Only Once

- Inference time:



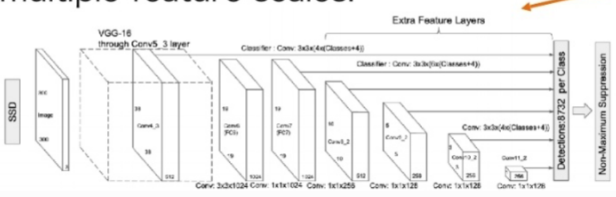
- Divide image to grids (SxS)
- Each cell has B anchors
- Each anchor has (x, y, w, h), confidence score and class distribution over C classes
- More efficient than Faster-RCNN
- Less robust to scale variation since no spatial pooling

- YOLO predicts bounding boxes from a single representation



- SSD uses multiple feature scales:

Only a few extra layers



- A bit more complex

6-) SSD

- We can add multiple feature scales to YOLO for fixing scale variation

Pro
- More Accurate YOLO
- Work better in Lower res
- Inference speed

Cons
- Still Lags behind two-stage
- Data Aug necessary

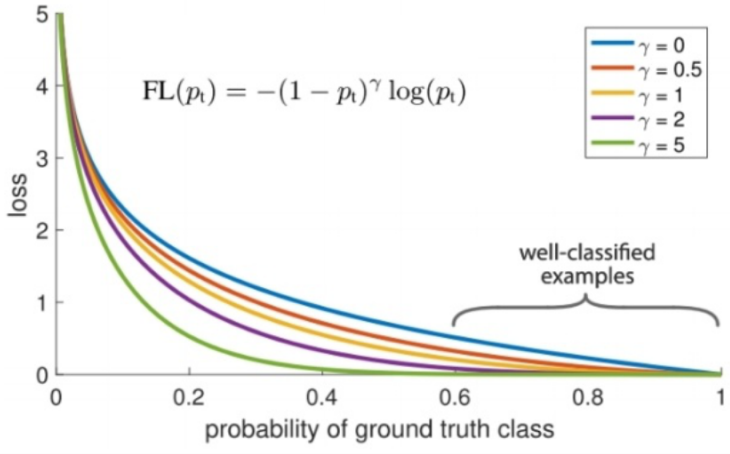
7-) Retina Net

One stage like YOLO and SSD with Focal Loss

- Feature extract with Resnet
- Multiscale prediction - now with Feature Pyramid Network
- Better accuracy then two-stage detectors+ more efficient.

Focal Loss: (Class imbalance problem

- Replace CE with focal loss (FL):



- When $\gamma = 0$, it is equivalent to the cross-entropy loss.
- As γ goes towards 1, the easy examples are down-weighted.
- Example: $\gamma = 2$, if $p_t = 0.9$, FL is $\times 100$ lower than CE.

Back-of-the-envelope calculation:
100 hard examples x 2.3 x 0.9^2 = 186.3
10^6 easy examples x 0.1 x 0.1^2 = 1000

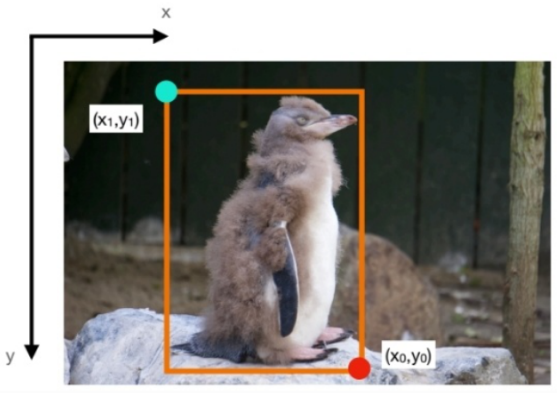
One Stage

- Eeasier model design
- If you apply data augmentation, competitive accuracy
- Fast
- Class inbalance problem (Focal Loss)
- Hard negative mining

Two Stage

- More Accurate (Robustness to scale variation)
- Hard to implement (Due to Pooling)
- Slower than one-stage
- Classificotion work on "interesting" foreground regions. Background objects filtered out.
- Class balance foreground and background is manggable.
- Classifier can concentrate on analyzing proposals with rich information content.

Box Representation

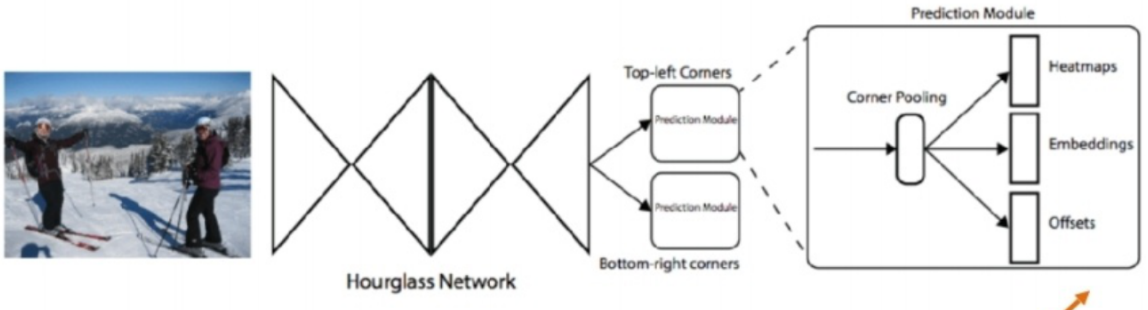


- (x_0, y_0, h, w)
- (x_0, y_0, x_1, y_1)

Corner Net

Hourglass network for keypoint detection

A. Newell et al. "Stacked hourglass networks for human pose estimation". 2016



We predict corners at a lower resolution and then regress an offset (bounding box correction as we have seen for all methods)

Problems

- Many Incorrect boundingboxes
- FP

There is Also centerNet which focuses on center of object proposal

⭐ Sequential Detection ⭐

1.) Recorrent Object Detection

- No anchors
- Used LSTM's
- Does not scale very well to high number of objects
- Classification start/stop criteria is challenging to learn well.

Hungarian matching using IoU score

Tracking

Challenges

- Changing apperance, object pose
- Dynamic background
- Occlusions
- Fast motion

Online vs Offline Tracking

Online
- Given observation so far, estimate current state
- Real time
- Hard to recover from errors or occlusions

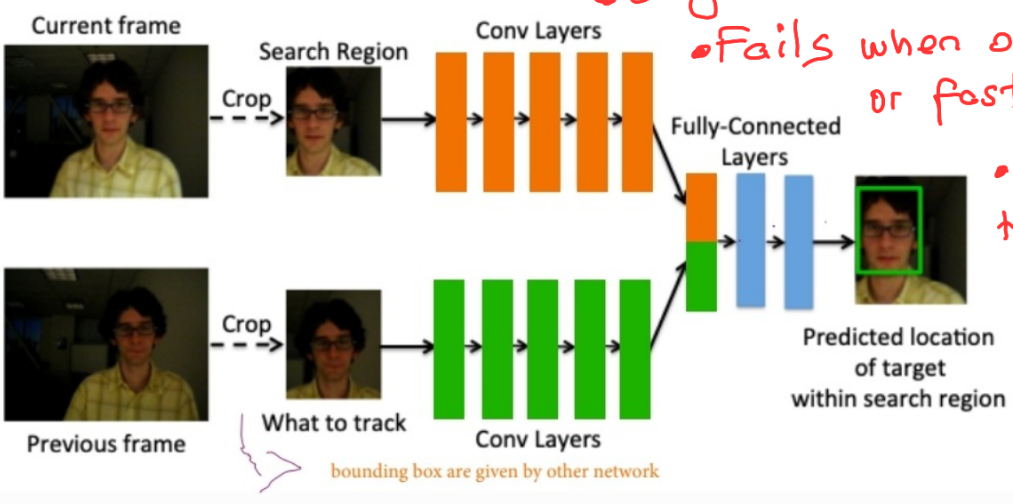
Offline
- Given all observations estimate any state
- Good to recover from occlusions
- Video analysis

Single Target Tracking (GOTURN)

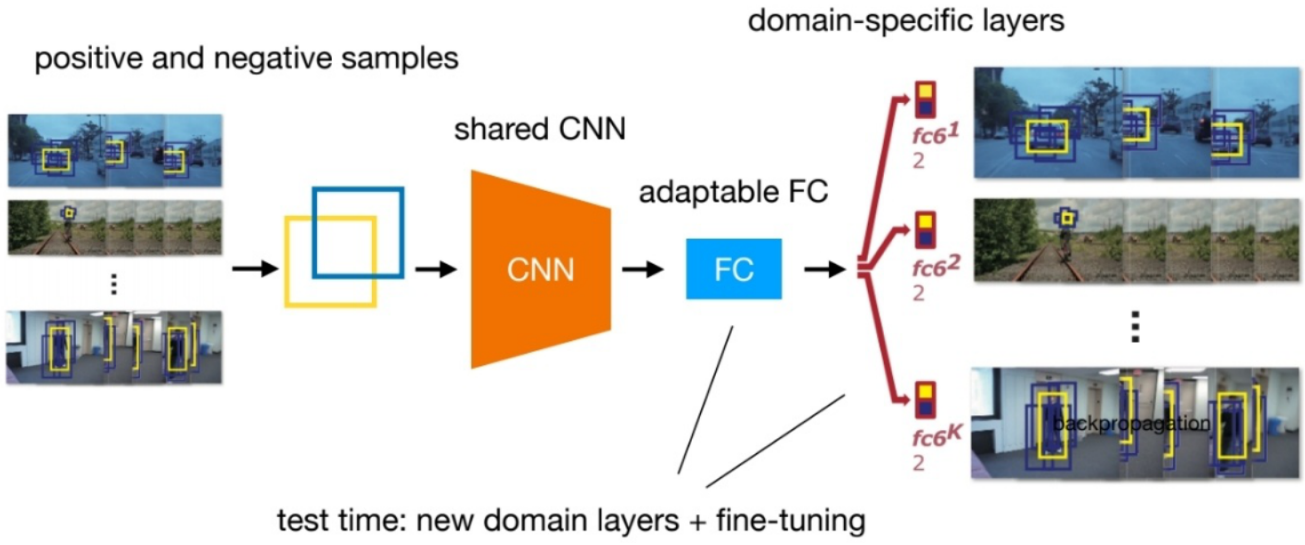
- Input: search region + template region (what to track)
- Output: bounding box coordinates in the search region

- Simple
- Efficient
- End-to-end

- Only track one object
- Fails when occlusions or fast motion
- Sensetive to temporal choice



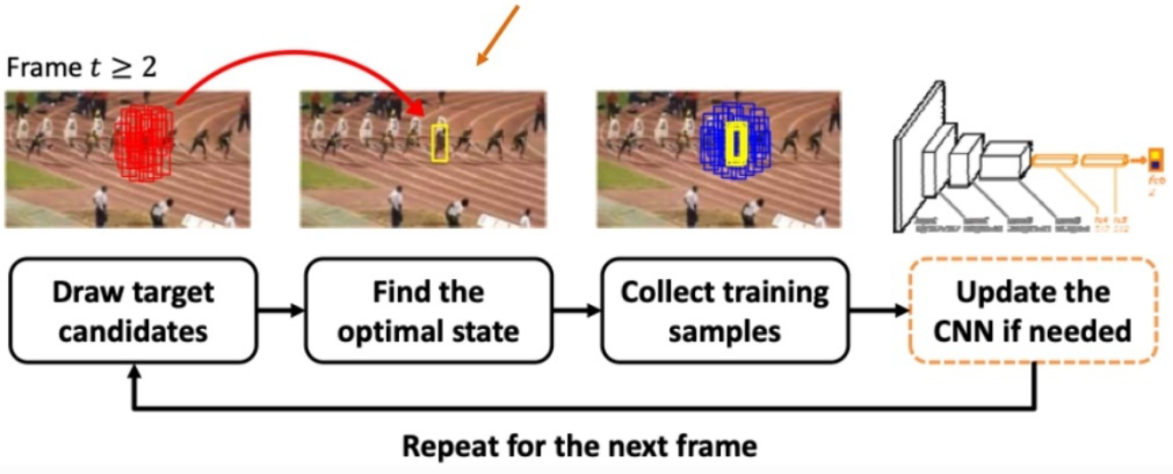
Single Target Tracking (MDNet)



Single Target Tracking (MDNet)

- Online tracking

R-CNN type of regression



Advantages

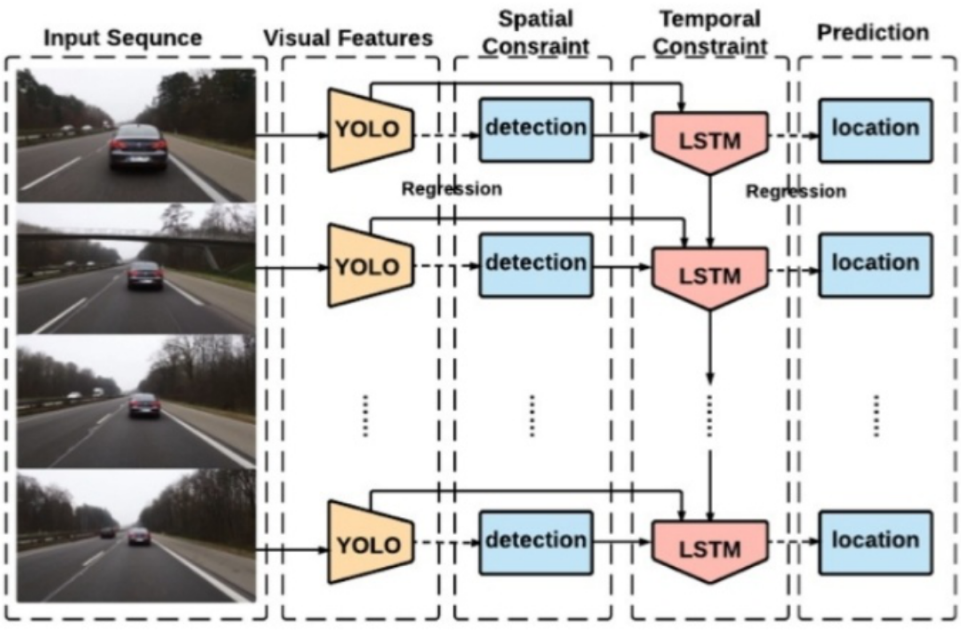
- adaptive appearance model.
- fine-tuning step is comparatively cheap.
- Winner of the VOT Challenge 2015 (http://www.votchallenge.net)

Disadvantages

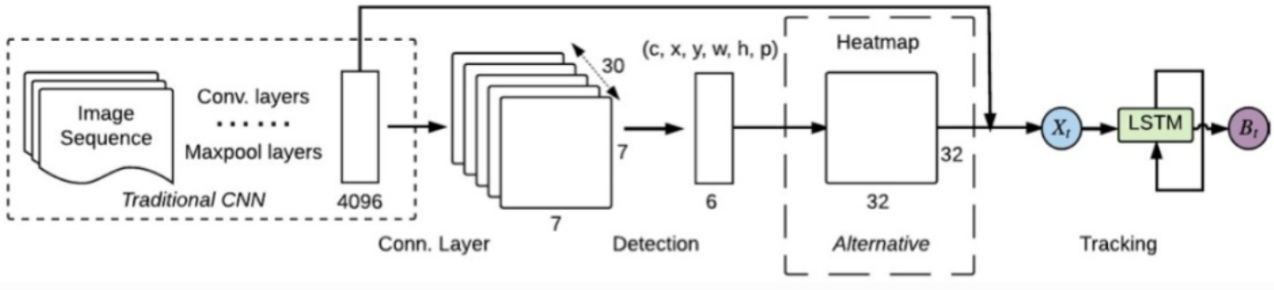
- not as fast as GOTURN;
- strong assumptions on the temporal prior

Single Target Tracking (ROLO)

- CNN for appearance + LSTM for motion



LSTM receives the heatmap for the object's position and the 4096 descriptor of the image

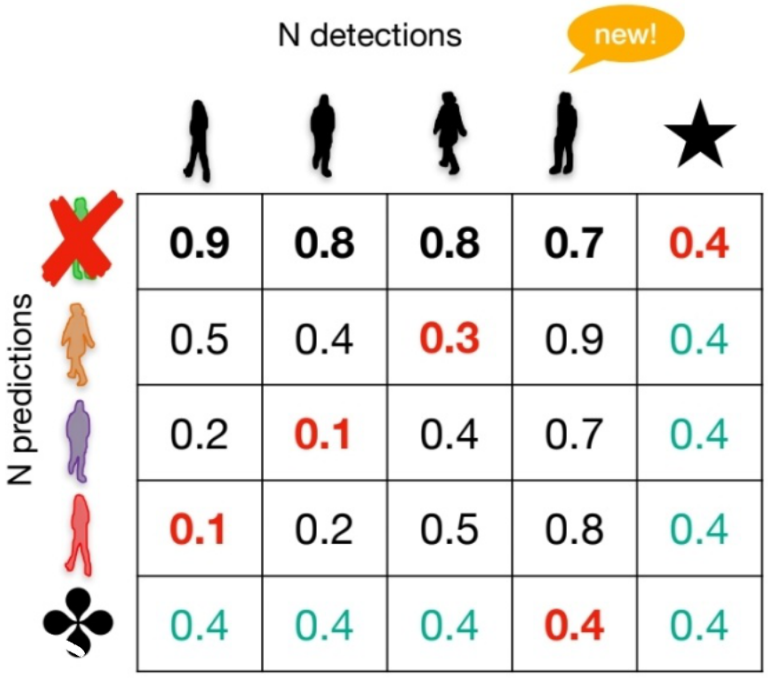


Multiple Object Tracking

Bipartite matching

- What happens if no prediction is suitable?
  - e.g. the object leaves the frame.
- Solution: pseudo node
  - its value will define a threshold
  - we may need to balance it out

- Remove tracks: ★
- New tracks: ✤



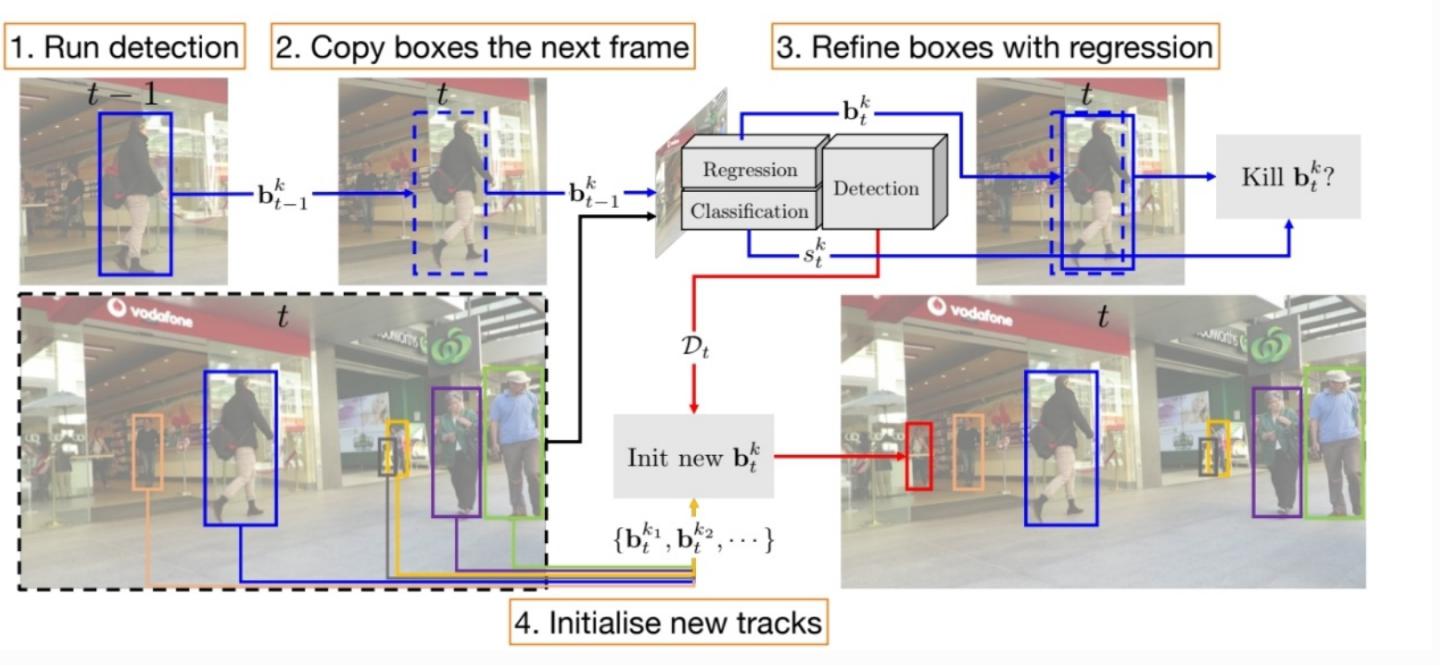
✕	**0.9**	**0.8**	**0.8**	**0.7**	**0.4**
	0.5	0.4	**0.3**	0.9	0.4
	0.2	**0.1**	0.4	0.7	0.4
	0.1	0.2	0.5	0.8	0.4
✤	0.4	0.4	0.4	**0.4**	0.4

1-) Define distances between boxes (1-IoU)

2-) Hungarian Algorith to find min cost.

3-) Supply min Threhod for distance

2-) Tracktor



Advantages

- We can use object detectors power
- Work well even if trained on still images since the regression head is agnostic to object ID and category

Disadvantages

- Work bad when motion is large (Fast object) (Low camera rate)
- Confusion in crowded places
- Temporary occlusions (track is killed)

3-) Metric Learning for re-identification (Re-ID)

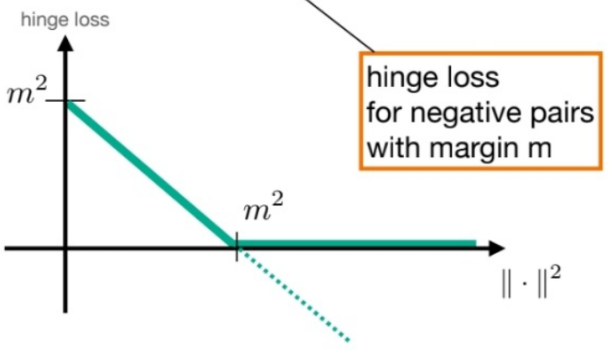
Hinge Loss

Unbounded loss for positive pairs, bounded for negative pairs:

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

1 if (A,B) is a positive pair
0 otherwise

L2 distance



hinge loss for negative pairs with margin m

Metric learning: triplet loss

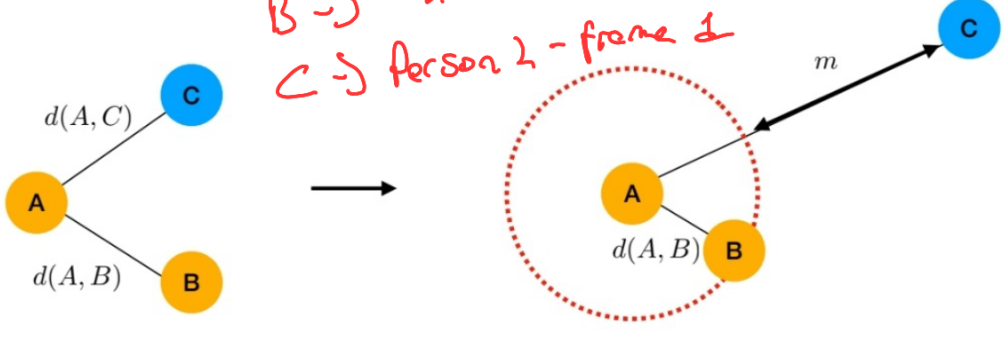
Should be min

Should be max

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

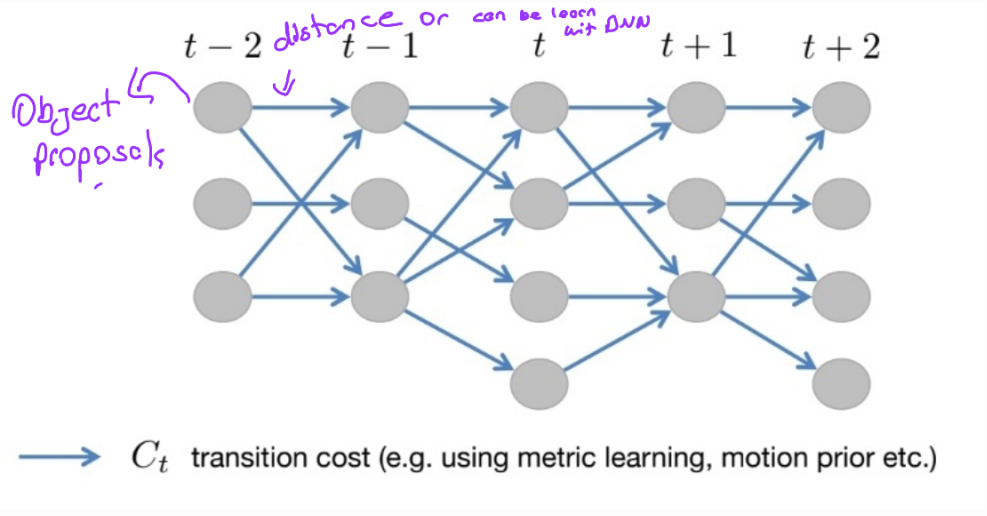
A -> Person 1 - frame 1
B -> " " 2
C -> Person 2 - frame 1

Intuitive idea:

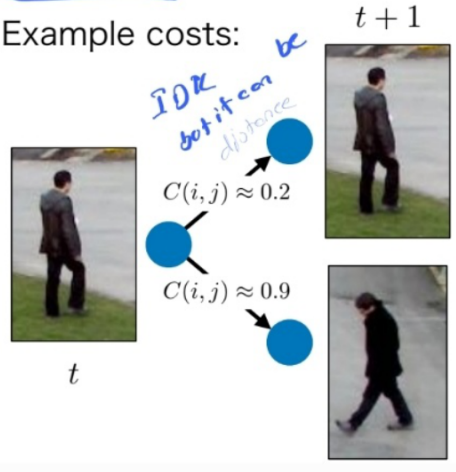


OFFLINE TRACKING

1. Graph-based MOT (Multi Object Tracking)



- Minimising the cost:
- Example costs:



$$f^* = \arg\min_f \sum_{i,j} C(i,j) f(i,j)$$

Cost of association of nodes

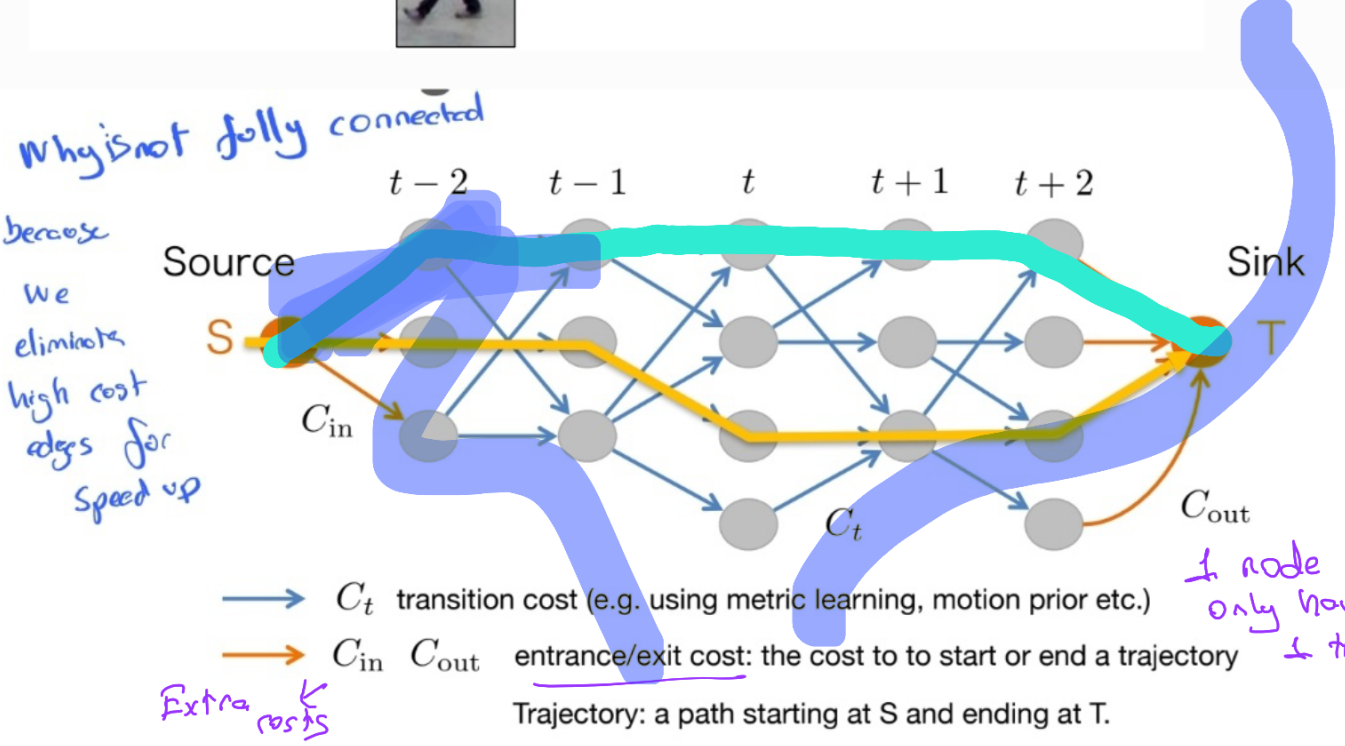
Disjoint set of trajectories

Indicator {0,1}

We are trying to find maximum flow with minimum cost.

Why is not fully connected

because we eliminate high cost edges for speed up

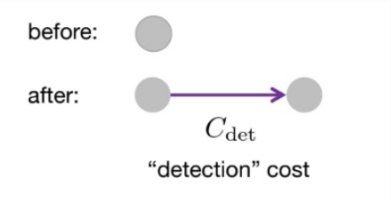


Extra costs

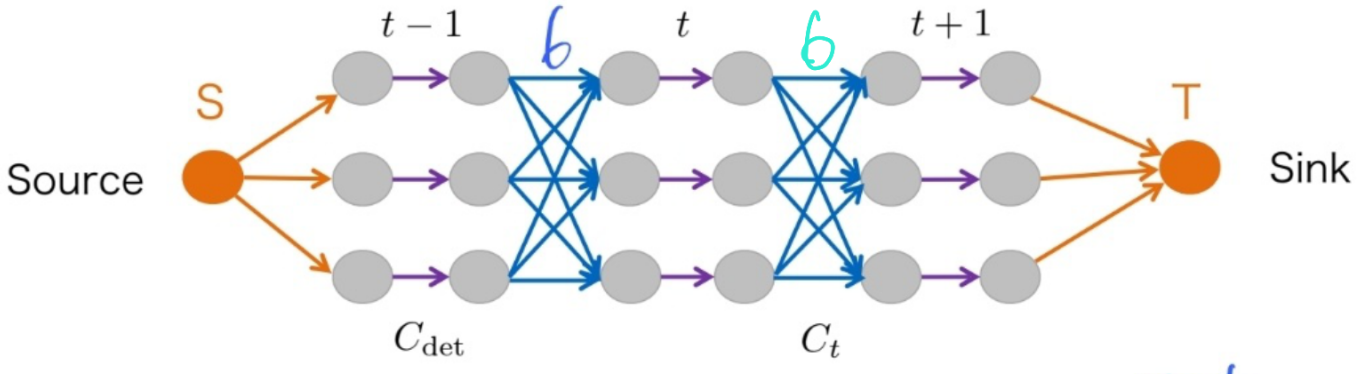
1 node only have 1 trajectory

Trajectory is a path from state "t" to "t+n"

How we get detection confidence?

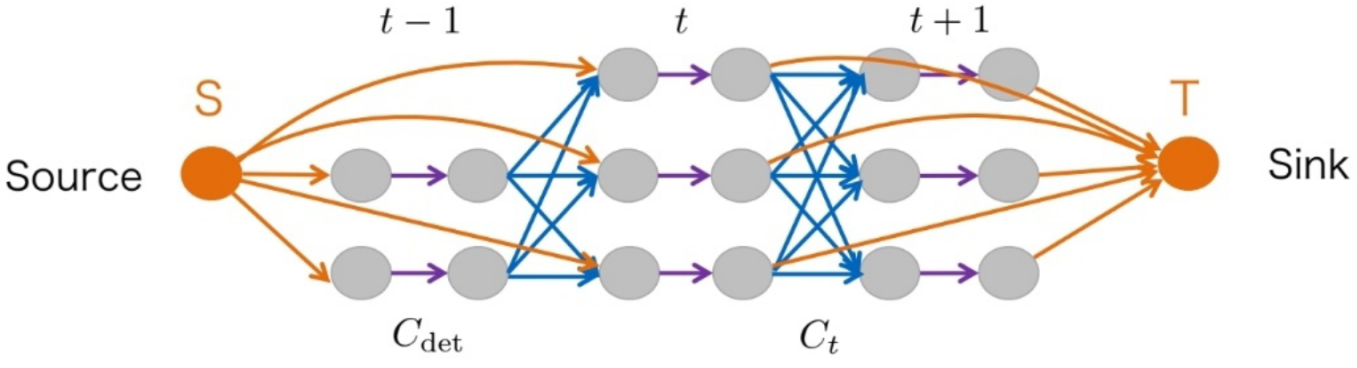


Split each node to 2. The edge is your detection cost.



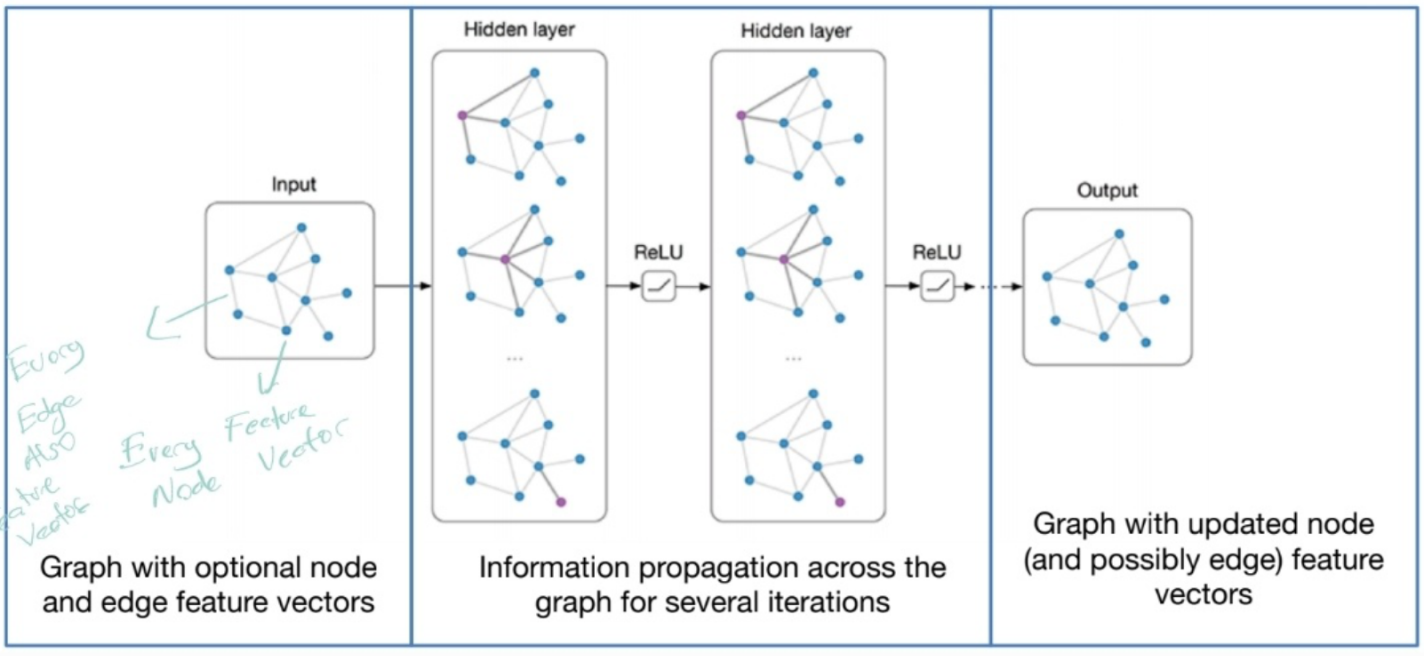
- The graph compactly encodes our problem.
  - QUIZ: How many trajectories (full-length) are possible? $31 = 6^5$ $\Rightarrow 6 \cdot 6 = 36$
- But... detections are not perfect.

What if an object only occurs in second frame? We will add source and sink connections.



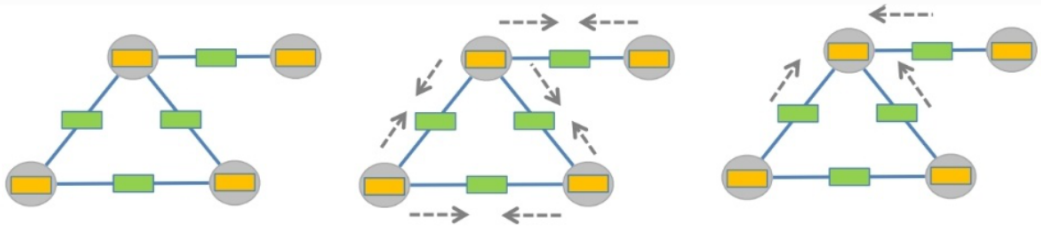
- Solution: Connect all nodes (detections) to entrance/exit nodes:

2. Message Passing Networks



Every Edge also Feature Vector

Every Node Feature Vector



The propagation consist 2 methods

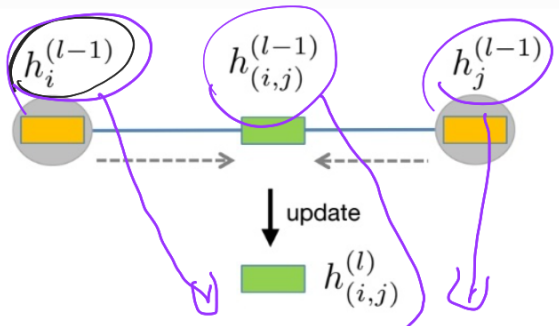
- Node to edge
- Edge to node

- We can divide the propagation process in two steps:
  - 'node-to-edge' and 'edge-to-node' updates.
- Alternate these two updates (message passing)

→ encodes context into embeddings.

Node embeddings

Edge embeddings



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

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

Use the updated edge embeddings to update nodes:

After a round of edge updates, each edge embedding contains information about its pair of incident nodes.

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

Edge to Node

Node to edge

So we have problem in Node to edge a node can have arbitary number edges. We need to aggregate with permutation-invariant(sroden bagimsiz) function.

Edge-to-node updates

- Define a permutation-invariant aggregation function:

$$\Phi^{(l)}(i) := \Phi\left(\{h^{(l)}(i,j)\}_{j \in Ne(i)}\right)$$

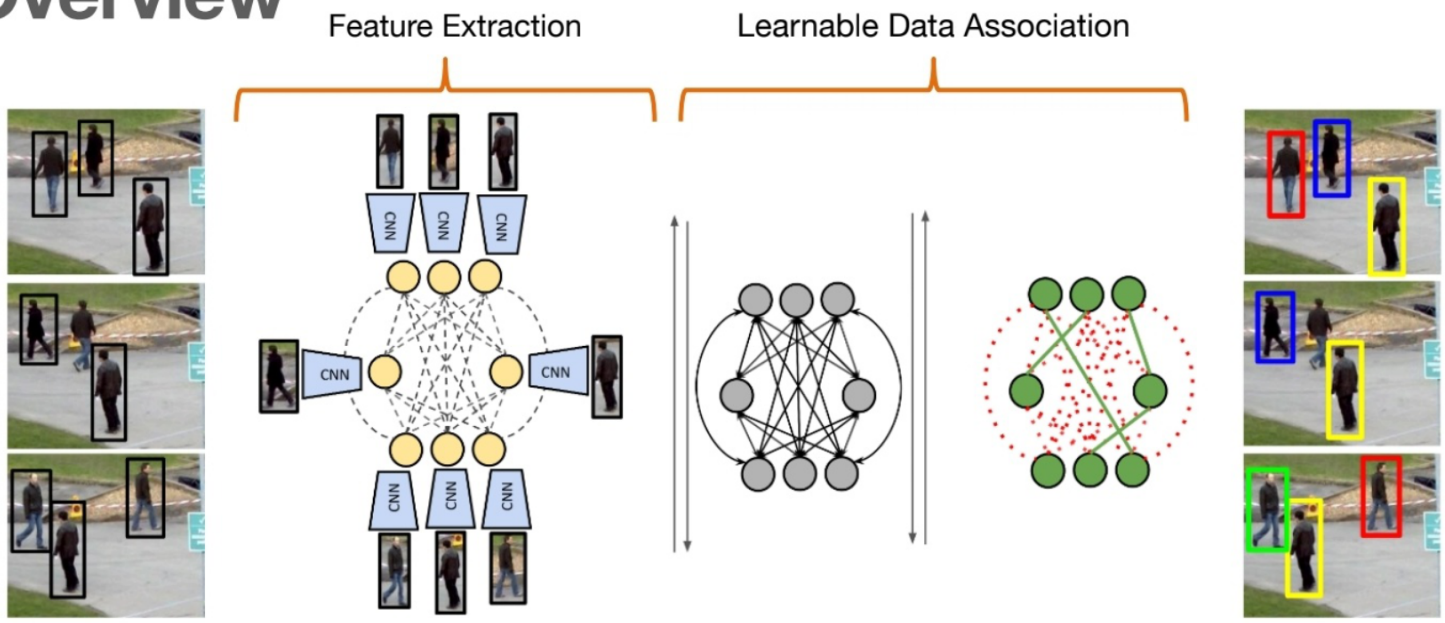
The input is a **set** of embeddings from incident edges

what does it mean?

- What are examples permutation-invariant functions? (QUIZ)
  - sum/mean, max/min (commutative operations).

How can we apply it into MIOT?

Overview



Brasó and Leal-Taixé. "Learning a Neural Solver for Multiple Object Tracking" (2019).

Encode appearance and scene geometry cues into node and edge embeddings

Propagate cues across the entire graph with neural message passing

Learn to directly predict solutions of the Min-Cost Flow problem by classifying edge embeddings

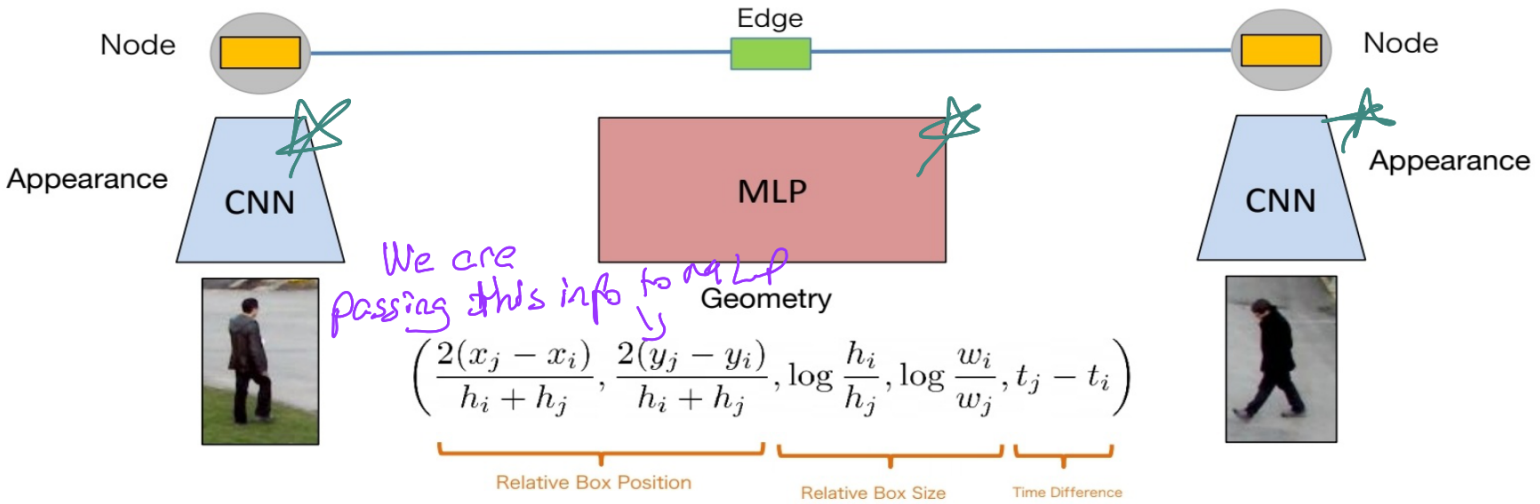
We are creating a node for each object proposal by giving them image CNN

Connect all nodes

Feature encoding

We have only 1 CNN and 1 MLP

- Appearance and geometry encodings

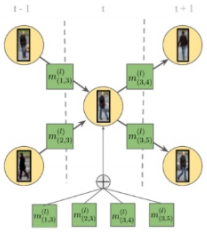


Temporal causality

- Recall edge-to-node aggregation:

$$\Phi^{(l)}(i) := \Phi\Big(\{h^{(l)}(i,j)\}_{j \in Ne(i)}\Big)$$

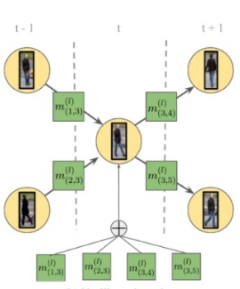
- Flow conservation at a node:
  - at most 1 connection to past
  - at most 1 connection to future



- Solution: decouple incident from outgoing edges

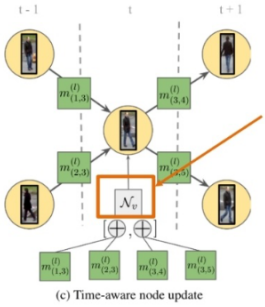
→ So we need to follow flow

Time-aware Message Passing



(b) Vanilla node update

All node embeddings are aggregated at once



(c) Time-aware node update

Aggregation of nodes is separated between past / future frames

Brasó and Leal-Taixé. "Learning a Neural Solver for Multiple Object Tracking" (2019).

→ That is why we seperate past / future when edge-to-node.

- 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

Evaluation metrics

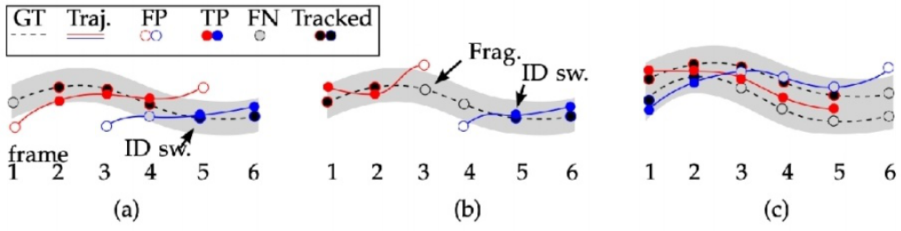
- Compute a set of measures per frame
  - Perform matching between predictions and ground truth (we will use exactly the same Hungarian algorithm)
  - FP = False positives
  - FN = False negatives (missing detections)
  - IDsw: identity switches

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

- How do we compute ID switches?



(a) An ID switch is counted because the ground truth track is assigned first to red, then to blue.

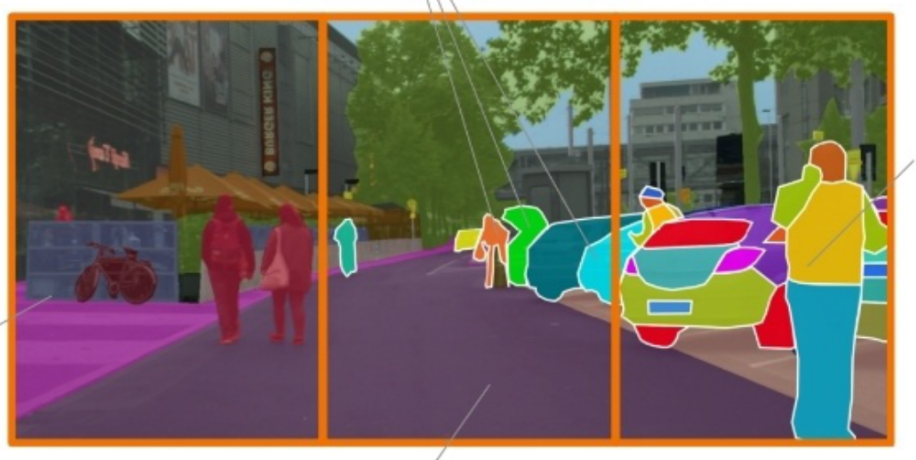
(b) Count both an ID switch (red and blue both assigned to the same ground truth), but also a fragmentation (Frag) because the ground truth coverage was cut.

(c) Identity is preserved. If two trajectories overlap with a ground truth trajectory (within a threshold), the one that forces least ID switches is chosen (the red one).

Semantic Segmentation

Label every pixel with semantic category

Example:

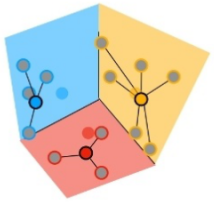


Segmentation as Clustering

Segmentation as clustering

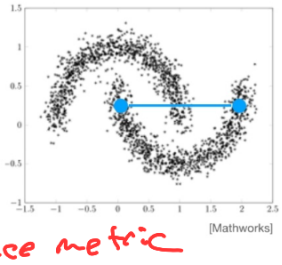
- K-means

1. Initialise (randomly) K cluster centres
2. Assign points to clusters
   using a distance metric
3. Update centres by averaging the points in the cluster
4. Repeat 2 and 3 until convergence



- K-means
- does it always work?

No, We need to define new distance metric

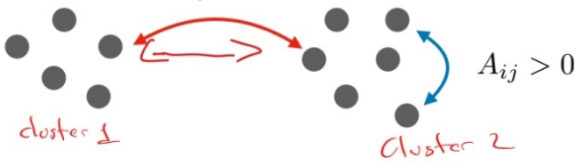


- Minimising the total within-cluster distance may not be ideal...

Spectral clustering

-> We can also use Spectral Clustering

Example



- We can represent these n points as an $n \times K$ matrix ($K = 2$): Number of cluster

$$\begin{matrix} \text{points in cluster 1} \\ \text{points in cluster 2} \end{matrix} \begin{cases} \\ \\ \\ \\ \\ \end{cases} \begin{pmatrix} 1 & 0 \\ 1 & 0 \\ \vdots & \vdots \\ 0 & 1 \\ 0 & 1 \end{pmatrix}$$

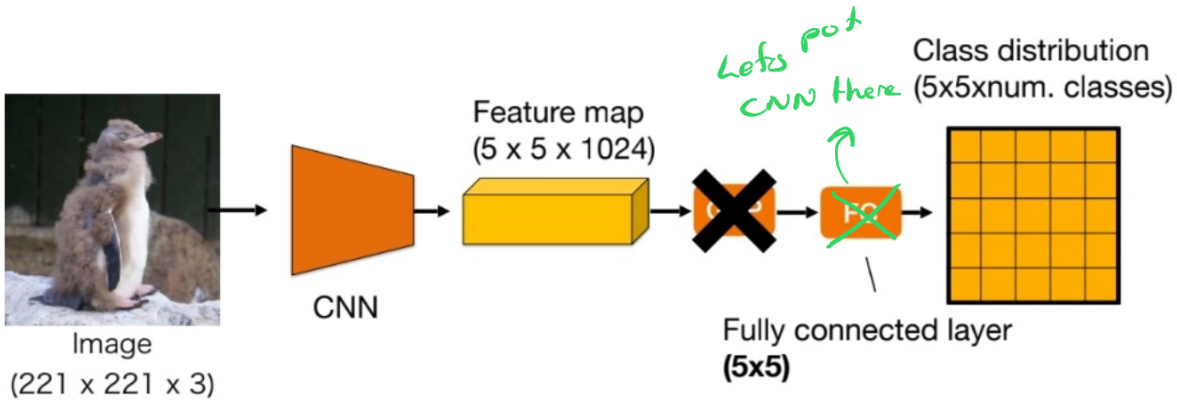
is in cluster 1 / is in cluster 2

- The columns are eigenvectors of L corresponding to eigenvalue 0;
- Clustering these points in the K-dimensional space with K-means would now be more meaningful!

- All points are node.

So we can use CRF to train something but not important. We have DL.

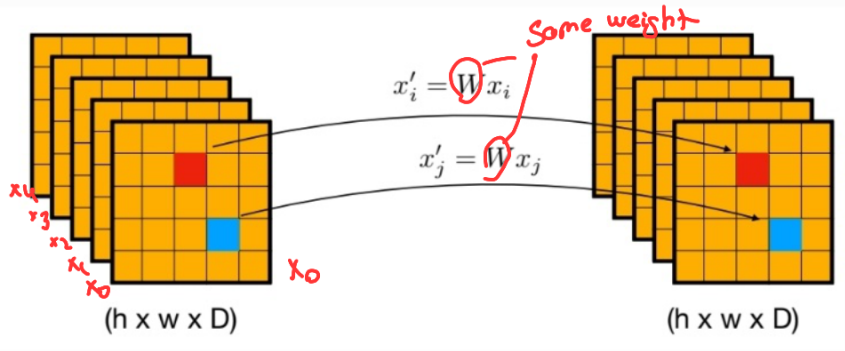
- Let us remove GAP:



- parameter increase; fixed input size only; no translation invariance.

1x1 Conv

- Same with applying shared fully connected Layer to every pixel.
- Keeps dimensions
- Why we use it? Well we keep dimensions but in practise we have multi dimensions. So it is a linear operator multiplying each value with a same weight.



Okay, How to produce segmentation result fore each image pixel ?

Keep original image resolution in the encoder

If you decrease stride, Acc ↑

Problem 2 → ! • But Expensive

Problem 1 → • Also receptive field size in decrease

if you lowered stride, you dont have possibilty to connect far pixels

Solution 1 : Dilated Convolations

- Receptive field increase
- Number of parameters same

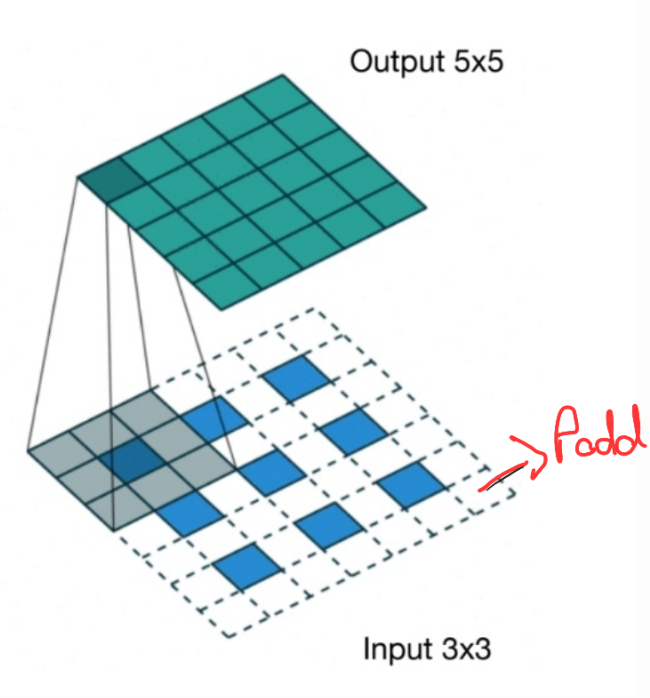
```
Original Matrix Shape :  (9, 9)
[[0. 0. 0. 0. 0. 0. 0. 0. 0.]
 [0. 0. 0. 0. 0. 0. 0. 0. 0.]
 [0. 0. 0. 0. 0. 0. 0. 0. 0.]
 [0. 0. 0. 0. 1. 1. 1. 0. 0.]
 [0. 0. 0. 0. 1. 1. 1. 0. 0.]
 [0. 0. 0. 0. 1. 1. 1. 0. 0.]
 [0. 0. 0. 0. 0. 0. 0. 0. 0.]
 [0. 0. 0. 0. 0. 0. 0. 0. 0.]
 [0. 0. 0. 0. 0. 0. 0. 0. 0.]]


Original kernel Shape :  (3, 3)
[[1 2 3]
 [4 5 6]
 [7 8 9]]


Dilated kernel Shape :  (7, 7)
[[1 0 0 2 0 0 3]
 [0 0 0 0 0 0 0]
 [0 0 0 0 0 0 0]
 [4 0 0 5 0 0 6]
 [0 0 0 0 0 0 0]
 [0 0 0 0 0 0 0]
 [7 0 0 8 0 0 9]]


DILATED CONVOLUTION RESULTS [Dilation Factor = 3]
Numpy Results Shape:  (3, 3)
[[4. 5. 5.]
 [4. 5. 5.]
 [4. 5. 5.]]
```

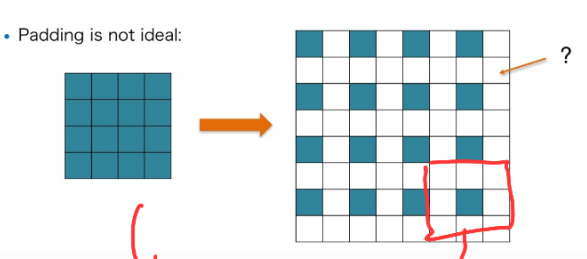
→ Normal Conv

→ Dilated Conv with factor=3

Solution 2 : Upsampling

We can use conv Layer for increasing dim.





Because 9 pixels take value from 1 feature

We can use interpolation instead of padding.

Interpolation is still not enough

We can use skip connections.

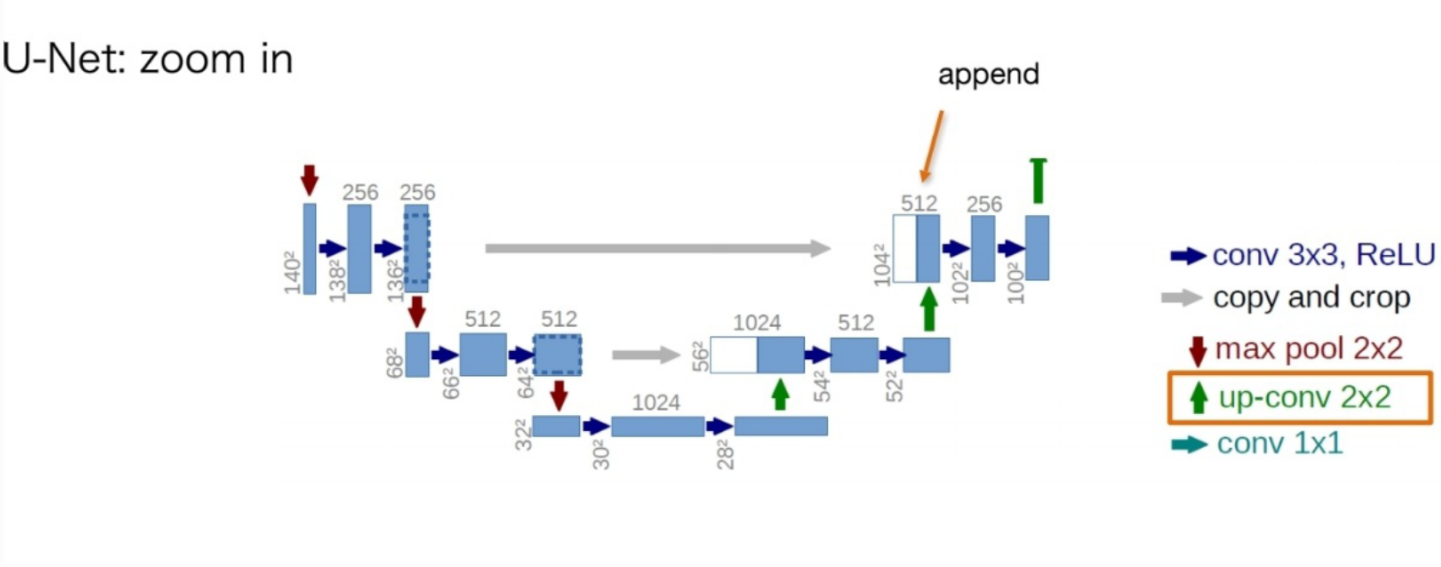
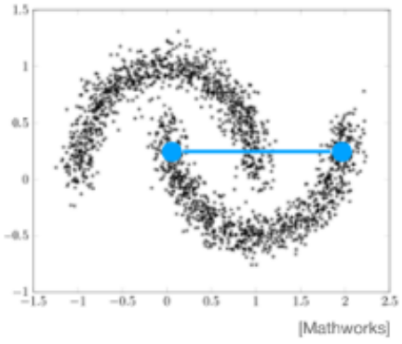


Image Segmentation

→ K means

Segmentation as clustering

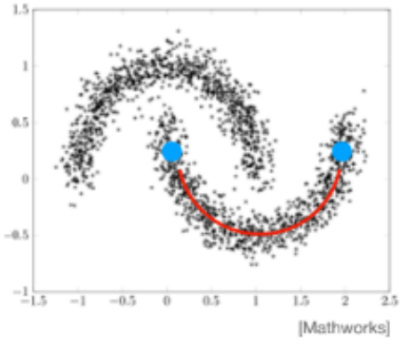
- K-means
  - does it always work?



- Minimising the total within-cluster distance may not be ideal...

Segmentation as clustering

- K-means
  - does it always work?



- Minimising the total within-cluster distance may not be ideal...
  - ...unless we re-define the distance metric (recall metric learning)

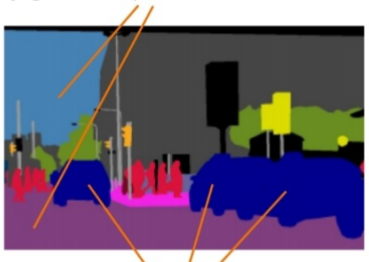
- Spectral Clustering

- Energy Based Methods

- DeeplabV3 → 1x1 conv, Upsample.

Instance Segmentation

Label every pixel, including the background (sky, grass, road)

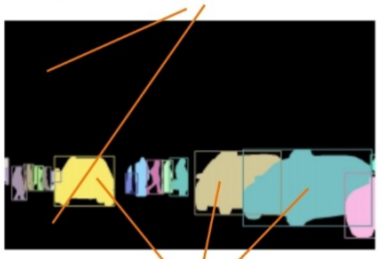


Does not differentiate between the pixels from objects (instances) of the same class

↓

Semantic

Do not label pixels coming from uncountable objects ("stuff"), e.g. "sky", "grass", "road"



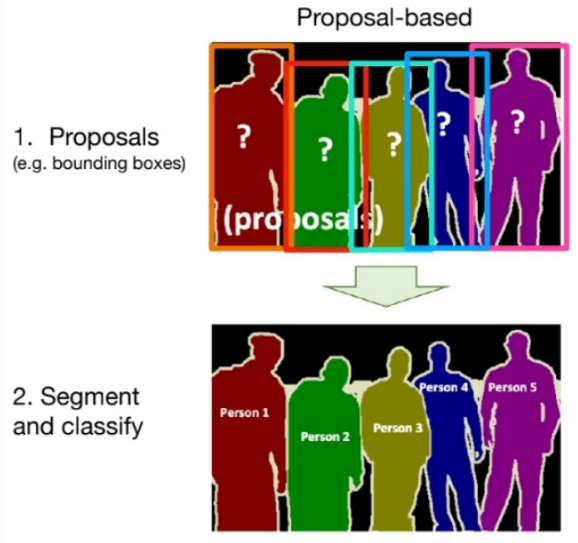
Differentiates between the pixels coming from instances of the same class

↓

Instance

Methods

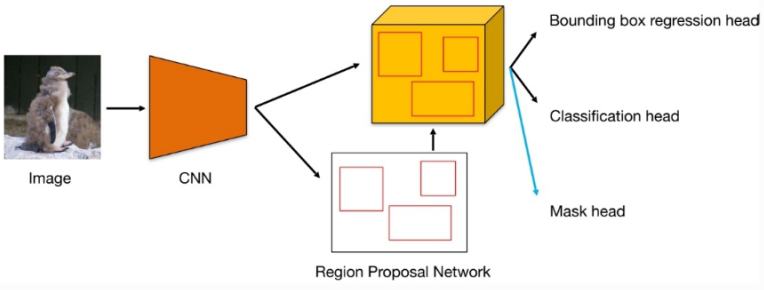
1. Proposal-based

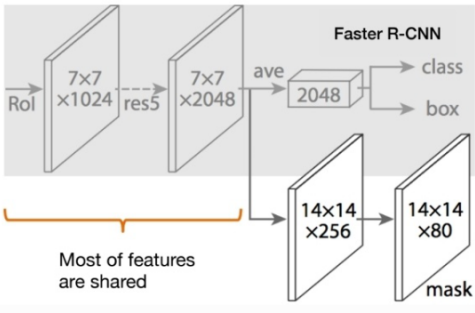


Extract Proposal as we do in prev chapters

2. Mask R-CNN

Basically, we add maskhead to Faster R-CNN.



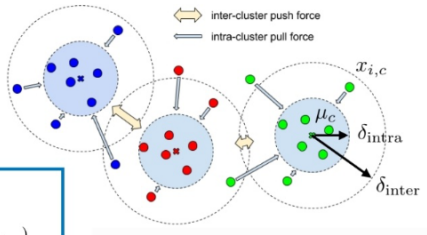


1. Instances via clustering

- Instance as a cluster in feature space:

Loss:

- Recall metric learning and hinge loss:



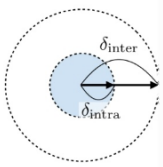
Intra-cluster term:

$$\max(0, \|\mu_c - x_{i,c}\|_2 - \delta_{\text{intra}})$$

Inter-cluster term: We don't want to intersect classes.

$$\max(0, 2\delta_{\text{inter}} - \|\mu_c - \mu_{c'}\|_2)$$

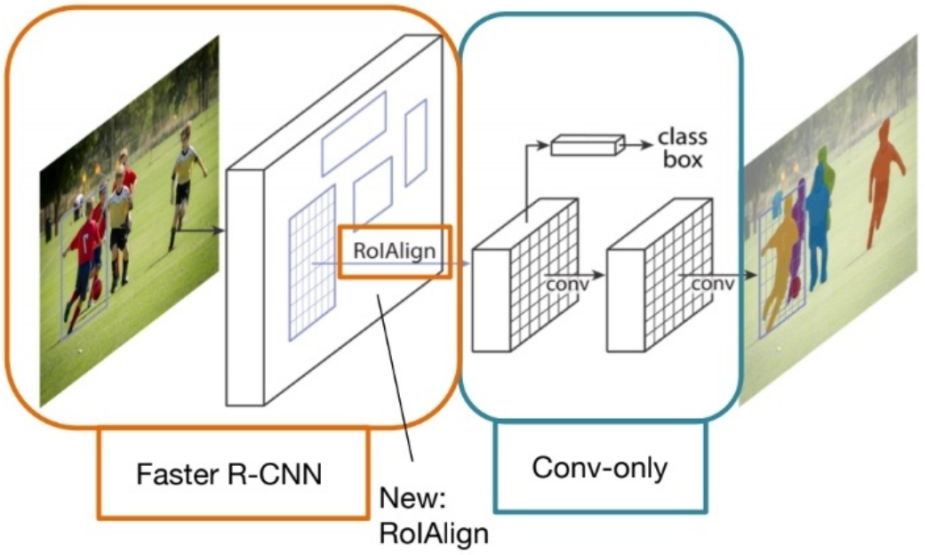
- We can set: $\delta_{\text{inter}} > 2\delta_{\text{intra}}$
- After training, any embedding is within distance $2\delta_{\text{intra}}$ to all embeddings of the same cluster (ideally).



Test-time strategy:

1. Select unlabelled pixel.
2. Find (unlabelled) neighbours.
3. Assign the pixels to a new cluster.
4. Repeat until all pixels are labelled.

- Faster R-CNN + mask head for segmentation



+ mask loss:
cross-entropy per pixel

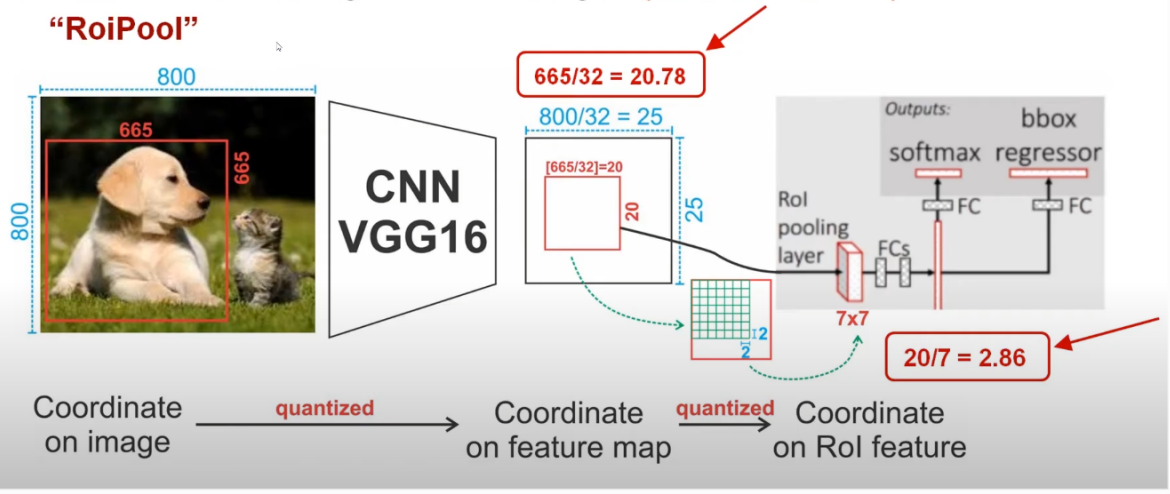
He at al. "Mask R-CNN" ICCV 2017

Also we are using ROI Align instead of ROI Pooling.

What was ROI Pooling:

1. We have 800 x 800 input image and we will find 665 x 665 object proposal after CNN.
2. VGG16 lowered our input image by scale 32. So when we divide our proposal by 32 we get 20.78 and we have to round these value to int. Actually we lose many pixels in here. $665 \times 665 \neq 20.32 \times 20.32$
   640×640
3. Also when applying 7x7 pooling we can not process some values too.
   $20/7 = 2.86$
   is rounded to 2

a. "RoiPool" is changed to "RoiAlign" (modification 1)



There is two main reason of not using ROI Pooling

Two quantisations:

1. Bounding box alignment
2. Bin alignment

Pooling within each bin

(max or average)

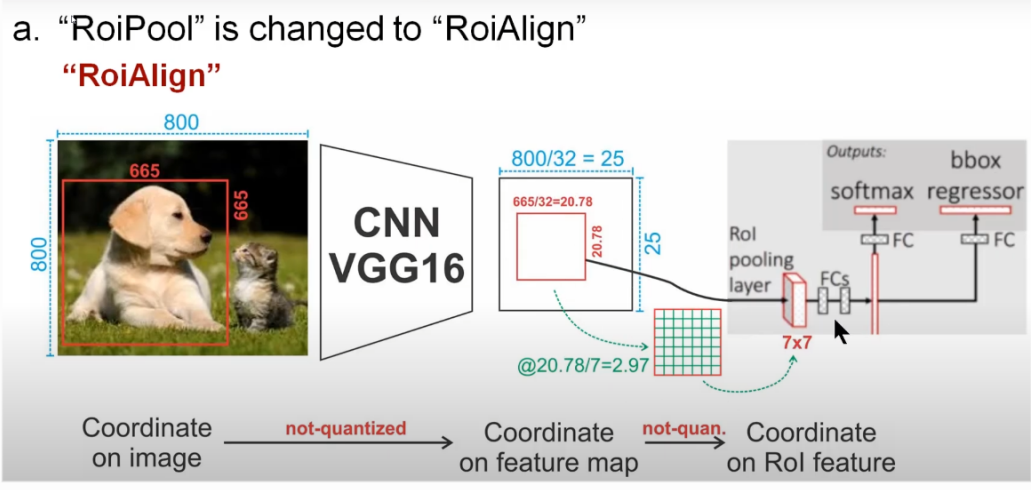
Works well for object detection

(we use this for classification only)

Not for segmentation

So how ROI Align Fix this

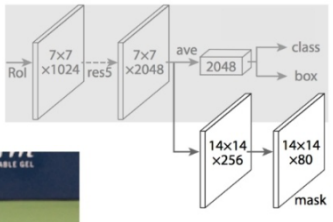
It calculates object proposal with floating points and it divide into 4 bins. For each bin it calculates 4 sampling point. and compute feature values with bilinear interpolation.

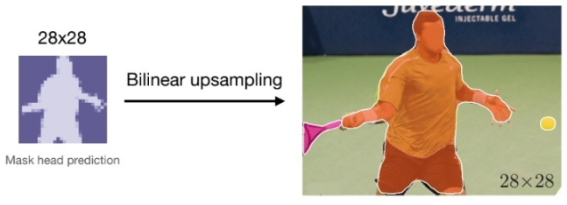


Still we have a problem our mask are in low res.

Mask R-CNN + PontRend

- Problem: low mask resolution
- Example:

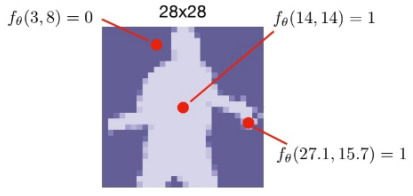




Instead we can train a function that takes floating points

Implicit neural representation

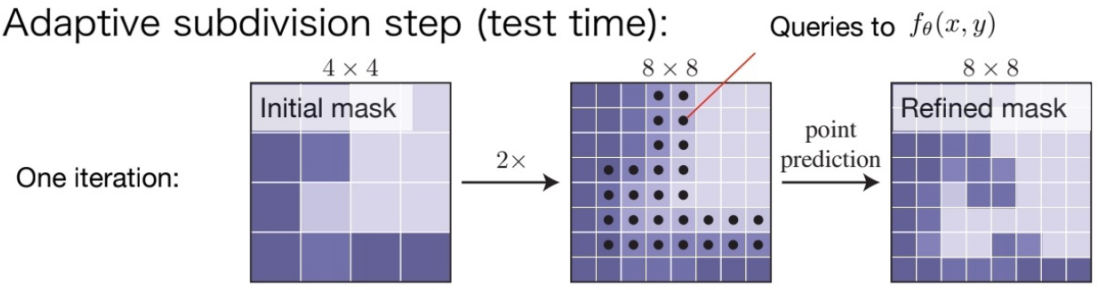
- Mask head is an example of mapping a discrete signal representation (e.g. a pixel) to a desired value (e.g. binary mask);
- Instead, we parameterise the mask as a continuous function that maps the signal domain (e.g. (x,y) coordinate):



- $f_\theta(x,y)$ is an example of an implicit (neural) representation;
- Why is it useful here?
- We can query fractional coordinates.

We have 4x4 mask output. We divide it sub cells and ask it to our function

- Idea:
  - Train implicit mask representation by focusing on the boundaries;
  - Test time: Use the learned implicit mapping to refine boundaries.
- Adaptive subdivision step (test time):



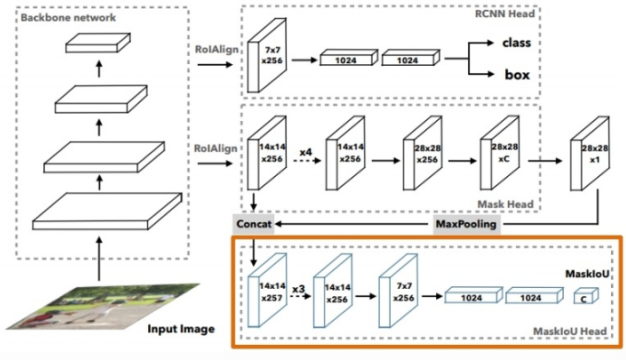
Masks Quality Detection

We can not use bounding box regression head directly because it indicates object score.

So we add mask regression head for it.

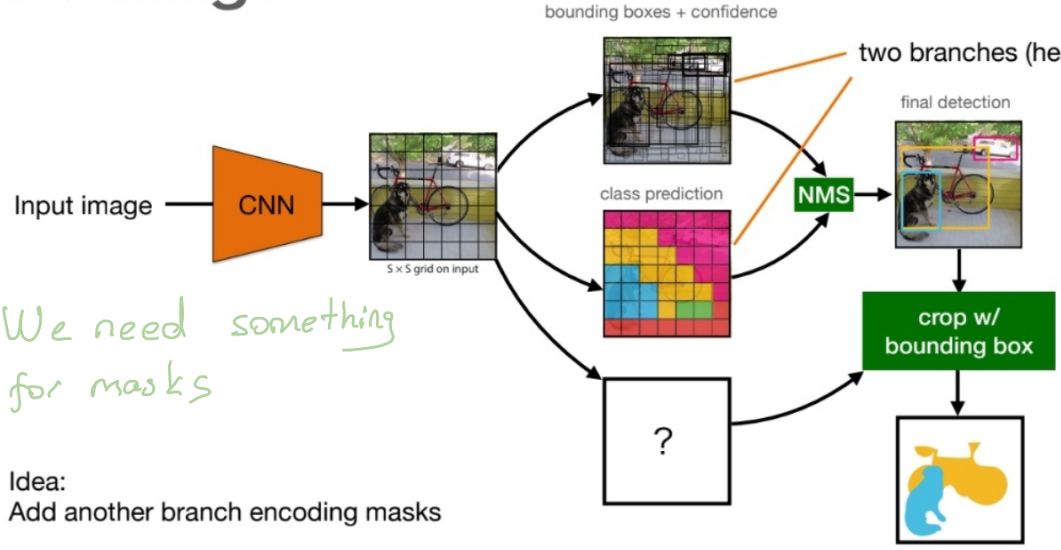
Mask Score R-CNN idea: Learn to predict mask IoU w.r.t. ground truth

- Add a regression head:
  - Use RPN proposal for training.
  - To compute IoU, binarise prediction with a threshold.
  - Uses L2 loss.



Instance Segmentations with One-stage

One-stage IS

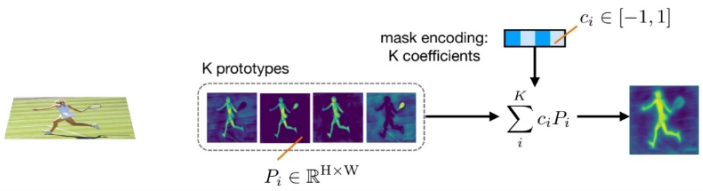


We need something for masks

Idea:
Add another branch encoding masks

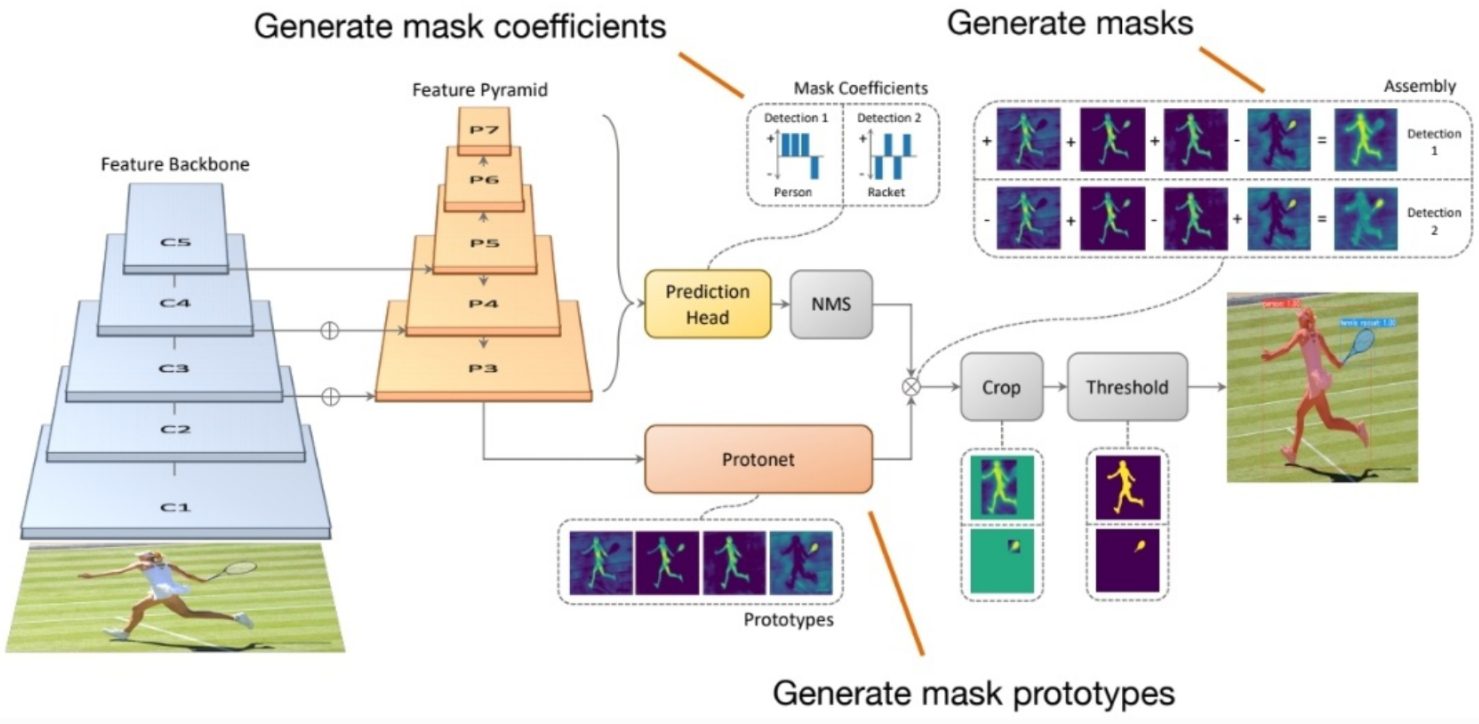
YOLACT

- How to encode masks?
  - YOLACT: A linear combination of "prototypes":



-> TBH, I didn't understand.

YOLACT: Overview

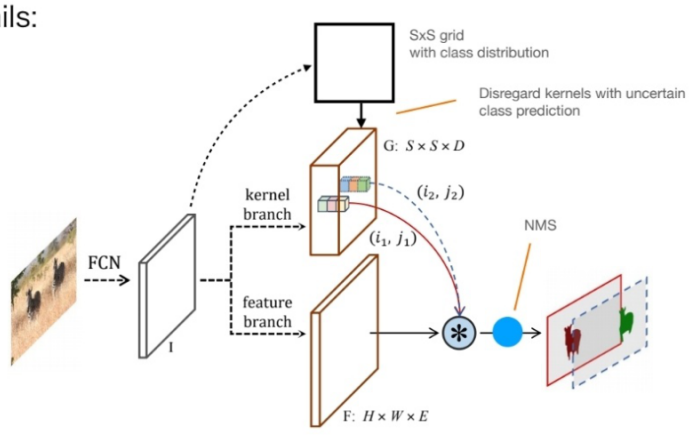


Quality of masks is even better than two stage detectors

Because we predicted boxes in object proposal in mask R-CNN, but in here we predict in whole image.

SOLOv2

- A few details:



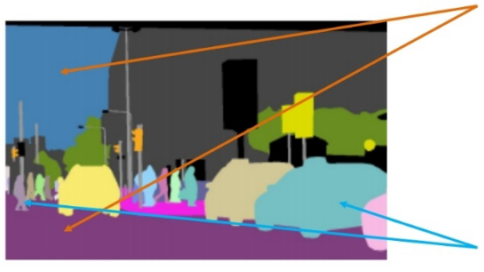
Summary

- One-stage and two-stage instance segmentation methods offer accuracy vs. efficiency trade-off.
- Similar to our conclusions about object detectors:
  - Two-stage methods are more accurate (robust to scale variation), but less efficient;
  - One-stage methods are faster and have competitive accuracy.
    - Accurate segmentation of large-scale objects.

Panoptic Segmentation

Semantic + Instance = Panoptic

Panoptic segmentation



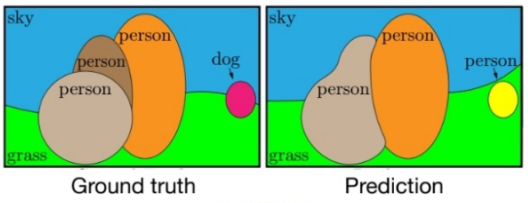
It gives labels to uncountable objects called "stuff" (sky, road, etc), similar to FCN-like networks.

It differentiates between pixels coming from different instances of the same class (countable objects) called "things" (cars, pedestrians, etc).

The metric:-

Panoptic quality (PQ)

1. Establish matches between the ground-truth and predictions;
2. Count TPs, FPs and FNs;
3. Compute PQ for each class:



Person — TP: { }; FN: { }; FP: { }

$$PQ = \frac{\sum_{(p,g)\in TP} IoU(p,g)}{|TP|} \cdot \frac{|TP|}{|TP| + \frac{1}{2}|FP| + \frac{1}{2}|FN|}$$

SQ RQ

...and then average.

- To compute PQ we specify that a prediction and a ground truth match only if their IoU is greater than 0.5.
- This match, if found, is unique.

SQ = Segmentation Quality ⇒ · Average mask IOU for TP
· Measures pixel-level accuracy of predicting masks

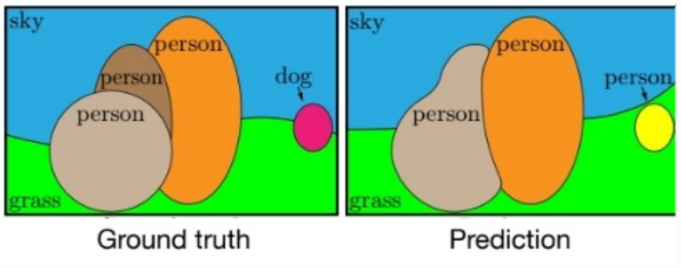
RQ = Recognition Quality ⇒ Object Level Accuracy (F1 Score)

Observations:

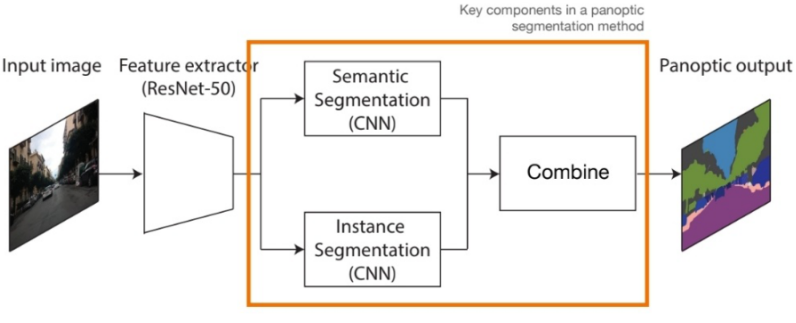
1. PQ, RQ, SQ ∈ [0,1]
2. For missing objects (Dog object)

For Dog class = FN ↑ PQ ↓
For Person " = FP ↑ PQ ↓

Use "unkown" class for not effect PQ



- Typical architecture:



Merge things and stuff:
- NMS on instances.
- Resolve stuff-things conflicts in favour of things. (Why?)
- Remove any stuff regions labelled "other" or with a small area.

Panoptic FPN

- Loss function:

Trade-off hyperparameter

$$L = L_c + L_b + L_m + \lambda_s L_s$$

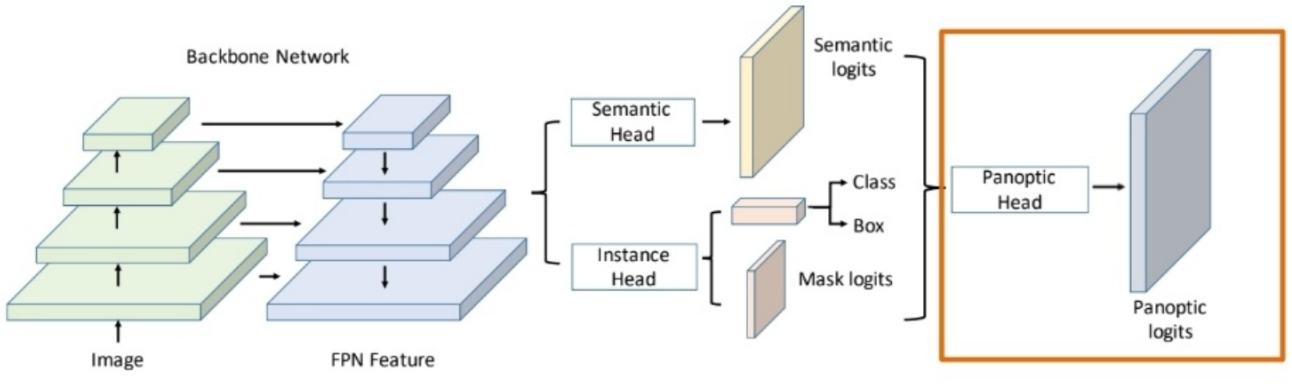
Instance segmentation branch loss

Semantic segmentation branch

- Remark: Training with multiple loss terms ("multi-task learning") can be challenging, as different loss terms may "compete" for desired feature representation.

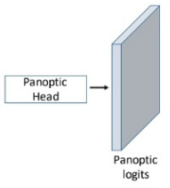
UPS NET

UPSNet



- What's different?

- Main idea: create a single logit tensor for stuff and things
  - number of stuff categories: N_{stuff}
  - number of instances (variable per image): N_{inst}
  - add 1 "unknown" category
  - panoptic tensor: $(N_{\text{stuff}} + N_{\text{inst}} + 1) \times H \times W$

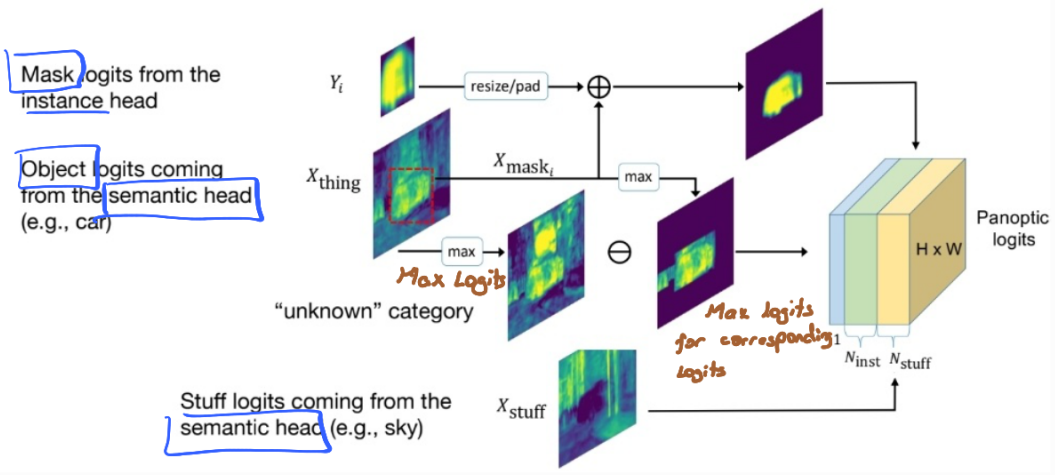


- Semantic segmentation head predicts both things and stuff classes.

Things -> In computer vision, the term things generally refers to objects that have properly defined geometry and are countable, like a person, cars, animals, etc.

Stuff -> Stuff is the term used to define objects that don't have proper geometry but are heavily identified by the texture and material like the sky, road, water bodies, etc.

The panoptic head



Advantage of panoptic logits:
- the channel argmax tells us if the pixel belongs to stuff, "unknown" or things;
- if the latter, which ID.

There is also new concept in UPSNET

Deformable conv

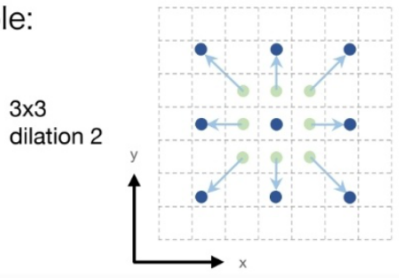
It is same logic with dilation but we generalize its parameters a bit more

Generalising dilation

- We can represent dilated convolution as parameterisation of a non-dilated convolutional kernel:

$$k'(x, y) := k(x + g_x(x, y), y + g_y(x, y))$$

- Example:



(g_x, g_y)

Can be fractional (use bilinear interpolation)

-1,1	0,1	1,1
-1,0	0,0	1,0
-1,-1	0,-1	1,-1

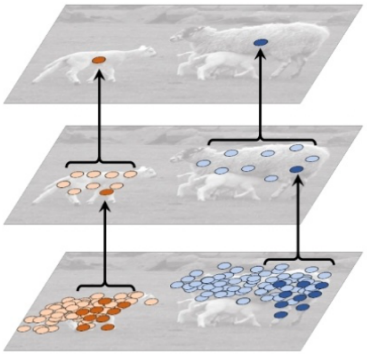
Dai et al., "Deformable Convolutional Networks" ICCV 2017.

-> So that we can take value randomly or different shape.

->Also these offsets Learnable.

Why is it useful?

- Consider non-rigid object recognition.
  - Regular convolution:
- Problem: When an object deforms, we need another kernel to work in that pose too.
- ... or deformable convolution:



UPSNET

Advantages

- Improved Accuracy over Panoptic FPN
  - Joint things and stuff training via Panoptic Head
  - Deformable Conv

Disadvantages

- Not Efficient
  - ROI Pooling

Video Object Segmentation

Challanges

- Strong viewpoint/apperence changes
- Occlusions
- Scale changes
- Illumination
- Shape

We need 2 model

Appearance Model
- Assumption : Constant Appearance
- Input: 1 frame
- Output: Segmentation Task

(optional) Motion Model
- Assumption: smooth displacement, brightness constancy
- Input: 2 frames
- output: motion

VOS: Tasks

"Semi-supervised" (one-shot) VOS

Inference time
- input: video + object mask in the first frame

-> Object mask given

"Unsupervised" (zero-shot) VOS

Inference time
- input: video (without any labels)

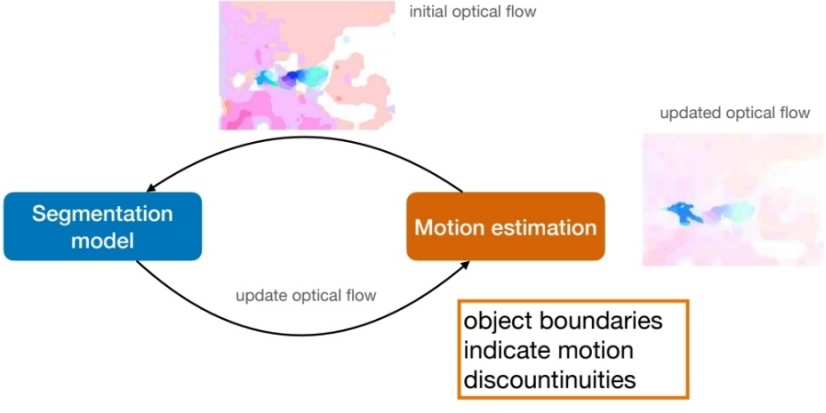
1. Identify objects of interest (e.g. using instance segmentation);
2. Establish temporal consistency.

Cons: Typically complex models (active research area).

Motion-based VOS

Can we do VOS with optical flow?

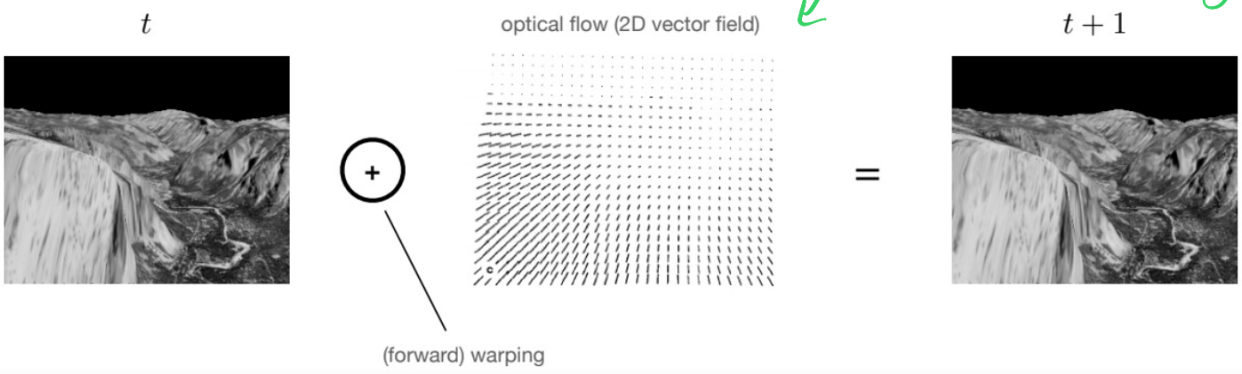
- Joint formulation of segmentation and optical flow estimation:



Optical flow

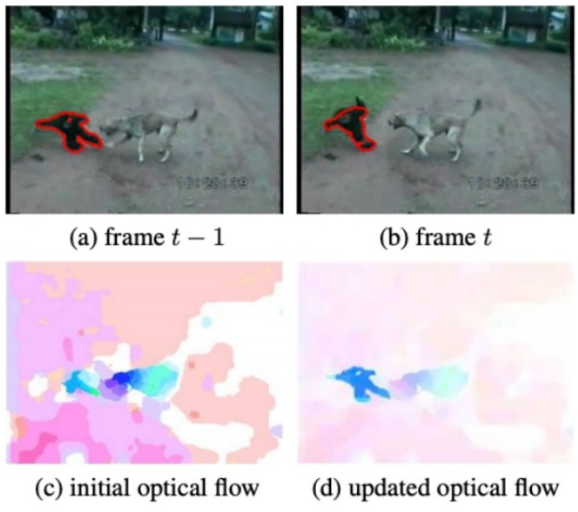
- Recall optical flow (from Computer Vision I):
  - a pattern of apparent motion (Lukas and Kanade, '81; Horn & Schunk '81);

if I move first image with this vectors, I would get



Can we do VOS with optical flow?

- Joint formulation:
  - iteratively improving segmentation and motion estimation.
- Slow to optimise:
  - runtime: up to 20s (excluding OF).
- Initialisation matters:
  - we need (somewhat) accurate initial optical flow.
- DL to the rescue?

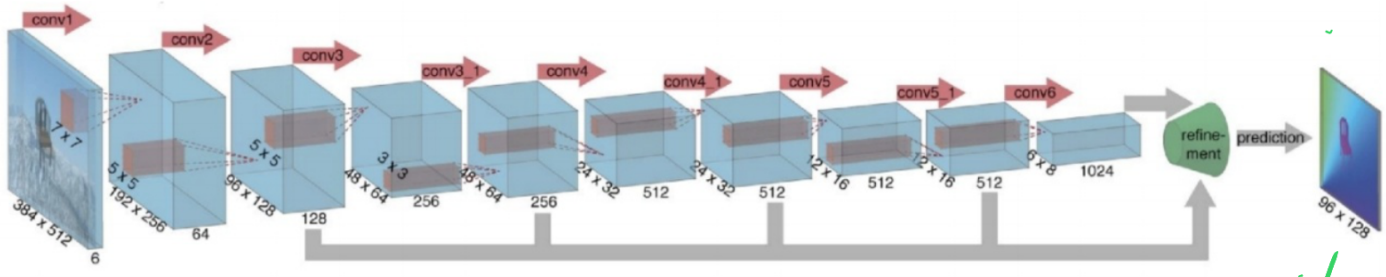

(a) frame $t-1$ (b) frame t

(c) initial optical flow (d) updated optical flow

Y.H. Tsai et al. "Video Segmentation via Object Flow". CVPR 2016

FlowNet: Architecture 1

- Stack both images → input is now 2 x RGB = 6 channels

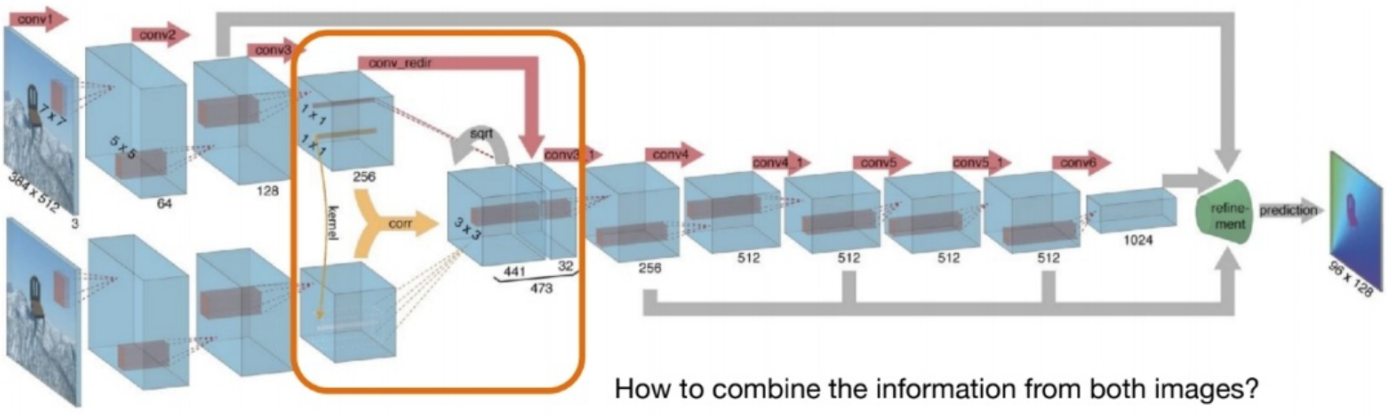


Vector Field

- Training with L2 loss from synthetic data

We can also use siamese architecture

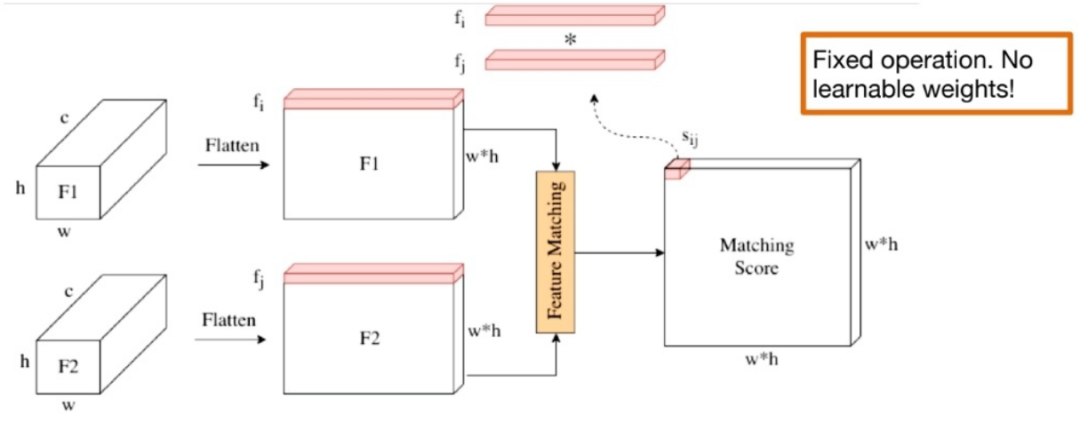
- Two key design choices



How to combine the information from both images?

Correlation layer

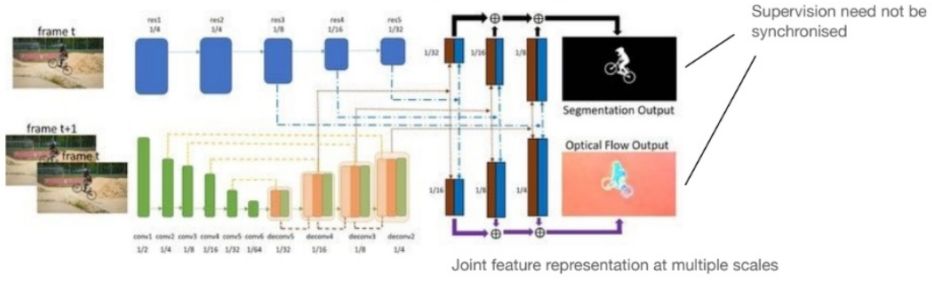
- The dot product measures similarity of two features



If we combine optical flow and object segment

SegFlow

- Joint estimation of optical flow and object segment:



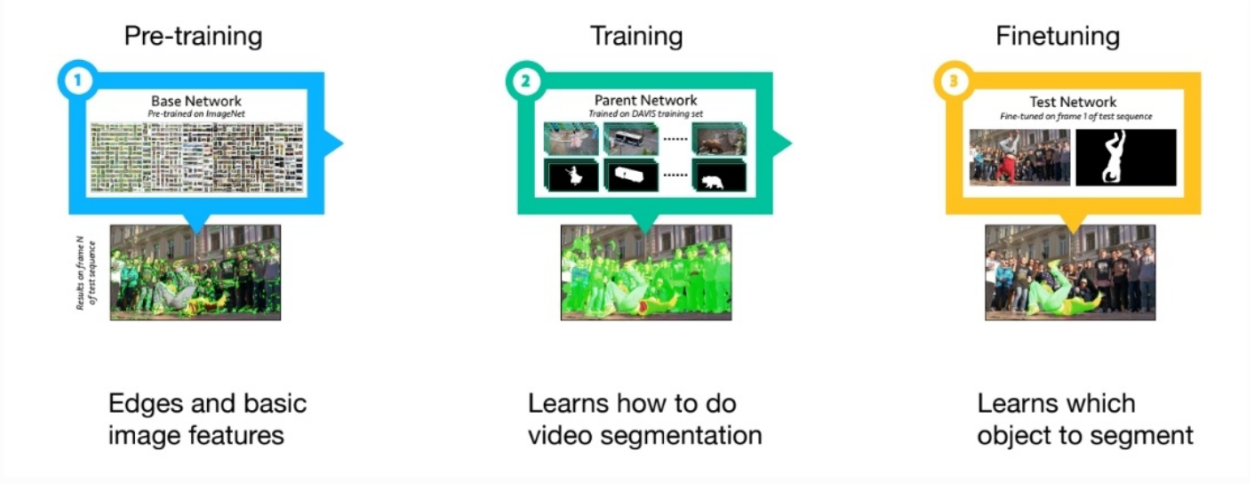
- Alternating optimisation:
  - fix one network to optimise the other.

Appearance-only VOS

Main idea:

- Train a segmentation model from available annotation (including the first frame);
- Apply the model to each frame independently;
- One-shot VOS (OSVOS): separate the training steps
- Pre-training for 'objectness'.
- First-frame adaptation to specific object-of-interest using fine-tuning.

OSVOS



It is a static model fails when background changes

- One-shot: learning to segment sequence from one example (the first frame).
- This happens in the fine-tuning step:
  - the model learns the appearance of the foreground object.
- After fine-tuning, each frame is processed independently → no temporal information.
- The fine-tuned parameters are discarded before finetuning for the next video.

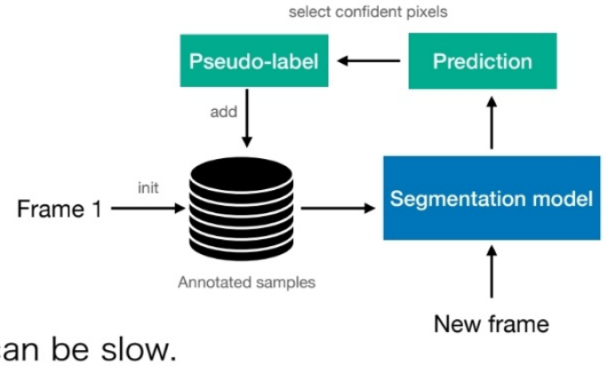
- Also fails when object apperence changes
- Also fails when camera position drifting

Online Adaption

OnVOS

OnAVOS: Online Adaptation

- Online adaptation: adapt model to appearance changes in every frame, not just the first frame.



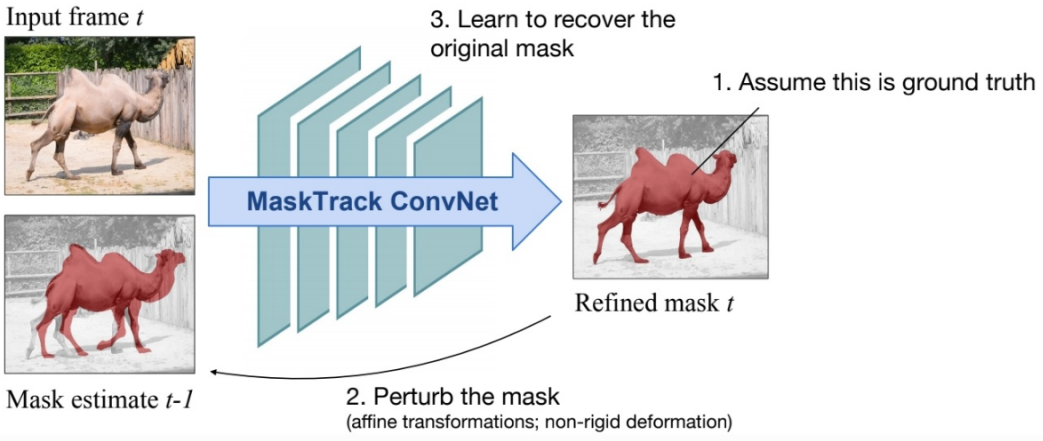
- Drawback: can be slow.

Issues

- OnAVOS is more accurate than One-Shot VOS;
- Instead of fine-tuning on a single sample, we fine-tune on a dynamic set of pseudo-labels;
- The pseudo-labels may be inaccurate, so their benefit is diminished over time.

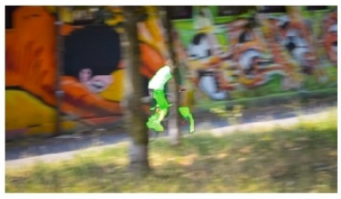
- Next: Can ensure we fine-tune the model with a correct signal?

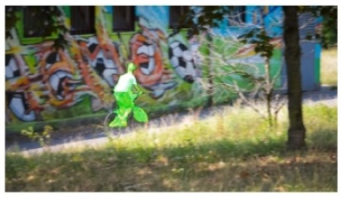
MaskTrack



Summary

- Advantages of appearance-based models:
  - can be trained on static images;
  - can recover well from occlusions;
  - conceptually simple.





- Disadvantages:
  - no temporal consistency;
  - can be slow at test-time (need to adapt);

Proposal-based VOS

Proposal Generation

Until now:	Now:
- Input is the whole image - Proposals could be used for refinement	- Generate proposals - Link them temporally (similar to tracking-by-detection)

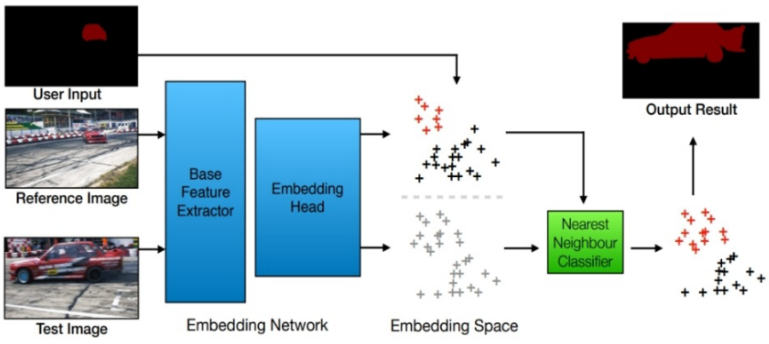
- Instance segmentation networks (e.g. Mask R-CNN) produce instance-level representation (masks with optional bounding boxes).
- One could link them over time to produce the tracking result.

PREMVOS → Çok işlemedi
çok derin dedi

Metric-based Approaches

Pixel-wise retrieval

- The user input can be in any form, first-frame ground-truth mask, scribble...



•Training: use the triplet loss to bring foreground pixels together and separate them from background pixels

•Test: embed pixels from both annotated and test frame, and perform
a nearest neighbour search for the test pixels.

Advantages:
•We do not need to retrain the model for each sequence, nor finetune;

•We can use unsupervised training to learn a useful feature representation

Gok oz
bokseti

Temporal LSTM: Summary

- Advantage: Improved temporal coherence.

Disadvantages:

- Typically more computationally greedy.
- We have to limit the sequence length to contain the computational requirement.
- May struggle with large motions.
- Error accumulation.

VOS vs MOTS

- "Semi-supervised" Video Object Segmentation (VOS) is limited by:
  - First frame mask given (in the supervised case)
  - Short video clips with objects present in almost all frames
  - Objects in a video are (mostly) of different categories
  - Few objects to track (max around 7 per video)
- Multi-Object Tracking and Segmentation (MOTS)
  - Scenarios with a large number of objects (20-40), mostly of the same category (e.g., pedestrians)
  - Long sequences
  - No first frame annotation provided, one has to deal with appearing and disappearing objects.

Metrics for VOS

- 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.

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

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

Metrics for VOS

- Region similarity: Jaccard index (IoU) of ground truth mask and predicted mask.
  - Mean: average for the dataset
  - Recall: fraction of sequences scoring higher than a threshold (e.g. 0.5 in DAVIS)
  - Decay: quantifies the performance loss (or gain) over time.
    - Can be F-Decay (contour), J-Decay (region);
    - In DAVIS: mean(25% first frames) - mean(25% last frames).

Transformers

Self-attention: Summary

Putting it all together. Define:

- a query vector $Q \in \mathbb{R}^{S \times n}$ We have S queries
- a key matrix $K \in \mathbb{R}^{T \times n}$ Our hash table has T (key, value) pairs
- a value matrix $V \in \mathbb{R}^{T \times m}$

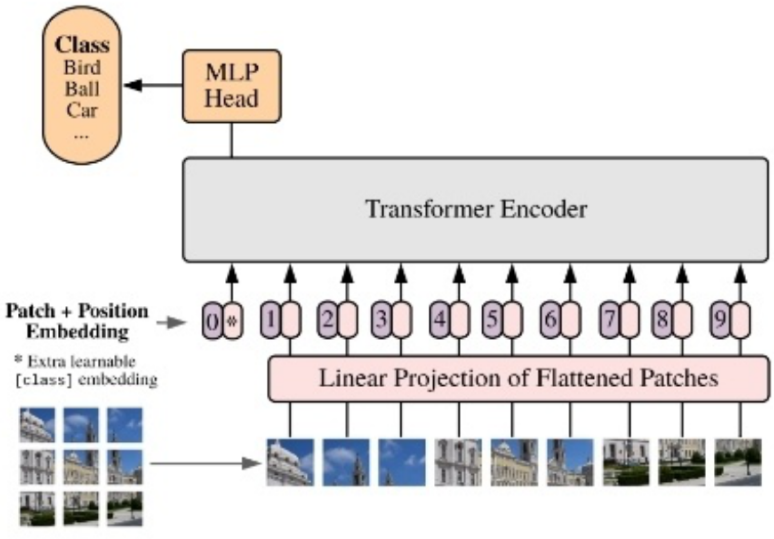
Self-attention: $\text{Attention}(Q, K, V) = \text{softmax}(QK^T/\sqrt{n})V$

Experiments with ViT

- ViT performs well only when pre-trained on large JFT dataset (300M images). (QUIZ: Why?)
- ViT does not have the inductive biases of CNNs:
  - locality (self-attention is global);
  - 2D neighbourhood structure (positional embedding is inferred from data);
  - translation equivariance.
- But still: now we use the same computational framework for two distinct modalities: language and vision!

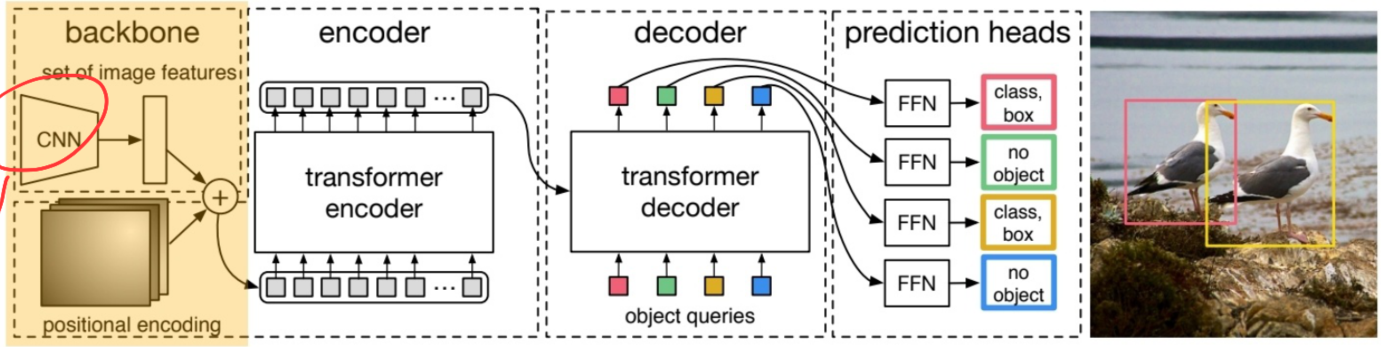
Vision Transformer

1. Split an image into fixed-sized patches.
2. Linearly embed each patch (a fully connected layer).
3. Attach an extra “[class]” embedding (learnable).
4. Add positional embeddings.
5. Feed the sequence to the standard Transformer encoder.
6. Predict the class with an MLP using the output [class] token.



(Dosovitskiy et al., 2020)

DETR: A closer look



- The model flattens it and supplements it with a positional encoding before passing it to the Transformer.

→ Uses CNN for 2D representation

•We supply object queries (learnable positional encoding) to the decoder, which jointly attends to the encoder output.

Each output embedding of the decoder is passed to a shared feed forward network (FFN) that predicts either a detection (class and bounding box) or a "no object" class.

DETR: The loss function

- Hungarian matching between the predictions (queries) and the ground truth.

→ kullanmamızın asıl sebebi mesela 2 tane kuş var büyük bir box çizerse loss yüksek olacak onları içeren

Loss for the class

Loss for the bounding box

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

Optimal assignment computed in the first step

→ Çünkü her predicted box değeriyle eşleşmeli

DETR: Summary

- Accurate detection with a (relatively) simple architecture.
- No need for non-maximum suppression
  - We can simply disregard bounding boxes with “empty” class, or with low confidence.
- Accurate panoptic segmentation with a minor extension.

Issues:

- High computational and memory requirements (especially the memory).
- Slow convergence / long training.

See: Zhu et al., “Deformable DETR: Deformable Transformers for End-to-End Object Detection” (ICLR 2021).

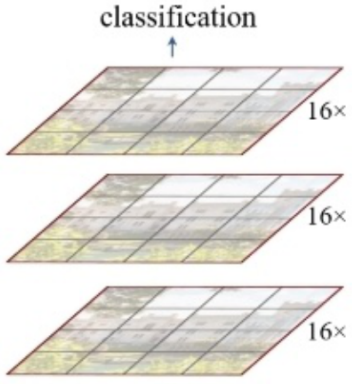
SWIN TRANSFORMER

Vit nin sıkıntılı yönü resmi 16 ya 16 ya böldüğünde büyük resimlerde her bir patch çok büyük kalıyor.

Swin Transformer

Recall ViT:

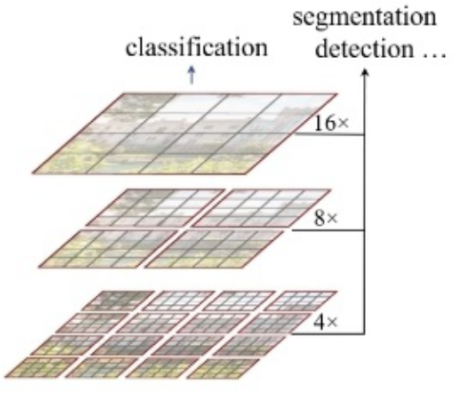
- We process patch embeddings with a series of self-attention layers;
  - Quadratic computational complexity.
- The number of tokens remains the same in all layers (QUIZ: How many / what does it depend on?)
- In CNNs, we gradually decrease the resolution, which has computation benefits (and increases the receptive field size).
- Do ViTs benefit from the same strategy?



Swin Transformer

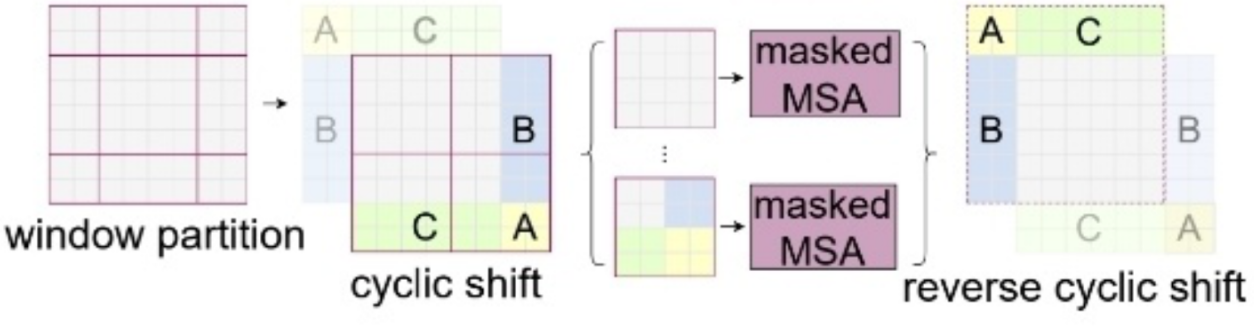
Swin Transformer:

- Local attention windows with a fixed size.
- Construct a hierarchy of image patches.
- Linear computational complexity.
- Output representation is compatible with standard backbones (e.g. ResNet)
  - We can test many downstream tasks (e.g. object



Efficient implementation of overlapping attention windows

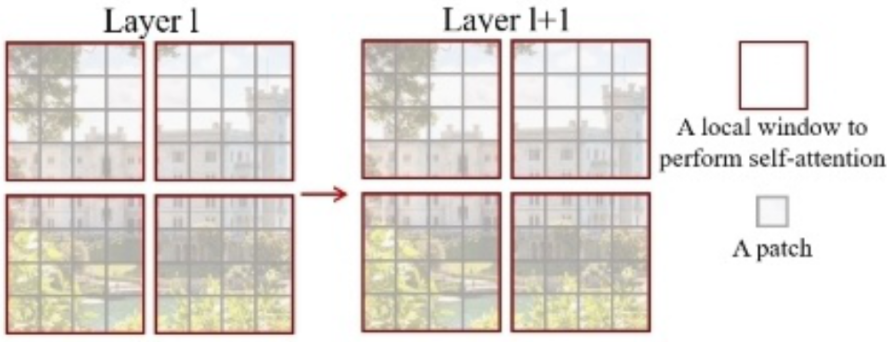
- equivalent to using "circular" padding (see PyTorch documentation).



Swin Transformer

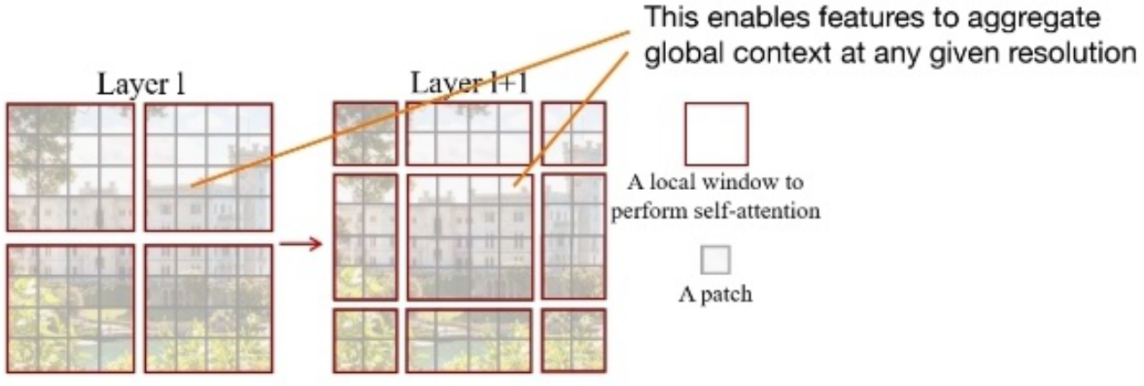
A naive implementation will hurt context reasoning:

- At any given level (except the last one), our context is not global anymore:



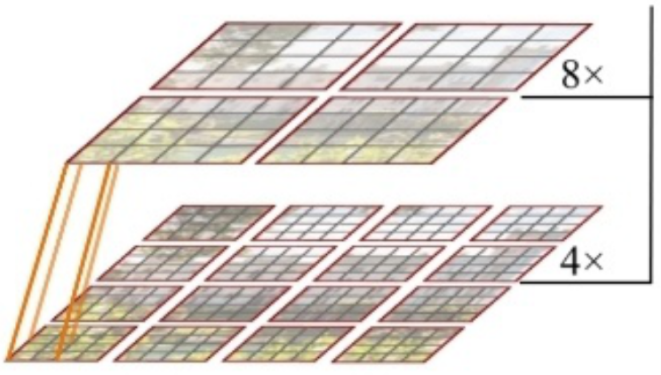
A naive implementation will hurt context reasoning:

- Solution: alternate the layout of local windows:



Successively decrease the number of tokens using **patch merging**:

- concatenate 2x2 C-dim patches into a feature vector (4xC-dim);
- linearly transform to a 2xC dimensional vector.



Swin Transformer: Results

- More efficient and accurate than ViT and some of CNNs (despite using lower resolution input).
- Note: No pre-training on on large datasets!
- Improved scalability on large datasets (compare ImageNet-1K and ImageNet-22K)

ImageNet-22K

method	image size	#param.	FLOPs	throughput (image / s)	ImageNet top-1 acc.
R-101x3 [38]	384^2	388M	204.6G	-	84.4
R-152x4 [38]	480^2	937M	840.5G	-	85.4
ViT-B/16 [20]	384^2	86M	55.4G	85.9	84.0
ViT-L/16 [20]	384^2	307M	190.7G	27.3	85.2
Swin-B	224^2	88M	15.4G	278.1	85.2
Swin-B	384^2	88M	47.0G	84.7	86.4
Swin-L	384^2	197M	103.9G	42.1	87.3

ImageNet-1K

method	image size	#param.	FLOPs	throughput (image / s)	ImageNet top-1 acc.
RegNetY-4G [48]	224^2	21M	4.0G	1156.7	80.0
RegNetY-8G [48]	224^2	39M	8.0G	591.6	81.7
RegNetY-16G [48]	224^2	84M	16.0G	334.7	82.9
EffNet-B3 [58]	300^2	12M	1.8G	732.1	81.6
EffNet-B4 [58]	380^2	19M	4.2G	349.4	82.9
EffNet-B5 [58]	456^2	30M	9.9G	169.1	83.6
EffNet-B6 [58]	528^2	43M	19.0G	96.9	84.0
EffNet-B7 [58]	600^2	66M	37.0G	55.1	84.3
ViT-B/16 [20]	384^2	86M	55.4G	85.9	77.9
ViT-L/16 [20]	384^2	307M	190.7G	27.3	76.5
DeiT-S [63]	224^2	22M	4.6G	940.4	79.8
DeiT-B [63]	224^2	86M	17.5G	292.3	81.8
DeiT-B [63]	384^2	86M	55.4G	85.9	83.1
Swin-T	224^2	29M	4.5G	755.2	81.3
Swin-S	224^2	50M	8.7G	436.9	83.0
Swin-B	224^2	88M	15.4G	278.1	83.5
Swin-B	384^2	88M	47.0G	84.7	84.5

Swin Transformer: Summary

- Reconciles the inductive biases from CNNs with Transformers;
- State-of-the-art on image classification, object detection and semantic segmentation;
- Linear computational complexity
  - local attention windows of fixed size (independent from the image resolution).
- Demonstrates that Transformers are strong vision models across a range of classic downstream applications.
- Best paper award at ICCV '21.