

利用深度学习进行复杂背景下的 物体检测

(申请暨南大学工学学士学位论文)

培养单位：计算机科学与技术系

学 科：计算机科学与技术

姓 名：王 宇 辰

指导教师：龙 舜 副 教 授

二〇一八年五月

Finding Butterflies in Your Photos

Thesis Submitted to
Jinan University
in partial fulfillment of the requirement
for the professional degree of
Bachelor of Engineering

by
Wong Yuchen
(Computer Science and Technology)

Thesis Supervisor : Associate Professor Long Shun

May, 2018

诚 信 声 明

我声明，所呈交的毕业论文是本人在老师指导下进行的研究工作及取得的研究成果。据我查证，除了文中特别加以标注和致谢的地方外，论文中不包含其他人已经发表或撰写过的研究成果，也不包含为获得其他教育机构的学位或证书而使用过的材料。我承诺，论文中的所有内容均真实、可信。

签 名：_____ 日 期：_____

摘要

计算机视觉近年在图像分类、物体检测和图像风格融合等领域取得了巨大的成功。而物体检测是计算机视觉领域的一个核心问题，近年来涌现了 Fu 等提出的基于图像分类的 R-CNN[9] 和 Redmon 等提出的基于图像分割的 YOLO[4][3] 等新方法，并在自动驾驶、智慧地图等领域中在实际应用的场景在实际应用中取得良好的应用。

本文在搜集并整理物体检测领域主要方法的基础上，以研究从一个生态蝴蝶数据集 [12] 为基础。首先对该数据集进行预处理，然后提出一个基于深度学习检测生态照片中蝴蝶位置的方法，并通过实验对该方法进行了测试。实验结果表明本文所提出的方法在所处理的数据集上无论是定位的准确度还是整体位置预测的精确度都取得了极好的效果。最后，我们展示了在实验中我们设计并采用的管理深度模型的工程性思路。

关键词：计算机视觉；物体检测；蝴蝶

Abstract

Computer vision had a great development in image classification and prisma in the last few years. Object detection is one of the core problems in computer vision. Recent years, with RCNN[9], an approach towards this problem by image classification introduced by Fu and yolo[4][3] by Redmon based on image segamentation. Object detection also has wide application in auto-driving and modern map applications.

We firstly investigated different methods towards object detection. Then we analysis an object detection task on a butterfly dataset[13]. Firstly we preprocess our dataset and come up with a solution based on YOLO and test this method with a seris of experiments. The result indicates that our method perform well on the dataset we are facing, both on precision and localization. We also display our way in managing deep learning models.

Key words: computer vision; object detection; butterflies

目 录

| | |
|----------------------------|-----|
| | 1 |
| 诚 信 声 明 | III |
| 第 1 章 概述 | 1 |
| 1.1 研究背景 | 1 |
| 1.2 常用的方法 | 2 |
| 1.3 解决框架 | 4 |
| 1.4 文章的结构 | 5 |
| 第 2 章 检测方法 | 7 |
| 2.1 概述 | 7 |
| 2.2 神经网络训练 | 8 |
| 2.2.1 数据集的聚类 | 8 |
| 2.2.2 训练参数的设计 | 9 |
| 2.3 网络架构和 loss 函数的设计 | 9 |
| 2.4 神经网络的测试 | 11 |
| 2.5 总结 | 11 |
| 第 3 章 数据集和对数据集的处理方法 | 12 |
| 3.1 概述 | 12 |
| 3.2 一种较简易的数据处理方法 | 12 |
| 3.3 另一种数据处理方案 | 13 |
| 3.4 总结 | 18 |
| 第 4 章 实验设计 | 19 |
| 4.1 概述 | 19 |
| 4.2 常见的物体检测算法的评价标准 | 19 |
| 4.2.1 IOU | 19 |
| 4.2.2 Recall Rate | 20 |
| 4.3 实验 | 21 |
| 4.3.1 实验结果分析 | 23 |
| 4.4 总结 | 23 |

目 录

| | |
|-------------------------------------|-----------|
| 第 5 章 一个简易的深度学习训练和测试框架 | 24 |
| 5.1 概述 | 24 |
| 5.2 模型的训练 | 24 |
| 5.3 模型的使用 | 26 |
| 5.4 总结 | 27 |
| 第 6 章 总结 | 28 |
| 参考文献 | 30 |
| 致 谢 | 31 |

第1章 概述

1.1 研究背景

蝴蝶是一类常于日间飞行的昆虫。在生物分类学中，蝴蝶是鳞翅目中凤蝶总科的一个总科级演化支，又名真蝶总科。目前大约存在 18800 种蝴蝶。如图 1.1，蝴蝶在大小、颜色和习性等方面具有很强的多样化特征。据大卫 [12] 等人的统计，世界上最大的蝴蝶是亚历山大鸟翼凤蝶，展翅宽度达到 280 毫米，而体型最小的是褐小灰蝶，长度只有 16 毫米。近年来，学界对蝴蝶各类生活习性的研究变得越来越热烈。同时积累了大量的蝴蝶生态和模式图片素材。如何在一张生态图片当中较为准确的标注出蝴蝶，已经成为了对蝴蝶进行研究的过程中所需要面对的一个重要的问题。

计算机视觉技术近年在图像分类、物体检测和图像风格融合等领域取得了迅猛发展。而物体检测是计算机视觉的一个核心问题，近年来涌现了 Fu 等提出的基于图像分类的 R-CNN[9] 和 Redmon 等提出的基于图像分割的 YOLO[4][3] 等新方法，并在自动驾驶、智慧地图等领域中在实际应用的场景在实际应用中取得良好的应用。

本文着重阐述在基于中国蝶类志的一个数据集上，通过数据清洗改良原始的数据集，然后通过深度神经网络对生态图片中的蝴蝶进行检测的研究。我们的数据集的每张图片当中存在大小不定位置不定的若干只蝴蝶。因此我们的核心问题就是如何在较模糊的图像或生态图像里准确的定位蝴蝶的位置。



(a) 世界上最大的蝴蝶: 亚历山大鸟翼凤蝶 (b) 世界上最小的蝴蝶: 褐小灰蝶

图 1.1 世界上最大和最小的蝴蝶

1.2 常用的方法

物体检测，是目前计算机视觉中的研究热点之一。Justing Jackson[2] 等人将这个问题定义为在一张图片当中，使用各种检测方法将图片中数目不等目标物体检测出来。当前，物体检测已经成了深度学习领域最重要的问题之一。

近年来，在计算机视觉领域对物体检测的方法进行了种类繁多的研究。取得了很大的进展。而这些方法主要可以分成两类。第一种思维基于一个非常朴素的想法，即在图片当中生成多个候选框，然后使用神经网络对这些候选框做图片分类。通过这样的一个思维，衍生出了许多著名的进行物体检测的方法，如 RCNN 和 Faster RCNN。下面是对这一类方法的简要介绍：

RCNN: 这是 Ross Girshick[9] 等人在 2014 年提出的一个利用深度神经网络来进行物体检测的一个方法。这个方法被广泛的应用，并且存在有多个改进的方案。RCNN 将物体检测的过程划分为 3 个子过程。第一个过程中，首先通过 selective search[5] 的方法得到一张图片的多个候选框。在第二个过程当中，使用一个深度神经网络对产生的候选框进行特征提取，生成特征向量。最后，使用 SVM 对特征向量进行分类工作。总的来说，这是一个基于图片分类的物体检测的方法。其优势在于将物体检测问题化归成了一个传统的图片分类的问题，使得网络架构的设计具有较大的灵活性。但是因为现有生成候选框的算法的效率较低，所以这类方法有很大的性能提高空间。

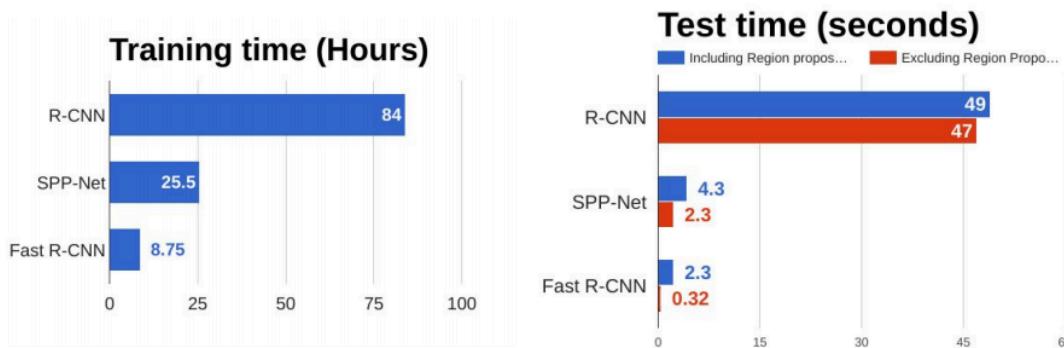


图 1.2 Justin Johnson[2] 等人关于多种物体检测方法效率的对比

Faster RCNN: 这是 Kaiming He[10][7] 等人于 2015 年在 RCNN 的基础上提出的一套进行物体检测的方法。相比于 RCNN，这个方法的改进点主要在于候选框的

选取方面。与 RCNN 相比, 该方法改用一个小规模的神经网络进行候选框选取, 如图 1.1, 这种方法在效率和效果上要高于 RCNN。

而另外一种解决检测问题的方法则是, 在检测的过程中不是去生成候选框。而是将图片网格化, 然后以各个网格为中心生成候选区域。与第一种方法将候选区域直接从图片中截取出来的行为不同, 这类的方法不会进行这些操作。这就使得这类方法能够得到更多的图片上下文的信息, 产生更好的检测效果。图 1.3 简要的展示了这一类方法解决物体检测问题的主要思路。

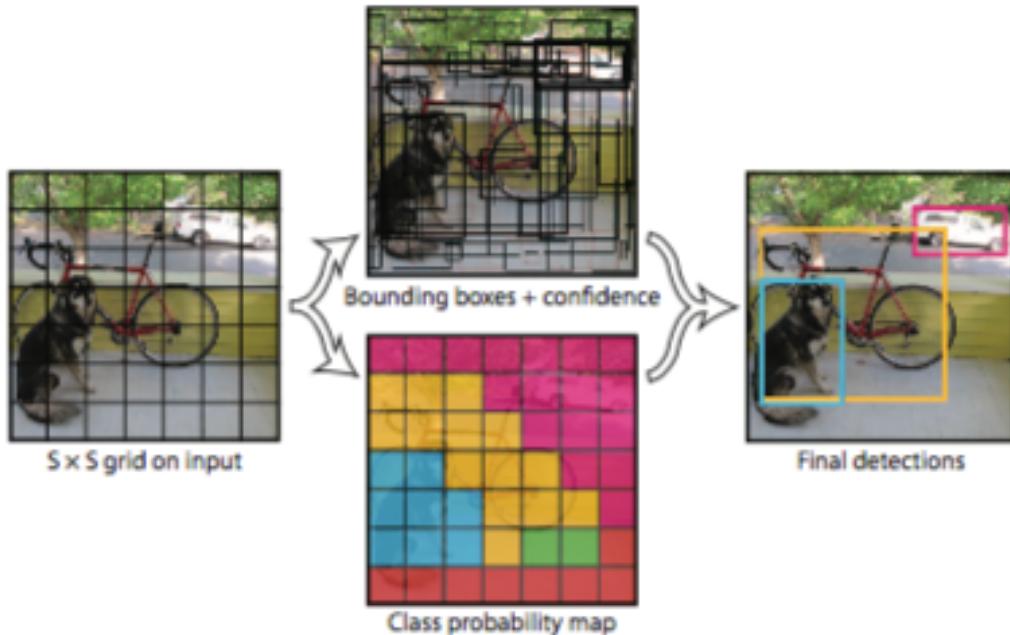


图 1.3 YOLO 主要现将一张图片网格化, 再以每个网格为中心生成多个选框。最后对所有的选框生成物体存在的概率, 以解决物体检测的问题。

YOLO: 这是 Redmon[4] 在 2015 年提出的一个对图片进行物体检测的方法。其主要的思维是, 将图片划分成为 $S \times S$ 个方格块。首先, 对于每个方块会产生 B 个选框。同时, 对于每个方块, 我们会产生 C 个物体是否存在的条件概率。最后, 我们会将两者的结果合并得到最后预测的结果。

YOLOv2: 这是 Redmon[3] 在 2017 年对于 YOLO 的改进版本。在原始的 YOLO 当中, 因为我们将图片使用大小为 $S \times S$ 的方格分割。在预测的时候这些方格是计算概率的基本单位。因为图片中往往会有存在大量比这样的基本单位更小的物体, 因

此 YOLO 在处理这些小物体的时候，往往存在较大的误差。因此而在新的版本当中，作者对这些问题做了一些很好的工作。首先是生成预选区域。YOLO2 会在生成的预选区域中生成选框并进行预测。其次则是提升输入层的大小。先前的 YOLO 使用的输入层大小一般为 224*224。(导致许多较小的物体在这样的分辨率下较难分辨)。许多较小的物体在这样的分辨率下较难分辨。而在 YOLOv2 当中作者将输入层的大小扩张为 432*432。较好的提升了神经网络在检测小物体时候的精确度。

这些方法的产生，使得物体检测技术产生了较大的发展。如表 1.1，我们可以看到，这些方法在一个用于评价计算机视觉模型的数据集 VOC 上，取得了较大的发展，目前，这些方法在实际应用当中也得到了广泛的应用。

表 1.1 多种物体检测框架在 VOC 上的训练效果对比

| Detection Framework | Training Dataset | mAP |
|-----------------------|------------------|------|
| Fast-RCNN SVM | 2007+2012 | 70.0 |
| Faster RCNN VGG16[11] | 2007+2012 | 73.2 |
| Faster-RCNN ResNet[6] | 2007+2012 | 76.4 |
| YOLO[4] | 2007+2012 | 63.4 |
| SSD300[1] | 2007+2012 | 74.3 |
| SSD500[1] | 2007+2012 | 76.8 |
| YOLOv2 288*288 | 2007+2012 | 69.0 |
| YOLOv2 352*352 | 2007+2012 | 73.7 |
| YOLOv2 416*416 | 2007+2012 | 76.8 |
| YOLOv2 480*480 | 2007+2012 | 77.8 |
| YOLOv2 544*544 | 2007+2012 | 78.6 |

1.3 解决框架

本文基于现有的物体检测的重要成果，通过对数据集进行必要的清洗和对 YOLO 进行必要的修改，提出一套较新的对生态照片中的蝴蝶进行定位识别的解决方案。这个解决方案较为新颖的地方是，有别于传统的基于分类器的解决方法。我们将所有种类的蝴蝶当作同一个种类进行识别，避免了某些珍惜种类的蝴蝶因为生态图片较少而造成的检测效果较差的弊端。这个解决方案主要解决如下几个问题：

- 数据集中图片大小过大的问题，据我们的统计，在我们的数据集中，图片的平均大小为 6000*4000，显然，这样大小的图片无法对当前任何规模的神经网络都是无法接受的。为此，我们采用了一些数据清洗的方法，对每一张图片通过了一些方式进行了减小。这样做，减小了数据集的大小，同时较好避免了因为图片压缩而造成的训练质量的下降。
- 传统的 YOLO 网络对于较小的物体检测效果较差的问题。因为 YOLO 本身神经网络自身的特点，较小的物体往往会产生误报或少报。在我们的解决方案当中，我们对神经网络的训练阶段和测试阶段的参数进行了不同的调整，提高了运行的效率和效果。
- 在进行深度学习实验的过程中进行模型管理的问题。因为深度学习往往参数众多，在管理模型的过程中难免出现问题。本文提出了一种基于 JSON 的管理深度学习模型的方法，在本文的实际操作中取得了较好的效果。

总之，本文提供了一个较新的进行单类物体识别的解决方案。在本文的实际实验当中，我们的解决方案在数据集上在定位精确度和整体预测准确度方面都获得了较好的效果。并在工程方面解决了在进行深度学习实验的过程当中对训练数据和生成的模型的管理问题。有较好的创新意义。

1.4 文章的结构

本文主要分为下面的 5 个章节：

第一章概述，简述了对生态照片中的蝴蝶进行检测的研究背景和在计算机视觉领域常用的进行物体检测的方法，并阐述本研究的大体思路。

第二章解决方案的设计。主要阐述在不同的数据清洗方案和网络架构的选择之下对于数据集的定位结果和对结果的分析。并根据产生的结果对经典的解决方案进行改动，从而生成一套较新的对于生态照片中的蝴蝶进行定位的解决方案。

第三章对数据的清洗，主要阐述了实验中所用的数据集的基本情况，并对数据集做必要的统计和分析。提出根据不同网络的特点对数据集进行改进的方案。

第四章首先阐述了我们在实验过程中评价模型好坏的标准，然后给出并且简

要分析了我们在实验过程中获得的结果。

第五章阐述了在实验过程中设计的一个较易使用的机器学习实验的工程性管理方案。这则方案将必要的文件使用一个 JSON 文件加以控制，为训练和使用模型提供了较大的方便。

第2章 检测方法

2.1 概述

本章主要讲述我们在研究的过程当中，根据实际数据集的情况和现有检测方法优劣势的分析而提出的一套新的对生态照片中的蝴蝶进行检测的模型。根据前文的描述，现有的物体检测的方法主要分为两类。第一类使用一些处理的方法在原图中生成一些预选框，再使用传统的图像分类的方法对预选框进行分类。但是，这类方法的问题主要在于生成预选框的效率往往较低。造成训练和测试的效率较低。第二类方法则是以 YOLO 和 SSD 为代表的，这一类方法则略去了生成候选框的过程，而是将图片网格化，以各个网格为中心进行检测工作。这一类方法往往效率较高，但是在检测小物体时精度较低。

根据我们对数据集的分析，在实际工作中，我们遇到的数据集中，有 95% 以上的图片只含有 1 只蝴蝶，且蝴蝶的分辨率往往较高。所以，我们在设计模型时，采用了以 YOLO 为基础的模型架构。这样做，可以使得训练和测试的效率较高，并且因为训练集的关系，YOLO 原本的缺点得到了克服。能够取得较好的效果。

我们的模型是基于 YOLO 而设计的，YOLO 的架构如下所示：

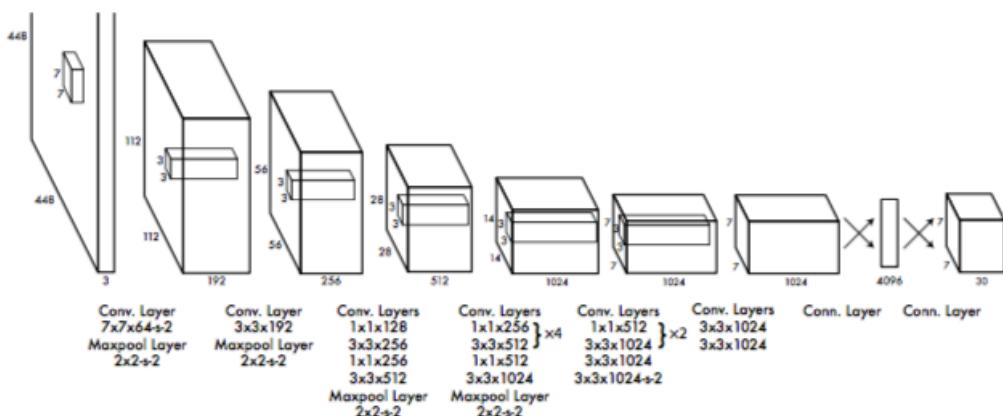


图 2.1 我们的解决方案所用的网络架构主要基于 Redmon 等人 [4] 提出的 YOLO 网络架构，有 24 个卷基层和 2 个全连接层组成，将物体检测的过程看成一个端到端的任务

2.2 神经网络训练

2.2.1 数据集的聚类

我们使用下面的四元组来描述矩形框相对于整张图片的位置：

Listing 2.1 Build.json Example

```
1 <classname>      <x>      <y>      <width> <height>
```

其中，`classname` 就是这个物体所属的种类编号，因为在我们的实验当中，我们只是想得到蝴蝶的大致位置而不是将其分类，所以我们的编号全部为 0。`x,y,width,height` 为选框的位置和大小相对于整张图片大小的比例。

前文我们提到了，yolo 会将在我们的实验当中，以每个网格为中心生成一定数量的候选框。而如何指导这些候选框的生成呢？我们采用聚类的方法对训练数据中候选框的大小进行了聚类。在我们的实验中，我们使用 k-means 的方法对训练集中所有的图片的候选框进行聚类。经过实验，在我们的实验中，我们使用聚类的方法将数据分成 5 类。使用的 k-means 的方法如下所示：

Listing 2.2 Processing 1

```
1 Input: E = {e_1,e_2...e_n} (set of entities to be clustered)
2           k (number of clusters)
3           MaxIters (limit of iterations)
4 Output: C={c_1,c_2...,c_k} (set of cluster centers)
5           L={l(\epsilon)|\epsilon=1,2,3...,n}
6
7 foreach c_i in C:
8   c_i = e_j is in E (random selection)
9 end
10 foreach e_i is in E:
11   l(e_i) = argminDistance(e_i, e_j) j is in {1...k}
12 end
13
14 changed = false
15 iter = 0
16
17 repeat
18   foreach c_i is in $C$:
19     UpdateCluster(c_i)
20   end
```

```

21
22     foreach e_i is in E:
23         minDist = argminDistance(e_i, e_j) j in {1...k}
24         if minDist != l(e_i):
25             l(e_i) = minDist
26             changed = true
27         end
28     end
29     iter++
30     until changed=false or iter>Maxiter

```

2.2.2 训练参数的设计

在我们实验设计当中，我们使用动态的 learning_rate 的方式。我们在正式进行训练之前，采用了测试 learning_rate 效果的方法。即进行预训练 10000 次，每 1000 次将当前的 learning_rate 降低为原来的 $\frac{1}{10}$ 。通过这种方式，我们就得到了下面的 learning_rate 的设置方式。

- 在 1-200 步，learning_rate 为 1e-4
- 在 200-8000 步，learning_rate 为 1e-3
- 8000-20000 步，learning_rate 为 1e-4
- 20000-35000 步，learning_rate 为 1e-5
- 35000-52500 步当中，learning_rate 为 1e-6

如图 2.2，在实验当中，这样的方式能够使得 loss 很好的收敛。

在我们的实验当中，我们的使用 Nvidia K80 来辅助进行模型的训练。因为实验环境的关系和训练集的实际大小，我们将 batchsize 设为 32，每个 batch 划分成为 4 个单元。这样做考虑到了实际环境的容量了训练的效率，我们实验中一共进行了 52500 次 batch。其中 loss 函数值随迭代次数的变化图如图 2.2 所示。我们可以发现我们的模型得到了很好的收敛。

2.3 网络架构和 loss 函数的设计

在我们的实验当中，我们采用了和 yolov2 相似的架构。由于我们所要面对的训练集的图片分辨率普遍较大。如图 2.1，在我们的实验当中，我们将输入层的大小设为 416*416，虽然因为这样做，在训练和测试的时候，图片往往需要较大的切割，但是，因为 95% 的图片当中蝴蝶的数量只有一个并且因为我们采用的较好的数据清洗的方法使得原本在图片当中占比较小的物体得到了放大，这样做不仅使得训练的速度得到了优化，并且对实际的精度影响也较小。

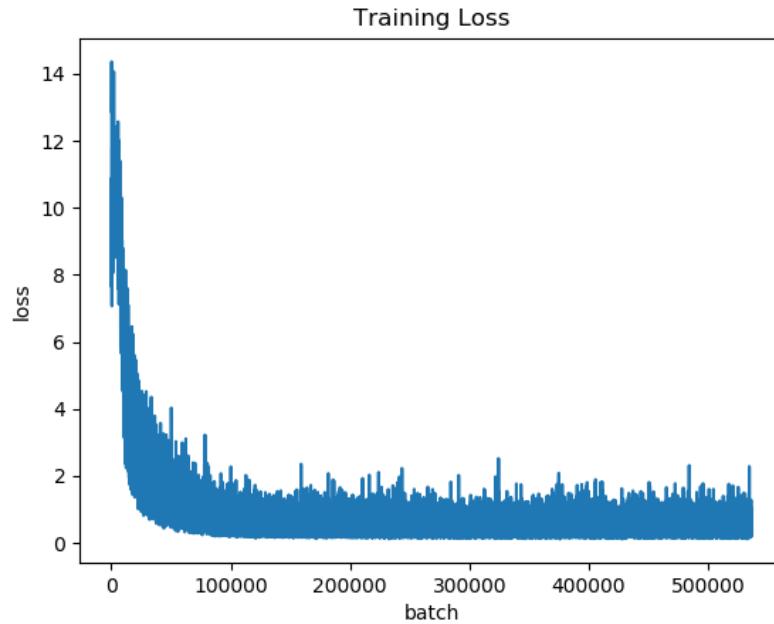


图 2.2 训练过程中 Loss 函数的变化

我们的损失函数的设计仍然沿用了 Redmon[4] 等人在论文中提出的损失函数的设计方案。我们的损失函数如下：

$$Loss = L_1 + L_2 + L_3$$

$$L_1 = \lambda_{coord} \sum_{i=0}^{S^2} \sum_{j=0}^B 1_{ij}^{obj} [(x_i - \hat{x}_i) + (y_i - \hat{y}_i)] + \lambda_{coord} \sum_{i=0}^{S^2} \sum_{j=0}^B 1_{ij}^{obj} [(\sqrt{w_i} - \sqrt{\hat{w}_i})^2 + (\sqrt{h_i} - \sqrt{\hat{h}_i})^2]$$

$$L2 = \sum_{i=0}^{S^2} \sum_{j=0}^B 1_{ij}^{obj} (c_i - \hat{c}_i)^2 + \lambda_{noobj} 1_{ij}^{noobj} \sum_{i=0}^{S^2} \sum_{j=0}^B [c_i - \hat{c}_i]^2$$

$$L3 = \sum_{i=0}^{S^2} 1_{ij}^{obj} \sum_{c \in classes} (p_i(c) - \hat{p}_i(c))^2$$

在这里, λ_{coord} 和 λ_{noobj} 是我们事先指定的。这个损失函数可以看作 3 个部分的综合:

- L_1 指的是位置预测损失, x_i 和 y_i 是预测的矩形框中心的坐标。而 w_i 和 h_i 则是矩形框的宽和高。我们使用这 4 个参数来表示一个矩形框。而 $\hat{x}_i, \hat{y}_i, \hat{w}_i, \hat{h}_i$ 则是标注好的结果。
- L_2 指的是预测置信度的损失。 c_i 指的是物体预测为某个种类的置信度。
- L_3 指的是类别预测的损失, $p_i(c)$ 指的是物体预测成某个类别的概率。

2.4 神经网络的测试

在我们的测试当中, 因为在进过我们的数据清洗后的训练集, 其选框相对于原图的大小相比于原来的数据集有所增大, 同时根据 Peiyun Hu[8] 等人在小物体检测方面的一些研究成果, 我们适当增大输入层的大小, 减少了因为图片输入时因为图片压缩而造成的图片失真的情况。但是在我们的实验中, 我们也发现, 过度的增大图片输入层的大小, 也会造成召回率的降低。后文中我们将会详细介绍这个现象。

在后文当中我们会较为详细的叙述我们最终的实验结果。

2.5 总结

在本章当中, 我们简要的介绍了我们关于检测生态照片当中蝴蝶的神经网络在训练阶段和测试阶段的参数设置, 网络架构调整和 loss 函数的分析。在实验过程当中, 因为训练集和测试集中蝴蝶大多分辨率较高且面积较大。我们的解决方案取得了较好的效果。但是, 在检测较小的蝴蝶或者分辨率较差的蝴蝶时, 这个解决方案仍然存在精度不足的问题。我们将在下面的章节当中着重分析和解决这些问题。

第3章 数据集和对数据集的处理方法

3.1 概述

本实验所使用的数据集来自中国蝶类志的一个子集。包含两种蝴蝶的照片。其一为模式照，即蝴蝶标本的标准照。如下图所示，此类照片的特点是分辨率较高，蝴蝶占图片的比重较大。第二种为生态照，即蝴蝶在自然状态下的照片，由于拍照和自然环境的特点。在此类照片当中蝴蝶的形态多种多样，占图片的大小比重也有较大的差别。



(a) 数据集模式照举例



(b) 数据集生态照举例

图 3.1 数据集照片举例

根据谢娟英 [13] 等人的统计。这个数据集下有 721 张蝴蝶生态照片，4270 张蝴蝶标准的生态照片。由于在我们的实验当中，我们的主要目标是解决在生态照片下对于蝴蝶的定位问题，因此对于下面讨论的主要是对于蝴蝶生态照片的处理方法。

3.2 一种较简易的数据处理方法

通过对数据集的简要分析，我们发现，本数据集当中，生态照片的分辨率普遍较大。经过统计，我们发现，生态照片的平均分辨率为 $6000*4000$ ，而蝴蝶的最大大小为 $3224*2448$ 。而普通的 yolo 网络其输入层只有 $416*416$ 。因此，在本实验当中，使用原图进行训练集的训练极难产生较为满意的结果。

基于这个发现，我们对这个数据集进行一次较简单的变换。对于分辨率大于

800*600 的图片，我们将其变换成 800*600 的较小的图片，而对于分辨率较小的图片，我们则不对这张图片进行处理。通过简单的处理，我们得到了一个进过压缩的数据集。我们下称这个数据集为 D1.

Listing 3.1 Processing 1

```

1
2 def imageProcess(S):
3     #S is the list of training set
4     for img in S:
5         read img from File
6         convert the size of picture to 800*600
7         save img to File1
8     return

```

通过简单的变换，我们发现，D1 中的图片大小大大减小，神经网络的训练速度明显加快。但是，这样的操作造成的影响也是显而易见的。如下图，因为我们未将图片等比例缩小，很多张图片中的蝴蝶出现了变形的情况。同时，有些蝴蝶在图片中占的比例较小，造成在数据处理后出现大量的信息丢失。对神经网络训练的效果造成了较大的影响。因此，我们对数据集的组成进行了较详细的分析，提出了下面的解决方案。



图 3.2 D1 照片举例

3.3 另一种数据处理方案

根据上面的叙述，我们可以发现，直接将数据集中较大分辨率的图片压缩成指定分辨率的图片，虽然数据集得到了压缩。但是可能会使得目标检测物体失真，从而无法得到较好的训练效果。更严重的是，如果目标物体所占原图片的比重较

小，这样的压缩方式会使得所得到的图片过小。在以 yolo 为基础的解决方案下，这样的图片会大大影响检测的效果。于是，我们需要对数据集进行进一步的分析，并且提升在处理过后的图片当中目标物体占整张图片的比重。

我们首先对生态照片当中所含蝴蝶的数量如下所示，我们发现，在本实验所用的数据集所含的 721 张生态照片当中，692 张照片只含有 1 只蝴蝶，其余 29 张图片所含蝴蝶的数量如下所示。我们发现，本数据集的生态照片当中只有 5 张照片所含蝴蝶的数量在 3 只以上，根据这个特点，我们可以根据图片中目标选框的数量来确定数据处理的方案。

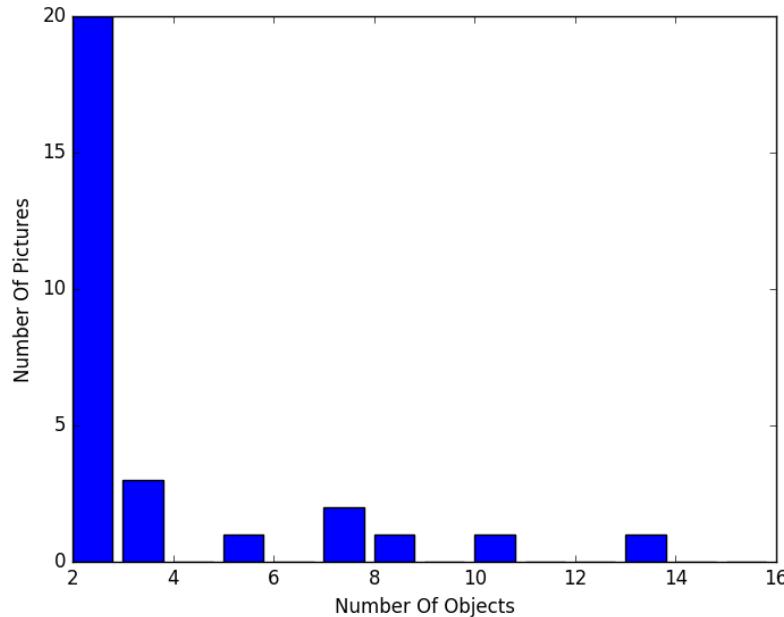


图 3.3 原数据集中蝴蝶数量多余 3 只的照片统计

首先，我们定义一个选框在对应的图片当中的有效率为下面的式子：

$$SR = \frac{S_1}{S}$$

其中 S_1 为选框的面积，而 S 为图片的总面积。

我们希望 SR 的大小在 25% 以上，同时，我们在这里希望目标物体在图片中

出现的位置具有随机性。于是，根据图片中含有不同蝴蝶的数量，我们对数据集做下面的改变：

- 在只含有一只蝴蝶的时候，我们以矩形选框为中心，从水平和竖直的两个方向分别扩张这个矩形选框的 40%-60%。这样我们就可以保证矩形框的面积和总面积的比例为 1:4 左右和并且重新计算这个矩形选框在整张图片中的相对位置。
- 在含有两只蝴蝶的时候，首先我们判断两个矩形选框是否存在重合，如果存在重合则按照一个矩形选框，使用上面的方法进行处理。如果不重合则分开通过上面的方法进行处理。
- 在含有 3 只或者 3 只以上的蝴蝶的时候，我们通过统计发现，蝴蝶的平均大小在 70*70 左右，且图片中呈现的结果是大堆的蝴蝶堆积在一起。所以，我们采用较为激进的处理方式，以每一个矩形选框为中心，向外扩张 0-30 个像素点，得到一张新的图片。这样做，使得每张图片当中只会存在 1 只蝴蝶。并且这些图片不会产生较大的损失。

这个处理方法的伪代码如下所示：

Listing 3.2 Processing 2

```

1 # 处理图片当中只有1只蝴蝶的情况
2 def ProcessOne(pic):
3     Read Image from File
4     object = readObjectLocation(pic)
5
6     im1 = Both width and height of pic resize to 1/4
7     write im1 to File1
8
9     #Compute the bounding of new picture
10    newxmin = int(random.uniform
11                  (int(object.xmin*0.6),object.xmin))
12    newxmax = int(random.uniform
13                  (object.xmax, int((pic.w-object.xmax)*0.4)+object.xmax))
14    newymin = int(random.uniform
15                  (int(object.ymin*0.6),object.ymin))
16    newymax = int(random.uniform
17                  (object.ymax, int((pic.h-object.ymax)*0.4)+object.ymax))
18
19    im2 = Clip [newxmin:newxmax, newymin:newymax] from original picture
20    write im2 to File2
21    generateNewObjectInformation
22    return
23

```

```

24 # 处理图片里面只有2只蝴蝶的情况
25 def processTwo(pic):
26     objects = GetObjectsFromPicture(pic)
27     if objects[0] and objects[1] is mergeAble:
28         Merge two objects into one
29         ProcessOne(pic)
30     else:
31         for object in objects:
32             # Computer the bounding of new picture
33             newxmin = int(random.uniform
34                           (int(object.xmin*0.6), object.xmin))
35             newxmax = int(random.uniform
36                           (object.xmax, int((pic.w-object.xmax)*0.4)+object.xmax))
37             newymin = int(random.uniform
38                           (int(object.ymin*0.6), object.ymin))
39             newymax = int(random.uniform
40                           (object.ymax, int((pic.h-object.ymax)*0.4)+object.ymax))
41
42             im2 = Clip [newxmin:newxmax, newymin:newymax] from original picture
43             write im2 to File2
44             generateNewObjectInformation
45     return
46
47 ## 处理蝴蝶数量大于3的情况的算法
48 def ProcessThree(pic):
49
50     xoffset = random.uniform(10, 30)
51     yoffset = random.uniform(10, 30)
52
53     objects = GetObjectsFromPicture(pic)
54
55     for object in objects:
56         # computer new bounding for each object
57         newxmin = object.xmin - xoffset
58         newxmax = object.xmax + xoffset
59         newymin = object.ymin - yoffset
60         newymax = object.ymax + yoffset
61         im2 = Clip [newxmin:newxmax, newymin:newymax] from original picture
62         write im2 to File2
63         generateNewObjectInformation
64
65     return
66
67 def imageProcess(S):
68     # S is the set of our training set
69
70     for pic in S:
71         objectNumber = CountObject(pic)
72         if objectNumber == 1:
73             ProcessOne(pic)
74         else if objectNumber == 2:
75             ProcessTwo(pic)
76         else:

```

```
77     Clip(pic)
78     return
```

经过如上所述的处理过程，我们得到了一个新的训练集(下称D2)。一些处理后的照片的例子如下所示：

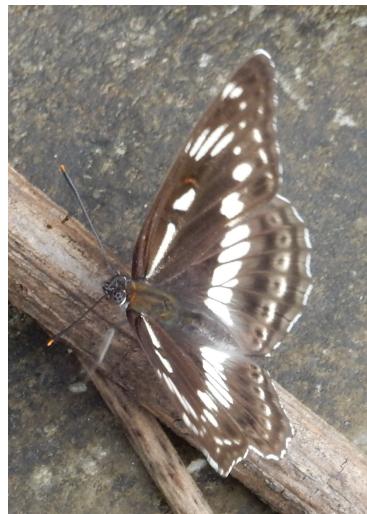


图 3.4 D2 照片举例

通过如上所示的过程，我们得到了一个新的数据集，新的数据集的大小较原数据集大小大大的减小了。同时因为避免了变换操作，使得图片质量得到了最大限度的保存。在后文中我们会得知，这样的数据集在训练的过程中有着较好的效果。但同时，这样对于数据集的处理，将数据集中含有多只蝴蝶的图片全部裁剪成为只有1只蝴蝶的图片。在实际的应用当中，生成的数据集对与有多只蝴蝶的图片，检测的效果较差。

3.4 总结

本章当中，我们主要介绍了在实验当中我们对于实验训练集的处理方案。首先，我们根据训练集的图片分辨率较大的问题，提出了直接将图片缩放的解决方案。但是，这样做使得训练集中的部分图片出现变形的情况，并且让训练集中的小物体失真。为此，我们对训练集进行了一次统计。设计出了一个以同比例缩放和部分关键区域放大为基础的对数据集处理的解决方案，并且取得了较好的效果。

第4章 实验设计

4.1 概述

本章，我们将着重介绍在研究过程中我们对实验的设计和对实验结果的一些分析。首先，我们将介绍当前在深度学习领域较为权威的几种衡量物体检测算法好坏的评价标准。然后，我们将介绍一些我们在产生最后的解决方案的过程当中所做的一些实验。最后，我们对实验的结果进行一些分析。

4.2 常见的物体检测算法的评价标准

4.2.1 IOU

如上文所说，一个物体检测算法，其核心的部分就是对一张图片，尽可能精确的得到目标检测物体的位置，并且将位置在图片中标注出来。因此，标注的精确度，就是体现一个物体检测算法好坏的重要标准之一。

IOU(Intersection over Union) 就是在深度学习领域一个十分重要的体现一个物体检测算法精度的指标之一。如下图所示，这个标准的计算方法就是计算两个选框的相交部分和两个选框的总部分的比值。

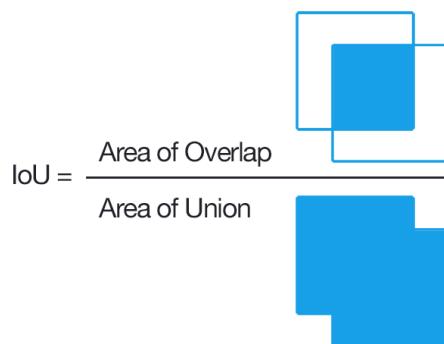


图 4.1 IOU 示意图

而在我们的实验当中，我们默认所有的选框的形状均为矩形。所以，我们实现的伪代码如下所示：

Listing 4.1 IOU

```

1 def IOU(area1, area2):
2     S1 = (min(area1.xmax, area2.xmax)-max(area1.xmin, area2.xmin))*(
3         min(area2.xmax, area1.xmax)-max(area1.xmin, area2.xmin))
4     S2 = area1.w*area1.h+area2.w*area2.h-S1
5     return s1/s2

```

4.2.2 Recall Rate

如上节所示，我们可以采用 IOU 这个指标来衡量一个物体检测算法在单张图片上的检测效果。但是，我们需要从整个数据集的角度更好的来衡量这个模型的好坏。从这个角度的考虑下，一个新的指标:recall rate, 便应运而生。

在了解 recall rate 的定义之前，我们定义下面的指标：

thresh: 两个矩形选框被判断重合的 IOU 最低阈值。简言之，如果两个矩形选框的 $IOU > recall$, 则我们就可以认为两个矩形是重合的，否则，我们就不认为两个矩形是重合的。

TN/True Negative: 验证集被标注为正样本且被模型处理为负样本的实例。

TP/True Positive: 验证集被标注为正样本且被模型处理为正样本的实例。

FN/False Negative: 验证集被标注为负样本且被模型处理为负样本的实例。

FP/False Positive: 验证集被标注为负样本且被模型处理为正样本的实例。

在实际处理的过程当中，我们不会显示的产生具体的负样本的具体信息。所以，我们所产生的矩形框只会有 1,3,4 三种情况。所以，我们产生 recall rate 的伪代码如下面所示：

Listing 4.2 Recall

```

1 iterate through entire list of predictions for all images
2
3 if IOU > threshold
4     if object not detected yet
5         TP++
6     else
7         FP++    // nms should reduce # of overlapping predictions
8 else
9     FP++
10
11 if no prediction made for an image
12     FN++

```

```

13 AP = TP / (TP+FP)
14 Recall = TP / (TP+FN)
15

```

4.3 实验

如上文所述，在实验的过程中，我们主要发现了，在这个问题当中，影响模型实际工作效率的因素，主要存在下面的几点：(1) 数据集的好坏，(2)learning_rate 的设置，(3) 网络架构的选取。基于上面的几点，我们设计并进行了下面的实验：

表 4.1 实验训练参数设置

| 参数 | Lab1 | Lab2 | Lab3 |
|-------------------|---------|---------|---------|
| Dataset | D1 | D2 | D2 |
| LearningRate | 0.0001 | 0.0001 | 0.0001 |
| InputLayer | 416*416 | 832*832 | 416*416 |
| Network Framework | Yolov2 | Yolov2 | Yolov2 |

在测试的过程中，我们发现，测试的时候，输入层大小的选择对检测效果具有很大的影响，为此，我们做的下面的工作：

表 4.2 实验结果

| Framework & Input Layer | Avg IOU | Avg Recall |
|-------------------------|---------|------------|
| Lab1 416*416 | 37.78% | 24.62% |
| Lab2 832*832 | 81.79% | 96.92% |
| Lab2 1024*1024 | 81.97% | 98.46% |
| Lab2 1664*1664 | 74.91% | 92.31% |
| Lab3 416*416 | 79.85% | 93.85% |
| Lab3 512*512 | 80.59% | 93.85% |
| Lab3 832*832 | 83.69% | 96.92% |
| Lab3 1024*1024 | 82.44% | 98.46% |
| Lab3 1664*1664 | 76.42% | 95.38% |

从上面的实验结果当中，我们可以发现，使用了数据清洗之后的数据集，使模型的准确度得到了相当大的提升。但同时我们要看到的是表格当中标注黑体的两行。虽然我们在测试的过程当中为了防止因为图片压缩而造成的失真而导致预测

准确度的下降。我们发现，到达了一定程度之后增大输入层的大小反而会造成预测精度的下降。根据我们的分析和 Peiyun[8] 这样是因为，我们的测试集，其数据的概率分布和原训练集类似。但我们在处理数据集的过程中对图片进行了裁剪，这就使得我们生成的模型会更适应含有较原图比例较大（大约 25%）的图片。因此，将输入层放大，能够取得较大的精度。

一些实验处理过后的图片如下所示：

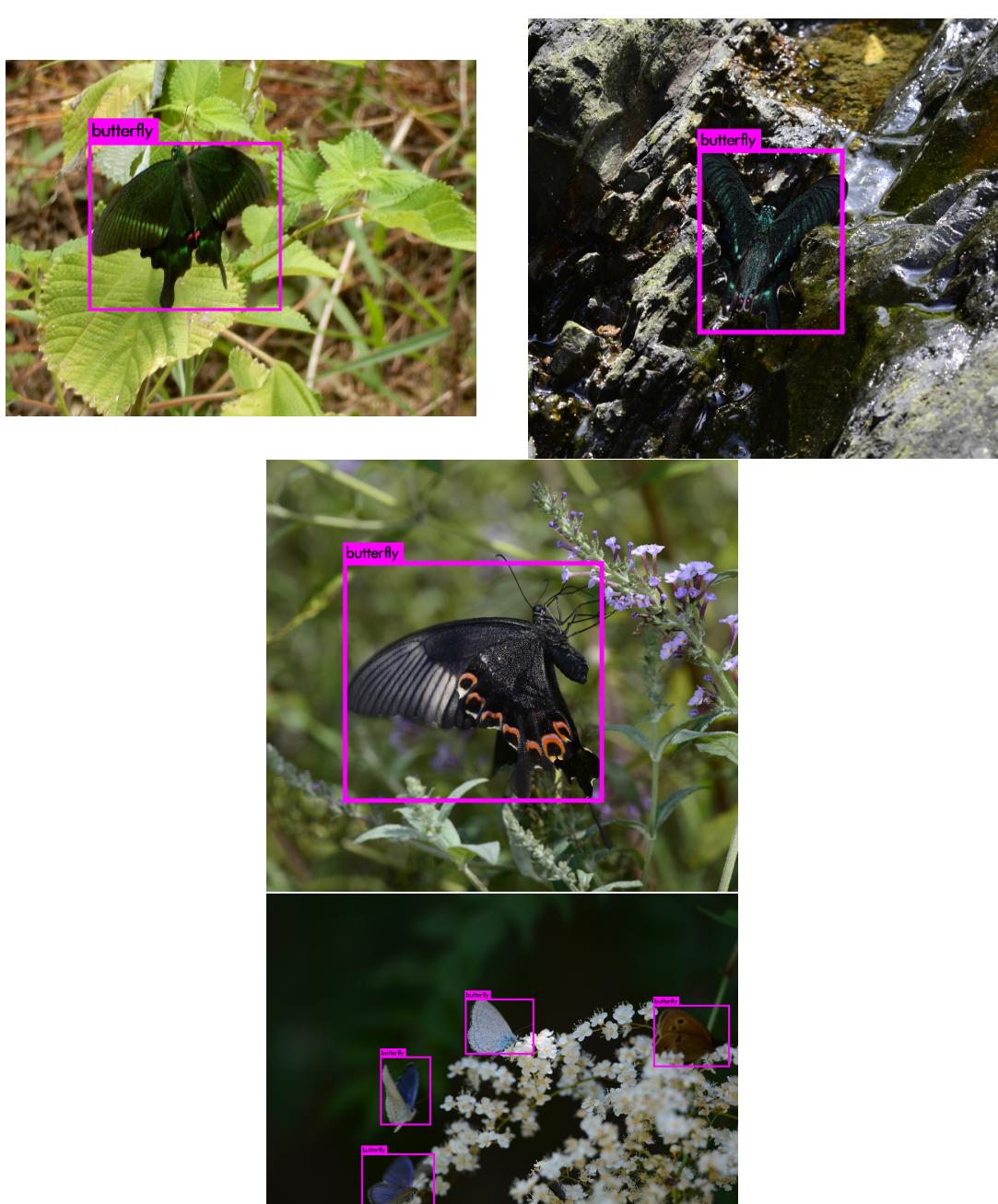


图 4.2 实验结果

4.3.1 实验结果分析

通过分析我们的实验结果, 我们不难发现, 数据集的好坏, 训练和测试参数的设置对最后模型的效果都有很重要的影响。

我们的第一个模型训练使用的是 D1, 这是一个将所有的照片都压缩成 800*600 的方式, 如前文所说, 在这样的压缩过程当中, 我们数据集中的图片产生了不同程度的失真。对测试造成了很不好的影响。同时, 在使用原图进行测试的时候, 因为输入层过小而使得图片得到了过分的压缩, 造成测试的不准确。

在我们的第二个数据集当中, 我们将矩形框的占有率提高到了 25% 以上。并且采用了较好的训练参数。但是, 在使用原图进行测试的过程当中, 我们需要将神经网络的输入层扩大。因为如果我们保持输入层不变, 则只会造成原图压缩之后目标矩形框的大小比训练集中的矩形框的大小小, 造成测试的不准确的情况。而如果在测试时输入层的大小过大, 则也会造成测试的精度下降, 因为矩形框的大小比训练集中矩形框的大小大。此时测试的效果也会降低。

4.4 总结

本章当中, 我们叙述了在我们的实验当中神经网络参数的设置和具体实验的结果。在我们的实验结果当中, 我们也可以发现, 大的输入层, 更多的特征提取不一定比小的输入层的效果要好。这也是和我们所要面对的数据集的质量所决定的。我们所面对的数据集所含的生态照片, 分辨率较高, 但是局部的噪声往往较大。对选框进行放大后使用较小的输入层, 能够较好的规避这方面的影响。

第5章 一个简易的深度学习训练和测试框架

5.1 概述

当前，因为深度学习的技术和研究发展迅速。出现了许许多多用于快速构建深度学习训练和测试的框架。例如 Google 的 Tensorflow 和 Facebook 的 Caffe2。这些框架预定义了很多深度学习过程中所需要的数学上的算法和构建深度神经网络所需要的卷积层等操作。下面是一些常用与深度学习的框架：

- **Tensorflow:** Tensorflow 是谷歌发布的用于深度学习的框架。其特点是将每一个计算过程看作一个完整的计算图。这样做的优势在于逻辑较为清晰，并且较为贴近原始的算法。让开发者可以直接计算过程进行构建，而不用考虑运行时较为复杂的计算情况。
- **Caffe:** Caffe 原是伯克利实验室对外公布的一个面向机器学习的框架。其特点是使用配置而不是代码来管理深度学习的计算过程。也就方便研究人员能够快速的构建其所需要的深度学习的计算过程，达到较好的效果。

在我们的实验当中，我们设计和使用了一个较为高级，也较为简便的面向深度学习训练和测试的构建规则 Easy_Builder。这套规则能够让我们更快的构建出较为严谨和易用的深度学习应用。简化工程的设计和实现过程。在我们的框架当中，所有的模型必须且只能实现两个公开的接口 train 和 predict。用于深度学习模型的训练和使用。

5.2 模型的训练

在深度学习领域模型的训练当中，往往有大量的信息需要工程人员进行处理。在我们的框架当中，我们将一次深度学习的训练需要使用的信息主要分成下面的两类。首先，是训练的数据信息，即训练的数据集和验证集。其次是控制信息，主要是训练的参数和预先训练的权重。在我们的框架中，我们使用一个 JSON 的文档来存储这些信息，使深度学习模型的训练能够高效的运行。一个示例的 JSON 文件如下所示：

Listing 5.1 Build.json Example

```

1  {
2      "traindatalist": "/train.txt",
3      "testdatalist": "/test.txt",
4      "classnameslist": "/classname.txt",
5      "classnum": 20,
6      "trainconfigfile": "/yolo-voc.cfg",
7      "pretrainedweightfile": "/darknet19_448.conv.23"
8  }

```

如上面的示例所展现的，所有的训练集数据文件和训练文件均使用相对路径来表示。这样做能够方便训练文件的迁移。在模型训练的过程当中，系统会自动加载这些相对路径并且将其转化成为绝对路径。从绝对路径当中读取对应的数据。这个系统的简要流程图如下所示：

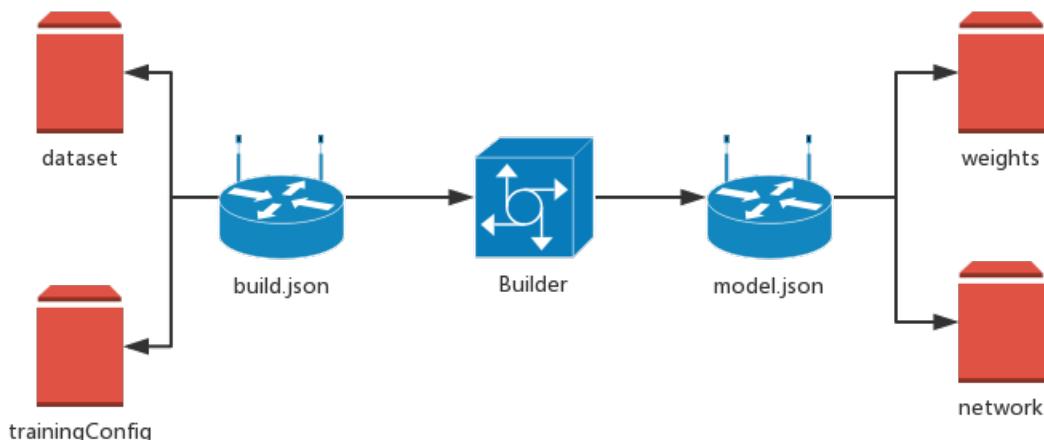


图 5.1 builder 系统结构图

训练完成之后，我们需要将训练好的模型存储下来。这个时候，我们仍然将一个训练好的深度学习的模型所拥有的数据分成两个方面。第一个方面是数据信息，这里包含了所生成的模型权重，另一个方面就是控制信息，即在训练模型的时候所涉及的所有的控制信息。我们将这些信息或存储这些信息的文件相对路径存储在一个 JSON 文件当中，使得这个模型在使用的时候能够被正确的加载。一个示例的 JSON 文件如下所示：

```

1 {
2     "classnum":1,
3     "trainconfigfile":"lab2/yolo-voc.2.0.cfg",
4     "dataconfig":"data",
5     "pretrainedweightfile":"lab2/yolo-voc_final.weights"
6 }
```

如上面所示，系统会将所有生成的文件和使用这个模型必要的文件全部复制到目标文件夹当中。并且在作为元数据的 JSON 文件当中使用相对路径来描述文件的位置。使得生成的模型能够更方便的进行移植。简化做实验的流程。

我们提供了一个 API 接口来进行模型的训练，如下所示：

```
1 static void train(char* tgtDir, char* tmpDir, char* srcDir, bool gpu);
```

我们的系统会在 srcDir 里面查找一个叫做 **build.json** 的文件。并且从中加载训练的所有信息进行训练。也就是，我们只需要很简单的代码便可以进行深度学习模型的训练。

5.3 模型的使用

上一节，我们描述了如何使用我们的框架方便的进行深度学习模型的训练。这一节，我们则介绍如何使用我们的系统所生成的模型。上一节，我们提到，我们的系统在生成模型的过程当中，会在目标文件夹中产生一个叫做”model.json”的文件。这个文件里包含所有加载模型所需要的数据和控制信息或者存储这些信息的文件的相对路径。那么，对于模型的使用，我们只需要让系统知道我们所需要处理的测试集的位置即可。而在这里，我们的框架大致提供了两种方法来加入测试集的信息。

第一种方法是修改 JSON 文件的信息，直接加入一个条目来表示测试数据集的位置。系统允许使用绝对路径和相对路径来表示测试集的位置。这样，这个 JSON 文件就从表示一个模型的元数据文件变成了表示一次深度学习测试的元数据文件。一个修改了的元数据文件如下所示：

Listing 5.2 model.json example

```

1 {
2     "classname":1,
3     "trainconfigfile":"lab2/yolo-voc.2.0.cfg",
4     "dataconfig":"data",
5     "testdatalist": "/test.txt",
6     "pretrainedweightfile":"lab2/yolo-voc_final.weights"
7 }
```

第二种方式则是使用编程的方式来加入测试集的信息，即这个框架存在一个 main.cpp 可供我们根据自身的需要进行修改。加入所有我们希望加入的信息。实现更加灵活的控制我们的深度学习过程。

一个使用这种方式进行测试的做法如下所示，我们只用区区几条代码便可以实现模型的测试。

Listing 5.3 example

```

1 int main()
2 {
3     char* darknetAddr = "./butterflyDetector/darknet";
4     char* tgtDir = "./tgtDir";
5     char* fileList = "./unittest/model/test.txt";
6
7     char* modelDir = "./unittest/model";
8     yoloDetection model;
9     model.loadFromExistedModel(string(modelDir));
10    model.predict(tgtDir, fileList, darknetAddr, 0.2);
11
12    return 0;
13 }
```

5.4 总结

本章，我们描述了在实验过程中设计的一个用于进行深度模型训练和使用的深度学习框架。在这个框架当中，我们要求所有的模型必须且只能实现两个公共的接口 train() 和 predict()。并且使用 JSON 文件来描述模型所需要的信息。这样做，无疑简化了结构的设计，方便深度学习实验的设计和执行，同时，便于这个框架的快速扩展。

第6章 总结

近年来，计算机视觉获得了很大的发展，产生了很多优秀的成果。而物体检测问题，则是其中的一个十分重要的核心问题。物体检测，因为图片中物体数量的随机性和图片处理的复杂性而成为了国内外众多学者研究的焦点。本文通过对目前较为典型的方法的调研，提出了一种基于深度学习的对生态照片中蝴蝶位置检测的研究。对未来生物学界对蝴蝶的研究，具有一定的意义。

本文通过与现有解决方案的对比，提出了一种新型的基于深度学习的对生态照片中的蝴蝶进行检测的解决方案。

本文的主要工作内容总结如下：

- 通过查阅大量相关文献，研究目前较为前沿的有关物体检测算法的研究和较为经典的方法，并分析这些方法在物体检测时的优势和劣势。
 - 对数据进行了较大幅度的清洗，首先，我们提出了一种较为简单的清洗方案，使得整个数据集的大小得到了压缩。然后，我们根据了第一种方法的弊端和我们对数据集内容的统计，提出了第二种基于随机化算法的解决方案。较好的对数据集进行了清洗。
 - 根据第一步中对现有的方法的认识和在第二步当中通过反复的比较和统计形成的对我们所要面对的数据集的认识，我们提出了一个以 Redmon[2] 等人提出的 yolov2 为基础的神经网络架构为基础的对生态照片中的蝴蝶进行检测的解决方案。并且通过调整学习率和测试与训练不同阶段的参数，不断的增强训练的效果。得到了最终的对于生态照片当中蝴蝶位置检测的解决方案。取得了较好的效果。
 - 一个方便深度学习实验搭建的思路。在我们的实验当中，我们使用了 JSON 来保存所有神经网络训练和运行所需要的信息。这样做，方便了我们快速进行神经网络的实验和测试。也为未来的扩展提供了巨大的方便。
- 针对未来的工作，这个解决方案还可以从以下几点进行扩展，使得这个解决方案对各种情形都能够较好的匹配。
- 如下图所示，我们在分析预测结果较差的测试样本的时候发现，我们发现，这

些样本的分辨率往往存在较大的问题。对此，我们可以采用一些图像处理的方法，将原本模糊的部分增强，以实现较好的检测效果。



图 6.1 一些较为模糊的照片

- 由于算法原本的关系，在涉及到距离较近的物体时，采用类似 yolo 思维的算法往往会将这些相距较近的物体识别成了一个物体，从而造成了较大的错误。这往往是因为 yolo 总是将图片按照固定大小的网格划分，然后以各个网格为中心生成候选框。如果两个物体相距较近，则很容易被选入一个候选区域，从而造成较大的错误。为此，我们可以缩小网格的尺寸，同时在检测的时候可以采用多次检测的策略，然后对检测的结果进行合并。

参考文献

- [1] Danyang Wang Jian Huang and Xiaoshi Wang. Novel single stage detectors for object detection. *arXiv*, 2016.
- [2] Justin Johnson. Detection and segmentation. *Stanford CS231 Lecture Notes*, 2017.
- [3] Ali Joseph. Yolo9000: Better, faster, stronger. *CVPR*, 2017.
- [4] Ross Joseph, Santosh and Ali. You only look once: Unified, real-time object detection. *CVPR*, 2015.
- [5] K.E.A. van de Sande J.R.R. Uijlings et al. Selective search for object recognition. *IJCV*, 2012.
- [6] Shaoqing Ren Kaiming He, Xiangyu Zhang and Jian Sun. Deep residual learning for image recognition. *arXiv*, 2015.
- [7] Andrew Karen. Very deep convolutional networks for large-scale image recognition. *ICLR*, 2015.
- [8] Deva Peiyun. Finding tiny faces. *CVPR*, 2016.
- [9] Trevor Ross, Jeff and Jitendra. Rich feature hierarchies for accurate object detection and semantic segmentation. *CVPR*, 2014.
- [10] Ross Shaoqing, Kaiming and Jian. Faster r-cnn: Towards real-time object detection with region proposal networks. *arXiv*, 2015.
- [11] Karen Simonyan and Andrew Zisserman. Very deep convolutional networks for large-scale image recognition. *arXiv*, 2015.
- [12] 大卫. 卡特. 蝴蝶. 中国友谊出版公司, 1993.
- [13] 侯琦谢娟英. 蝴蝶种类自动识别研究. *arXiv*, 2017.

致 谢

时光飞逝，如白驹过隙，四年的本科生活接近尾声。四年的时间里，曾经欣喜过也彷徨过，经历过悲伤也收获过快乐，很庆幸生命中能够遇到我的老师和同学，能够和他们成为朋友，在此非常感谢他们。

首先，感谢我的导师，龙舜老师。本文的工作就是在龙老师的指导下完成的，龙老师的渊博的知识开阔了我的视野给了我深深的启迪。在论文的写作过程和日常学习中，龙老师的严谨的治学态度和对学生负责任的精神给我留下了深刻的印象，也深深的影响了我的工作和学习中的做事态度。在此我向龙老师致以最诚挚的感谢。

在大三后的暑假，我在微软度过了3个月的学习时光，遇见了很多很好的同事和同学，他们不仅在学术上给了我巨大的帮助，同时他们对待工程和科学上严谨的态度和活跃的思维也深深的打动了我。希望未来能够再和你们度过一段愉快的时光。

最后，感谢我的父母。感谢父母多年来对我的无条件的付出和关爱，感谢他们一直以来的理解与支持