Image classification, detection and localization are two major challenges in compute vision field. In this chapter, we briefly introduce these two problems and the current state-of-art algorithms. Then we analyze how we can use these algorithms to solve our sea lion counting problem.

0.1 Image classification

Image classification is the task of giving an input image and output the corresponding label. Before convolution neural network is used in image classification, we manually extract features from images and exploit these features for classifying images. It is a challenge, even for experts to design a feature extraction algorithm. Convolutional neural network(CNN) is a special kind of deep learning architecture used in computer vision. The convolution layer makes use of a set of learnable filters. A filter is used to detect the presence of specific features or patterns present in the original image. The filter is convolved across the width and height of the input image, and a dot product is computed to give an activation map. Different filters which detect different features are convolved with the input image and the activation maps are stacked together to form the input for the next layer. As we have more and more activation maps, we can consume too much memory and in order to solve this problem, pooling layers are used to reduce the dimension of the activation maps. There are two kinds of pooling layers: max pooling and average pooling. As the name states, max pooling keeps the maximum value within the filter and discards all the rest, while average pooling keeps the average value. By discarding the rest values, we reduce the dimension of the activation maps and thus reduce the number of parameters we need to learn.

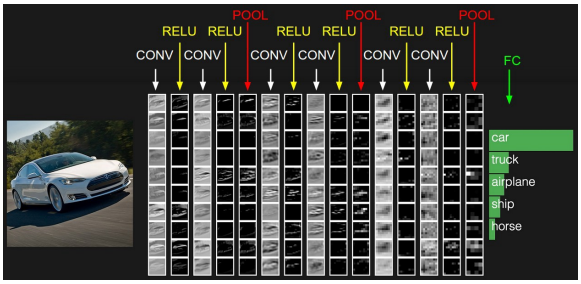


Figure 1: CNN architecture for image classification

After each convolution operation, an activation function is added to decide whether a certain neuron fires or not. There are different kinds of activation functions as illustrated in figure 1.2 and they have different characteristics. Sigmoid functions squashes the output into a value between zero and one and it was the most popular activation function back in days since

it has nice interpretation as a saturating «firing rate» of a neuron. However sigmoid activation function has three major problems:

1. Saturated neurons «kill» the gradients.
2. Sigmoid outputs are not zero-centered.
3. Exponential function is a bit compute expensive.

Later ReLU(Rectified Linear Unit) activation function is invented. ReLU activation function does not have saturation problem and while the largest gradient value for sigmoid function is 1/4, the gradient for ReLU function is 1 or 0. Theoretically ReLU activation function has larger convergence rate than sigmoid function. The problem for sigmoid function is that when we have a negative input value, the gradient is zero. It seems to behave like our neurons which can fire or not, but in reality this can create dead ReLU nodes. Since the value and gradient are all zero when the input value is negative, it can happen that some neurons can never fire again. This is called the «dead neuron phenomenon». In order to solve this problem, leaky ReLU is invented. Leaky ReLU does not have zero gradient at the negative part of the axis, but a small positive value, thus when necessary it can grow back to non zero, avoiding dead neuron problem. Nowadays leaky ReLU is the most common used activation function in deep learning architectures.

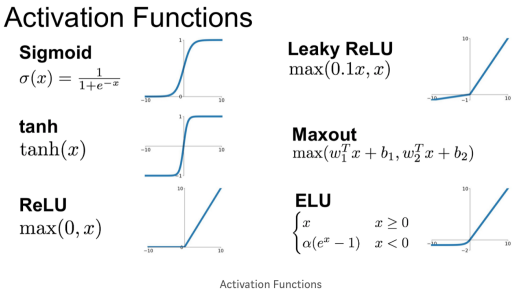


Figure 2: Activation functions

The general idea for CNN architecture is to stack several convolution and pooling layers to extract features from images, then use fully connected layers to exploit these features for classification. By using CNN architecture we don't need to design feature extraction algorithm manually, instead we can use training images to make convolution filters to learn the weights itself. After fully connected layers, we pass the output through a soft-max layer to convert numbers between 0 and 1, giving probability of image being of a particular class.

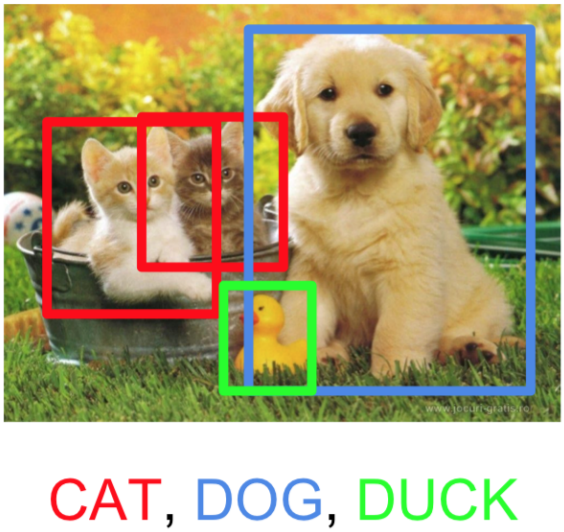


Figure 3: Object detection and localization

0.2 Image detection and localization

Image detection and localization is a more difficult task than image classification, because you not only need to classify the objects in the image, but also need to find the object location as well. Given an input image possibly containing multiple objects, we need to generate a bounding box around each object and classify the object type, as illustrated in figure 1.3. The general idea is to output the class label as well as the coordinates of the four corners of the bounding box.

Image detection and localization is a two-step problem, first we need to find which parts of the image may contain an object, second we need to classify each part. We call the region of image where may exist an object «region of interests»(ROI). One straight forward way to find ROI is to use a slide window to generate all possible ROI. However this method needs to test many positions and scales which is time consuming in both training phase and testing phase. There exists various kinds of image detection and localization algorithms differing in ROI proposing technique.

0.2.1 R-CNN

R-CNN is short for Regional Convolutional Neural Network. The basic idea is to use CNN to classify each ROI of the image. The pipeline for this algorithm is the following:

1. Train from scratch or download a pre-trained image classification model for ImageNet.

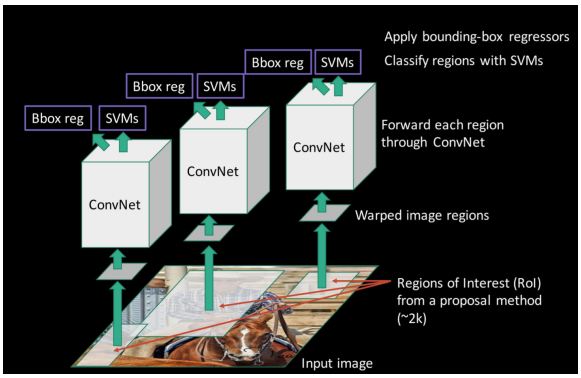


Figure 4: R-CNN architecture

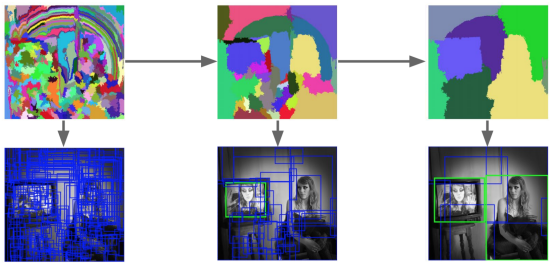


Figure 5: Selective search

2. Fine-tune model for detection. Throw away the final fully connected layer and modify it according to domain dependent problem.

3. Extract region proposals for all images, and for each region, wrap it to CNN size and use CNN to classify its type. It can be a certain object or background.

4. Train one binary SVM per class to classify region features.

5. For each class, train a linear regression model to map from cached features to offsets to ground truth boxes to make up for «slightly wrong proposals»

In R-CNN algorithm, the region proposal method is independent from the algorithm itself, and we can use whatever region proposal algorithms we like. In the paper, the author suggests to use «selective search». Selective search is a greedy algorithm, starting from bottom-up segmentation merging regions at different scales. This method can find «blob-like» regions containing similar pixel values. For each image, there is around 2k regions of proposals.

R-CNN was the start-of-art algorithm for object detection in 2010. There are three major disadvantages for this algorithm, first of all it is slow at test

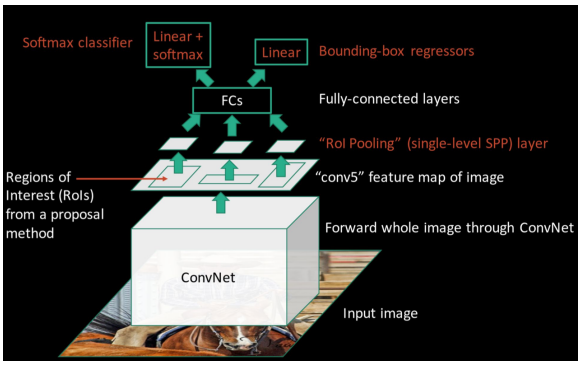


Figure 6: Fast R-CNN architecture

time because it needs to run both the selective search and full forward path for each region proposal. Second, SVMs and regressors are post-hoc which can only be used in domain specific problem. For different problems we need to train SVMs and regressors from scratch.

0.2.2 Fast R-CNN

Fast R-CNN is invented in order to break the bottleneck of R-CNN architecture. Instead of generating regions of proposals first and then use CNN to classify each region, fast R-CNN architecture first forwards the whole image into a CNN and then generates regions of proposals from the feature map. By swapping the order, fast R-CNN architecture shares computation of convolutional layers between proposals for an image. After regions of proposals are generated, fast R-CNN architecture uses «ROI Pooling» to wrap ROI into a predefined size. ROI pooling is actually a pooling operation with changeable filter size. This operation is necessary because we need to forward the feature vector through fully connected layers and the feature vector size is fixed once we set up the architecture. Fast R-CNN architecture can be trained end-to-end and we don't need to train separate modules like SVMs in R-CNN architecture. Fast R-CNN is a lot faster than R-CNN, according to the author, faster R-CNN training phase is 8 time faster than R-CNN and it only takes 0.32 seconds to make inference at test time. However this inference time does not include generating regions of proposals, we still need to use other methods like selective search to find ROI.

0.2.3 Faster R-CNN

Faster R-CNN inserts a «Region Proposal Network»(RPN) after the last convolutional layer of fast R-CNN and thus avoids using selective search to generate ROI. RPN is trainable to produce region proposals directly and there is no need for external region proposals. The rest of modules are the

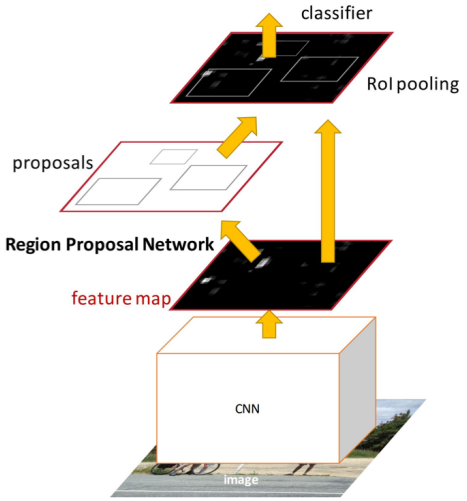
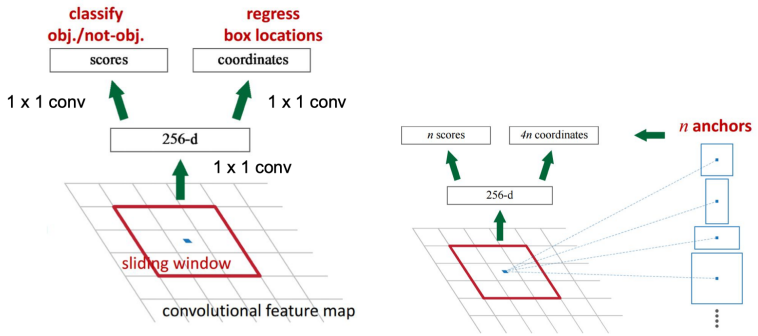


Figure 7: Faster R-CNN architecture



(a) RPN architecture (b) Anchor boxes

Figure 8: Region proposal network

same as fast R-CNN.

RPN is a small network for classifying object or non-object and it can regress bounding box locations as well. A sliding window is used on the convolutional feature map and the position of the sliding window provides localization information with reference to the image. Box regression provides finer localization information with reference to this sliding window. Since we don't know the shape of objects in the image, in order to increase performance we use N anchor boxes at each location. Regression gives offsets from each anchor boxes to justify each ROI. Faster R-CNN is about ten times faster than fast R-CNN architecture and achieves similar performance result with respect to fast R-CNN.

0.3 Counting objects by classification and detection

Can we use image classification and detection techniques here in our object counting problem? Recall that the problem we face is giving an input image, we need to output the number of each object and this requires both classification and object counting. Also note that the dataset we have does not include a bounding box around each object, but a colored dot at the center. Here is how we may use image classification and detection techniques to solve object counting problem:

- Classification: We can use a sliding window to generate patches from an image, and then use CNN architecture to classify each patch. If originally we have 5 types of sea lions to classify, here we are doing 6-class classification, including the background patch. During inference, we first separate the image into patches and then classify each patch to sum up the counts.

- Detection and localization: Since we do not have a bounding box around each object, and in order to train object detection network we need to first create a bounding box from the center dot. By approximating the size of each sea lion, we could create the bounding box manually. Then we can do a faster R-CNN architecture to localize each object. During inference, each bounding box prediction indicates an object.

Theoretically both of the methods should work, but problems do exist for each of them. If we generate object counts by sliding window and image classification, we naturally assume that each patch maximumly contains a single object. In order to make count prediction precise, we need to carefully design the patch size so that most of the patches contain only one sea lion or no sea lion at all. As for object detection, it seems over-killing here because we don't need to predict object location when we only want to generate object counts. In our sea lion counting problem, we tried to use sliding window and image classification as a start-up.