FAST R-CNN

➢ What is Fast R-CNN….?

One of the deep-learning object detectors among YOLO, Single Shot Detectors(SSDs).

I. Why object Classification is a more challenging one?

II.

Object detection is a more challenging task than image classification, requiring more complex methods to solve. Due to this complexity, current approaches train models in multi-stage pipelines that are slow and inelegant.

Complexity arises due to:

- Numerous candidate object locations (often called “proposals”) must be processed.
- These candidates provide only rough localization that must be refined to achieve precise localization

III. What’s the need for the development of Fast R-CNN?

In 2014, a group of researchers at UC Berkeley developed a deep convolutional network called **R-CNN** (short for region-based convolutional neural network) that can detect 80 different types of objects in images. It achieves excellent object detection accuracy by using a deep ConvNet to classify object proposals. However, it has notable drawbacks:

- It is a multi-stage model, where each stage is an independent component. Thus, it cannot be trained end-to-end.
- It caches the extracted features from the pre-trained CNN on the disk to later train the SVMs. This requires hundreds of gigabytes of storage.
- R-CNN depends on the Selective Search algorithm for generating region proposals, which takes a lot of time. Moreover, this algorithm cannot be customized to the detection problem.
- Each region proposal is fed independently to the CNN for feature extraction. This makes it impossible to run R-CNN in real time.

Some Common Terms:

- ✓ Spatial pyramid pooling networks (SPPnets)

  A single convolution layer or a set of convolution layers takes an image and produces a feature map proportional to a particular ratio (called the sub-sampling ratio) w.r.t the input image. But for a fully connected layer, the input has to be a fixed-length vector. The authors replaced the last pooling layer (the one just before the FC layer) with a Spatial Pyramid Pooling(SPP) layer to overcome this issue.

  In this layer, we had set up fix stride (of 2) and fixed window size (of [2 x 2]). Because of this, our output is always proportional to the input.
  Now **if we make the pooling window and stride proportional to the input image, we can always get a fixed-sized output.** Moreover, the SPP layers do not just apply one pooling operation, it applies a couple of different output-sized pooling operations (that's where the same comes from — Spatial Pyramid Pooling) and combine the results before sending them to the next layer.

  These were proposed to speed up R-CNN by sharing computation. SPPnet accelerates R-CNN by 10 to 100× at test time. Training time is also reduced by 3× due to faster proposal feature extraction. SPPnet also has notable drawbacks:

  - Like R-CNN, training is a multi-stage pipeline that involves extracting features, fine-tuning a network with log loss, training SVMs, and finally fitting bounding-box regressors. Features are also written to disk.
  - Unlike R-CNN, the fine-tuning algorithm proposed cannot update the convolutional layers that precede the spatial pyramid pooling(which limits the accuracy of very deep networks

- ✓ Precision:

  Measures how accurate are your predictions i.e the percentage of the predictions your correct

- ✓ Recall:

  How good you find all the p

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

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

TP = True positive

TN = True negative

FP = False positive

FN = False negative

- ✓ AP is averaged over all categories. Traditionally, this is called "mean average precision" (mAP). We make no distinction between AP and mAP (and likewise AR and mAR) and assume the difference is clear from the context.
- ✓ Training R-CNN is a multi-state pipeline which means it is a more unwieldy model to build and means sufficient training data is needed at several stages.
- ✓ Fine-tuning refers to using the weights of an already trained network as the starting values for training a new network
- ✓ Forward propagation in convolution operation consists of overlapping filter weights on the input, multiplying them, and summing the results to get the output.
- ✓ Backpropagation is just a way of propagating the total loss back into the neural network to know how much of the loss every node is responsible for and subsequently updating the weights in a way that minimizes the loss by giving the nodes with higher error rates lower weights, and vice versa.
- ✓ Stochastic Gradient Descent(SGD) :

  In SGD, instead of using the entire dataset for each iteration, only a single random training example (or a small batch) is selected to calculate the gradient and update the model parameters. This random selection introduces randomness into the optimization process, hence the term "stochastic" in stochastic Gradient Descent

The advantage of using SGD is its computational efficiency, especially when dealing with large datasets. By using a single example or a small batch, the computational cost per iteration is significantly reduced compared to traditional Gradient Descent methods that require processing the entire dataset.

➢ Fast R-CNN Architecture & Training

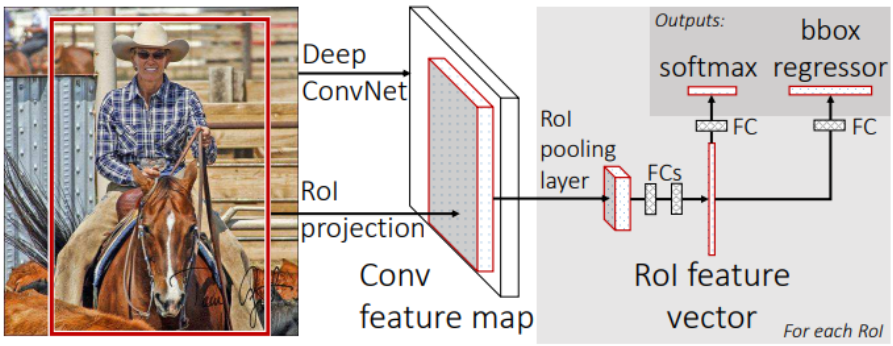
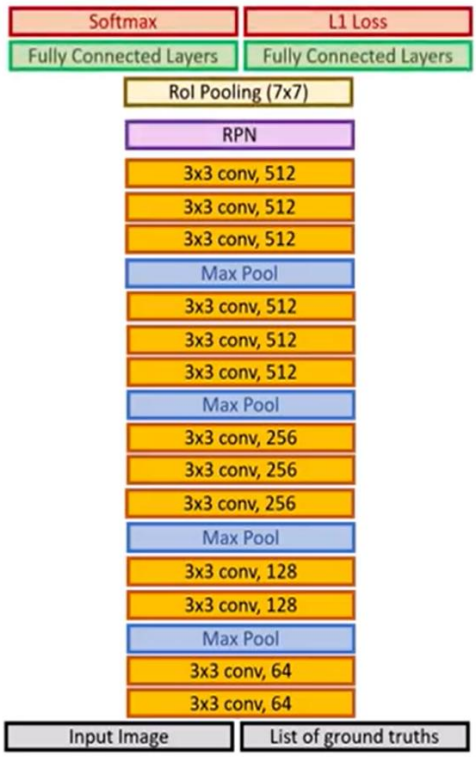
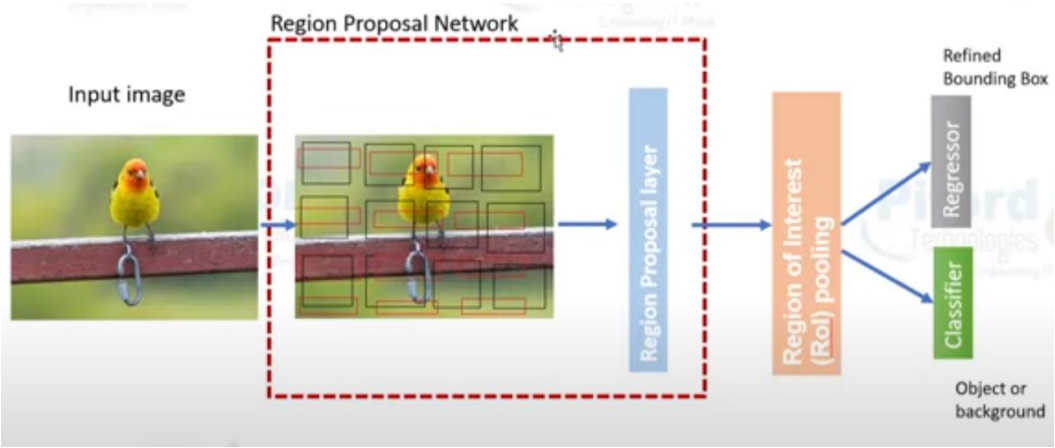


Figure 1. Fast R-CNN architecture. An input image and multiple regions of interest (RoIs) are input into a fully convolutional network. Each RoI is pooled into a fixed-size feature map and then mapped to a feature vector by fully connected layers (FCs). The network has two output vectors per RoI: softmax probabilities and per-class bounding-box regression offsets. The architecture is trained end-to-end with a multi-task loss.

- ✓ A Fast R-CNN network takes as input an entire image and a set of object proposals
- ✓ The network first processes the whole image with several convolutional (conv) and max pooling layers to produce a conv feature map
- ✓ Then, for each object proposal a region of interest (RoI) pooling layer extracts a fixed-length feature vector from the feature map.
- ✓ Each feature vector is fed into a sequence of fully connected (FC) layers that finally branch into two sibling output layers: one that produces softmax probability estimates over K object classes plus a catch-all “background” class and another layer that outputs four real-valued numbers for each of the K object classes. Each set of 4 values encodes refined bounding-box positions for one of the K classes





Background Class: The area where the object is not present
Foreground Class: The area where the object is present(will move to the next stage of the algorithm)

Region Proposal Network:

- Generate anchor boxes
  Anchor boxes: A set of predefined bounding boxes of some height and width. Size can vary according to the object
- Find out IoU(Intersection over union – measures the overlap between 2 boundaries)

If IoU is>=0.5 => the object will be detected by the box
(*Note: Higher IoU will be labeled as foreground class*
IoU<0.5 => background class)

The target of RPN is to predict foreground and background anchor boxes
The output of the Region Proposal Network will be the feature maps of those anchor boxes which are labeled as foreground classes

How to train a model……?

Region of Interest (RoI pooling layer):

- The RoI pooling layer, a Spatial pyramid Pooling (SPP) technique is the main idea behind Fast R-CNN and the reason that it outperforms R-CNN in accuracy and speed respectively
- RoI pooling layers divide a (h,w) rectangular window into an H×W set of sub-windows of approximate size h/H × w/W, and afterward perform max-pooling in each sub-window. The RoI pooling layer performs a max pooling operation in any proposed RoI of an image individually**.**

Classifier:

- Find out whether there's an object or background in an image

Regressor:

- To draw a bounding box on the object which we classified(refine the bounding box)

Initializing from pre-trained networks

When a pre-trained network initializes a Fast R-CNN network, it undergoes three transformations:

- The last max pooling layer is replaced by an RoI pooling layer that is configured by setting H and W to be compatible with the net's first fully connected layer (e.g., H = W = 7 for VGG16).
- The network's last fully connected layer and softmax (which were trained for 1000-way ImageNet classification) are replaced with the two sibling

layers described earlier (a fully connected layer and softmax over K + 1 categories and category-specific bounding-box regressors).

- The network is modified to take two data inputs: a list of images and a list of RoIs in those images.

Fine-Tuning for Detection

Here we need to know how we sample region proposals out of all possible outputs given by selective search and what's the exact loss function

? Why SPPnet is unable to update weights below the spatial pyramid pooling layer…?

> The root cause is that back-propagation through the SPP layer is highly inefficient when each training sample (i.e. RoI) comes from a different image, which is exactly how R-CNN and SPPnet networks are trained. The inefficiency stems from the fact that each RoI may have a very large receptive field, often spanning the entire input image. Since the forward pass must process the entire receptive field, the training inputs are large (often the entire image).

? Which layers to fine-tune?

- ✓ For the less deep networks considered in the SPPnet paper, fine-tuning only the fully connected layers appeared to be sufficient for good accuracy. We hypothesized that this result would not hold for very deep networks.
- ✓ To validate that fine-tuning the conv layers is important for VGG16, we use Fast R-CNN to fine-tune, but freeze the thirteen conv layers so that only the fully connected layers learn
- ✓ This ablation emulates single-scale SPPnet training and decreases mAP from 66.9% to 61.4%. This experiment verifies our hypothesis: training through the RoI pooling layer is important for very deep nets

Mini-Batch Sampling

- ✓ During fine-tuning, each SGD mini-batch is constructed from N = 2 images, chosen uniformly at random
- ✓ We use mini-batches of size R = 128, sampling 64 RoIs from each image. (Selective search gives us 2000 RoIs and we're sampling just 64 of them- The reason is: We take 25% of the RoIs from object proposals that have intersection over union (IoU) overlap with a ground truth bounding box of at least 0.5. These RoIs comprise the examples labeled with a foreground object class, i.e. u ≥ 1. The remaining RoIs are sampled from object proposals that have a maximum IoU with ground truth in the interval [0.1, 0.5). These are the background examples and are labeled with u(ground truth class) = 0. The lower threshold of 0.1 appears to act as a heuristic for hard example mining. During training, images are horizontally flipped with a probability of 0.5. No other data augmentation is used.)

Multi-task loss

- ✓ A Fast R-CNN network has two sibling output layers:
  - p = (p0, . . . , pK), over K + 1 categories => discrete probability distribution(per RoI)

  p is computed by a softmax over the K(object classes)+1(background classes) outputs of a fully connected layer.

  - $t^k = (t^k_{\mathrm{x}}, t^k_{\mathrm{y}}, t^k_{\mathrm{w}}, t^k_{\mathrm{h}})$ **=>** bounding box regression offsets for each of the K object classes, indexed by k

  t^k specifies a scale-invariant translation and log-space height/width shift relative to an object proposal.
  ground-truth bounding-box regression target = v

- ✓ $L(p, u, t^u, v) = L_{\mathrm{cls}}(p, u) + \lambda[u \geq 1] L_{\mathrm{loc}}(t^u, v)$

Lcls(p, u) = − log pu is log loss for true class u
Lloc is defined over a tuple of true bounding-box regression targets for class u, v = (vx, vy, vw, vh), and a predicted tuple $t^u = (t^u_{\mathrm{x}}, t^u_{\mathrm{y}}, t^u_{\mathrm{w}}, t^u_{\mathrm{h}})$, again for class u.

- ✓ The Iverson bracket indicator function [u ≥ 1] evaluates to 1 when u ≥ 1 and 0 otherwise. By convention, the catch-all background class is labeled u = 0. For background RoIs, there is no notion of a ground-truth bounding box and hence Lloc is ignored. For bounding-box regression, we use the loss

$$L_{\text{loc}}(t^u, v) = \sum_{i \in \{\text{x,y,w,h}\}} \text{smooth}_{L_1}(t_i^u - v_i),$$

in which

$$\text{smooth}_{L_1}(x) = \begin{cases} 0.5x^2 & \text{if } |x| < 1 \\ |x| - 0.5 & \text{otherwise,} \end{cases}$$

$$L(p, u, t^u, v) = L_{\text{cls}}(p, u) + \lambda[u \geq 1] L_{\text{loc}}(t^u, v),$$

- $L_{\text{cls}}(p, u) = -\log p_u$ $\quad (L_{CE} = -\sum_i p_i^* log p_i)$

  True class distribution — Predicted class distribution

- $[u \geq 1]$ evaluates to 1 when $u \geq 1$ and 0 otherwise.
- $L_{\text{loc}}(t^u, v) = \sum_{i \in \{\text{x,y,w,h}\}} \text{smooth}_{L_1}(t_i^u - v_i),$

  in which

$$\text{smooth}_{L_1}(x) = \begin{cases} 0.5x^2 & \text{if } |x| < 1 \\ |x| - 0.5 & \text{otherwise,} \end{cases}$$

Back-Propagation through RoI pooling layers

Backpropagation routes derivatives through the RoI pooling layer. For clarity, we assume only one image per mini-batch (N = 1), though the extension to N > 1 is straightforward because the forward pass treats all images independently

$$\frac{\partial L}{\partial x_i} = \sum_r \sum_j [i = i^*(r, j)] \frac{\partial L}{\partial y_{rj}}.$$

In words, for each mini-batch RoI r and for each pooling output unit yrj, the partial derivative ∂L/∂yrj is accumulated if i is the arg max selected for yrj by max pooling. In back-propagation, the partial derivatives ∂L/∂yrj are already computed by the backward function of the layer on top of the RoI pooling layer

SGD hyper-parameters

- ✓ The fully connected layers used for softmax classification and bounding-box regression are initialized from zero-mean Gaussian distributions with standard deviations of 0.01 and 0.001, respectively.
- ✓ Biases are initialized to 0. All layers use a per-layer learning rate of 1 for weights and 2 for biases and a global learning rate of 0.001.
- ✓ When we train on larger datasets, we run SGD for more iterations, as described later. A momentum of 0.9 and parameter decay of 0.0005 (on weights and biases) are used.

Scale Invariance:

There're 2 ways of achieving scale-invariant object detection:

- ✓ Brute Force
  Each image is processed at a pre-defined pixel size during both training and testing. The network must directly learn scale-invariant object detection from the training data.

- ✓ Multi-Scale approach
  The multi-scale approach, in contrast, provides approximate scale-invariance to the network through an image pyramid. At test-time, the image pyramid is used to approximately scale-normalize each object proposal. During multi-scale training, we randomly sample a pyramid scale each time an image is sampled, as a form of data augmentation. We experiment with multi-scale training for smaller networks only, due to GPU memory

From the observation, we found out that the multi-scale approach offers only a small increase in mAP at a large cost in computing time

Fast R-CNN Detection

- ✓ Once a Fast R-CNN network is fine-tuned, detection amounts to little more than running a forward pass (assuming object proposals are pre-computed). The network takes as input an image (or an image pyramid, encoded as a list of images) and a list of R object proposals to score
- ✓ At test-time, R is typically around 2000, although we will consider cases in which it is larger (≈ 45k). When using an image pyramid, each RoI is assigned to the scale such that the scaled RoI is closest to 2242 pixels in the area
- ✓ For each test RoI r, the forward pass outputs a class posterior probability distribution p and a set of predicted bounding-box offsets relative to r (each of the K classes gets its own refined bounding-box prediction). We assign a detection confidence to r for each object class k using the estimated probability Pr(class = k | r) Δ= pk. We then perform non-maximum suppression independently for each class using the algorithm and settings from R-CNN

  Non-maximum Suppression: We select the predictions with the maximum confidence and suppress all the other predictions having overlap with the selected predictions greater than a threshold. In other words, we take the maximum and suppress the non-maximum ones, hence the name non-maximum suppression.

SVD(Singular Value Decomposition)

SVD is basically a matrix factorization technique, which decomposes any matrix into 3 generic and familiar matrices

Singular Value Decomposition (SVD)

$$W_{m\times n}=\begin{bmatrix}\vec{w}_1 & \vec{w}_2 & \dots & \vec{w}_n\end{bmatrix}=U_{m\times m}\Sigma_{m\times n}V^T_{n\times n}$$

$\sigma_1 \geq \sigma_2 \geq \dots \geq \sigma_n$

$$=\begin{bmatrix}\vec{u}_1 & \vec{u}_2 & \dots & \vec{u}_m\end{bmatrix}\begin{bmatrix}\sigma_1 & 0 & \cdots & 0 & 0\\ 0 & \sigma_2 & 0 & 0 & 0\\ \vdots & 0 & \ddots & 0 & 0\\ 0 & 0 & 0 & \sigma_n & 0\\ 0 & 0 & 0 & 0 & 0\end{bmatrix}\begin{bmatrix}\vec{v}_1 & \vec{v}_2 & \dots & \vec{v}_n\end{bmatrix}^T$$

1. $m > n$ Coefficients of $\vec{u}_i$ where $i > n$ would be zero.
2. $m < n$ σ_i where $i > m$ would be zero.

Recall from Linear Algebra:

$$\begin{bmatrix}\vec{a}_1 & \vec{a}_2 & \dots & \vec{a}_n\end{bmatrix}_{m\times n}\begin{bmatrix}d_1 & 0 & 0 & 0\\ 0 & d_2 & 0 & 0\\ 0 & 0 & \ddots & 0\\ 0 & 0 & 0 & d_n\end{bmatrix}_{n\times n}=\begin{bmatrix}d_1\vec{a}_1 & d_2\vec{a}_2 & \dots & d_n\vec{a}_n\end{bmatrix}_{m\times}$$

Truncated SVD for faster detection

- ✓ In this technique, a layer parameterized by the u × v weight matrix W is approximately factorized as

$$W \approx U\Sigma_t V^T$$

using SVD

$$W_{m\times n}=\begin{bmatrix}\vec{w}_1 & \vec{w}_2 & \dots & \vec{w}_n\end{bmatrix}=U_{m\times m}\Sigma_{m\times n}V^T_{n\times n}$$

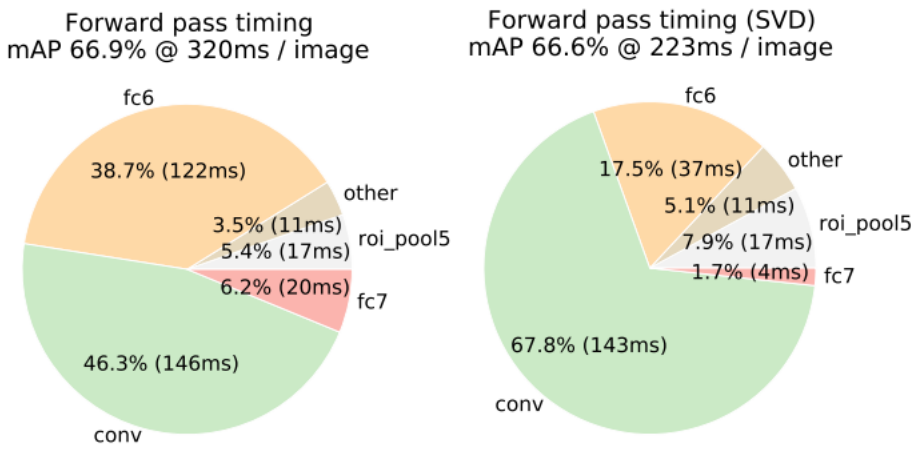
$$=\begin{bmatrix}\vec{u}_1 & \vec{u}_2 & \dots & \vec{u}_m\end{bmatrix}\begin{bmatrix}\sigma_1 & 0 & \cdots & 0 & 0\\ 0 & \sigma_2 & 0 & 0 & 0\\ \vdots & 0 & \ddots & 0 & 0\\ 0 & 0 & 0 & \sigma_n & 0\\ 0 & 0 & 0 & 0 & 0\end{bmatrix}\begin{bmatrix}\vec{v}_1 & \vec{v}_2 & \dots & \vec{v}_n\end{bmatrix}^T$$

If σ_i is less than a certain value, then it means the corresponding column on matrix U is not very important

We can simply round it to zero!

- ✓ In this factorization, U is a u × t matrix comprising the first t left-singular vectors of W, Σt is a t × t diagonal matrix containing the top t singular values of W, and V is v × t matrix comprising the first t right-singular vectors of W. Truncated SVD reduces the parameter count from uv to t(u + v), which can be significant if t is much smaller than min(u, v). To compress a network, the single fully connected

layer corresponding to W is replaced by two fully connected layers, without a non-linearity between them. The first of these layers uses the weight matrix ΣtV T (and no biases) and the second uses U (with the original biases associated with W). This simple compression method gives good speedups when the number of RoIs is large.



- ✓ Truncated SVD can reduce detection time by more than 30% with only a small (0.3 percentage point) drop in mAP and without needing to perform additional fine-tuning after model compression

Advantages:

- ☐ Fast training and testing compared to R-CNN, SPPnet
- ☐ No disk storage is required for feature caching
- ☐ Higher detection quality (mAP) than R-CNN, SPPnet
- ☐ Training is single-stage, using a multi-task loss
- ☐ Training can update all network layers
- ☐ Fine-tuning conv layers in VGG16 improves mAP

State-of-the-art mAP on VOC07(mAP range is from 63.1% to 66.9%), 2010(achieves a higher mAP than Fast R-CNN (67.2% vs. 66.1%)), and 2012(mAP of 65.7% (and 68.4% with extra data))

Disadvantages:

- ☐ One drawback of Faster R-CNN is that the RPN is trained where all anchors in the mini-batch, of size 256, are extracted from a single image. Because

all samples from a single image may be correlated (i.e. their features are similar), the network may take a lot of time until reaching convergence.

VOC 2010 and 2012 results

- ✓ Fast R-CNN achieves the top result on VOC12 with an mAP of 65.7% (and 68.4% with extra data)
- ✓ On VOC10, SegDeepM achieves a higher mAP than Fast R-CNN (67.2% vs. 66.1%)
- ✓ SegDeepM is trained on VOC12 train-val plus segmentation annotations; it is designed to boost R-CNN accuracy by using a Markov random field to reason over R-CNN detections and segmentations from the O2P semantic segmentation method
- ✓ Fast R-CNN can be swapped into SegDeepM in place of R-CNN, which may lead to better results.
- ✓ Fast R-CNN's mAP increases to 68.8%, surpassing SegDeepM

VOC 2007 results

- ✓ All methods start from the same pre-trained VGG16 network and use bounding-box regression.
- ✓ SPPnet uses five scales during both training and testing. The improvement of Fast R-CNN over SPPnet illustrates that even though Fast R-CNN uses single-scale training and testing, fine-tuning the conv layers provides a large improvement in mAP (from 63.1% to 66.9%).
- ✓ R-CNN achieves a mAP of 66.0%.

Training and testing time

- ✓ For VGG16, Fast R-CNN processes images 146× faster than R-CNN without truncated SVD and 213× faster with it. Training time is reduced by 9×, from 84 hours to 9.5.
- ✓ Compared to SPPnet, Fast RCNN trains VGG16 2.7× faster (in 9.5 vs. 25.5 hours) and tests 7× faster without truncated SVD or 10× faster with it
- ✓ Fast R-CNN also eliminates hundreds of gigabytes of disk storage, because it does not cache features.

? Does multi-task training help?

- ✓ Multi-task training is convenient because it avoids man□aging a pipeline of sequentially-trained tasks. But it also has the potential to improve results because the tasks influence each other through a shared representation (the ConvNet)
- ✓ Across all three networks we observe that multi-task training improves pure classification accuracy relative to training for classification alone. The improvement ranges from +0.8 to +1.1 mAP points, showing a consistently positive effect from multi-task learning

? Do we need more training data?

- ✓ A good object detector should improve when supplied with more training data.
- ✓ Here we augment the VOC07 trainval set with the VOC12 trainval set, roughly tripling the number of images to 16.5k, to evaluate Fast R-CNN. Enlarging the training set improves mAP on the VOC07 test from 66.9% to 70.0%

? Do SVMs outperform softmax?

- ✓ SVM and Softmax are usually comparable. SVM provides stables results and trains faster while softmax might be bogged down by all the calculations if you have complex training data with many labels.

? Are more proposals always better?

There are (broadly) two types of object detectors: those that use a sparse set of object proposals (e.g., selective search [21]) and those that use a dense set (e.g., DPM – Deformable Parts Model).

Sparse Proposals:

Classifying sparse proposals is a type of cascade [22] in which the proposal mechanism first rejects a vast number of candidates leaving the classifier with a small set to evaluate. This cascade improves detection accuracy when applied to DPM detections [21]. We find evidence that the proposal□classifier cascade also improves Fast R-CNN accuracy. If proposals serve a purely computational role, increasing the number of proposals per image should not harm mAP

AR correlates well with mAP for several proposal methods using R-CNN, when using a fixed number of proposals per image. AR must be used with care; higher AR due to more proposals does not imply that mAP will increase