Understanding and Designing a Regional Proposal Network to Identify Abnormalities in Chest X-Rays

Ross T. Wilhite
Department of Mechanical Engineering
Tufts University
419 Boston Ave, Medford, MA 02155
wilhiteross@gmail.com

Abstract

Machine learning is becoming increasingly important in modern computer vision. Especially in the field of image processing and feature recognition, neural networks are becoming more ubiquitous. Much of this success is attributed to the widely appreciated CNN, including it's recent renewed support in the last decade. There are however, newer methods of feature classification that offer significant benefits over the older methods. In 2014, Girshick revolutionized this field by introducing the RCNN. Then, in 2016 again, he iterated on the RCNN, making a faster way to identify significant regions in a feature map. This paper explores my attempts to implement an RPN using x-ray training data from a Kaggle competition.

1 Introduction

1.1 Competition Information

In our project, I participated in a competition by the Vingroup Big Data Institute to identify and classify 14 types of thoracic abnormalities in images of chest x-rays. The goal of this competition was to produce a model that would serve as a valuable second opinion to radiologists when making their diagnosis. I implemented our model using a region-based convolutional neural net based on the work of Girshick et al [**?**].

1.2 Given Data

We were given 15,000 independently labeled images of chest x-rays in dicom format. Each image had a few corresponding annotations, which either labeled and classified an abnormality in the scan or declared the scan free of abnormalities. For each image it is possible that multiple radiologists made annotations, and it is also possible that the same radiologist identified more than one abnormality.

1.3 Abnormalities Definition

The categories of abnormalities included diagnoses like aortic enlargement, calcification, and pulmonary fibrosis. "No finding" was also a possible category. On the training set of x-rays, radiologists identified a bounding box for where they saw the abnormality on the image, then classified it using one of the categories.

1.4 Convolutional Neural Networks in Image Processing

The convolutional neural network (CNN) was first introduced in 1989 by LeCun et al []. But the CNN did not see itself become a significant part of computer vision until 2010 when the AlexNet was developed. The AlexNet was highly successful because it greatly accelerated the speed of training to 6 times faster than its predecessors. In recent years, RCNNs have become increasingly close to identifying objects in real time.

2 Related Works

The world of machine-learning assisted computer vision is massive and expanding rapidly. One of the main reasons for this is the vast amount of applications. The reason machine learning is popular in computer vision is because of the highly promising results achieved from applying convolutional neural networks to image recognition.

The workhorse of machine learning assisted computer vision is the Convolutional Neural Network (CNN). The CNN was originally introduced in 1989 by Lecun et al. [**?**], and has been used in a wide range of applications. CNNs are popular for computer vision because they are highly effective at isolating anomalies. Microsoft, for example, used CNNs as a means to monitor their systems and detect anomalies in time-series data: [**?**].

In recent years, the larger focus has been on real time image recognition. The reason this field has recently started expanding is due to the renewed interest in autonomous robotics. With self-driving cars surging in interest, there have been a few studies involving using RCNNs (Regional Convolutional Neural Networks) in autonomous vehicle markets, for purposes of identifying objects in real time such as pedestrians [**?**]. RCNN's have also seen usage in medical image processing. Medical image processing can be incredibly nuanced, abnormalities can be difficult to detect, so machine learning models are highly effective in this field. In a very parallel study, a mask-RCNN was used to perform segmentations of microscopy images of cell nuclei. [**?**]

3 Methodology

3.1 RCNN

RCNNs were introduced in 2014 by Girshick et al [**?**]. They were introduced as a means of improving accuracy of identifying regions in images, and were capable of improving the average precision by more than 30% than the previous methods. They consist of 3 main components: a region generation algorithm, a convolutional neural network, and an SVM. The interesting feature of this proposal is that the Convolutional Neural network is not actually used for classification and is merely existing for the purpose of generating a feature map from the proposed regions.

3.1.1 Region Proposals

There are a few existing methods of region proposals. A current method which is exceedingly popular is known as selective search, introduced in 2012 by Uijlings et al. [**?**]. Selective search makes use of image convolutions to identify similar regions on a page. The idea is to segment images by combining similar colors. I then take regions from each individual segment, and iterate recursively on this same workflow. This method is effective, and less computationally expensive then previous methods, such as sliding window detection.

Sliding Window detection is a classic method which has been used since it's introduction in 1995 [**?**]. It was a popular method before the popularization of using neural networks in image recognition because it was easy to implement and didn't require large amount of memory or storage. It was, however, computationally expensive and therefore took a long time per image to analyze each region. The regional proposals were done iteratively, as the boundaries of a region are figuratively slid across the image. For each point, the region must also be analyzed in multiple scales and aspect ratios. This method actually serves as a base for one of the more promising recent proposed techniques, The region proposal network (RPN).

The RPN is a more recent idea proposed in 2016 by Girshick as a vital step to the Fast-RCNN [**?**]. The network itself is the most complicated part of the RCNN and requires its own training. It has become exceedingly popular though, because of it's vastly improved performance, and has already spawned areas of research that previously would've been impossible.

The first step in an RPN is generating a feature map of the original image. Usually the image is downsampled and passed through an incomplete CNN to generate a feature map. This feature map becomes further downsampled in H and W, but has significantly increased depth, making it easily classified. Then, over the feature map, I generate something called anchors, another concept introduced by Girshick. Anchors are points spread evenly across an image that represent the center of regions I will propose. From here, a vector consisting of c different side lengths is applied across each anchor. The end result is a series of anchor boxes surrounding each anchor point. Since each scale is applied to each side and at each anchor point, I end up possibly thousands of anchor boxes applied to each feature map. The designers will pick a feature stride, s which represents the amount of points on the original image between each adjacent anchor. Given the original dimensions, H, W, the amount of anchor boxes for any image is calculated via the following: (H/s)*(W/s)*c2(Note that the distinction between H and W is insignificant and simply for comprehension, because the images are reshaped to a square before anchors are calculated.)

These features are then compared to the ground truth bounding boxes. I generate delta bounding box dimensions, which represent the translational movement of the top left corner, and center of each region that would have to be applied in order to move the box to the closest ground truth bounding box location. The system then calculates IOU (Intersection over union) for each box to each bounding box, and labels each bounding box as either foreground or background. If the proposed region has a significantly high IOU, I return it as foreground, and it is sent to the neural network, it's delta bounding box dimensions, and the feature maps.

The CNN itself consists of both a regression and classification system. The regression generates predicted bounding box deltas for the proposed anchor regions, and the classification labels each anchor region as foreground or background.

When trained properly, the RPN can be used to generate proposed bounding boxes as well as the probability that they are significant objects. Then these proposals can be sent to the main CNN of the R-CNN system.

4 Experimental Setup and Results

Earlier in the paper, I mentioned legacy, easy to implement methods of region generation such as selective search and sliding window. Neither of these methods suffice for systems that either have a large amount of data, or need faster identification. In our experimentation, I could not make the region generation with either of those methods to work faster than 15 seconds per image. At that rate, with over 3000 images, it would've taken around 15 hours to generate regions. Since I was expecting to have to iterate on our network, I wanted to reduce turnaround time by utilizing a faster method of image recognition. This is what led us to utilize the RPN, but also because it would prove to be more of a challenge over other methods.

4.1 Feature Map Extraction

As stated earlier in the paper, I was mostly interested in utilizing an AlexNet to extract a feature map from the input data. The first step was to resize each image to a square and convert to RGB, then I sent them through the Alex Net. Originally, I was experimenting with a way to take the output from each individual layer of the AlexNet, these outputs would then be sent to the CNN and deltas and labels of each individual layer would be processed in the loss function. There were two main issues with this that led me away from that implementation method. The first issue was incompatibilities associated with adding an extra dimension to the input. I had many issues with incompatible dimensions while developing and for simplicity decided to work with as few pieces of data at once as possible. The second issue with this method was that the depth of every AlexNet filter would have to be the same, or I would have to individually reshape each outgoing layer's feature map so they can all be processed in the CNN together. This also was adding unnecessary complexity, and I did not believe there would be enough to gain, so I decided against it.

4.2 Anchor Generation

Similar to the feature map extraction, I decided to keep the anchor generation as simple as possible. In terms of constants, I decided to use a stride of 2 for the anchors, and scales of (16, 33, 66, 113, 227). With the feature maps downsized to 13x13, this gave me 25 anchor sizes per location and 36 anchor points, for a total of 900 proposed anchor regions. The anchors are then compared to the ground truth bounding boxes. First I calculate the deltas between the anchor regions and the ground truth boxes in terms of the top left point and the center of the boxes. Then I calculate the IOU of each anchor box and the ground truth bounding boxes. The IOU of two rectangles is the ratio of how much of the rectangles overlap to the amount that the rectangles don't overlap, measured in 2-D area. If the IOU is > 0.5, I count it as foreground, and less than 0.1, I count it as background. This data is compiled and then sent to the core CNN.

4.3 CNN Core and Loss Functions

The core CNN which makes up our RPN is a simple CNN with 3 significant layers. The input is in the form a feature map, derived from passing the resized, original image into the first 5 layers of an AlexNet. The first layer is a convolution on the input which morphs the channels to 512. Then there are two other layers, a classification layer with a depth of two multiplied by the amount of anchor boxes at each anchor point. The two scalar represents the binary classification. The final layer has a depth of four multiplied by the amount of anchor boxes at each point. Here, the four represents the movement of the top left corner of the bounding box, expressed in (x,y) coordinates, and movement of the center of the bounding box, expressed in (x,y) coordinates.

The loss function is another main focus in the RPN. This function is actually a combination of two loss functions at once, one which calculates the categorical cross entropy of the labels, and one which calculates the smooth l1 loss of the box deltas. Essentially, I calculate these values by comparing the values of each anchor to the ground truths

There are some manipulations I have to do in order to get the data to work though. For example, since I do not care about calculating the delta loss for incorrect labels, I only use anchors indicated as foreground. There is also one special case. If there are no anchors identified as foreground in the image, either because there wasn't an accurate enough anchor box proposal, or because there does not exist any abnormalities in the image, I instead use the background anchors for label loss and assume the delta loss is 0.

We use a learning rate of .0001, momentum of 0.9, and clipnorm of 0.5. These values are empirically derived from a Mask RCNN developed by Matterport, Inc in Sunnyvale, CA. Alot of the internal infrastructure of my implementation is actually based off of their work, with major modifications of course to simplify and adapt the implementation to our data set.

4.4 Additional Note

As a means of trying to create a more compact RPN, computationally, I decided to design the system to use as little memory as possible. I ended up succeeding on getting the system to completely train on less than 1 GB of memory.

5 Discussion/Conclusion

I tried training with the above specifications. My main metric I used to measure the performance is the loss of the network. The lowest loss I was able to achieve was 4.5850. While it was passable, I was hoping for better performance. I believe that my biggest issue was with the size of the feature map, 13x13. A decrease in size for the feature map is to be expected, but I thought that maybe a larger feature map would give the trainable parameters more data to actually manipulate. However, when I tried to run another version which upscaled the feature map to 62x62, the loss actually increased to around 6.