Politecnico di Milano
Scuola di Ingegneria Industriale e dell'Informazione
Corso di Laurea Magistrale in Ingegneria Informatica
Dipartimento di Elettronica, Informazione e Bioingegneria

Thesis
Title

Relatore: Prof. Giacomo Boracchi
Correlatore: Diego Carrera

Tesi di laurea di: Zhou Yinan
Nome COGNOME Matr. 872686

Anno Accademico 2018-2019

To someone very special...

Acknowledgments

Abstract

Sommario

Contents

List of Figures

List of Tables

List of Algorithms

Introduction

Steller sea lions in the western Aleutian Islands have declined 94 percent in the last 30 years. The endangered western population, found in the North Pacific, are focus of conservation efforts which require annual population counts. Specifically trained scientists at NOAA Fisheries Alaska Fisheries Science Center conduct these surveys using airplanes and unoccupied aircraft systems to collect aerial images. Having accurate population estimates enables scientists to better understand factors that may be contributing to lack of recovery of Stellers in this area. Currently, it takes biologists up to four months to count sea lions from the thousand of images NOAA Fisheries collects each year. Once individual counts are conducted, the tallies must be reconciled to confirm their reliability. The results of these counts are time-sensitive. Automating the annual population count will free up critical resources, and allow experts to focus on the core research issue. Plus, advancements in computer vision applied to aerial population counts may also greatly benefit other endangered species.

Since Alex re-introduced CNN architecture into computer vision field, deep learning methods become the state-of-art for image classification, localization, and segmentation, etc... Image classification is the task of giving an input image, outputting a corresponding label. The algorithm is usually trained with massive labeled dataset and the outputs are constrained to the labels we have in the training samples. Image localization and segmentation are more difficult than classification. Localization algorithm outputs a bounding box and a corresponding label around each object in the image. In order to do this, the training data needs to have both the labels and the bounding boxes indicating the object location. This kind of dataset requires large amount of time for human labelers and thus is more difficult to construct. Segmentation algorithm requires pixel-leveled prediction for each object and constructing this kind of training set is tedious.

The purpose for this thesis is to develop an algorithm for simultaneous classification and counting, and we don't need to produce exact location for each object. Given an input image, we output the label and the corresponding count for each object. There are various kinds of situations where we could apply this algorithm to, for example, the counting problem for endangered animals like the sea lions described

above. Also for medical pictures, we could use this algorithm to produce precise counts for cells and tissues, helping doctors to analyze the disease. Moreover, the training dataset is a lot easier to construct: we only need to have a labeled dot for each object, no bounding boxes are required. This kind of dataset is more common than bounding box labeled or pixel-leveled dataset.

In the thesis, we use the sea lions dataset provided by NOAA Kaggle Competition and build a fully convolutional network based on Count-ception Architecture introduced in [1]. The thesis is structured as:

- In Chapter 1, we briefly talk about image classification, detection and localization algorithms.
- In Chapter 2, Count-ception Architecture is introduced.
- In Chapter 3, our algorithm is discussed, simultaneously doing object classification and counting.
- In Chapter 4, we deal with dataset construction and preprocessing techniques.
- In Chapter 5, algorithm performance is analyzed.
- In Chapter 6, conclusions are provided.
- In Chapter 7, issues and further improvements are discussed.

Chapter 1

Deep learning methods in computer vision

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

1.1 Image classification

Image classification is the task of giving an input image and outputting the corresponding label. Before convolution neural network is used in image classification, people 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 operation 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. By stacking more activation maps, we can get more abstract features. As the architecture becomes deeper, 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 some values in each filter, we reduce the dimension of the activation maps and thus reduce the number of parameters we need to learn and this makes deep CNN architecture

possible.

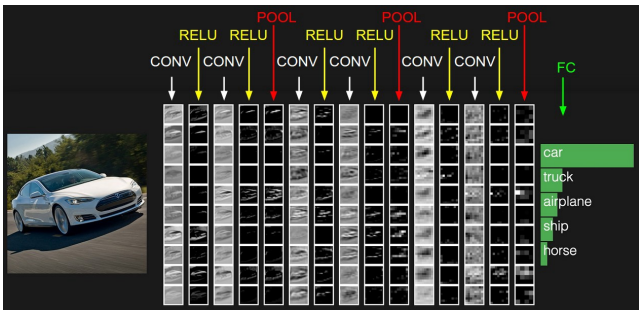


Figure 1.1: CNN architecture for image classification

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 reLU 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 the output value can grow back to non zero, avoiding dead neuron problem. Nowadays leaky ReLU is the most common used activation function in deep learning architectures.

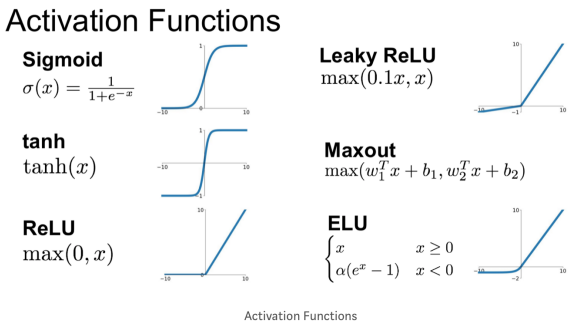


Figure 1.2: Activation functions

The general idea for CNN architecture is to stack several convolution and pooling layers to extract features from images, then it uses 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 make convolution filters to learn the weights itself. After fully connected layers, we pass the output through a soft-max layer to squash numbers between 0 and 1, giving probability of an image of a particular class.

1.2 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. Outputting the class label is a classification problem and generating bounding box coordinates is a regression problem.

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 sliding window to generate all possible ROI. Since we don't know neither the location nor the size of each object, we need to test too many positions and scales which is time consuming. There exists various kinds of image detection and localization algorithms differing in network structure and ROI proposing techniques.

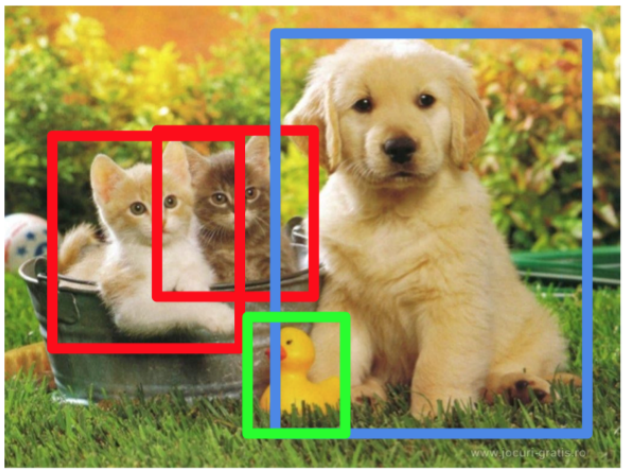


Figure 1.3: Object detection and localization

1.2.1 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.

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

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 and merging regions at different scales. This method can find "blob-like" regions containing similar pixel values. For each image, there are around 2k regions of proposals.

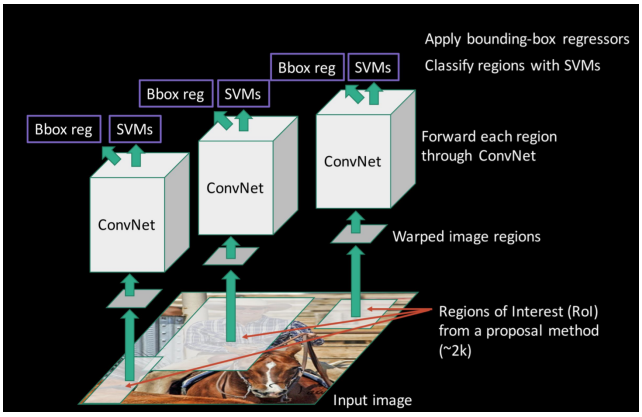


Figure 1.4: R-CNN architecture

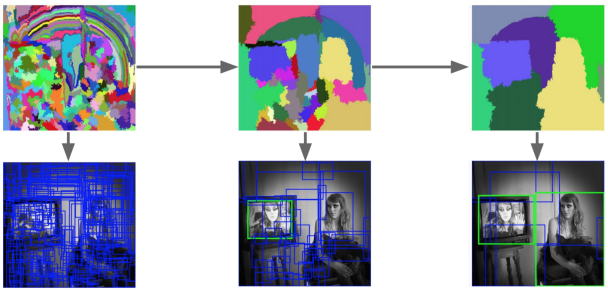


Figure 1.5: Selective search

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 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. Third, the whole architecture is not end-to-end and this complex multistage training pipeline is difficult to implement.

1.2.2 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

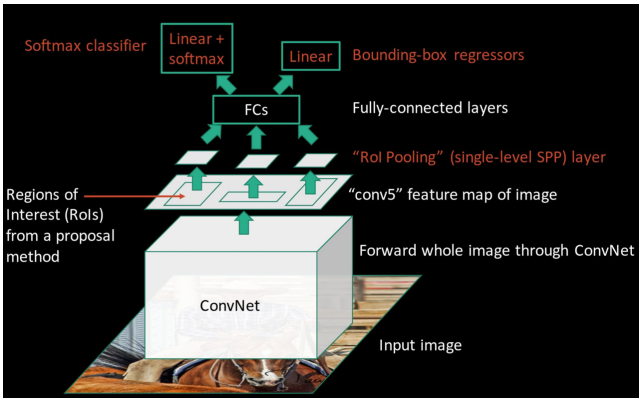


Figure 1.6: Fast R-CNN architecture

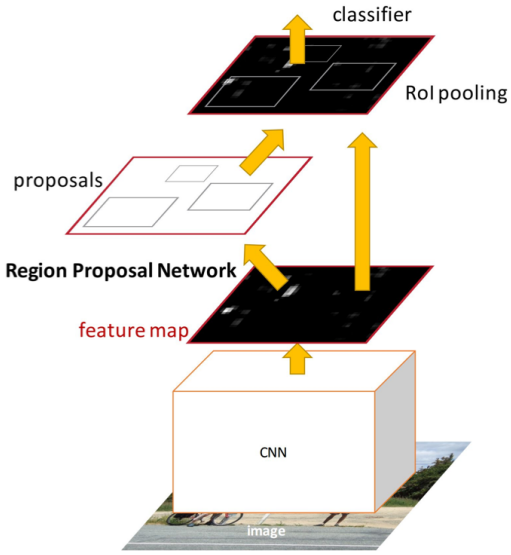


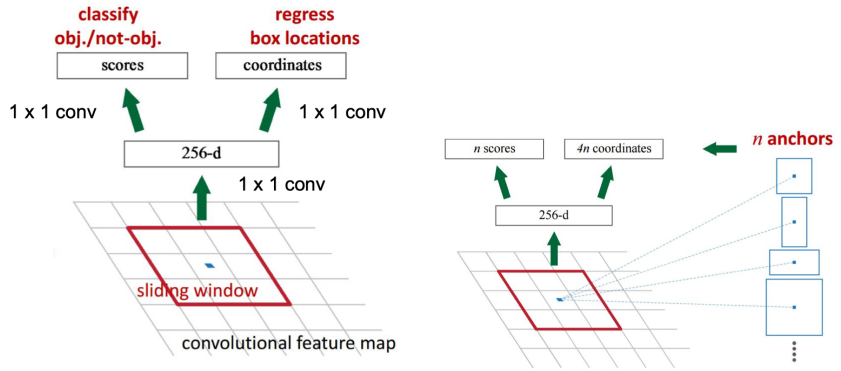
Figure 1.7: Faster R-CNN architecture

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 times faster than R-CNN and it only takes 0.32 seconds to make an inference at test time. However this inference time does not include generating regions of proposals, and we still need to use other methods like selective search to find ROI.

1.2.3 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 same as fast R-CNN.

RPN is a small network for classifying object or non-object and it can regress



(a) RPN architecture (b) Anchor boxes

Figure 1.8: Region proposal network

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.

1.3 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. Finally we sum up all the patches to get the result.

- 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.

Chapter 2

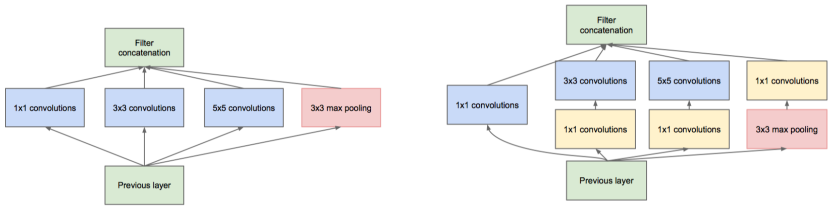
Count-ception

In this chapter, we talk about Count-ception architecture which is based on inception modules and fully convolutional network.

2.1 Inception Network

The Inception network is an important milestone of CNN classifiers. Before Inception network, most popular CNN networks just stack convolution layers deeper and deeper, hoping to get better performance. However, deeper network has more parameters which make gradient descent less effective and lead to overfitting. The Inception network is carefully designed in terms of speed and accuracy while keeping the computational budget constant. The network is organized in the form of "Inception module" which makes it possible for the network to grow both in width and depth. The performance is verified by GoogleLeNet, a 22 layers deep network which won ILSVRC14 competition.

The main difference between Inception module and normal CNN convolution layer is that Inception uses various sizes of filters in each layer while CNN uses only one. The idea of Inception architecture is based on finding out how an optimal local sparse structure in a convolutional vision network can be approximated. One straight forward way is to use more than one filter size and let the training phase to decide the best approximation area. The outputs of each filter are stacked together to form the input of the next stage. As we can see in the figure, there are three shapes of filter: 1x1, 3x3 and 5x5. The 1x1 filter is used for dimension reduction by decreasing output channels. In figure 2.1 (a), we can see the naive implementation of this idea. This implementation, however, has one big issue, even a modest number of 5x5 convolutions can be prohibitively expensive on top of a convolutional layer with a large number of filters. This problem becomes even more pronounced when we stack all the outputs together. This leads to the second implementation structure, as we can see in figure (b). Whenever the computational requirements increase too



(a) Inception module, naive version

(b) Inception module with dimension reductions

Figure 2.1: Inception module

much, we can simply apply a 1x1 convolution to reduce the dimension. Another thing to mention is the max pooling layer which exists only due to historical reason. Back in time, good performance CNN architectures have pooling structures and the inventor of Inception module decided to add the max pooling layer into the structure. Nowadays people start to argue about pooling layers, on one hand, pooling layers reduce the dimension of tensors to make deeper network possible, but on the other hand, pooling layers miss pixel information which can hurt the performance. In Count-ception architecture the author does not use pooling layers.

Inception module is the fundamental element in inception architectures. By concatenating various sizes of filters in each layer and using 1x1 filters to reduce dimensions, the network can grow wider and deeper. Since Inception module is invented, several improvements are made over the years. However in this thesis, we only use the idea of stacking various sizes of filters and ignore other tricks. So the details of the improvements of Inception networks are not discussed here, only a summary is provided.

- Inception v1 concatenates Inception modules to make the network wider and deeper.
- Inception v2 uses two 3x3 filters to replace 5x5 filters, decomposes nxn filers into 1xn and nx1 filters in order to increase computation speed.
- Inception v3 introduces RMSprop optimization algorithm, factorized 7x7 filters and batch normalization.
- Inception v4 uses the idea of ResNet.

2.2 Fully Convolutional Network

F-CNN is short for fully convolutional neural net which is a convolutional network without fully connected layers. The whole network is built by convolution layers and pooling layers only. Normal CNN architectures can be seen as a pipeline structure:

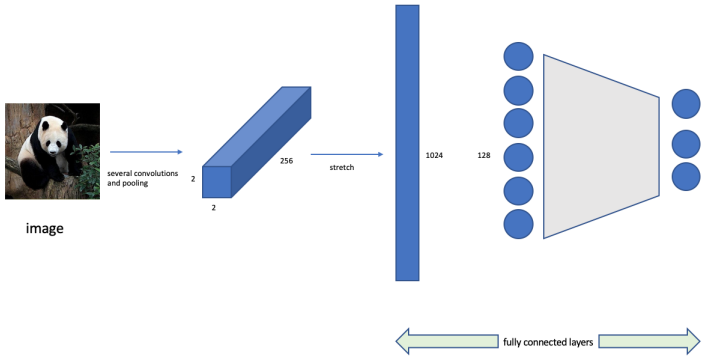


Figure 2.2: Normal CNN architecure

first using several convolution and pooling layers to extract features from images, then using fully connected layers to exploit these features for classification. This structure has one disadvantage, once we set up the architecture we can not change the image size anymore, otherwise we can not forward the tensor into fully connected layers. So it is very common to see that in many CNN architectures, the first step is to wrap the input image into a certain size. Wrapping images could hurt the performance because it changes the aspect ratio, causing distortion.

In fact, passing tensors through fully connected layers can be seen as a convolution operation. We can convert any fully connected layers into convolution layers, with one-to-one map on the weights. Let's see an example, suppose we have a CNN network doing three class classification with one hidden layer of size 128 neurons. After convolution and pooling operations we get a tensor with size 2x2x256. If we want to pass it through fully connected layers, we need to first stretch it into a long vector of size 1024. If we ignore the bias, the weight matrices in fully connected layers are 1024x128 and 128x3. We can convert fully connected layers into convolution layers using the following steps:

1. Do not stretch the 2x2x256 tensor, instead keep the dimension.
2. Build 128 filters, each filter is 2x2x256. Passing 1024 length vector through fully connected layers can be seen as doing convolution with no padding and stride one.
3. Build 3 filters, each filter is 1x1x128.

If we do the math, there is a one-to-one map between weights in fully connected layers and weights in convolution filters, as we can see in the figure. In general, converting any fully connected layers into convolution operations has the following rules:

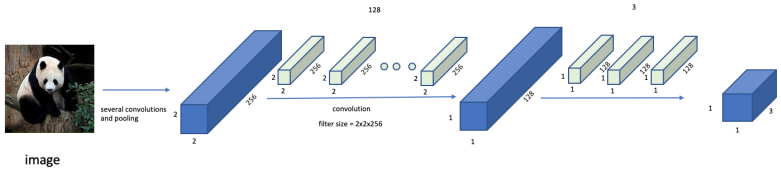


Figure 2.3: Fully Convolutional Network

- Passing vector through fully connected layers is equivalent to doing convolution with no padding and stride one.
- The filter size is equal to the size of input tensor, and number of filters is equal to the number of neurons in the next fully connected layer.

Converting fully connected layers into convolution layers has several benefits. First, we do not need to reshape the image when we have different image size. As long as the input image size is no smaller than the filter size, we can directly forward it through the network. Second, we do not get a single vector at the output, instead we get a tensor. This means if we are doing image classification and we feed the network with an image with larger size, we will not get a single probability vector but a heat map. Thus we can exploit the heat map for further processing. Third, modern deep learning frameworks like tesorflow and pytorch have optimization for convolution operations, so by doing computation in a fully convolutional manner, we can get the result faster than doing computation batch-wise.

2.3 Count-ception Architecture

Count-ception network is introduced in paper [1]. The author uses inception modules to build a network targeting at counting objects in the image. The whole network is a fully convolutional neural net and no pooling layer is used. The author didn't use pooling layers in order to not lose pixel information and make calculating receptive field easier. The network is shown in figure 2.4, each 32x32 region produces a 1x1x1 tensor indicating the number of objects contained in this region. After each convolution, batch normalization and leaky reLU activation are used in order to speed up convergence.

The network is a fully convolutional neural net and we can input image of different sizes. Each 32x32 patch generates a single output value and if we feed the network with 320x320 image, the output tensor will be 289x289x1. In order to train this network we need to construct our training dataset. The network is targeting at producing object counts in the image, and each object has a dot label at center, as

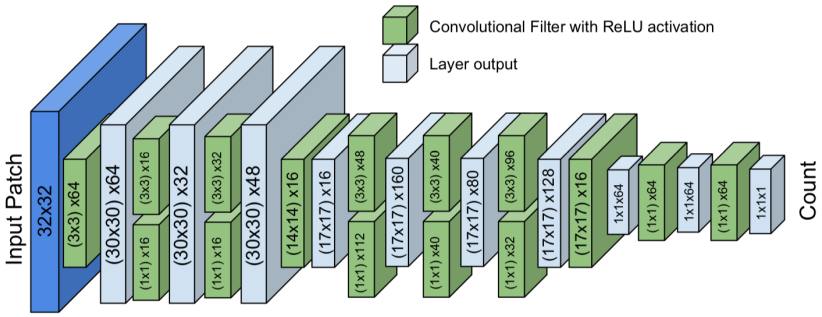
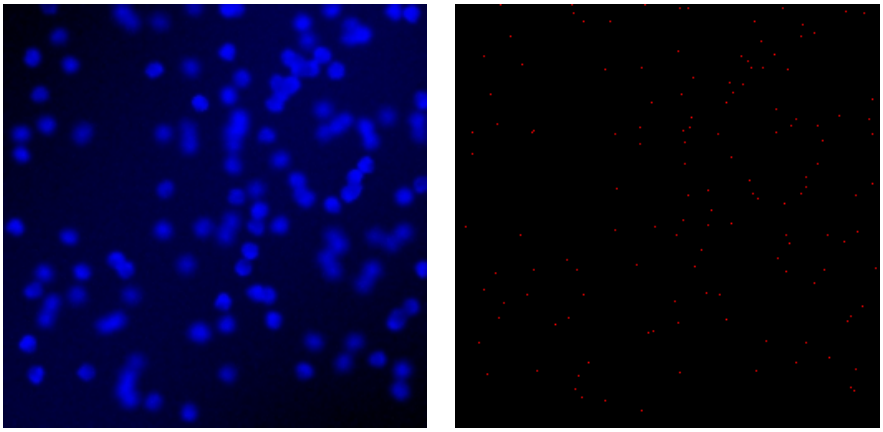


Figure 2.4: Count-ception Architecture



(a) Original image (b) Labeled image

Figure 2.5: Count-ception Dataset

we can see in figure 2.5. Figure a is the original image and we need to count the number of cells in it. Figure b is the labeled image where each dot corresponds to the center point of the cell.

The prediction map is calculated using the original image and Count-ception network. In order to calculate the loss, we need to construct the target manually. The target construction network takes the labeled image and produces a target heat map. If an output value in the heat map is not zero, it means the receptive field of it contains no object. The target construction network is a simple one layer CNN network containing only a convolution operation of 0 padding and stride 1. The filter is filled with all ones and its size is 32x32, same as the Count-ception architecture's receptive field.

The whole procedure for Count-ception is the following:

1. Pad the image image in order to deal with objects near the boarder.

2. Calculate the prediction map using Count-ception architecture.

3. Calculate the target map using target construction network.

4. Calculate the loss between prediction map and target map. The loss function is L1 loss.

5. Use gradient descent to update the weights. Note that the target construction network's weights are fixed. We only need to update the weights for Count-ception architecture.

After we have trained the Count-ception network, we can calculate the prediction map for each input image. In order to get the number of objects in the image, we can simply sum all the pixel values in the prediction map. Note that due to convolution operation, each pixel in the input image is counted redundantly. We can adjust the object counts using formula (1). F(I) stands for the heat map, and r is the receptive field size. In the example above, the receptive field of one pixel in the output is 32x32, so r equals 32. Each pixel in the input image is counted 32x32 times.

$$\#counts = \frac{\sum_{x,y} F(I)}{r^2} \tag{2.1}$$

Chapter 3

Third chapter

In this chapter, we talk about how to do object counting with sliding window and image classification approach.

3.1 Problem formation

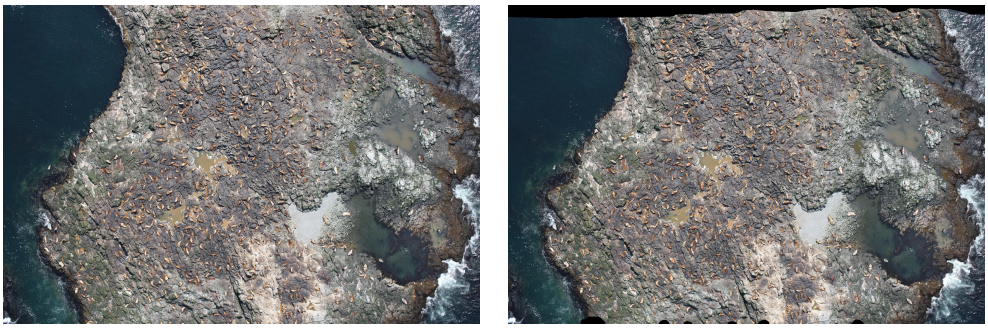
We have a set of images and each of them may contain objects of different types, and we want to predict the object counts for each one of them.

Chapter 4

Dataset

4.1 Steller Sea Lion Dataset

The goal of Steller Sea Lion Count competition is to estimate each type of sea lion in each one of test images. The different types of sea lion are: adult males, subadult males, adult females, and pups. There are totally 947 training images and 18639 testing images. For each training image, we have two versions: the original one and the one with colored dots in the center of each sea lion. Different images may have different sizes but all the sizes are around 4500x3000x3, thus the image is quite large occupying around 5MB space. The large amount of high resolution images bring us two major problems. First, during training we need to deal with memory consumption and we can not have a large number of batch size, otherwise they won't fit into GPU memory. Second, in the testing phase, we need to have short inference time due to huge amount testing images. In figure 4.1, a sampled training image pair is provided. Different color indicates different sea lion types:



(a) Original image (b) Labeled image

Figure 4.1: Training image pair

- red: adult males
- magenta: subadult males

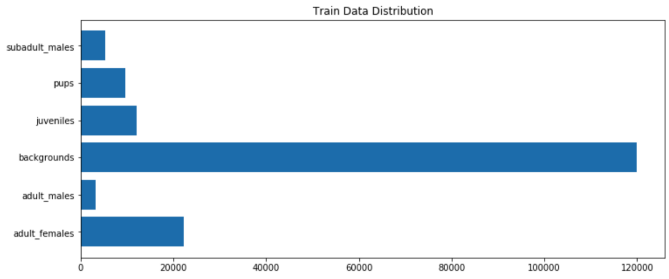


Figure 4.2: Sea lion types distribution

- brown: adult females
- blue: juveniles
- green: pups

As we can see in the figure, there are some black regions in the labeled images. The black regions are added by the data provider in order to filter out controversial sea lions. Another thing to notice is the number of sea lions in each image. In the image pair example above, we have around 900 sea lions but the number of sea lions varies a lot in different images. More specifically, we can have only one or two sea lions in some images. Also it is not an uniform distribution for sea lion types. Here is a summary of sea lion type distribution in the training dataset:

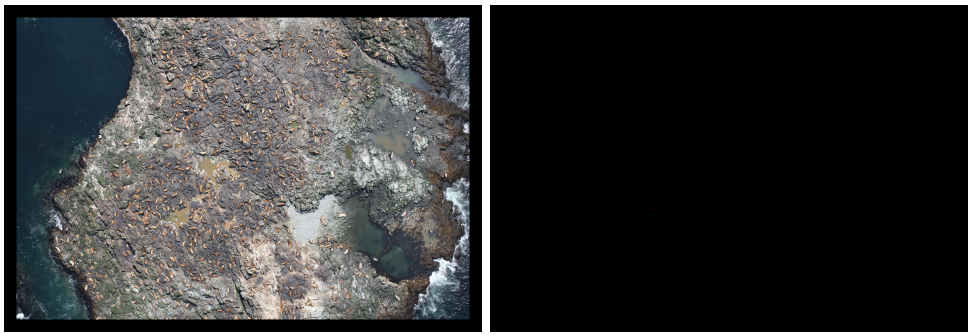
4.2 Data Preprocessing

In order to construct the training dataset, we need to do some data preprocessing. First of all, we use blob detection to find the color of each centered dot and its coordinates. Then we can use these coordinates to construct target maps which are used in Count-ception architecture. Data augmentation is used to balance sea lion types and improve classification performance.

4.2.1 Blob Detection

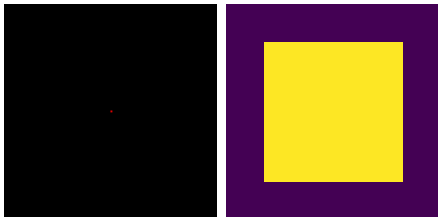
In computer vision, blob detection methods are aimed at detecting regions in a digital image that differ in properties, such as brightness or color, compared to surrounding regions. Informally, a blob is a region of an image in which some properties are constant or approximately constant; all the points in a blob can be considered in some sense to be similar to each other. The most common method for blob detection is convolution.

A dot in labeled images is a blob which contains the same pixel values within a small region, thus we can use blob detection to find the center coordinates of the



(a) Original image (b) Virtual image

Figure 4.3: Preprocessed image



(a) Virtual image (b) Activation map

Figure 4.4: Receptive field

dot. After we get the center coordinates, we can use R, G, B values to decide the color and get the corresponding sea lion type. Luckily we don't need to implement blob detection algorithm from scratch, there are many open source implementations for this algorithm and in this thesis we use the version provided by opencv.

4.2.2 Target Map Construction

In order to make Count-ception architecture work, we need to construct the target map manually. After we get the coordinate for each sea lion, we can construct a virtual image indicating positions of each sea lion. The virtual image has the same dimension as the original image and has 255 pixel value at each sea lion postition, all the other pixel values are zero.

As mentioned before, target map network is a simple CNN architecture with one convolution layer and it is used on the virtual image in order to get the activation map which is a heat map indicating positive object area. The convolution operation is implemented by pad zero and stride one and the filter is filled with ones. When we convolve the filter with the virtual image, we will get a positive value if the object dot is inside the filter, otherwise we will get a zero. Let us see an example, we have a virtual image of size 120x120 and there is an object dot at center of the image. After convolution, the virtual image turns into activation map.

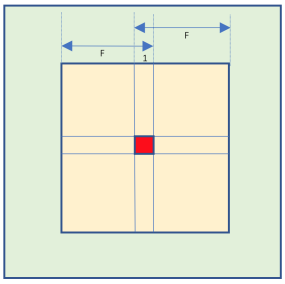


Figure 4.5: Activation area

Recall that in the labeled images, we have a colored dot at the center of each sea lion and when we design the target map network, we need to pay attention to the filter size. We call the filter is activated when we get a positive output. In the example above, the filter size is 48x48 and it is activated when the red dot hits the right bottom corner of the filter. It is easy to see that the activation area in the virtual image is twice the size of the filter minus one, as illustrated in figure 4.7. When we try to design the filter size, we need to keep in mind that the object should lie inside the activation area. Note that the original image and virtual image has the same size. In Steller sea lion dataset, the largest type of sea lion is the adult male which is around 96x96 pixels, so we can make the filter size equals to 48.

4.2.3 Data Augmentation

Data augmentation is a common technique used to improve the performance of neural networks. Overfitting is a challenge for deep learning models due to insufficient amount of data and complex network structures. We can not simplify the deep learning architecture when we have a non trivial problem but we can increase the training data by data augmentation. Given an input image, we can rotate, flip and zoom it to enrich our dataset. In Steller sea lion dataset, we use data augmentation to balance the sea lion types and improve classification performance.

Chapter 5

Fifth chapter

Chapter 6

Sixth chapter

Chapter 7

Seventh chapter

Chapter 8

Eighth chapter

Chapter 9

Ninth chapter

Conclusions

Bibliography

[1] Joseph Paul Cohen, Henry Z. Lo, and Yoshua Bengio. Count-ception: Counting by fully convolutional redundant counting. *CoRR*, abs/1703.08710, 2017.

Appendix A

First appendix

Appendix B

Second appendix

Appendix C

Third appendix