Object Detection:

FasterRCNN and SSD for custom object detection

David Hughes, CS5990

Content Outline

1. Problem description - to learn details/architectures of modern object detectors
    a. Domain - computer vision, deep learning
    b. Motivation - To be able to localize and classify objects efficiently and effectively
    c. Importance - In applications like autonomous cars it is important to be able to locate and identify a person for example
2. Approach used - FasterRCNN and SSD architectures
3. Analysis of results
4. Conclusions and future work - automatic custom object detection

Object Detection

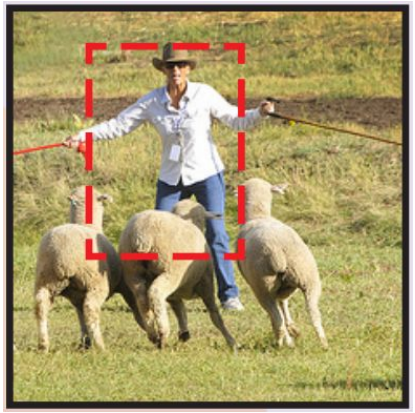
- Detecting objects of a specific class from an image
- Computer vision/image processing task
- Examples
  - Face Detection
  - Pedestrian detection in autonomous cars

https://github.com/tensorflow/models/tree/master/research/object_detection

Keywords / Ideas in Object Detection

Proposals

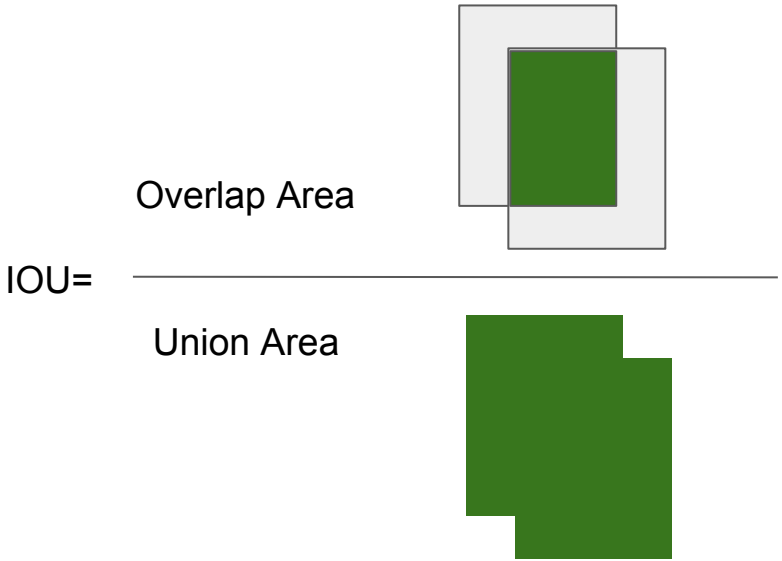
- rectangular region that might contain an object
- Other names: bounding box proposal, region of interest, region proposal, box proposal
- Storage Representations
  - Two corners: (x0, y0, x1, y1)
  - Center Location: (x, y, w, h)



https://medium.com/@nikasa1889/the-modern-history-of-object-recognition-infographic-aea18517c318

Intersection over Union (IoU)

- Evaluation metric
- Requires two sets of bounding boxes
  - Ex: ground truth and predicted proposals
- Divide the area of overlap by the area of union



IoU Code

```python
#https://www.pyimagesearch.com/2016/11/07/intersection-over-union-iou-for-object-detection/
def bb_intersection_over_union(boxA, boxB):
    # determine (x, y)-coord of intersection rectangle
    xA = max(boxA[0], boxB[0])
    yA = max(boxA[1], boxB[1])
    xB = min(boxA[2], boxB[2])
    yB = min(boxA[3], boxB[3])

    # compute the area of intersection rectangle
    interArea = max(0, xB - xA + 1) * max(0, yB - yA + 1)

    # compute area of both prediction and ground-truth
    boxAArea = (boxA[2] - boxA[0] + 1) * (boxA[3] - boxA[1] + 1)
    boxBArea = (boxB[2] - boxB[0] + 1) * (boxB[3] - boxB[1] + 1)

    # compute the intersection over union
    iou = interArea / float(boxAArea + boxBArea - interArea)

    # return the intersection over union value
    return iou
```

Recall / Precision

$$TP = \text{True positive}$$
$$TN = \text{True negative}$$
$$FP = \text{False positive}$$
$$FN = \text{False negative}$$

$$Precision = \frac{TP}{TP + FP}$$

$$Recall = \frac{TP}{TP + FN}$$

https://medium.com/@jonathan_hui/map-mean-average-precision-for-object-detection-45c121a31173

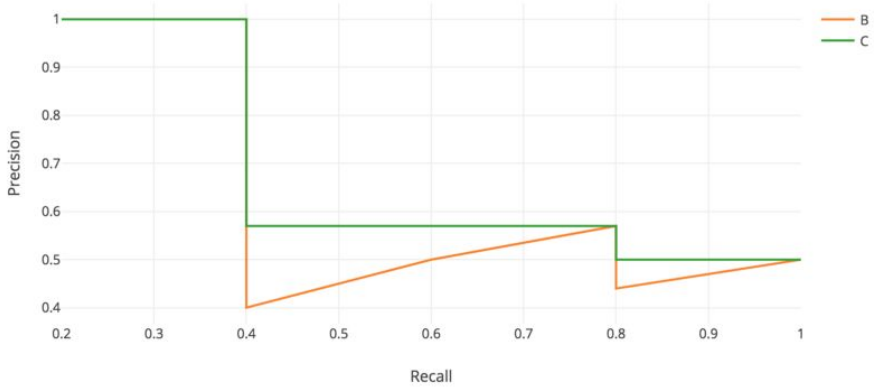
Mean Average Precision (mAP)

- AP = average of maximum precision at 11 recall levels
- mAP is just the average over all classes
- mAP@0.5 refers to IoU used

Iterate one prediction at a time

Rank	Correct?	Precision	Recall
1	True	1.0	0.2
2	True	1.0	0.4
3	False	0.67	0.4
4	False	0.5	0.4
5	False	0.4	0.4
6	True	0.5	0.6
7	True	0.57	0.8
8	False	0.5	0.8
9	False	0.44	0.8
10	True	0.5	1.0

Recall	Max
0	1.0
0.1	1.0
0.2	1.0
0.3	1.0
0.4	1.0
0.5	0.57
0.6	0.57
0.7	0.57
0.8	0.57
0.9	0.5
1.0	0.5



$$AP = \frac{1}{11} \sum_{r \in \{0.0,\ldots,1.0\}} AP_r$$

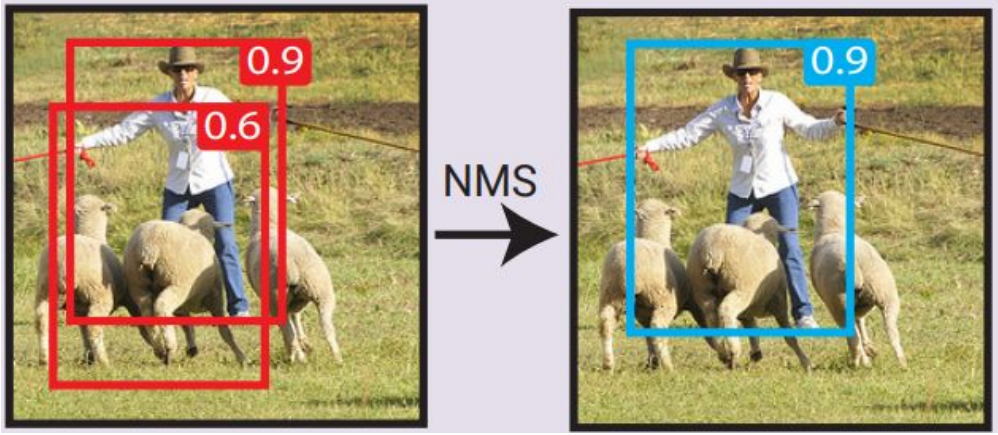
$$= \frac{1}{11} \sum_{r \in \{0.0,\ldots,1.0\}} p_{interp}(r)$$

$$p_{interp}(r) = \max_{\tilde{r} \geq r} p(\tilde{r})$$

https://medium.com/@jonathan_hui/map-mean-average-precision-for-object-detection-45c121a31173

Non Maximum Suppression (NMS)

- Merge overlapping proposals
- IoU > IoU_threshold -> remove lower confidence



https://medium.com/@nikasa1889/the-modern-history-of-object-recognition-infographic-aea18517c318

NMS in TF

tf.image.non_max_suppression

```
tf.image.non_max_suppression(
    boxes,
    scores,
    max_output_size,
    iou_threshold=0.5,
    score_threshold=float('-inf'),
    name=None
)
```

- https://www.tensorflow.org/api_docs/python/tf/image/non_max_suppression
  - Calls another function from a gen_ suffix file, python module with bindings to c++ implementation
- https://github.com/tensorflow/models/blob/master/research/object_detection/core/post_processing.py
  - To view code example

Bounding Box Regression

- Infer the bounding box that better fits the object inside
- Train the regressor to look at input region box and predict offset to the ground truth box

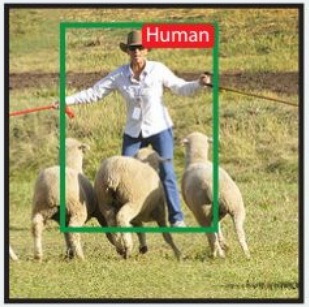
https://medium.com/@nikasa1889/the-modern-history-of-object-recognition-infographic-aea18517c318

Examples of Computer Vision Tasks

Image Classification

Classify an image based on the dominant object inside it.

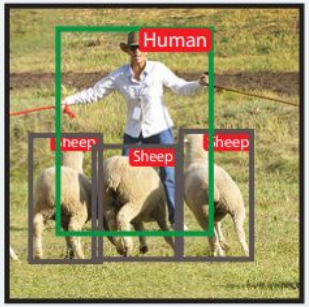
datasets: MNIST, CIFAR, ImageNet



Object Localization

Predict the image region that contains the dominant object. Then image classification can be used to recognize object in the region

datasets: ImageNet



Object Recognition

Localize and classify all objects appearing in the image. This task typically includes: proposing regions then classify the object inside them.

datasets: PASCAL, COCO

Semantic Segmentation

Label each pixel of an image by the object class that it belongs to, such as human, sheep, and grass in the example.

datasets: PASCAL, COCO

Instance Segmentation

Label each pixel of an image by the object class and object instance that it belongs to.

datasets: PASCAL, COCO

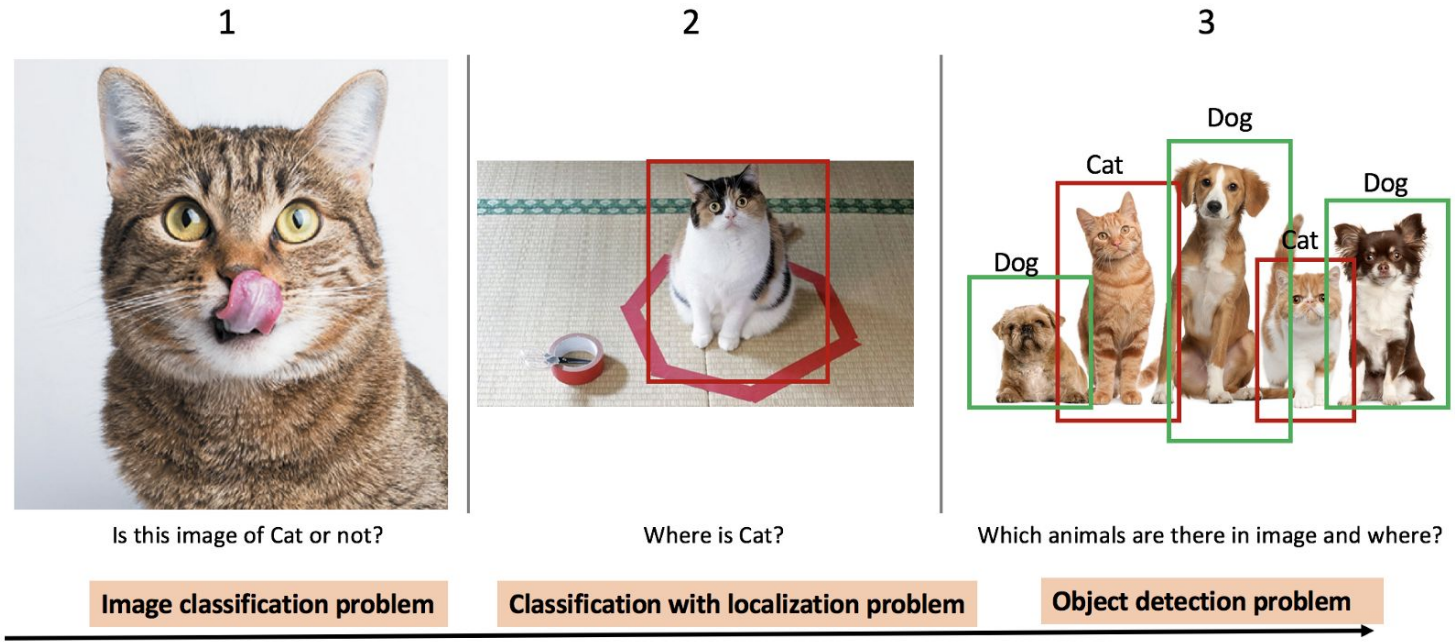
Keypoint Detection

Detect locations of a set of predefined keypoints of an object, such as keypoints in a human body, or a human face.

datasets: COCO

https://medium.com/@nikasa1889/the-modern-history-of-object-recognition-infographic-aea18517c318

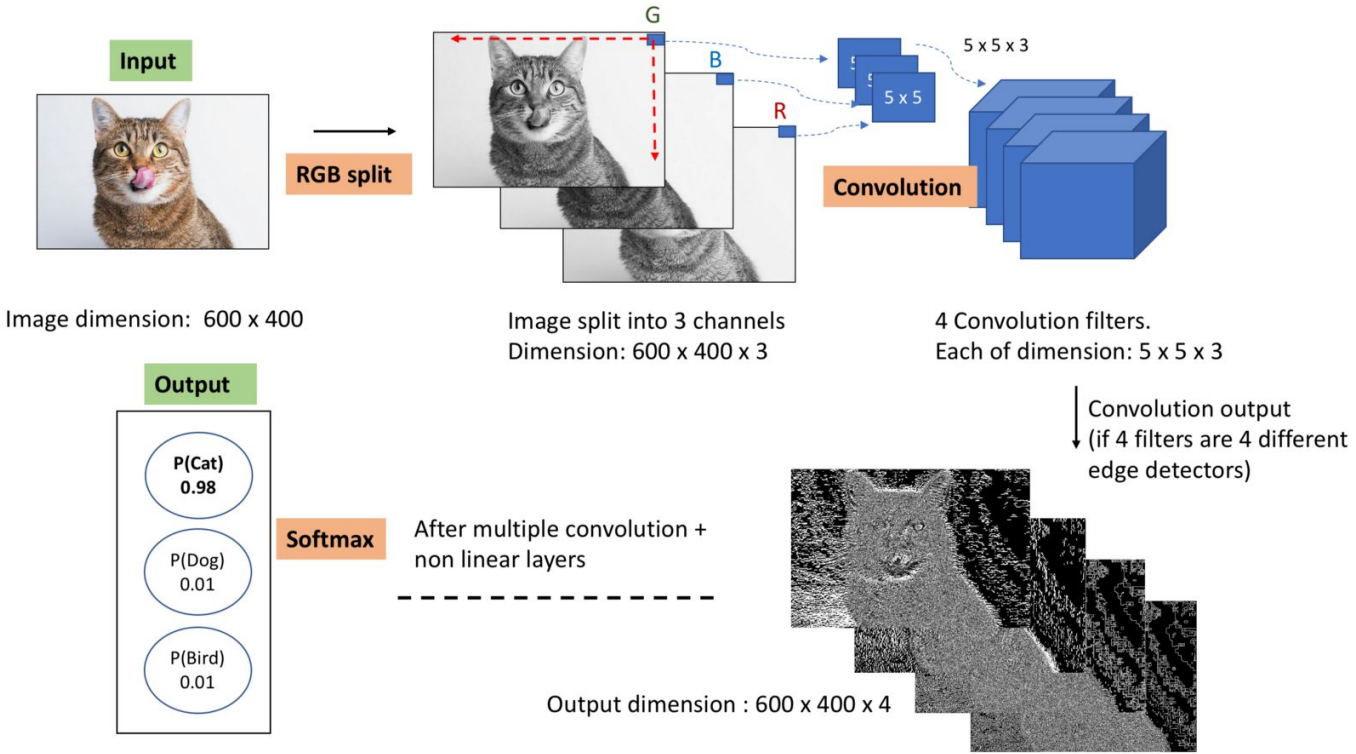
Progression of Computer Vision Tasks



https://towardsdatascience.com/evolution-of-object-detection-and-localization-algorithms-e241021d8bad

Image Classification

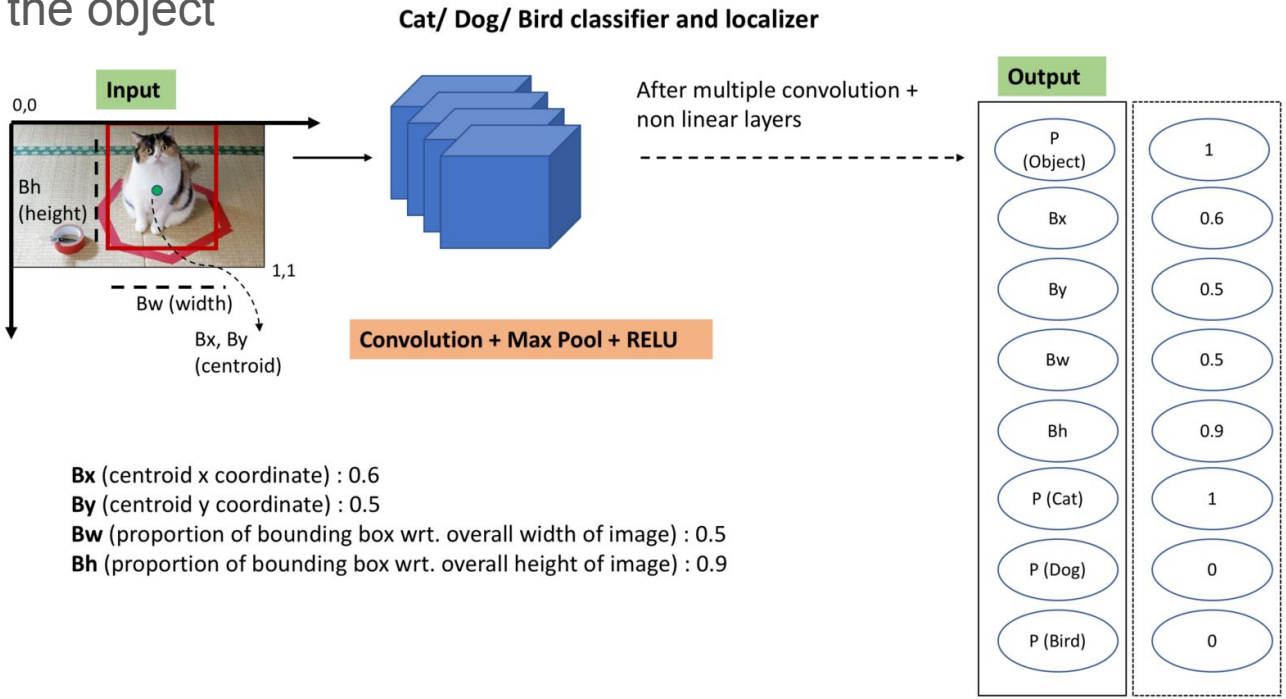
- Image => object class



https://towardsdatascience.com/evolution-of-object-detection-and-localization-algorithms-e241021d8bad

Object classification and localization

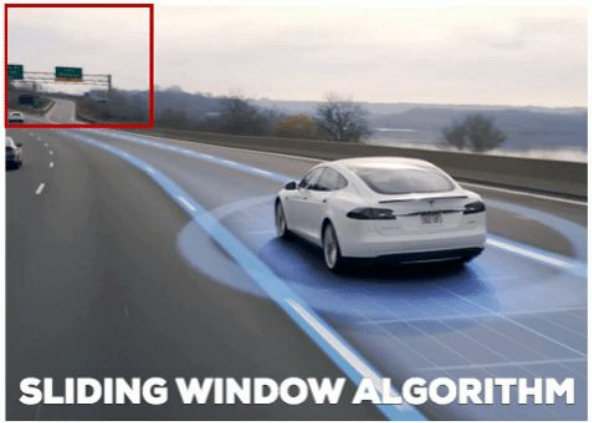
- Image => one object class and location
- Output landmarks for the object

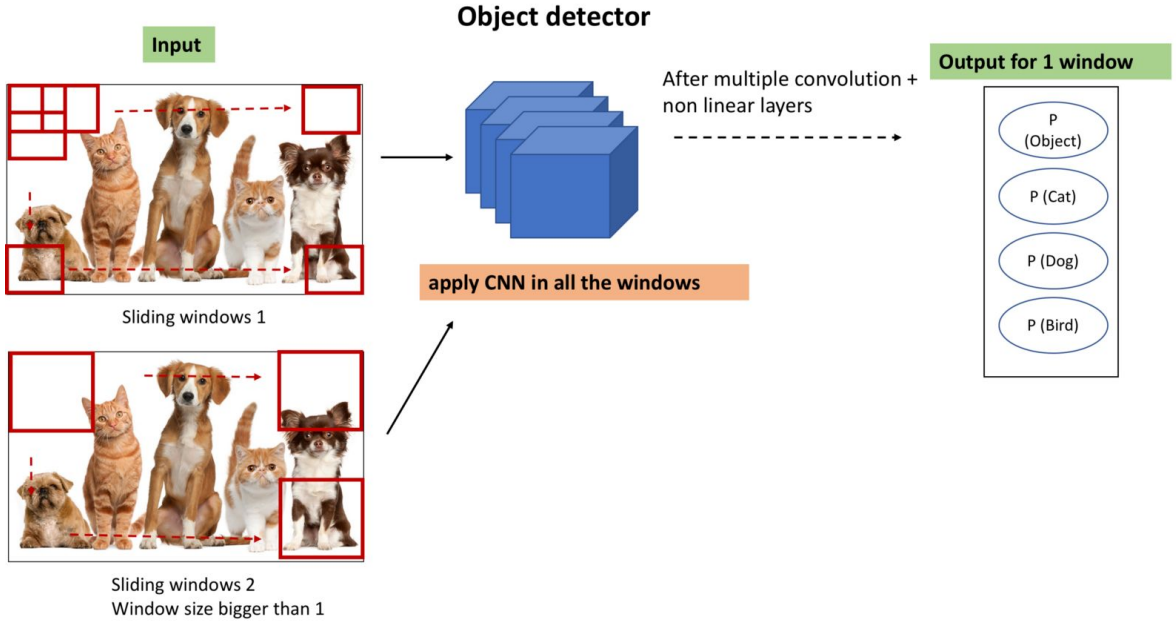


https://towardsdatascience.com/evolution-of-object-detection-and-localization-algorithms-e241021d8bad

Multiple objects detection and localization

- Sliding window object detection
- Cons
  - Computationally expensive (large number of regions)
  - Inaccurate bounding boxes (objects have different aspect ratios /spatial locations)





https://towardsdatascience.com/evolution-of-object-detection-and-localization-algorithms-e241021d8bad

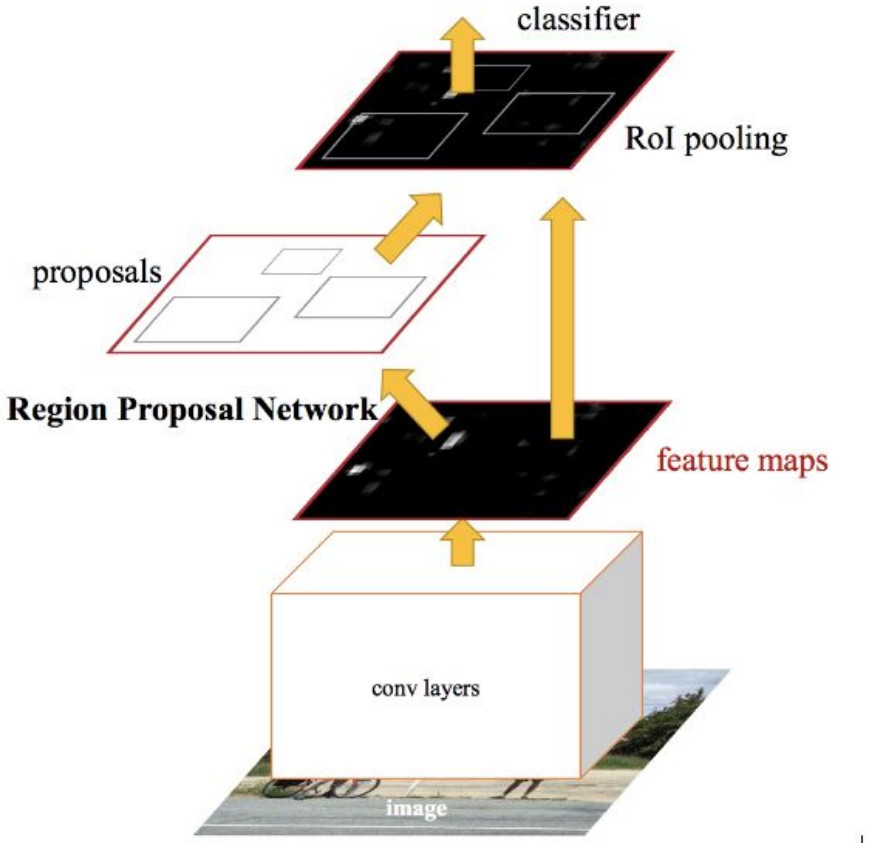
Is there a better way?

Architecture code

- Tensowflow’s general FasterRCNN
  - https://github.com/tensorflow/models/blob/master/research/object_detection/meta_architectures/faster_rcnn_meta_arch.py
  - 2321 Lines
- Tensorflow’s general SSD
  - https://github.com/tensorflow/models/blob/master/research/object_detection/meta_architectures/ssd_meta_arch.py
  - 1209 Lines

https://github.com/kbardool/keras-frcnn

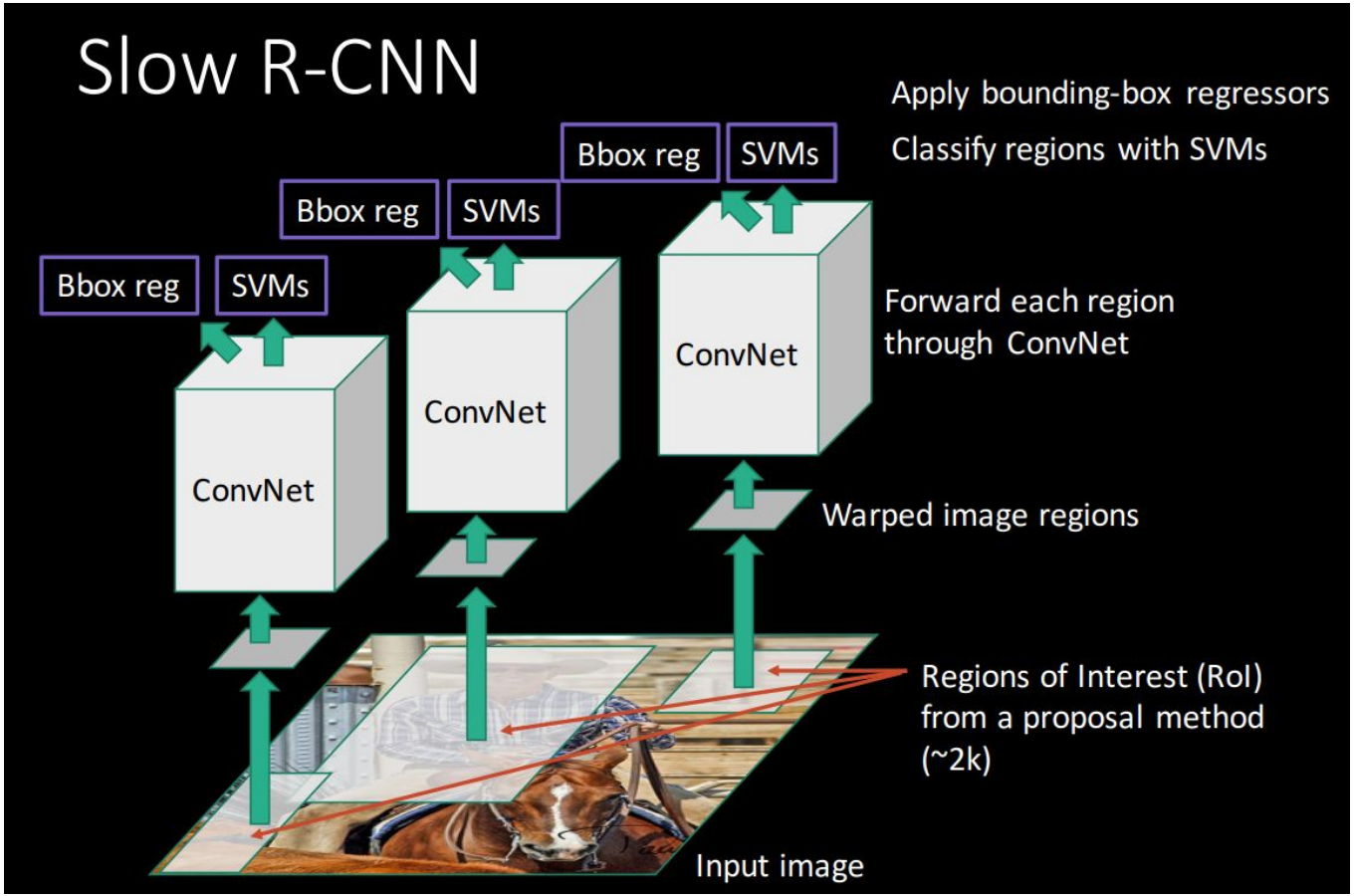
FasterRCNN



https://www.analyticsvidhya.com/blog/2018/10/a-step-by-step-introduction-to-the-basic-object-detection-algorithms-part-1/

But first….RCNN

- Region-Based Convolutional Neural Network
- 2013 (CVPR 2014) paper by Ross Girshick, et al.
- Combination of heuristic region proposal and CNN feature extractor
  - Heuristic = selective search
- Steps:
  1. 2000 Regions of Interest generated by selective search
  2. Regions are reshaped to 227x227 images
  3. AlexNet is used to extract 4096 features for each region image
  4. SVM trained to classify object in region image by 4096 features
  5. Class-specific bounding box regressors trained to refine proposals using 4096 features



http://www.robots.ox.ac.uk/~tvg/publications/talks/fast-rcnn-slides.pdf

Selective Search

- Heuristic region proposal algorithm
- Compute hierarchical grouping based on grouping/similarity criteria
  - Color, texture, brightness, size, shape

1. Generate Over-segmented Image
2. Recursively combine similar regions into larger regions (bottom up approach)
3. Use Generated regions to produce region proposals

Input Image

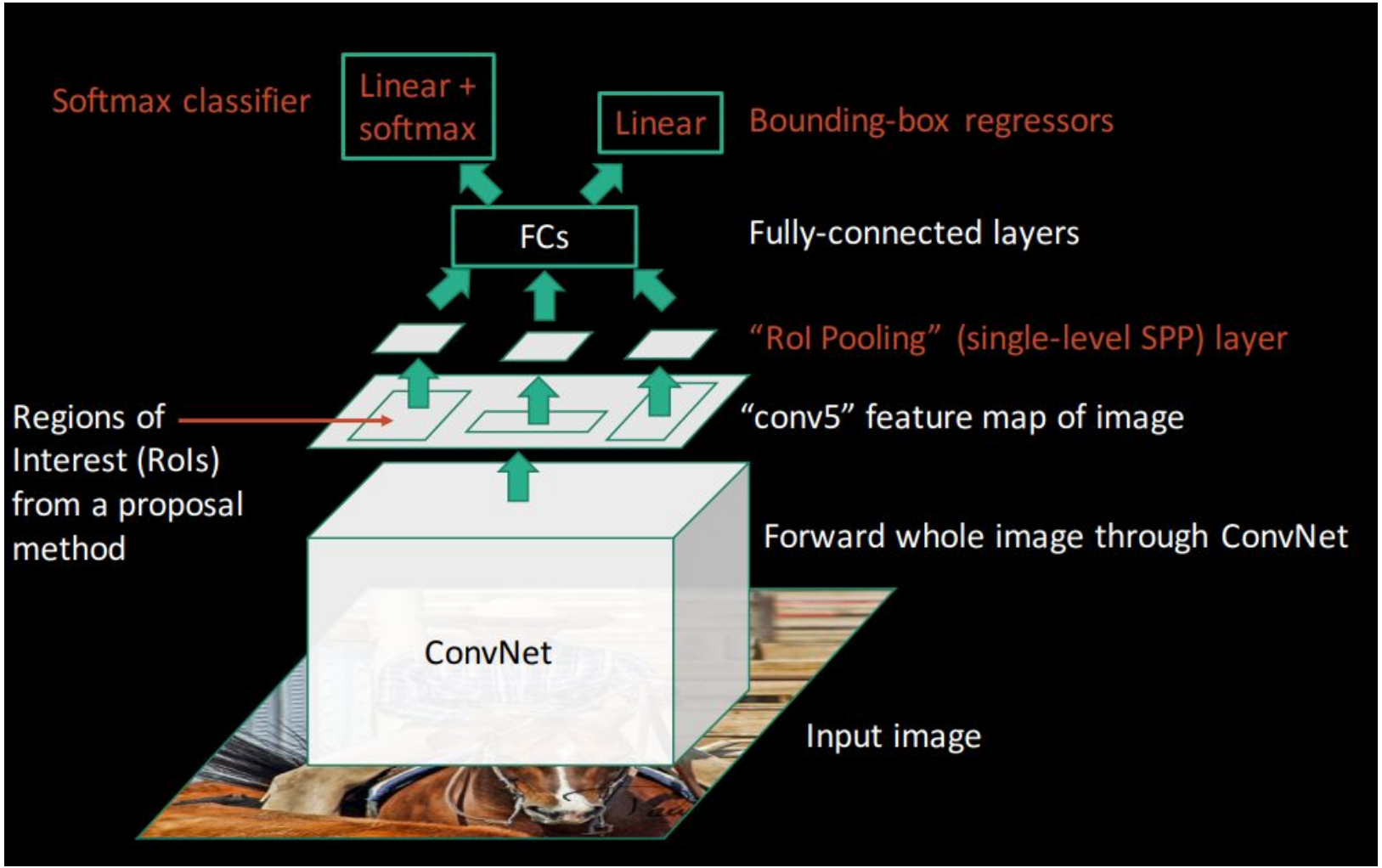
http://vision.stanford.edu/teaching/cs231b_spring1415/slides/ssearch_schuyler.pdf

RCNN Problems

- Expensive to extract features with CNN for 2000 regions
  - Forward pass of CNN for every region proposal image
- Uses three models together
  - Makes “end-to-end” object recognition pipeline hard to train
- Too slow for a large dataset
  - For N images in dataset, N*2000 forward passes of CNN

But Second….FastRCNN

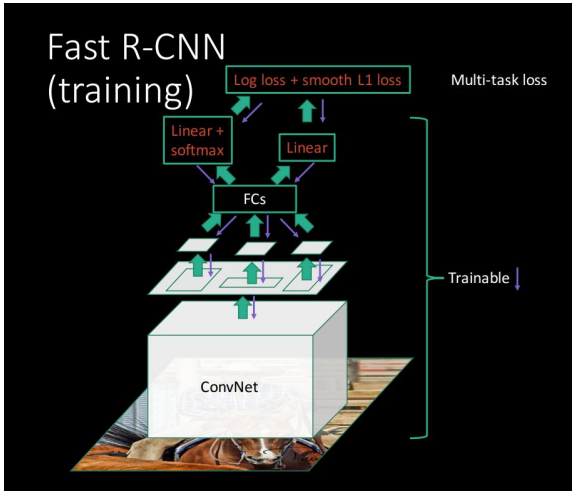
- 2015, by Ross Girshick again
- Use a single model to extract regions, predict classes, and return proposals
- Run CNN once per image
- Steps
  1. Train one CNN on image classification
  2. Propose 2k Regions by Selective Search
  3. Replace last max pooling of CNN with RoI pooling layer
     a. RoI pooling outputs fixed-length feature vectors of region proposals
  4. Pass regions to FC layer
  5. Output: softmax layer for class, box regression for predicting offsets relative to original RoI



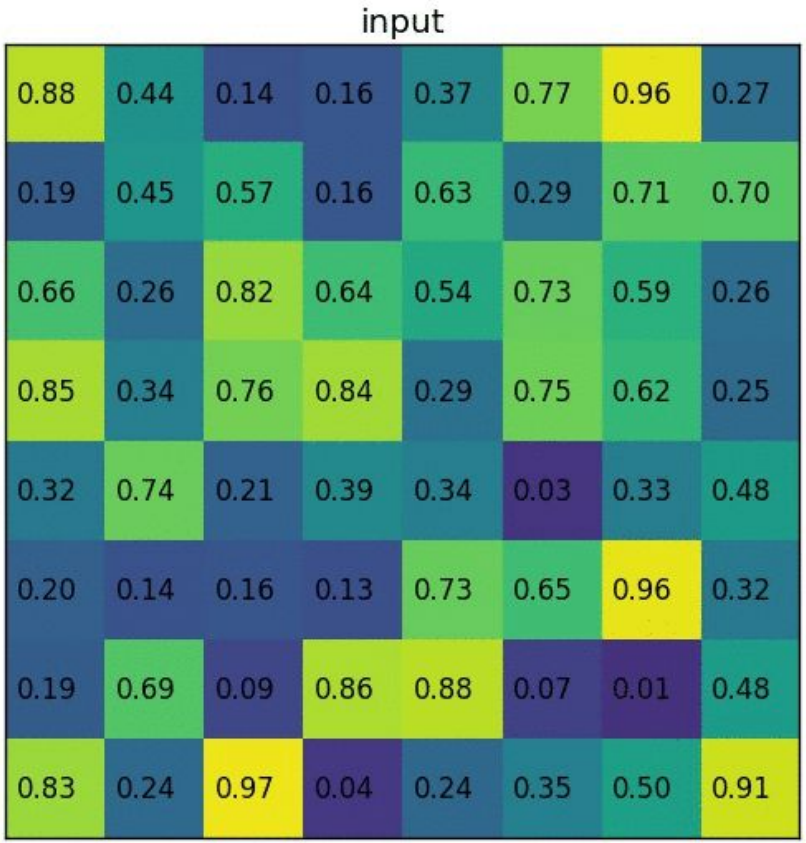
http://www.robots.ox.ac.uk/~tvg/publications/talks/fast-rcnn-slides.pdf

Region of Interest (RoI) Pooling

- Improvement over Spatial Pyramid Pooling Net (SPPnet)
- Differentiable layer
  - Allows the whole pipeline to be differentiable and trainable
  - For same reason as why Max Pooling is differentiable
- Reduce feature maps to the same size
- Steps:
  1. Image with region proposals
  2. Project region proposal onto conv feature map
  3. Divide projected region into hxw grid
  4. Max-pool within each grid cell
  5. Pass to FC layer



http://www.robots.ox.ac.uk/~tvg/publications/talks/fast-rcnn-slides.pdf



https://deepsense.ai/region-of-interest-pooling-explained/

RoI Pooling Code

Note: Code is actually from FasterRCNN

"How to crop proposals from a feature map" is another way of saying RoI Pooling Layer

```
1528    def _compute_second_stage_input_feature_maps(self, features_to_crop,
1529                                                 proposal_boxes_normalized):
1530      """Crops to a set of proposals from the feature map for a batch of images.
1531
1532      Helper function for self._postprocess_rpn. This function calls
1533      `tf.image.crop_and_resize` to create the feature map to be passed to the
1534      second stage box classifier for each proposal.
1535
1536      Args:
1537        features_to_crop: A float32 tensor with shape
1538          [batch_size, height, width, depth]
1539        proposal_boxes_normalized: A float32 tensor with shape [batch_size,
1540          num_proposals, box_code_size] containing proposal boxes in
1541          normalized coordinates.
1542
1543      Returns:
1544        A float32 tensor with shape [K, new_height, new_width, depth].
1545      """
1546      cropped_regions = self._flatten_first_two_dimensions(
1547          self._crop_and_resize_fn(
1548              features_to_crop, proposal_boxes_normalized,
1549              [self._initial_crop_size, self._initial_crop_size]))
1550      return slim.max_pool2d(
1551          cropped_regions,
1552          [self._maxpool_kernel_size, self._maxpool_kernel_size],
1553          stride=self._maxpool_stride)
```

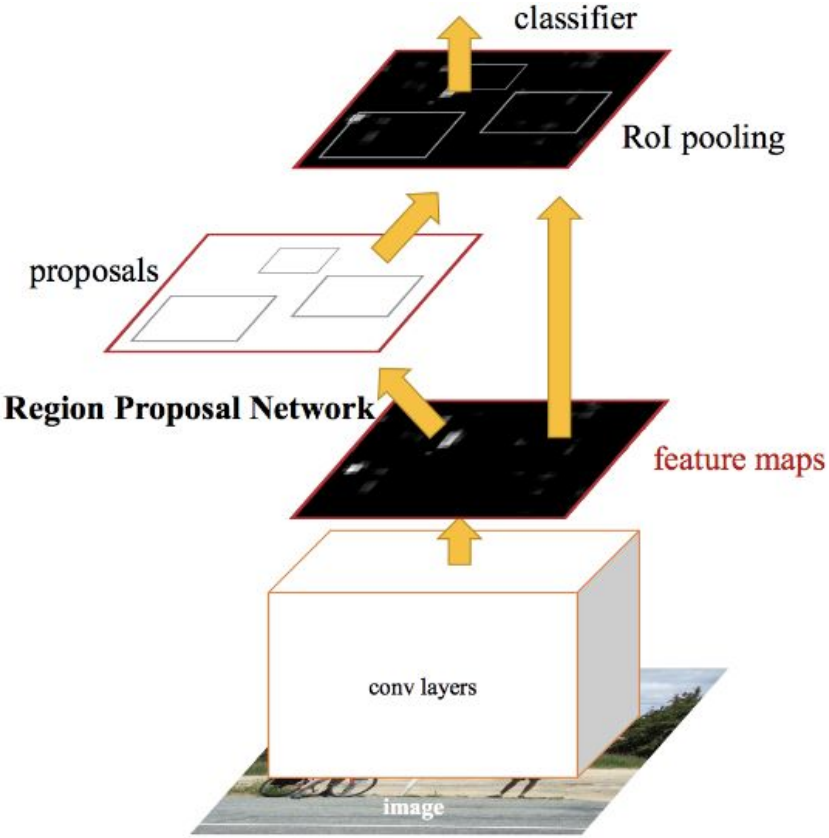
FastRCNN Problems

- The bottleneck is now the regional proposal heuristic, selective search

FasterRCNN

FasterRCNN (2015)

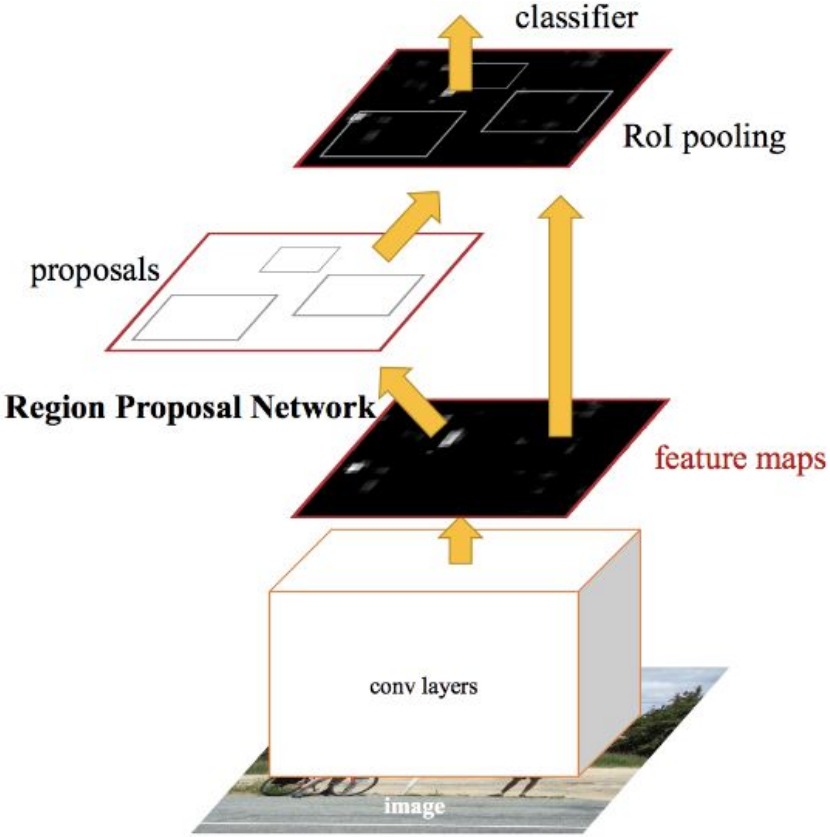
- FastRCNN with selective search replaced by Region Proposal Network (RPN)
- Shares Conv layer
- RPN takes feature maps and generates object proposals and objectness score



https://www.analyticsvidhya.com/blog/2018/10/a-step-by-step-introduction-to-the-basic-object-detection-algorithms-part-1/

FasterRCNN Steps

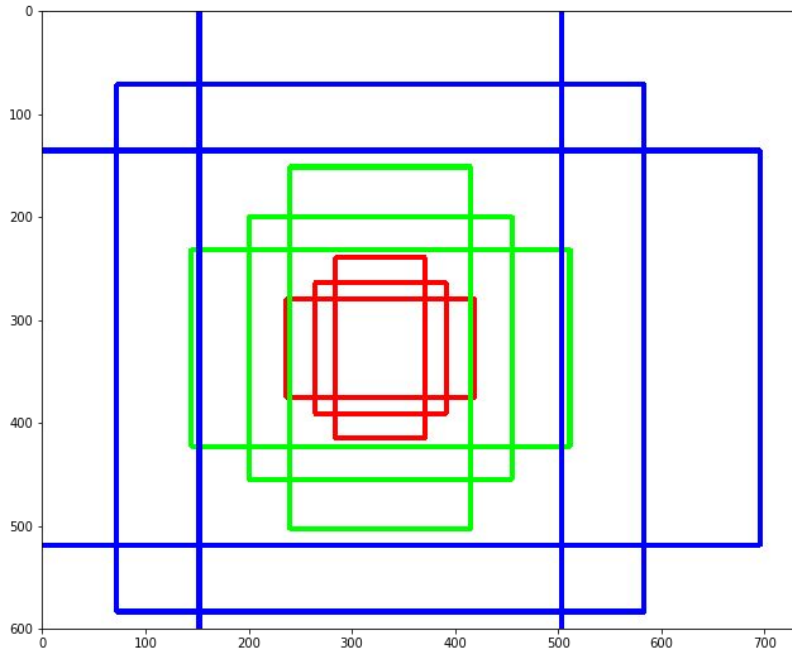
1. Image as input
2. Pass to CNN to generate feature maps
3. RPN applied to feature maps
4. RoI Pooling applied to proposals and feature map
5. Resized proposals passed to FC layer
6. Output: classification and bounding box



https://www.analyticsvidhya.com/blog/2018/10/a-step-by-step-introduction-to-the-basic-object-detection-algorithms-part-1/

Anchors/Prior Boxes

- Default: 3 scales x 3 aspect ratios
  - K = 9 anchors
- Total number of anchors is W*H*K for stride of one



https://medium.com/@smallfishbigsea/faster-r-cnn-explained-864d4fb7e3f8

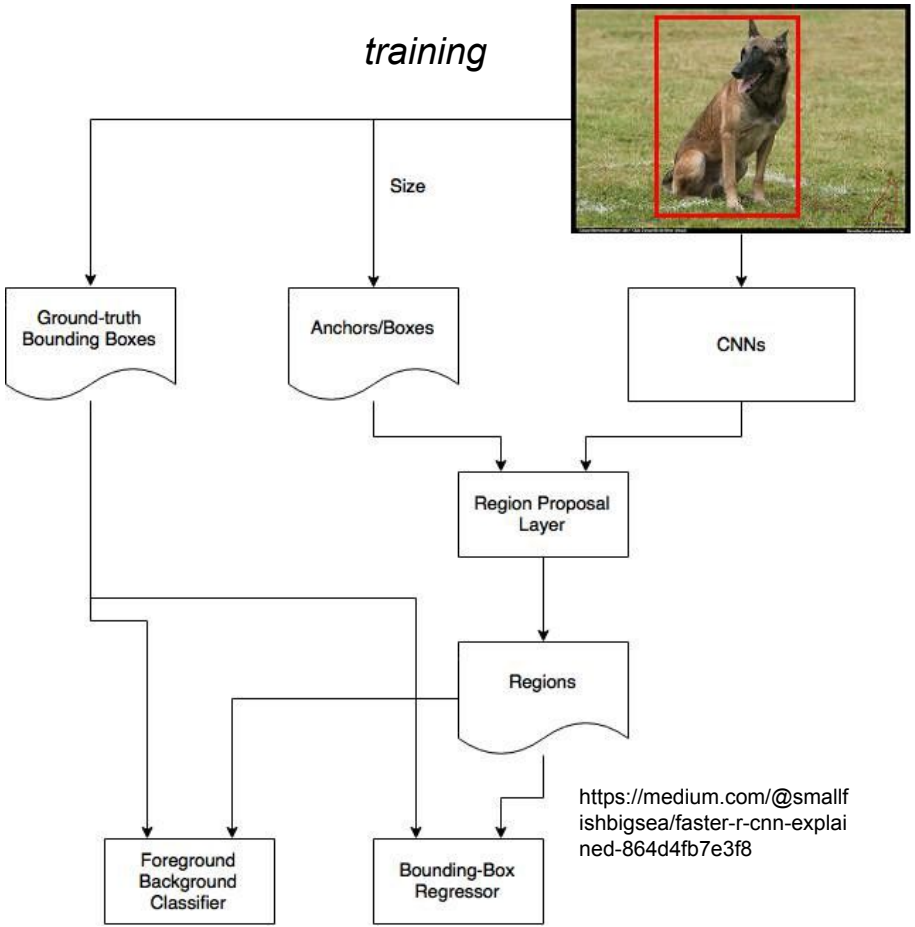
FROM: `_extract_rpn_feature_maps(self,preprocessed_inputs)`

```
973       image_shape = tf.shape(preprocessed_inputs)
974
975       rpn_features_to_crop, self.endpoints = (
976           self._feature_extractor.extract_proposal_features(
977               preprocessed_inputs,
978               scope=self.first_stage_feature_extractor_scope))
979
980       feature_map_shape = tf.shape(rpn_features_to_crop)
981       anchors = box_list_ops.concatenate(
982           self._first_stage_anchor_generator.generate([(feature_map_shape[1],
983                                                         feature_map_shape[2])]))
```

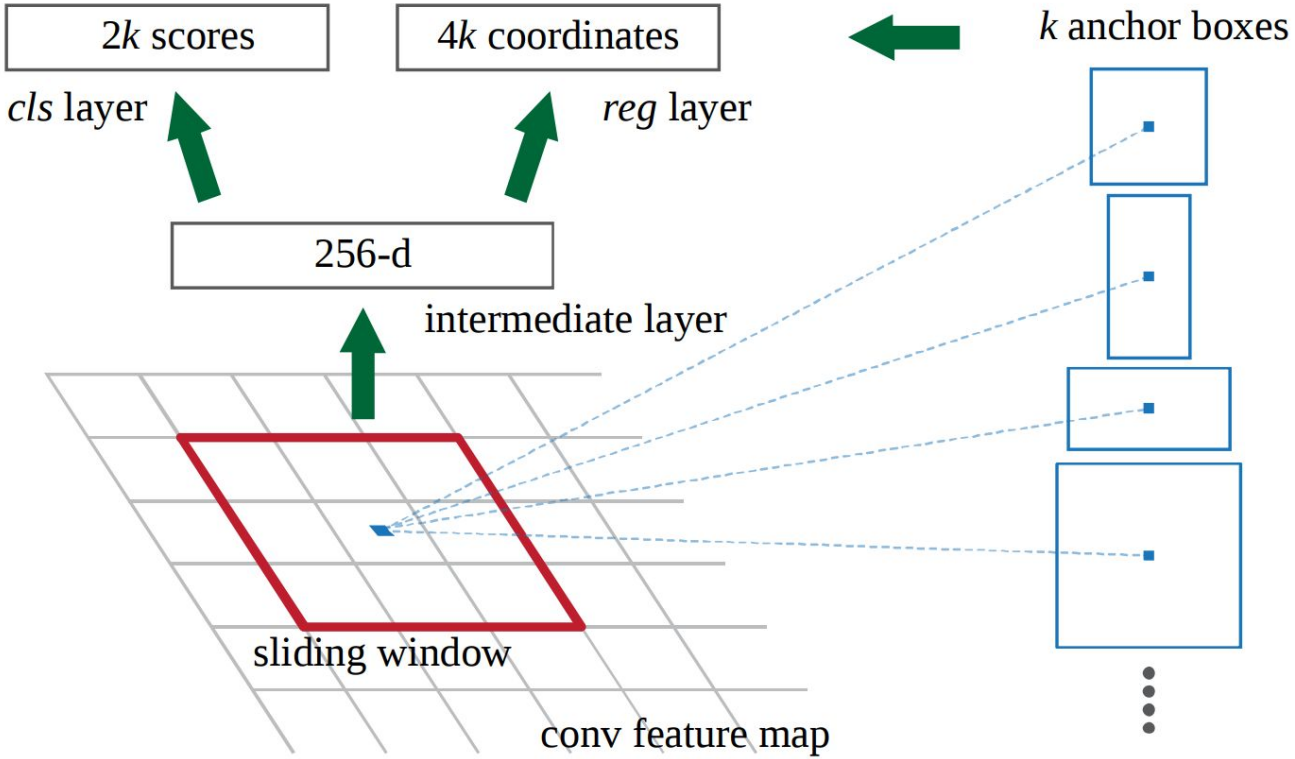
When anchor boxes are generated

Region Proposal Network (RPN)

- Small ConvNet looking at the global feature volume of the CNN
- Slide spatial window over conv feature map
- Predicts Probability of anchor being object/background
  - Training labels: label anchors with higher overlaps with ground-truth boxes as foreground/object, the ones with lower overlaps as background
- Refines anchor (Regressor)
- Receptive fields of different Anchors may overlap
  - Property: translational invariant



Sliding Window Visualized



W. Liu, et al.

TF RPN Code

```
570     def predict(self, preprocessed_inputs, true_image_shapes):
571       """Predicts unpostprocessed tensors from input tensor.
572
573       This function takes an input batch of images and runs it through the
574       forward pass of the network to yield "raw" un-postprocessed predictions.
575       If `number_of_stages` is 1, this function only returns first stage
576       RPN predictions (un-postprocessed).  Otherwise it returns both
577       first stage RPN predictions as well as second stage box classifier
578       predictions.
579
```

Inputs:

Args:

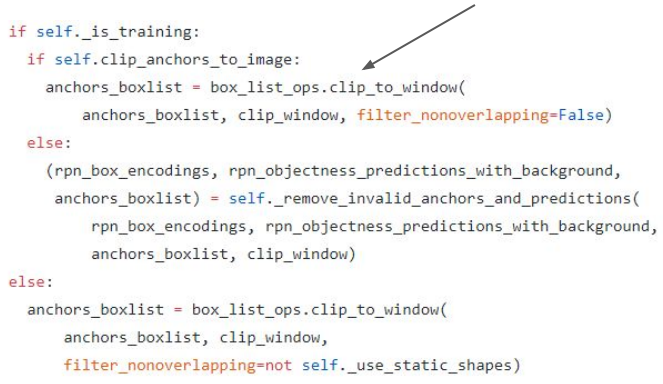
- preprocessed_inputs: [batch, height, width, channels] float tensor
- true_image_shapes: int32 tensor of shape [batch, 3] where, row is [height, width, channels] indicating the shapes of true images in the resized images

RPN Code cont

The two main components of the RPN: generating anchors and probabilities for the anchors

```
646       (rpn_box_predictor_features, rpn_features_to_crop, anchors_boxlist,
647        image_shape) = self._extract_rpn_feature_maps(preprocessed_inputs)
648       (rpn_box_encodings, rpn_objectness_predictions_with_background
649       ) = self._predict_rpn_proposals(rpn_box_predictor_features)
```

Note, there are boxes removed and clipped after this step as preprocessing for the next step



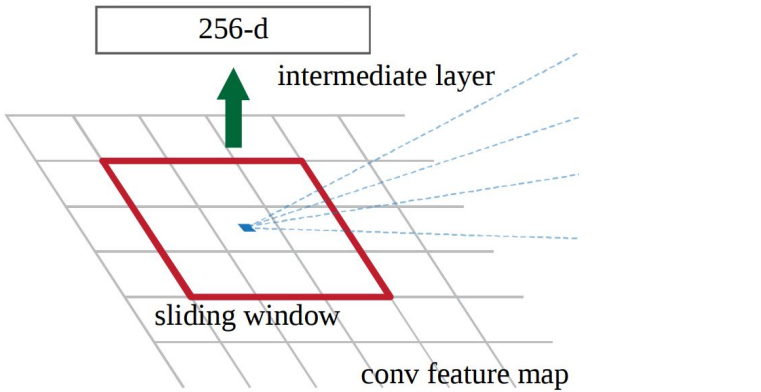
```
654     if self._is_training:
655       if self.clip_anchors_to_image:
656         anchors_boxlist = box_list_ops.clip_to_window(
657             anchors_boxlist, clip_window, filter_nonoverlapping=False)
658       else:
659         (rpn_box_encodings, rpn_objectness_predictions_with_background,
660          anchors_boxlist) = self._remove_invalid_anchors_and_predictions(
661              rpn_box_encodings, rpn_objectness_predictions_with_background,
662              anchors_boxlist, clip_window)
663     else:
664       anchors_boxlist = box_list_ops.clip_to_window(
665           anchors_boxlist, clip_window,
666           filter_nonoverlapping=not self._use_static_shapes)
```

Deeper look at Generating Regions

```
951     def _extract_rpn_feature_maps(self, preprocessed_inputs):
952       """Extracts RPN features.
953
954       This function extracts two feature maps: a feature map to be directly
955       fed to a box predictor (to predict location and objectness scores for
956       proposals) and a feature map from which to crop regions which will then
957       be sent to the second stage box classifier.
```

Generate Anchors

```
981       anchors = box_list_ops.concatenate(
982           self._first_stage_anchor_generator.generate([(feature_map_shape[1],
983                                                         feature_map_shape[2])]))
```



Apply the sliding window (conv layer)

```
984       with slim.arg_scope(self._first_stage_box_predictor_arg_scope_fn()):
985         kernel_size = self._first_stage_box_predictor_kernel_size
986         reuse = tf.get_variable_scope().reuse
987         rpn_box_predictor_features = slim.conv2d(
988             rpn_features_to_crop,
989             self._first_stage_box_predictor_depth,
990             kernel_size=[kernel_size, kernel_size],
991             rate=self._first_stage_atrous_rate,
992             activation_fn=tf.nn.relu6,
993             scope='Conv',
994             reuse=reuse)
995       return (rpn_box_predictor_features, rpn_features_to_crop,
996               anchors, image_shape)
```

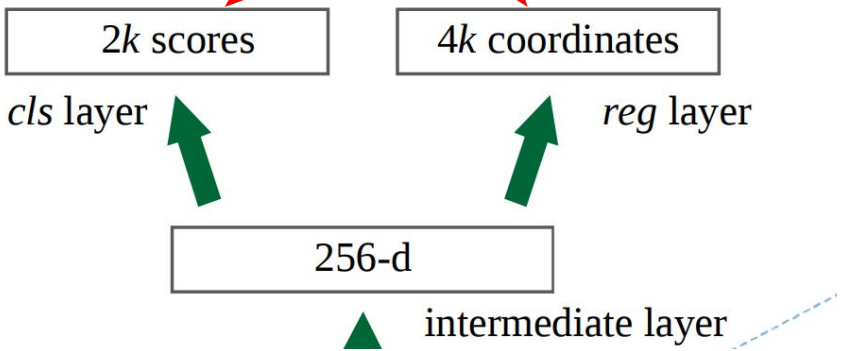
Deeper look at Generating Regions Cont

```
998     def _predict_rpn_proposals(self, rpn_box_predictor_features):
999       """Adds box predictors to RPN feature map to predict proposals.
```

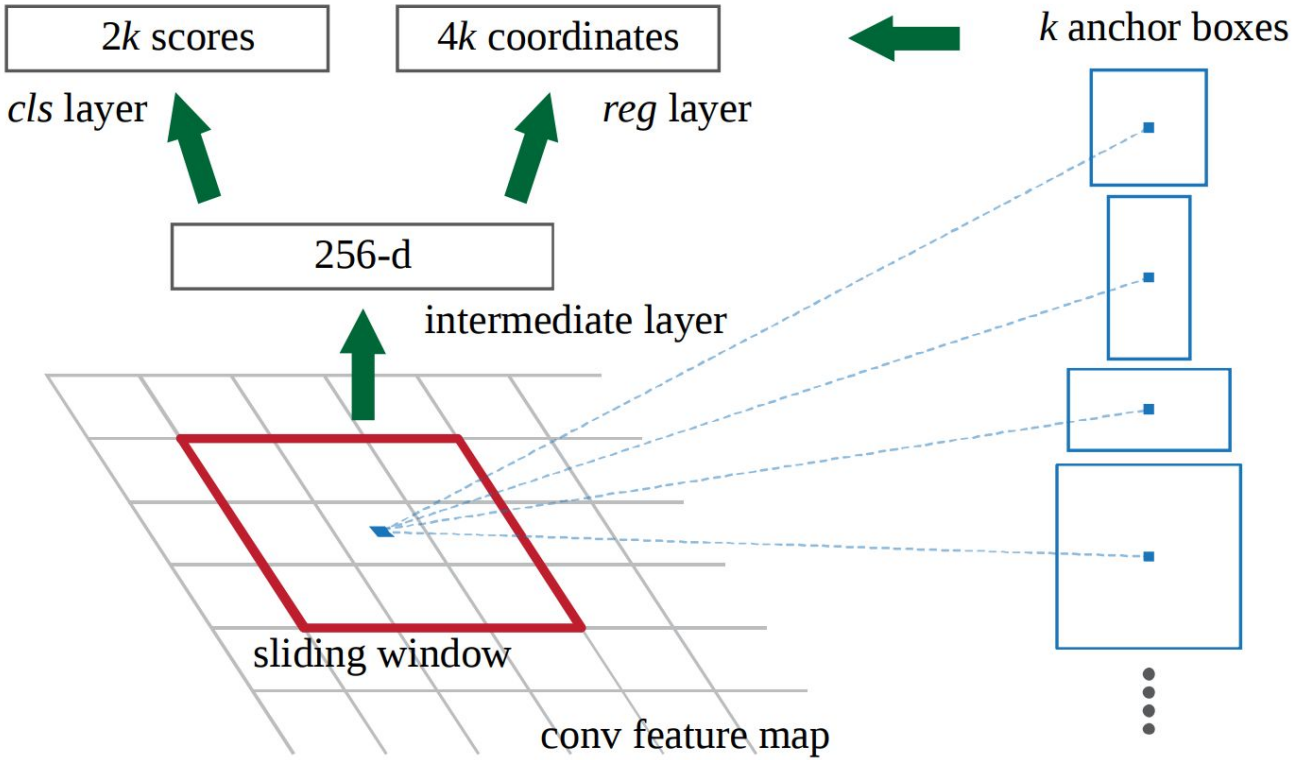
Perform the box regression and objectness step of RPN

```
1027        if self._first_stage_box_predictor.is_keras_model:
1028          box_predictions = self._first_stage_box_predictor(
1029              [rpn_box_predictor_features])
```

```
1036        box_encodings = tf.concat(
1037            box_predictions[box_predictor.BOX_ENCODINGS], axis=1)
1038        objectness_predictions_with_background = tf.concat(
1039            box_predictions[box_predictor.CLASS_PREDICTIONS_WITH_BACKGROUND],
1040            axis=1)
```



RPN Visualized



W. Liu, et al.

RPN Code cont., Returns

prediction_dict: a dictionary holding "raw" prediction tensors:

1) rpn_box_predictor_features: [batch_size, height, width, depth] for predicting proposal boxes and corresponding objectness scores.

2) rpn_features_to_crop: [batch_size, height, width, depth] representing image features to crop using the proposal boxes predicted by the RPN.

3) image_shape: a 1-D tensor of shape [4] representing the input image shape.

4) rpn_box_encodings: [batch_size, num_anchors, self._box_coder.code_size] containing predicted boxes.

5) rpn_objectness_predictions_with_background: [batch_size, num_anchors, 2] containing class predictions (logits) for each of the anchors. *includes* background class predictions (at class index 0).

6) anchors: [num_anchors, 4] representing anchors for the first stage RPN (in absolute coordinates).

RPN Code cont., Returns

Second Stage of FasterRCNN, but can be performed at different times according to this architecture

7) refined_box_encodings: [total_num_proposals, num_classes, self._box_coder.code_size] representing predicted (final) refined box encodings, where total_num_proposals=batch_size*self._max_num_proposals.

8) class_predictions_with_background: [total_num_proposals, num_classes + 1] containing class predictions (logits) for each of the anchors *includes* background class predictions (at class index 0).

9) num_proposals: [batch_size] number of proposals generated by the RPN. `num_proposals` allows us to keep track of which entries are to be treated as zero paddings and which are not since we always pad the number of proposals to be `self.max_num_proposals` for each image.

10) proposal_boxes: [batch_size, self.max_num_proposals, 4] representing decoded proposal bounding boxes in absolute coordinates.

11) mask_predictions: (optional) a 4-D tensor with shape [total_num_padded_proposals, num_classes, mask_height, mask_width] containing instance mask predictions.

RPN Loss

- combination of the classification loss and the regression loss

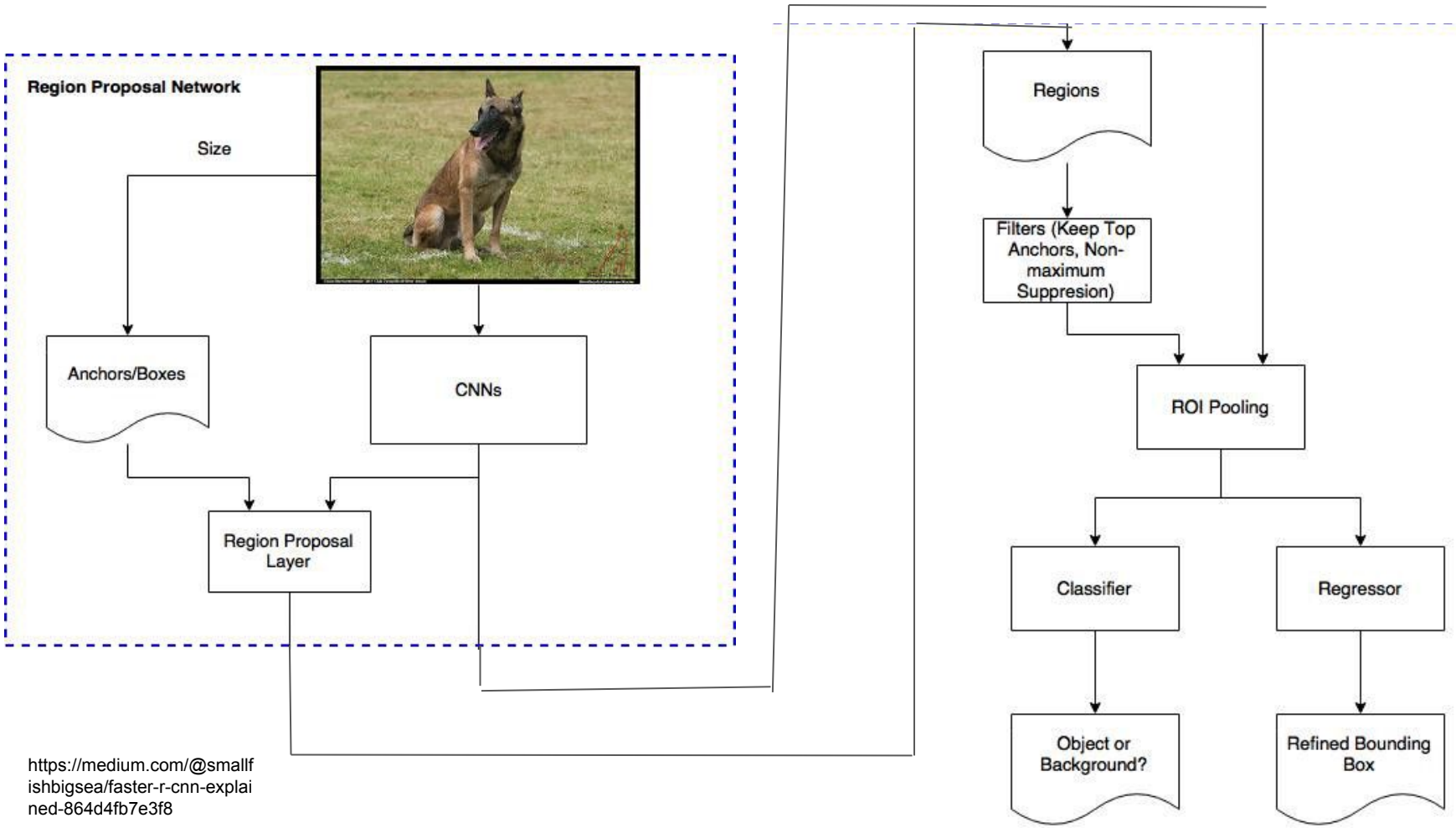
Symbol	Explanation
p_i	Predicted probability of anchor i being an object.
p_i^*	Ground truth label (binary) of whether anchor i is an object.
t_i	Predicted four parameterized coordinates.
t_i^*	Ground truth coordinates.
N_{cls}	Normalization term, set to be mini-batch size (~256) in the paper.
N_{box}	Normalization term, set to the number of anchor locations (~2400) in the paper.
λ	A balancing parameter, set to be ~10 in the paper (so that both $\mathcal{L}_{\text{cls}}$ and $\mathcal{L}_{\text{box}}$ terms are roughly equally weighted).

$$\mathcal{L} = \mathcal{L}_{\text{cls}} + \mathcal{L}_{\text{box}}$$

$$\mathcal{L}(\{p_i\}, \{t_i\}) = \frac{1}{N_{\text{cls}}} \sum_i \mathcal{L}_{\text{cls}}(p_i, p_i^*) + \frac{\lambda}{N_{\text{box}}} \sum_i p_i^* \cdot L_1^{\text{smooth}}(t_i - t_i^*)$$

$$\mathcal{L}_{\text{cls}}(p_i, p_i^*) = -p_i^* \log p_i - (1 - p_i^*) \log(1 - p_i)$$

https://lilianweng.github.io/lil-log/2017/12/31/object-recognition-for-dummies-part-3.html#loss-function-1



https://medium.com/@smallfishbigsea/faster-r-cnn-explained-864d4fb7e3f8

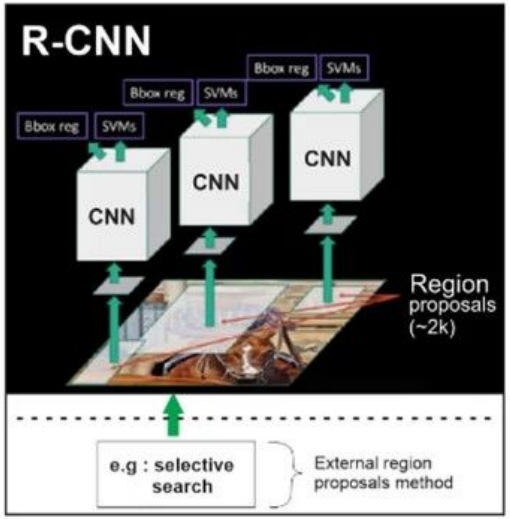
Why is this better than the other sliding window technique?

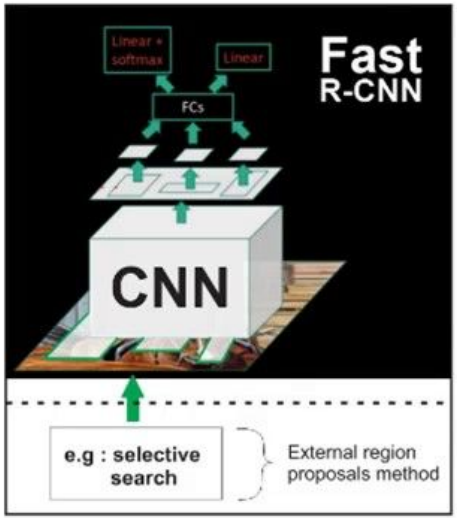
- fatal mistake of the previous sliding-windows is that we use the windows as the final boundary boxes.
  - need too many shapes to cover most objects.
- more effective solution is to treat the window as an initial guess.
  - Then we have a detector to predict the class and the boundary box from the current sliding window simultaneously.

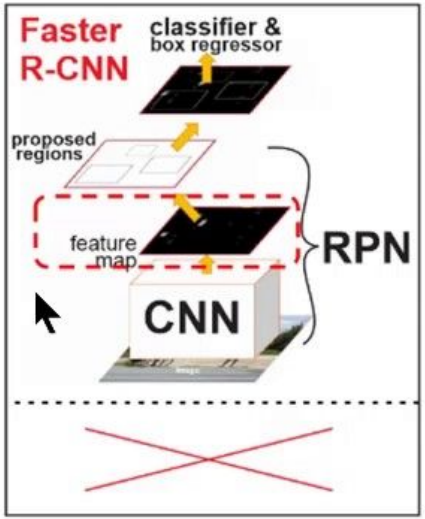
https://medium.com/@jonathan_hui/what-do-we-learn-from-single-shot-object-detectors-ssd-yolo-fpn-focal-loss-3888677c5f4d

Problems with FasterRCNN

- Multiple passes over the image
  - First, get the proposals using the sliding window RPN and a objectness predictor
  - Second, refine the boxes by using the conv layer again in the ROI pooling layer and use another classifier and regressor to output the final predictions



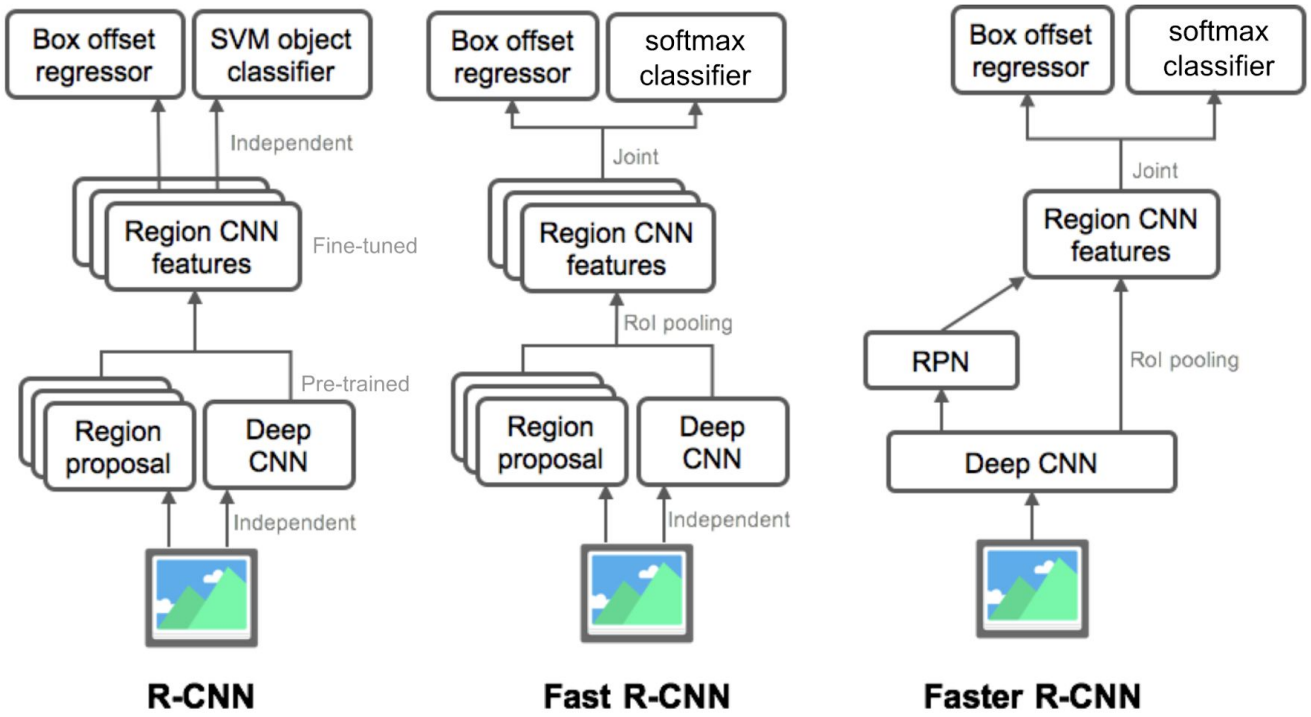




	R-CNN	Fast R-CNN	Faster R-CNN
Test time per image	50 seconds	2 seconds	0.2 seconds
Speed-up	1x	25x	250x
mAP (VOC 2007)	66.0%	66.9%	66.9%

* Standford lecture notes on CNN by Fei Fei Li and Andrej Karpathy

By Ardian Umam



https://lilianweng.github.io/lil-log/2017/12/31/object-recognition-for-dummies-part-3.html#loss-function-1

SSD

Direct Classification VS. Refined Classification

- Direct classification
  - YOLO, SSD
  - Regresses prior box and classifies object directly from same input region
  - Faster but less accurate (features used to classify are not extracted exactly)
- Refined Classification
  - FasterRCNN, MaskRCNN
  - First regresses prior box for refined bounding box, then pools features of refined box to form common feature volume and classify object by these features
  - Slower but more accurate

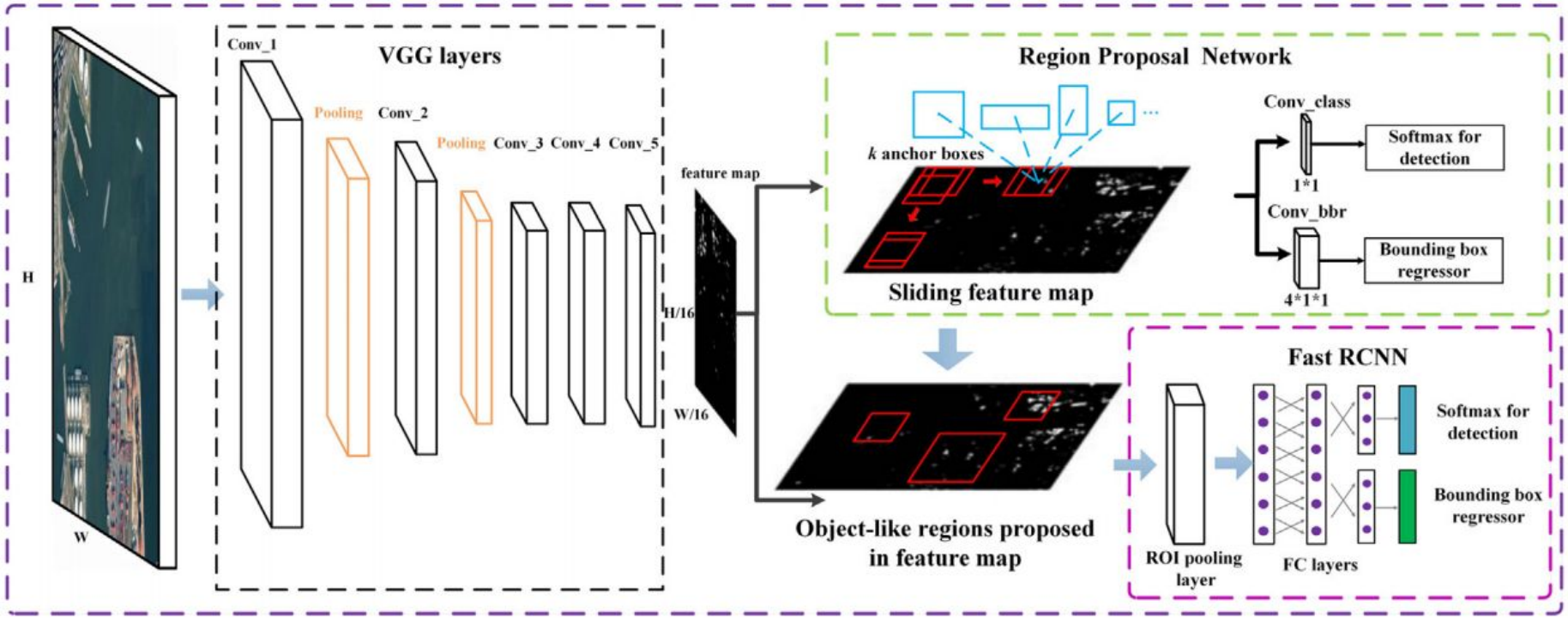


Fig. 2. The architecture of Faster R-CNN.

https://www.researchgate.net/publication/324903264_Multi-scale_object_detection_in_remote_sensing_imagery_with_convolutional_neural_networks

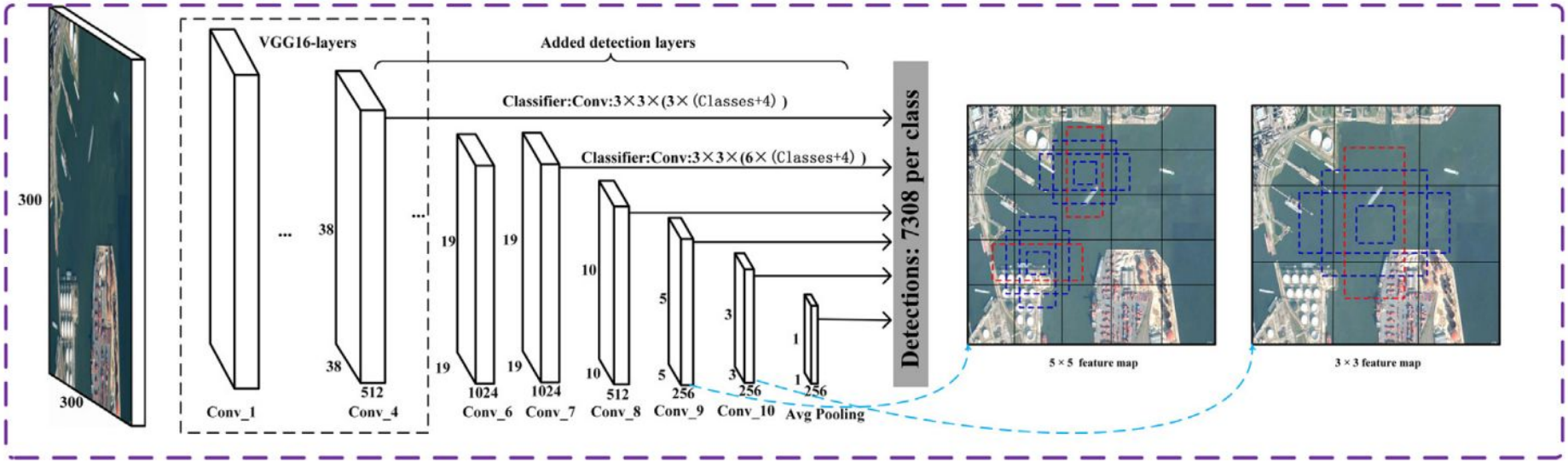
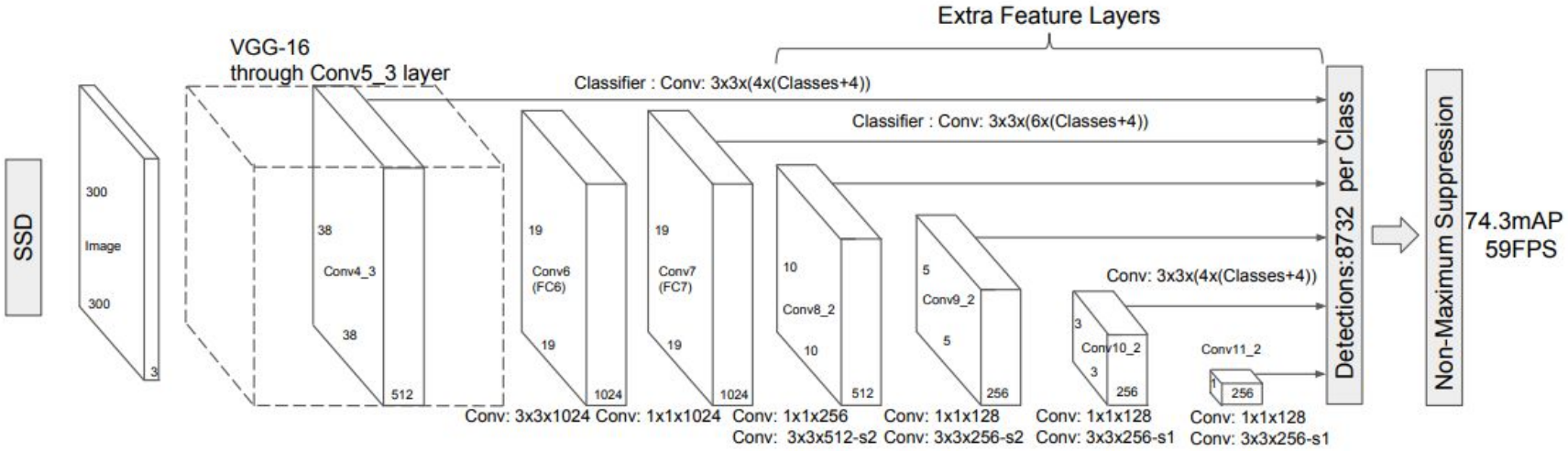


Fig. 5. The architecture of SSD.

https://www.researchgate.net/publication/324903264_Multi-scale_object_detection_in_remote_sensing_imagery_with_convolutional_neural_networks

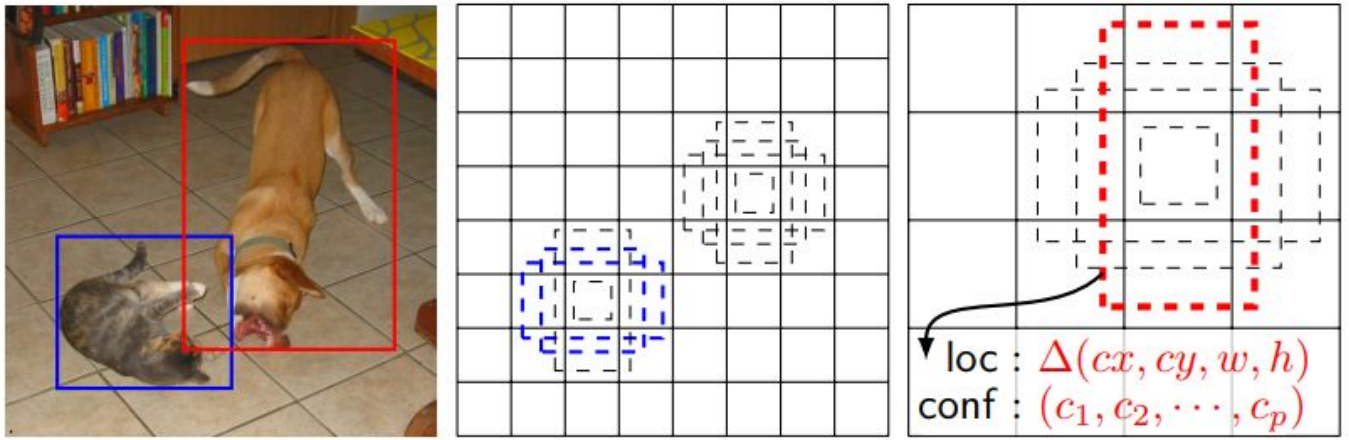
SingleShot Multibox Detector (SSD) 2016

- SingleShot = classification/localization in one forward pass
- Multibox = bounding box regression technique
- directly classify object of each prior box instead of scoring object confidence
- Run classification/localization on multiple conv layers at different depth levels



SSD default boxes

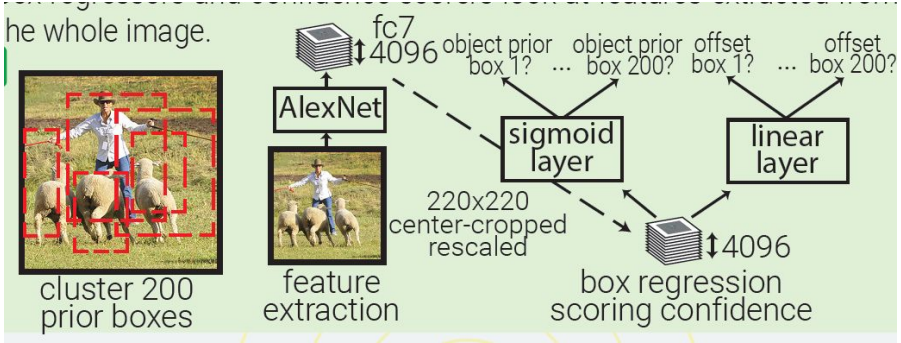
- Eliminates bounding box proposals and resampling of features stage
- “predict category scores and box offsets for a fixed set of default bounding boxes using small convolutional filters applied to feature maps”
- “For each default box, we predict both the shape offsets and the confidences for all object categories”



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

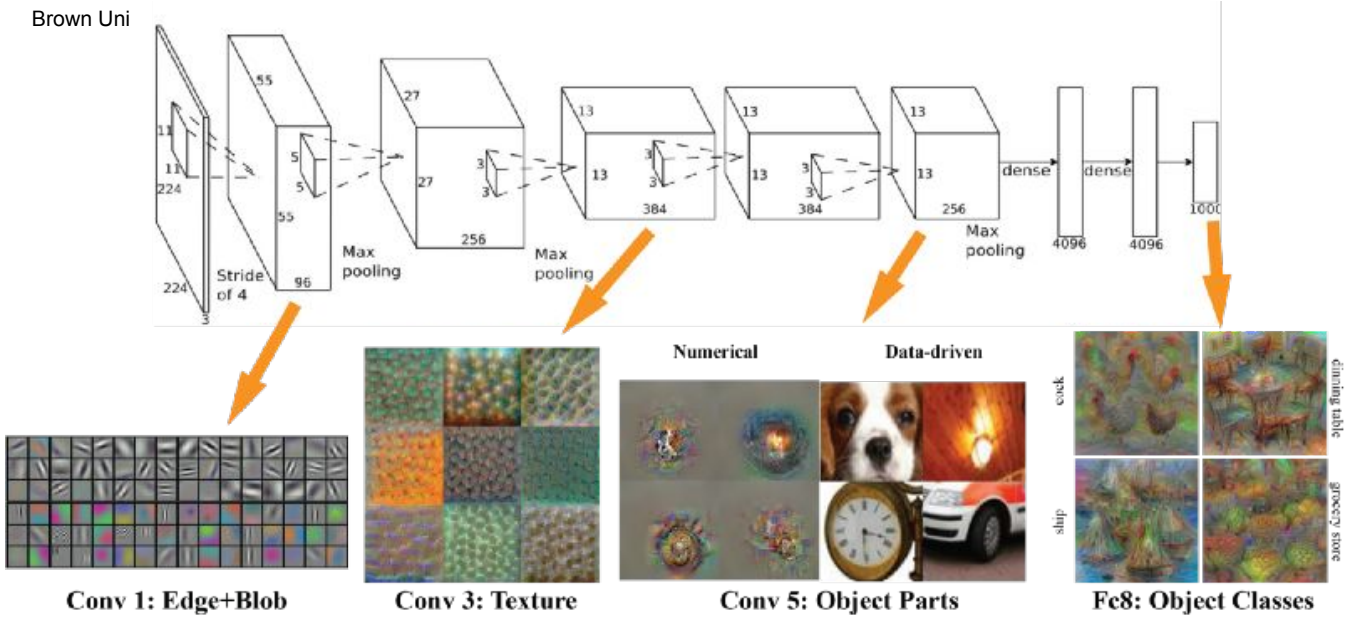
Multibox (2014)

- Not object recognition, but a proposal network
- Clusters ground truth boxes in dataset to find 200 centroids as prior box centers
- Output layers: sigmoid layer for object confidence, linear layer for offset
- SSD differences
  - Fixed priors - set of default bounding boxes
  - Location loss - smooth L1 instead of L2 (to avoid aiming for pixel perfect)
  - SSD performs classification on regions



https://medium.com/@nikasa1889/the-modern-history-of-object-recognition-infographic-aea18517c318

Brown Uni



- Running multibox on different conv layers increases the chance of finding an object whether it is large or small

SSD Code

```
522     def predict(self, preprocessed_inputs, true_image_shapes):
523       """Predicts unpostprocessed tensors from input tensor.
524
525       This function takes an input batch of images and runs it through the forward
526       pass of the network to yield unpostprocessesed predictions.
527
```

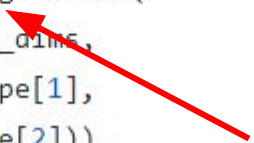
First, use a CNN to create feature volumes at different levels

```
558       if self._feature_extractor.is_keras_model:
559         feature_maps = self._feature_extractor(preprocessed_inputs)
560       else:
```

SSD Code cont

Next generate priors/anchors/etc. that are projected onto the feature maps at different levels

```
570       feature_map_spatial_dims = self._get_feature_map_spatial_dims(
571           feature_maps)
572       image_shape = shape_utils.combined_static_and_dynamic_shape(
573           preprocessed_inputs)
574       self._anchors = box_list_ops.concatenate(
575           self._anchor_generator.generate(
576               feature_map_spatial_dims,
577               im_height=image_shape[1],
578               im_width=image_shape[2]))
```



SSD Code Cont

Then make predictions on the boxes projected onto the feature maps

```
579       if self._box_predictor.is_keras_model:
580         predictor_results_dict = self._box_predictor(feature_maps)
```

Apply postprocessing to refine the results

```
def postprocess(self, prediction_dict, true_image_shapes):
  """Converts prediction tensors to final detections.

  This function converts raw predictions tensors to final detection results by
  slicing off the background class, decoding box predictions and applying
  non max suppression and clipping to the image window.
```

SSD Loss

- weighted sum between localization loss (Smooth L1) and confidence loss (Softmax)

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

$$L_{loc}(x, l, g) = \sum_{i \in Pos}^{N} \sum_{m \in \{cx, cy, w, h\}} x_{ij}^{k} \text{smooth}_{\text{L1}}(l_i^m - \hat{g}_j^m)$$

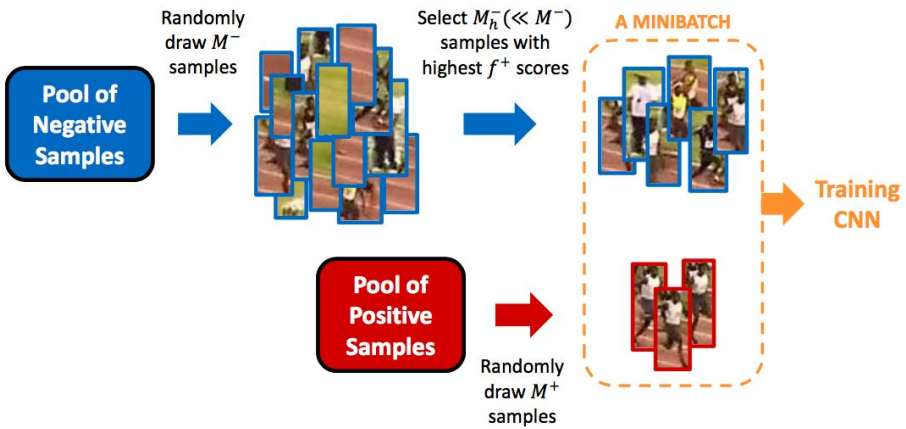
$$\hat{g}_j^{cx} = (g_j^{cx} - d_i^{cx})/d_i^w \qquad \hat{g}_j^{cy} = (g_j^{cy} - d_i^{cy})/d_i^h$$

$$\hat{g}_j^w = \log\left(\frac{g_j^w}{d_i^w}\right) \qquad \hat{g}_j^h = \log\left(\frac{g_j^h}{d_i^h}\right)$$

$$L_{conf}(x, c) = -\sum_{i \in Pos}^{N} x_{ij}^p log(\hat{c}_i^p) - \sum_{i \in Neg} log(\hat{c}_i^0) \quad \text{where} \quad \hat{c}_i^p = \frac{\exp(c_i^p)}{\sum_p \exp(c_i^p)}$$

Note on Negative Mining

- Generally most anchors/priors will have low IoU
- Leads to imbalance in training data towards making negative predictions
- Rebalance to 3:1 ratio, negative:positive



https://jamiekang.github.io/2017/05/28/faster-r-cnn/

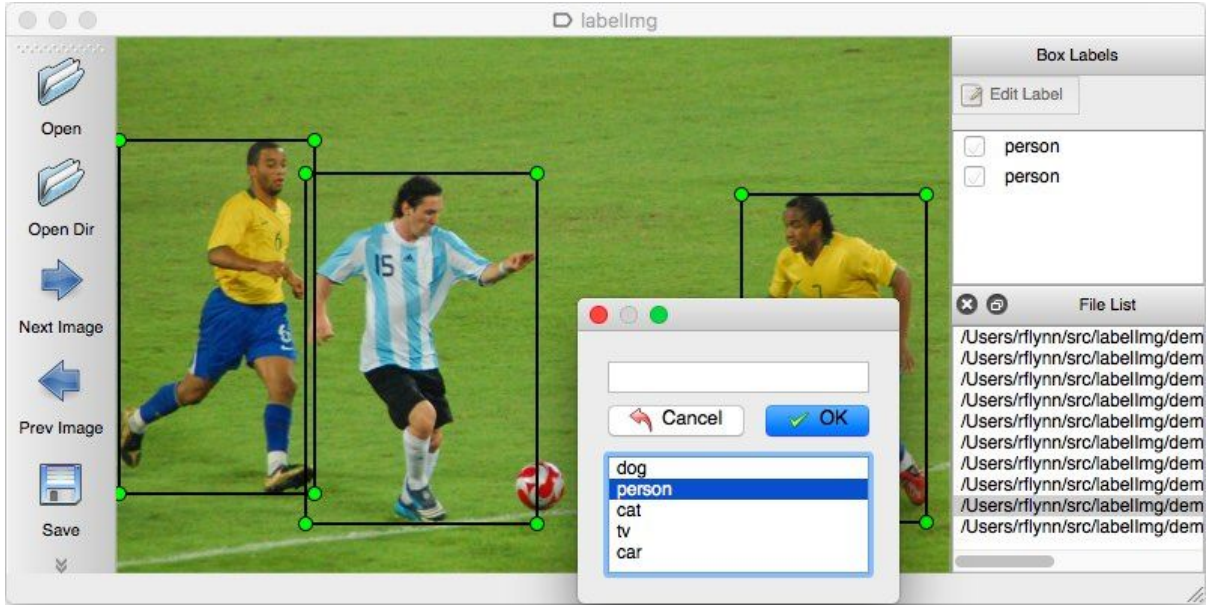
Other notes

- Authors of SSD recommend data augmentation
  - Horizontal flipping, patches of original picture at different ratios
- Prune the large number of boxes with NMS
  - Discard: confidence < 0.01, IoU < 0.45
- More priors per feature map cell = more accurate but slower

Custom Object Detection

How to create labels for your data?

LabelImg = graphical image annotation tool and label object bounding boxes in images



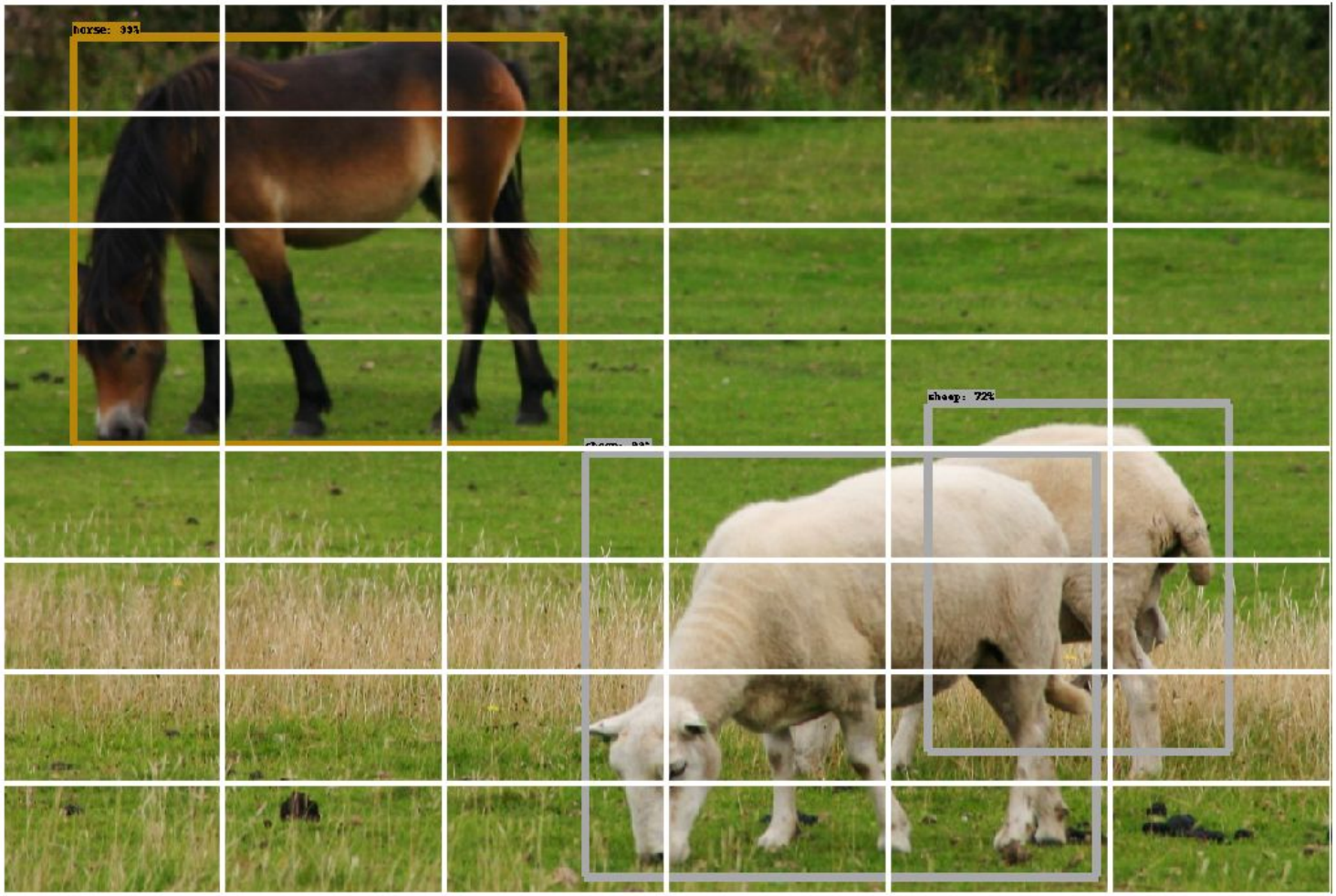
Tutorial / Assignment

- https://github.com/EdjeElectronics/TensorFlow-Object-Detection-API-Tutorial-Train-Multiple-Objects-Windows-10
  - This tutorial covers how to set up the TF Object Detection API for windows 10
  - Provides different scripts that help with the custom object tasks
- Assignment is set up on Google Colab to run code from object_detection_tutorial.py from the API
  - https://github.com/David-Hughes3/CS5990_project/tree/master/assignment

Results

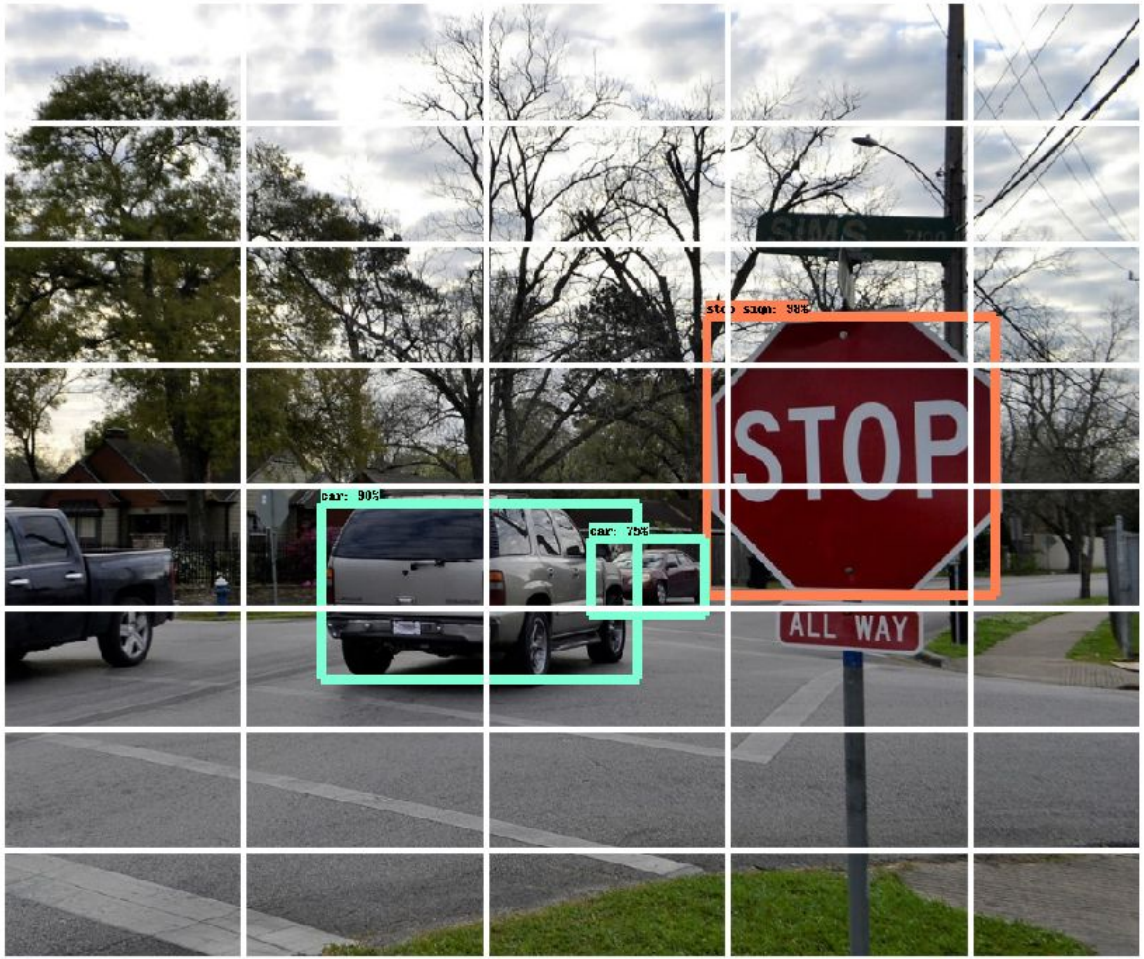
Model:
Ssd_mobilenet_v1_coco_
2017_11_17

Time: 3.3052144050598145



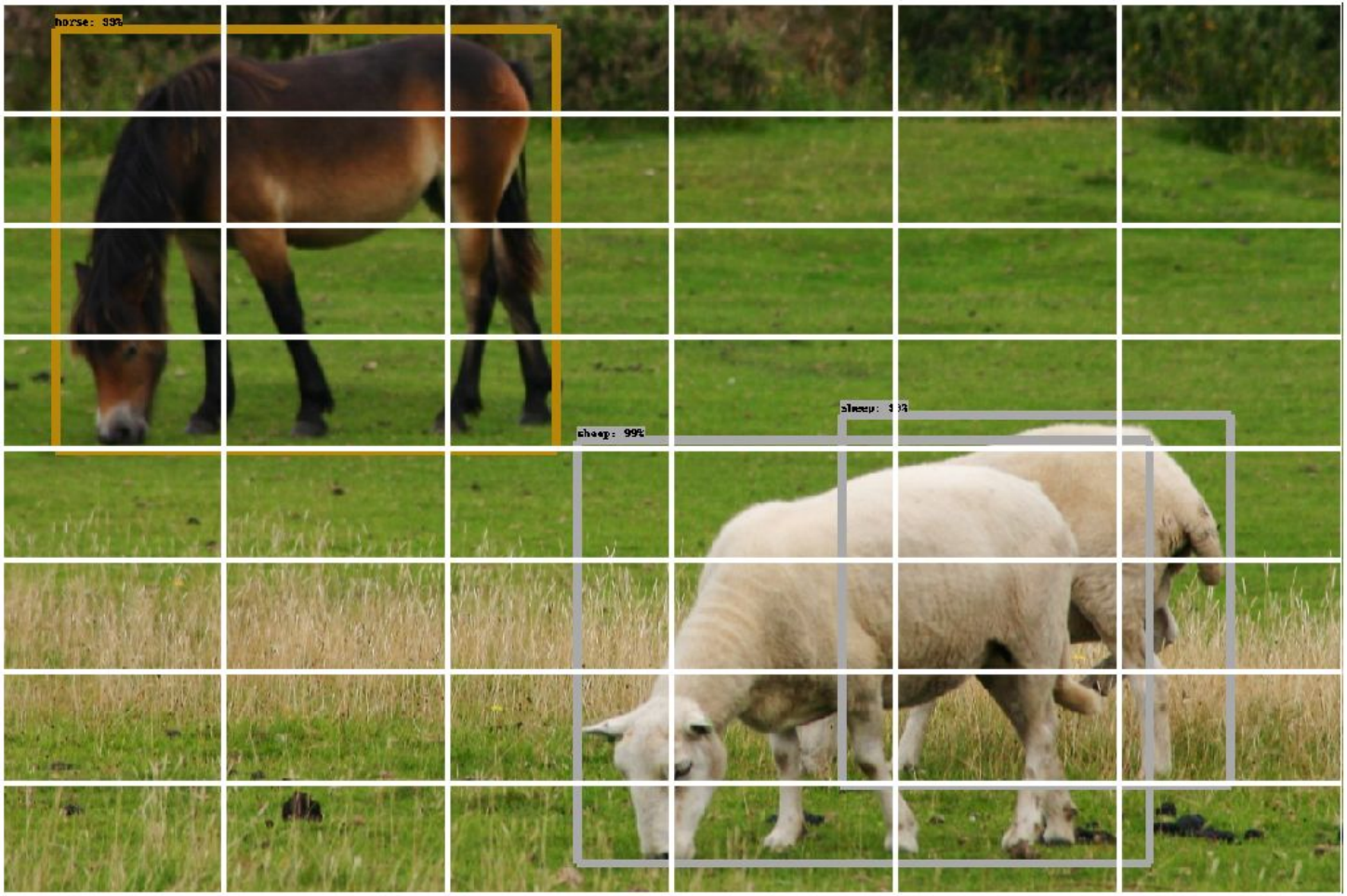
Model:
Ssd_mobilenet_v1_coco_
2017_11_17

Time: 3.1751675605773926



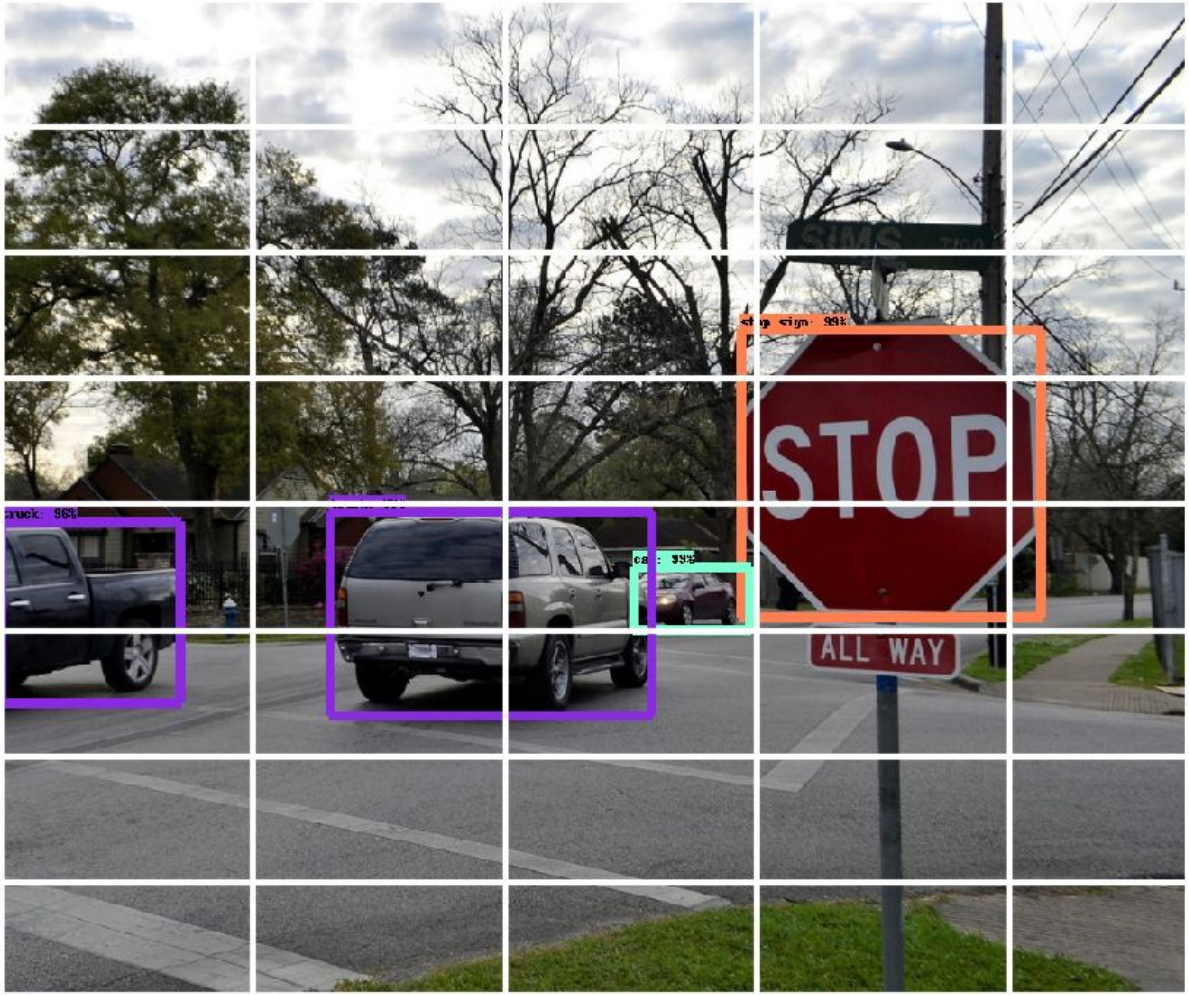
Model: Faster_rcnn_resnet101_coco_2018_01_28

Time: 9.9892797470092



Model:
Faster_rcnn_resnet101_coco_2018_01_28

Time: 10.854303359985352



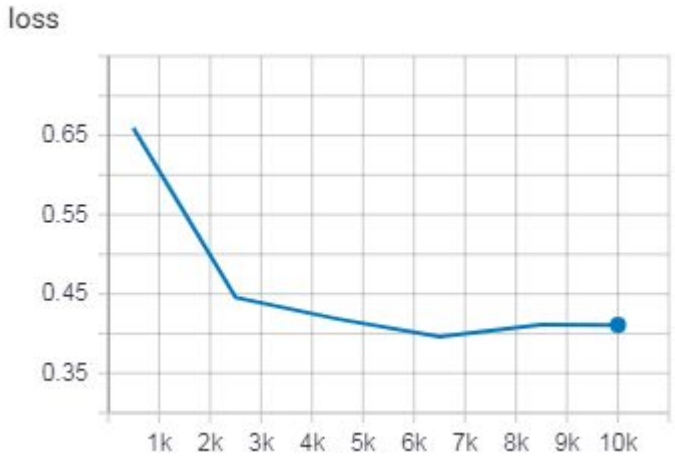
Results of custom training

Google Colab Results with model_main.py

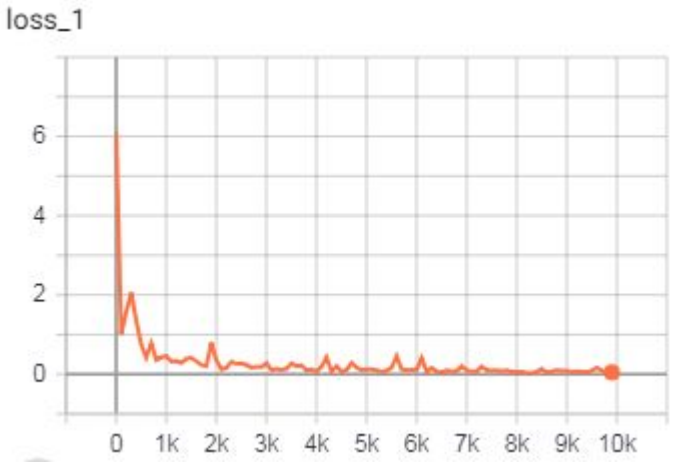
10000 steps

[EdjeElectronics playing card data]

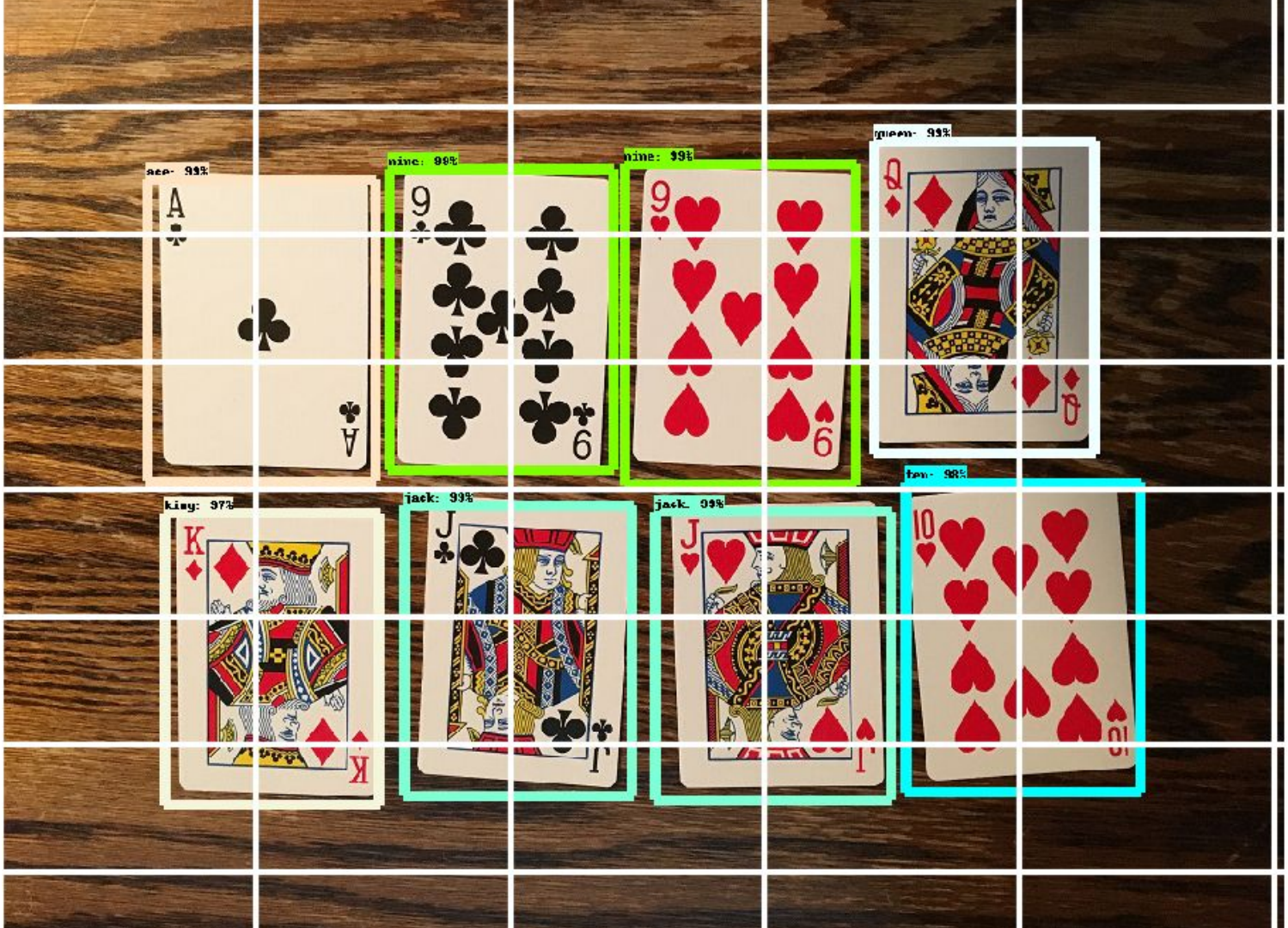
Model: faster_rcnn_inception_v2_coco_2018_01_28 from TF Model Zoo



Validation Loss



Training Loss


ace: 99%
nine: 99%
nine: 99%
queen: 99%
king: 97%
jack: 99%
jack: 99%
ten: 98%

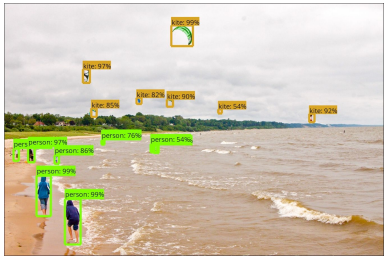
Automatic Custom Object Detection

Future work

Typically

Already one pipeline in FasterRCNN and SSD

New Data:
Webcam
Images
Video
etc

Gather Data → Train Model → Run Detector → 

=

Take Images
+
Draw bounding boxes by hand

=

Train
Evaluate
Optimize

Automatic Custom Object Detector

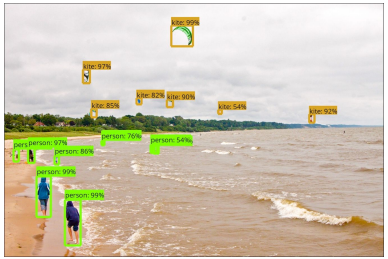
New Data:
Webcam
Images
Video
etc

Automatically generate data then train model
in one end-to-end custom object detector

Generate Data → Train Model → Run Detector

Take Images
Build a 3d model (2d -> 3d)
Superimpose model onto backgrounds (3d -> 2d)
Build bounding box from image mask

Train
Evaluate
Optimize



References

[1] V. S. Chandel.Selective search for object detection(c++/python), Sep 2017.

[2] EdjeElectronics. Edjeelectronics/tensorflow-object-detection-api-tutorial-train-multiple-objects-windows-10, Sep 2018.

[3] P.-Y. Gao, M. Farraj, and Y.-L. Tsou. Automated data prepa-ration for custom object detection.

[4] W. Liu, D. Anguelov, D. Erhan, C. Szegedy, S. Reed, C.-Y. Fu,and A. C. Berg. Ssd: Single shot multibox detector. InEuro-pean conference on computer vision, pages 21–37. Springer,2016.

[5] pkulzc, V. Rathod, and N. Wu. Tensorflow detection modelzoo, Dec 2018.

[6] S. Ren, K. He, R. Girshick, and J. Sun. Faster r-cnn: Towardsreal-time object detection with region proposal networks. InAdvances in neural information processing systems, pages91–99, 2015.

[7] Tensorflow. Tensorflow object detection api, Sep 2018.

https://medium.com/@nikasa1889/the-modern-history-of-object-recognition-infographic-aea18517c318