Project → Presentation
→ Software implementation
→ Written report

If the project is too difficult → Simplify it
If the project is too easy → Make it difficult

Presentation (5%) → Problem statement and background material
→ Description of the proposed solution
→ Implementation details
→ Discussion of the obtained results

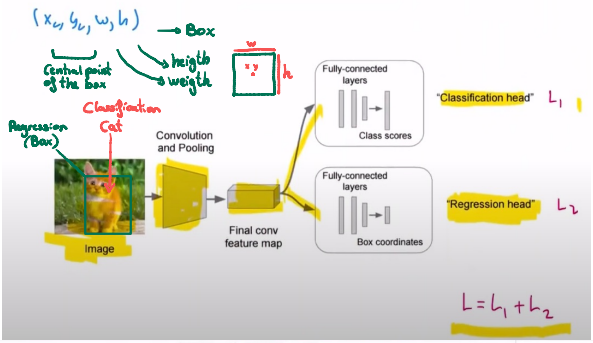
↓ Demonstration (video)

Project should be something similar to an assignment

Software and report → They expect an executable notebook
→ Same structure of a presentation
→ At least 8 pages, does not necessary and 2 column format
→ Citations and IEEE/ACM conventions

Object localization

- Localization is achieved through regression of the 4 bounding box coordinates



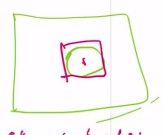
Classification head: It's a cat, dog...
Regression head: Will find a box

Object localization

- Training:
  - For each example mark bounding box and label class
  - Include background (no object) boxes
  - One hot encode class labels

$y^{(i)} = [p_c, x_c, y_c, w, h, c_1, \dots, c_K]$

p_c: 1 if object box, 0 if not object box
bounding box
one-hot encoded class



- Classification:

$\hat{y} = [\hat{p}_c, \hat{x}_c, \hat{y}_c, \hat{w}, \hat{h}, \hat{c}_1, \dots, \hat{c}_K]$

object probability (there is a object or not)
bounding box
class probabilities

L15: object detection (cont.)

Object localization

- Loss:

$L = L_{reg} + L_{cls}$

regression: e.g. cross entropy
classification: e.g. L_2

$$L_i(y^{(i)}, \hat{y}^{(i)}) = \begin{cases} (y_1^{(i)} - \hat{y}_1^{(i)})^2 & \text{if } y_1^{(i)} = 0 \\ \sum_j (y_j^{(i)} - \hat{y}_j^{(i)})^2 & \text{otherwise} \end{cases}$$

Truth, Predicted probability

no object - ignore all predictions

otherwise ← There is a object there

e.g. cross entropy for class labels

Object detection

- Localize and classify multiple objects
- Approach:
  - Find candidates
  - Classify candidates
  - Possibly refine coordinates
- Finding candidates: (different approaches)
  - Sliding window
  - Grid cells
  - Region proposals Network that propose where are the objects

Ask if there are an object

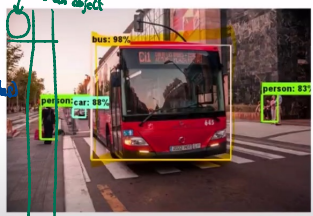


Figure 6. Example objection detection results using MobileNet SSD.

Grid cells: Divide the picture into cells and then we'll ask if there are are object in each cell

Sliding window detection

- Slide a window through the image. At each location:
  - Crop sub window
  - Classify each sub window (object/background)
- To deal with different object scales, crop different sub windows with different sizes and aspect ratios at each location
- This may be slow because there are many sub windows to classify

1000 × 1000 × 10

1000 × 1000 image → 10,000,000 sub-windows with 10 sub-windows at each location to classify for each image

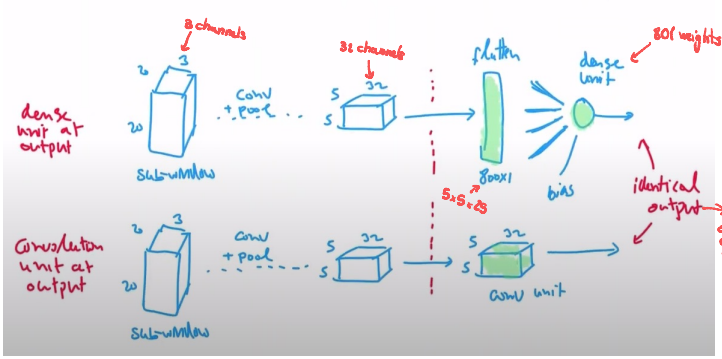
It is similar to convolution except that instead to multiplying the corresponding elements of the kernel with the image. I just slide the image and I crop a section and then I will ask the question is there an object in that crop section (normally bigger than 3×3)

Sliding window detection

- Problems:
  - Running dense layers for classifying each sub window is inefficient
  - In addition, dense layers need a fixed input size and so the image input size needs to be fixed
- Fully convolutional implementation:
  - A fully convolutional implementation of the dense layers is more efficient: no need to process each sub window separately, and convolutions can be implemented efficiently (e.g. as in mobileNet)
  - A fully convolutional implementation can work on variable size images

Sliding window detection

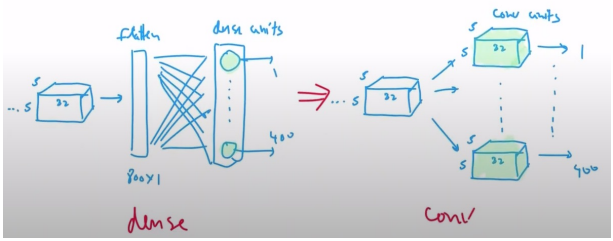
- A single fully connected (Dense) unit may be replaced by a single convolutional unit that has the exact dimensions of the input tensor (without flattening it)



Convolution and dense do the same thing (take away the sum of the inputs as well)

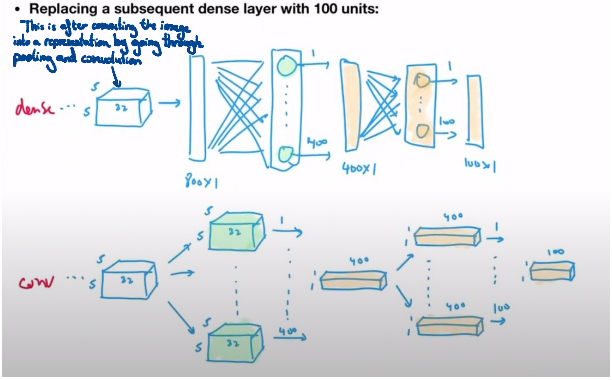
Sliding window detection

- Consider replacing 400 fully connected (FC) units by convolution filters
  - The output of 400 FC (dense) units is a 400×1 vector
  - We can replace 400 FC units with 5×5×32 convolution filters and so their output will be a 400×1 vector



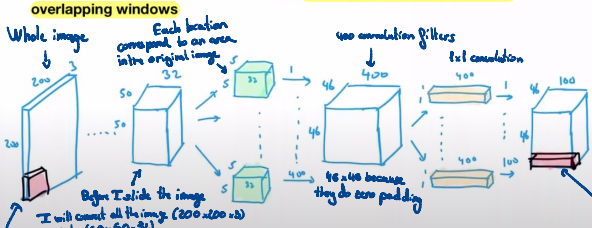
Sliding window detection

- Replacing a subsequent dense layer with 100 units:



Sliding window detection (benefit of change)

- So far we performed a single sub window classification
- For a single sub window there is no reduction in computations
- The advantage of convolutions is when sharing computations between overlapping windows



- Each column in the output provides probability for 100 classes in a corresponding image region (sub window)

Sliding window detection

- **Summary:**
  - Use convolution + pooling to extract features and reduce spatial resolution
  - Each location in a reduced resolution layer corresponds to a larger patch in the original image (patches overlap)
  - Use a dense layer or 1×1 convolution to classify or regress at each location in reduced resolution layer (classify class and regress bounding box)

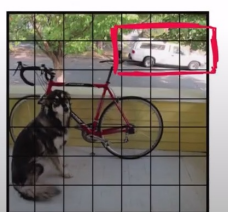
- **Problem:**
  - Detection using a sliding window does not produce accurate detection because:
    1. It is based on classification in a pooled (lower resolution) layer
    2. The region covered by filters is of fixed size and so we detect fixed size bounding boxes
  - To solve this we can, in addition to classifying each location, regress bounding boxes

Grid cell detection

- Instead of scanning the image, divide the image into grid cells and attempt to detect an object in grid cell
- Cells do not necessarily match objects and so use regression to find a bounding box for each detected object
- Similar to object localization (single object) except that now at each grid cell detect and localize objects
- At each grid cell train and predict using:

$$y = [p_c, x_c, y_c, w, h, c_1, \dots, c_k]$$

p_c: classification (object yes/no) — x_c, y_c, w, h: regression (bounding box) — $c_1, \dots, c_k$: classification (object class)

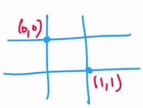


S × S grid on input

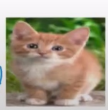
Grid cell detection

* x_c, y_c, w, h are relative to each grid cell

$x_c, y_c \in [0,1]$ whereas w and h may be bigger or smaller than 1 for boxes larger than a cell

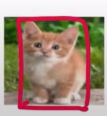


grid cell (proposal)



(0, 0, 0, 0)
Proposal is good

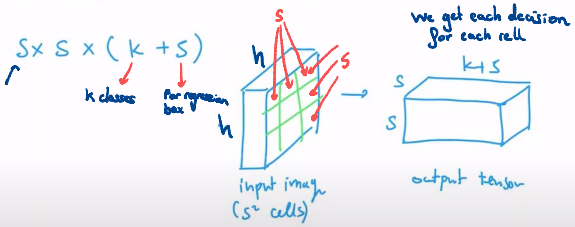
(.25, 0, 0, 0)
Proposal too far to left



(0, 0, -0.125, 0)
Proposal too wide

Grid cell detection

- For an S x S grid and K classes the output is a tensor of size:



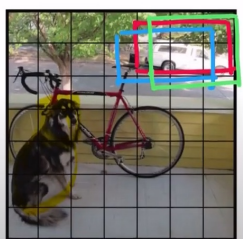
- At each grid cell with an object find a precise bounding box instead of using a fixed size window
- Only one object is detected at each grid cell

Grid cell detection

- Non-maximum suppression (NMS):
  - Select a detection in the grid cell with the highest detection score (probability) and delete detections that have an IOU (with the selected detection) that is greater than a threshold
  - Perform NMS for each class separately to allow for object overlap

I only choose one

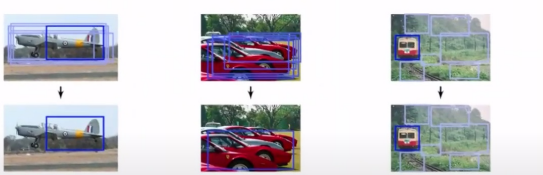
- Example: 3 detections because object occupies 3 grid cells. Select the one with highest probability.



S × S grid on input

Grid cell detection

- Possible problems with NMS:



(a) The top-scoring box may not be the best fit. (b) It may suppress nearby objects. (c) It does not suppress false positives.

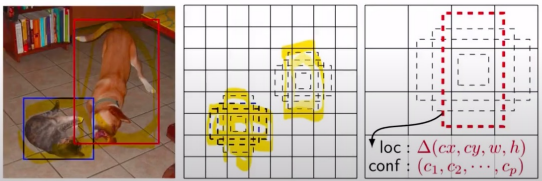
Grid cell detection

- Problems:
  - What should be the grid cell size?
    - A too small cell will fail in classifying big objects
    - A too large cell will fail in classifying small objects
  - What should be the aspect ratio of candidate cells?
    - Different objects have different aspect ratios of boxes that should be used for their detection
  - How to detect more than one object in each grid cell?

Which one should I use?

Grid cell detection

- **Anchor boxes:**
  - To solve three problems we use multiple anchor boxes of each grid location
  - Anchor boxes have different sizes and different aspect ratios (selected according to what we see in the training data)



(a) Image with GT boxes (b) 8×8 feature map (c) 4×4 feature map

Grid cell detection

- **Multiple anchor boxes:**
  - To allow detecting two different (overlapping) objects at each location (e.g. objects with different aspect ratios) double the target vector **y**

- Most grid cells will have zero or one anchor boxes

$y = [p_c^1, x_{c_1}^1, y_{c_1}^1, w^1, h^1, c_1^1, \cdots, c_k^1 ; p_c^2, x_{c_1}^2, y_{c_1}^2, w^2, h^2, c_1^2, \cdots, c_k^2]$

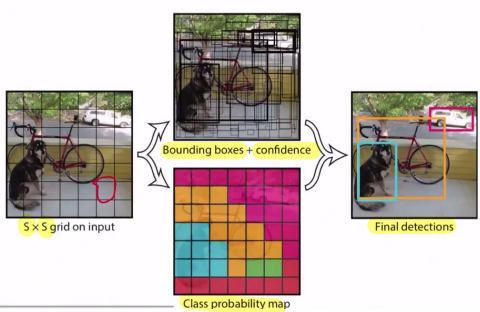
first cell second cell

$p_c^1 = 0$ and $p_c^2 = 0$ if object does not exist

- **Anchor box assignment during training:**
  - During training, the anchor box assigned to an object is the one having the highest IOU with a ground truth box

Grid cell detection

* Summary:



Single shot detectors

- Single shot versus double shot methods:
  - Single shot detectors perform object detection in a single network that looks at the image once
  - Two shot detectors look at the image twice: one to propose object regions (region proposals) and a second to refine and classify regions
  - Two shot methods have a higher computational cost (lower frame rate) but may be more accurate

- **Single shot methods:**
  - Consider predefined anchor boxes at grid cells
  - Refine and classify region proposals

- **Two short methods:**
  - Consider predefined anchor boxes at grid cells
  - Classify anchor boxes as objects or not to get region proposals
  - Refine and classify region proposals (higher quality classifier)

First shot: Network that makes the proposal where the objects are
Second shot: will do the classification and regression of the box

Single shot detectors

* Methods:
  - YOLO (You only Look Once)
  - MultiBox
  - Singh Shot MultiBox detector (SSD)

Single shot detectors

* YOLO (You only Look Once): A fast Singh Shot detector — different versions: $V_1, V_2, V_3, \dots$
* Algorithm:
  - divide image to SxS grid cells (e.g. S=19)
    A grid cell is responsible for detecting objects with centers in it.
  - Each grid cell predicts B bounding boxes and confidence scores. (confidence that the box contains the object and is accurate)

Confidence = $P(\text{object}) * IoU(\text{truth}, \text{prediction})$

Score ← It's the combination of probability of the object * IOU

If the value is small you don't want detect it as an object

If confidence score = 0 → There is not an object

Single shot detectors

* Yolo alg. (cont.):
  - Training vector with B bounding boxes per cell: B times

$y = [p_c^1\ b_x^1\ b_y^1\ h^1\ w^1\ c_1^1 \dots c_k^1 \dots ; p_c^B\ b_x^B\ b_y^B\ h^B\ w^B\ c_1^B \dots c_k^B]$

$p_c^j = 0$ if bounding box does not contain object

  - Predict using dense layers of 1x1 convolutions.
  - Apply non-maximum Suppression (NMS)

Single shot detectors

* YOLO Architecture

He did not explain this slide

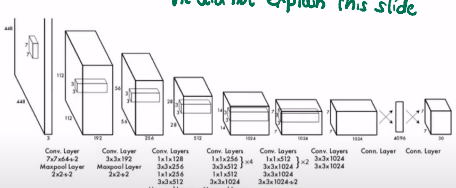


Figure 3: The Architecture. Our detection network has 24 convolutional layers followed by 2 fully connected layers. Alternating 1×1 convolutional layers reduce the features space from preceding layers. We pretrain the convolutional layers on the ImageNet classification task at half the resolution (224×224 input image) and then double the resolution for detection.

* use leaky ReLU activations inside network

Single shot detectors

YOLO loss function:

ignore empty bounding boxes

Center regression

$$\lambda_{\text{coord}} \sum_{i=0}^{S^2} \sum_{j=0}^{B} \mathbb{1}_{ij}^{\text{obj}} \left[(x_i - \hat{x}_i)^2 + (y_i - \hat{y}_i)^2 \right]$$

H, W regression

$$+ \lambda_{\text{coord}} \sum_{i=0}^{S^2} \sum_{j=0}^{B} \mathbb{1}_{ij}^{\text{obj}} \left[\left(\sqrt{w_i} - \sqrt{\hat{w}_i} \right)^2 + \left(\sqrt{h_i} - \sqrt{\hat{h}_i} \right)^2 \right]$$

object classification

$$+ \sum_{i=0}^{S^2} \sum_{j=0}^{B} \mathbb{1}_{ij}^{\text{obj}} \left(C_i - \hat{C}_i \right)^2$$

no-object classification

$$+ \lambda_{\text{noobj}} \sum_{i=0}^{S^2} \sum_{j=0}^{B} \mathbb{1}_{ij}^{\text{noobj}} \left(C_i - \hat{C}_i \right)^2$$

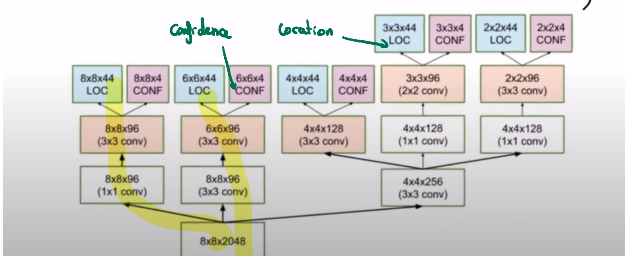
box classification

$$+ \sum_{i=0}^{S^2} \mathbb{1}_{i}^{\text{obj}} \sum_{c \in \text{classes}} \left(p_i(c) - \hat{p}_i(c) \right)^2$$

Single shot detectors

* MultiBox:
  - use prior for anchor boxes determined per class
  - Inception - like detection (detection at different scales)

Like inception we take different path to make the detection



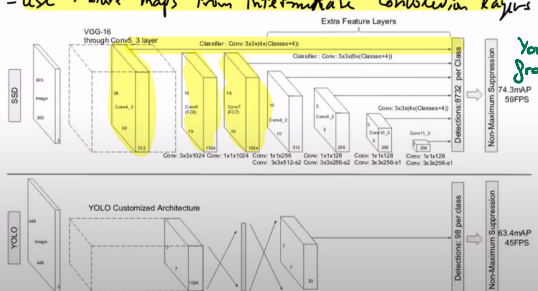
Single shot detectors

* MultiBox loss:
  - Confidence loss - class categorical cross entropy
  - Location loss - bounding box SoA

multiBox_loss = confidence_loss + α * location_loss

Single shot detectors

* Single-shot multibox detector (SSD):
  - SSD = YOLO + MultiBox
  - use feature maps from intermediate convolution layers



You take features from early layer and later layer

Single shot detectors

* SSD Contd.: Negative examples: location with no object

- use hard negatives (FP predicted by the model) to train the model (instead of all negatives)
  → this helps balancing negatives & positive examples

* Loss

Confidence Location

$$L(x, c, l, g) = \frac{1}{N}(L_{conf}(x, c) + \alpha L_{loc}(x, l, g))$$

$$L_{\text{conf}} = \frac{1}{n_{\text{postives}}}\left(\sum_{positives} \text{cross-entropy loss} + \sum_{\text{hard negatives}} \text{cross-entropy loss}\right)$$