

项目 II：人脸关键点检测（建议 PyTorch 完成。不反对学员用个人喜好的其他框架。）

紫色：要学习的内容

红色：项目任务（或注意事项）

绿色：要回答的问题

## 一、素材：

数据：地址由班班告诉大家

包括 2000 张图片以及相应标注信息。

文件夹 2 个：

A. stage1：里面有第一阶段参考 prototxt 文件

B. stage3：里面有第二阶段参考 prototxt 文件

python 文件：

A. generate\_train\_test\_list.py  
生成训练与测试数据列表。

训练列表：train.txt

测试列表：test.txt

B. detector.py  
程序主体，用来进行模型训练/验证

C. data.py  
用来处理数据。由 detector.py 进行调用

D. predict.py  
调用训练好的模型预测关键点。同样由 detector.py 进行调用

## 二、stage 1：

**任务：**本阶段，阁下的任务，就是参照所给 python 程序以及项目说明，深入体会流程，并完成相应的学习任务(【紫色括号中的内容】)。所给的各个 python 程序，为参考代码。方便同学们在有困难时，进行查阅。最终写出自己的代码，比如 detector\_yourself.py，并运行，进而完成训练，生成第一阶段的 detector 模型。再次运行 detector\_yourself.py 并由其调用 predict\_yourself.py，应当可以用所训模型在人脸图片上画出关键点。

**综述：**

人脸关键点检测，目的是通过 CNN 方法，进而在已有人脸检测框的基础上【注意：不是在全图上】，进行关键点检测，输出关键点坐标。

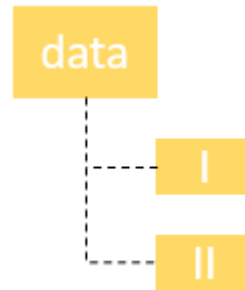
所谓人脸关键点，是一系列人为定义的点。人们主观认为，这些点，最能体现人脸信息，比如轮廓、五官样貌等。人脸关键点并不唯一，少到 4 点 5 点，多到 100 点 200 点都有。具体需要多少，要视实际情况而定。本项目训练结果，如果好，可以作为成果保留，所以点数适中，采用 21 点。

原始数据:

数据是任何 CNN 项目最为关键的一环，没有之一。甚至整个项目，如果没有数据，那么生成训练用数据能够占到整个项目一半以上的时间，有时，甚至贯穿整个项目。

为了方便各位的学习，我们已经将图片进行整理。**【实际工作中，这一步很可能将由各位亲手完成，千万不要忽略这一步的重要性以及耗时】**

各位拿到的数据，结构如下：



文件夹 I 与 II 中, 分别有 1000 张图共 2000 张。

I 与 II 中除图片外，还另有 label.txt，为标注信息。每行为一张图像的具体标注内容。共分 3 部分：1. 图像位置；2. 人脸框；3. 相对于整张图像的人脸关键点信息，顺序为 x1 y1 x2 y2... x21 y21。具体请看实例：

[illegible]

这是 label.txt 中一行所包含的信息。标注中，可能有<0 的情况出现，大家可以当做 0 或者忽略该行信息即可【**请牢记：此时所有数据坐标均是关于原始图像大小的。各位也请注意，通常情况下，原始坐标信息总是关于原始图像的。虽然，我们几乎不可能应用原始坐标，但是为了数据的可重用性，原始坐标信息总是应该保持为关于原始图像大小的**】

因此，在训练中，为了让程序知悉图片位置以及得以应用标注信息，我们需要完成“生成数据列表”的任务

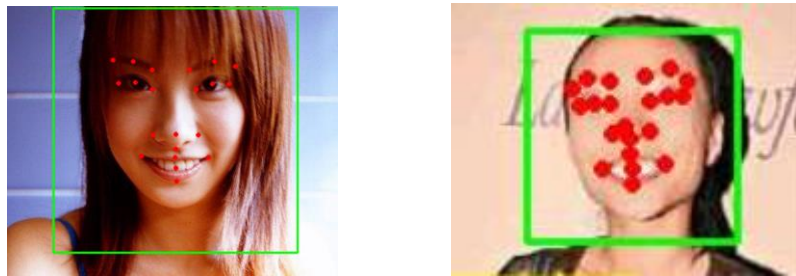
### 任务一：生成 train/test.txt

- A. 建议同学首先画出人脸边框以及相应关键点以熟悉程序操作、坐标表示以及检验标注数据是否正确如下图所示：



【为省地方，只截取了部分图】

- B. 我们可以看到，有时，由于标注的不仔细或不同标注任务标注标准不同，关键点可能会超出人脸框的范围。所以适当扩大人脸框的范围是必要的。这里，可以选取原始人脸框的 0.25 倍进行 expand。expand 时，请注意扩增后的人脸框不要超过图像大小。如下图：



【建议大家独立完成这部分。如果实在有困难，可以参考 generate\_train\_test\_list.py 中 expand\_roi 函数】

- C. 真正有用的部分，是人脸，以及人脸关键点。所以，希望对人脸进行截取。同时，截取过后，人脸关键点坐标即变为相对于截取后的人脸图了。此步非常简单，只需要用关键点坐标减去人脸边框左上角点坐标即可：

```
landmarks -= np.array([roi_x1, roi_y1])
```

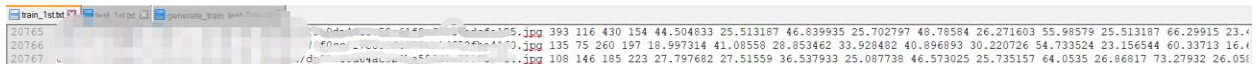
另外，还需注意的是，

- D. 生成 train/test.txt

此时，我们已可以生成训练以及测试数据集。(如果严谨些，还可生成验证数据集，这里只是项目举例，所以只简单生成训练与测试数据。) 训练与测试数据的比例，依个人喜好，通常 7:3 至 9:1 均能接受。并且，此时可以选择对数据进行 shuffle。

(不 shuffle 也可。Caffe 或 pytorch 里均有可以 shuffle 的功能)

最终，大家的训练集/测试集应该长成这样：



1. 每行对应一张扩增人脸
2. .jpg 为原图位置
3. 后续四个数字，为 expand 后的人脸边框坐标
4. 后续 42 个数字，为相对于人脸边框的人脸关键点坐标

- E. 验证：

为确保生成数据的准确性。生成后，仍需要验证。

具体可为，利用生成的数据截取人脸，并画出关键点，检验正确性。

以上，为数据准备的全部流程。请同学们体会其流程。

网络：

在 stage1 中，有 detector.prototxt。【.prototxt 为 caffe 的网络文件。这里，是一个非常简单的直线型网络，第一阶段大家可以直接应用这个网络架构。给大家这个文件的目的，一是给大家一个参考；二是想告诉大家用 caffe 的 .prototxt 可能是最简单的，可以直接看到网络具体长什么样子。大家可以把 detector.prototxt 中的内容直接 copy 到这个地址中：<https://ethereon.github.io/netscope/#/editor>，这个是 netscope，在线绘制 caffe 模型的工具。Copy 完成后，shift+enter 就可以看到网络结构了。如下图：】



所以，这里有任务：

## 任务二: 网络搭建

- A. 利用 netscope 查看示例网络结构
- B. 参考 detector.py 中 class Net, 回答问题:
  - 【1. 数据在网络中的维度顺序是什么? 】
  - 【2. nn.Conv2d()中参数含义与顺序? 】
  - 【3. nn.Linear()是什么意思? 参数含义与顺序? 】
  - 【4. nn.PReLU()与 nn.ReLU()的区别? 示例中定义了很多 nn.PReLU(), 能否只定义一个 PReLU? 】
  - 【5. nn.AvgPool2d()中参数含义? 还有什么常用的 pooling 方式? 】
  - 【6. view()的作用? 】
  - 【7. 体会 forward 中, 网络如何被构建。】
  - 【8. 注意返回值返回的并不是 loss】
- C. 如果可以, 请开始新建一个 detector\_yourself.py, 开始写自己的 class Net (当然可以用提供的网络, 但请一定重写一遍, 找找手感)
- D.

### 训练框架的搭建:

打开 detector.py, 由主函数 main()出发, 咱们开始开车。

#### A. 第一部分

首先, 是参数设置部分, 这些参数, 控制着整个程序。

```
def main_test():
    parser = argparse.ArgumentParser()
    parser.add_argument('--data_dir', type=str, default='./data', help='data directory')
    parser.add_argument('--model_dir', type=str, default='./model', help='model directory')
    parser.add_argument('--img_size', type=int, default=224, help='image size')
    parser.add_argument('--batch_size', type=int, default=32, help='batch size')
    parser.add_argument('--num_epochs', type=int, default=10, help='number of epochs')
    parser.add_argument('--num_workers', type=int, default=4, help='number of workers')
    parser.add_argument('--device', type=str, default='cuda:0', help='device')
    parser.add_argument('--seed', type=int, default=1, help='random seed')
    parser.add_argument('--log_dir', type=str, default='./log', help='log directory')
    parser.add_argument('--save_dir', type=str, default='./save', help='save directory')
    parser.add_argument('--load_dir', type=str, default='./load', help='load directory')
    parser.add_argument('--test_dir', type=str, default='./test', help='test directory')
```

希望大家在这里掌握:

- 【1. 由命令输入参数的方式】
- 【2. 尝试明白各参数含义】

#### B. 第二部分

```

# For single GPU
# For multi GPUs, nothing need to change here

print('==> Loading Datasets')

print('==> Building Model')
# For single GPU

```

此部分，是一些“程序控制代码”，包括：

- 【1. 如何设置 GPU】
- 【2. 如何将数据/网络传入 CPU/GPU】
- 【3. 如何读取数据】 (此部分会有后续任务与讲解)

#### C. 第三部分

```

criterion_pts =
optimizer =

```

此部分，是关于“训练控制代码”，希望大家学习的包括：

- 【1. 如何设置 loss】
- 【2. 配合后续周学习，loss 都有哪些。分别有什么作用(常用的即可)】
- 【3. 如何设置优化器】
- 【4. 配合第 8 周内容，常用的优化器有哪些】

#### D. 第四部分

```

if args.phase == 'Train' or args.phase == 'train':

elif args.phase == 'Test' or args.phase == 'test':

elif args.phase == 'Finetune' or args.phase == 'finetune':

elif args.phase == 'Predict' or args.phase == 'predict':

```

此部分是定义“程序处于什么阶段”的部分。

这里，我们看到有四个阶段。分别是：

##### a. 训练

此部分将引导我们进入“训练代码”。训练代码是程序主体。模型学习来源于此部分

##### b. 测试

此部分将引导我们进入“测试代码”。如何总体评价我们训练的模型好坏，将由此部分做出 (此部分会有后续任务)

##### c. Finetune

有时，我们会用别人训好的模型进行 finetune，或接着训练自己的模型。 (此部分会有后续任务)

##### d. 预测

有时，我们也会直观检测我们训好的代码。比如，应用训好代码，画出人脸关键点。 (此部分会有后续任务)



这里，希望大家学习的有：

【明确程序各个阶段的含义，有能力独自完成各个阶段】

### 任务三: 主体程序框架的搭建

请大家在理解的基础上，在 `detector_yourself.py` 中，完成自己的 main 程序主体。

注意，此时，你还不知道：

A. 如何读取数据

B. 如何写出 train/test/predict/finetune 部分代码

所以，这些可以空出来，我们接着开车

### 如何读取数据：

请大家聚焦

```
print('==> Loading Datasets')
train_set, test_set = get_train_test_set()
train_loader = 
valid_loader = 
```

此时，`get_train_test_set()` 函数一定显得无比突兀。

是的，数据的预处理与读取，就藏于这里。我们可以通过文档加载，了解到，这个函数，应该藏于 `data.py`。我们去看看。

#### A. 第一部分，关于主体：

首先，找到主体：

```
def get_train_test_set():
    train_set = load_data(
    valid_set = load_data(
    return train_set, valid_set
```

不难看到，根源在 `load_data()` 中。

另外，main 主函数是用于检验处理后的数据的准确性的。切记，随时检查准确性非常重要。我们一会再来看 main 函数

#### B. 第二部分，关于 load\_data()：

```
def load_data(phase):
    data_file = phase + '.txt'
    with open(data_file) as f:
        lines = f.readlines()
    if phase == 'Train' or phase == 'train':
        tsfm = transforms.Compose([
            Normalize(),           # do channel normalization
            ToTensor()             # convert to torch type: NxCHxW
        ])
    else:
        tsfm = transforms.Compose([
            Normalize(),
            ToTensor()
        ])
    data_set = FaceLandmarksDataset(lines, phase, transform=tsfm)
    return data_set
```

希望大家学习流程：

【1. 请注意，train 与 test 的数据处理可以不同。虽然这里是相同的】

【2. 按 train 与 test 不同，先定义数据变换及其顺序，这里是 `Normalize+ToTensor`，再将其作用在数据上，此处是 `FaceLandmarksDataset()`】

#### C. 【第三部分，关于变换 Normalize 与 ToTensor】：

请大家注意 `data.py` 中，class `Normalize` 与 class `ToTensor`。大家注意到这里面的 `def __call__` 是非显示调用的。大家只管写，pytorch 会帮我们进行调用的。

另外，传入的数据为 `sample`。这是一个 dictionary，代表了我们的数据包含了什么。这里，我们的数据只有两部分：

'image': 为我们的图像

'landmarks': 为我们的脸部关键点

Normalize: 作用由 channel\_norm 函数体现, 为了将图片 normalize。【请大家注意, 这是一个相对传统的做法, 目前, 人们发现, 做不做这个 normalize, 对最终结果的影响似乎不是很大。如果大家有兴趣, 欢迎大家去掉这一步, 并给出实验结果】

ToTensor: 作用是将数据转成 pytorch 可使用的格式

#### D. 【第四部分, FaceLandmarksDataset()】:

此部分为最终“将变换作用于数据”的部分。

可以看到, 这个 class 有三个部分, 分别是: \_\_init\_\_, \_\_len\_\_ 以及 \_\_getitem\_\_。此三部分, 均不需显示调用。

它们分别对应: 类的初始化; 一共有多少数据; 对于每批数据应当做何变换。

最关键的部分为 \_\_getitem\_\_, 这是变换的主体。

```
def __getitem__(self, idx):
    img_name, rect, landmarks = ...
    # image
    img = ...
    img_crop = ...
    landmarks = ...

    # you should let your landmarks fit to the train_boarder(112)
    # please complete your code under this blank
    # your code:

    sample = ...
    sample = self.transform(sample)
    return sample
```

我们返回的, 将是真正用于训练的数据, 因而, 它是一个 dictionary。代表了:

'image': 图像是什么

'landmarks': 关键点是什么

需要格外注意的是, 我们在处理图像的过程中, 有将图像 resize 到 train\_boarder(这里是 112x112), 但是我们的 landmarks 得到的确是相对于 expand 过后的人脸 crop。所以,

#### 任务四: 补全 FaceLandmarksDataset()

- 请大家新建 data\_yourself.py
- 按照刚才的流程, 重写读取数据部分
- 补全在 FaceLandmarksDataset() 中, 对于 landmarks 的操作。使你的 landmarks 是针对 train\_boarder size 的 (示例中是 112x112), 而非原始 expand facial crop 的。

【另, 这里处理数据是用的 PIL 库, 大家完全可以将其替换成熟悉的 OpenCV 库来进行图像操作。因为大家较熟悉 OpenCV, 所以这里选择 PIL, 为让大家知道, pytorch 中, 还可利用别的库来处理图像。不至于今后看到别人的代码陌生。另外请注意, PIL 库中图像是 RGB 顺序, OpenCV 为 BGR, 两者不同】

#### E. 第五部分, 关于 main 函数:

如前所述, 此部分的目的在于检验数据处理后是否依然正确。



### 任务五：补全 main 函数：

请在 data\_yourself.py 中完成：

```
if __name__ == '__main__':
    train_set = load_data('train')
    for i in range(1, len(train_set)):
        sample = train_set[i]
        img = sample['image']
        landmarks = sample['landmarks']
        ## 请画出人脸crop以及对应的landmarks
        # please complete your code under this blank

    key = cv2.waitKey()
    if key == 27:
        exit(0)
    cv2.destroyAllWindows()
```

以便验证自己的数据变换是正确的。

### **Train 部分：**

请大家移回目光到 detector.py。请看 def train()。

这里，就是训练代码的主体。包含两部分，其一是真正的训练部分；其二是 validating 部分。因为除了 train，我们要在 train 的过程中，实时监测训练结果，避免过拟。

请大家仔细阅读代码，相信以大家的能力，利用注释，足以理解代码。如果不懂，请大家查阅资料，并回答以下问题：

- 【1. print 的格式化如何实现的】
- 【2. optimizer.zero()与 optimizer.step()的作用是什么？】
- 【3. model.eval()产生的效果？】
- 【4. model.state\_dict()的目的是？】
- 【5. 何时系统自动进行 bp？】
- 【6. 如果自己的层需要 bp，如何实现？如何调用？】

### 任务六：完成训练的任务：

至此，训练的部分已经完整。大家可以尝试训练了。

- A. 在自己的 detector\_yourself.py 中完成 def train 函数。
- B. train 函数应由 main 函数 train 部分调用
- C. 请大家保存训练好的 model
- D. 代码中，没有存留 log 信息的代码，请大家自行完成这部分
- E. train 函数 return 的部分，作用在于可以存下各个阶段的 loss，用于后续绘制 loss 走势。示例中没有这部分，请大家自行完成
- F. 如果顺利，你应当至少在 train 时，loss 低至 3 以下；在 validation 时，loss 低至 9 以下。如果你在应用自己的 criterion，那么请忽略此条。一切以画出的结果是否准确为准

### Test、Predict 与 Finetune 部分：

在 detector.py 中，还有 Test、Predict 与 Finetune 三部分没有代码，分别在 main 函数 args.phase==Test、Predict 与 Finetune 中定义。

请注意，Test 的目的是直接利用已训练好 model，作用在 test 数据集上。（为简便，这里 test 和 valid 可用一个数据集。）看平均 loss。

Predict 目的为利用已训练好 model 作用在某张图片上，画出预测的 landmarks，直观看效果。相当于 model 的应用。

Finetune 为利用已训练好 model，继续训练。（通常，finetune 会需要更小的 lr）。

#### 任务七：完成 Test\Predict\Finetune 代码：

A. 请学会 load 已有 model

B. Finetune 时，有时还要固定某些层不参与训练，请回答如何 freeze 某些层。

至此，一套完整的流程已经全部建立。

我们涉及到了：处理原始数据、网络搭建、流程控制、数据预处理、训练、检验等多个步骤。请好好消化这些内容。如果你觉得车速尚可，那么，我们，又要提速了。

### 三、stage 2

**任务：**此阶段，是对于上一阶段的补充。大家应该也能感受到，stage1 仅仅是起步。毕竟，我们的网络很简单、我们 loss 很简单、我们的训练策略很简单、我们的数据很简单等等。总之，一切都很简单。所以，这个阶段，是使 stage1 的各个部分变得 fancy。没有一定之规，大家各显神通。总之，希望大家能给出一个秒杀 stage1 的 model。

#### 任务一：关于数据：

我们的数据预处理非常简单。所以还可以尝试很多：

1. 不要 normalize 可以么？
2. 数据增广没做啊，做下试试？比如：水平翻转、小角度旋转、是否可以平移？

#### 任务二：关于训练方法：

我们的训练采用最基本的 SGD。课上，我们讨论了更多方法，所以，为何不尝试呢？

1. 如果换用 Adam 呢？请找出 Adam 的 lr
2. 如果一开始用 Adam，之后换成 SGD 呢？
3. 如果用 step 改变 lr 呢？
4. 如果加上 batch normalization 呢？

#### 任务三：关于网络：

我们的网络非常简单，就是个线性网络。如果用其他的呢？

1. 比如，resnet？
2. 比如 fpn？
3. 比如其他的？
4. 尽量避免过拟

以下为选做内容：

任务四：关于目标：

我们的目标非常简单，就是直接回归坐标。但有的时候，我们也会回归 heatmap。这也可以是一个可以尝试的部分。

任务五：关于 loss：

我们的 loss 非常简单，就是 MSE，可以尝试下其他 loss，比如 smooth l1 loss。如果更改了训练目标，也应当配合训练目标适当更改 loss。

#### 四、stage 3

现实项目往往更加复杂。比如：现实情况下，你怎么能保证输入就是张人脸呢？因此，真实场景中，是否应该再加入分类分支呢？

任务：完成真实场景下人脸关键点检测：

A. 为网络加入分类分支。

真实场景下，不能保证输入就是人脸。

如果没有分类分支，那么一旦输入为非人脸，虽然此时不应输出任何关键点，但是网络势必仍然会输出伪关键点。因而一种解决方法就是，为网络加入分类分支。如果分类认为输入图片为人脸，则输出人脸关键点；如果分类认为输入为非人脸，则此时不输出人脸关键点。

B. 生成非人脸数据。

非人脸数据的生成可由图片中不含有脸的部分得到。可以认为，如果一个 image crop，其与人脸重叠部分  $iou < 0.3$ ，就是非人脸。

C. 重新生成 train/test.txt

我们以前生成的 train/test.txt 格式如下：

文件名 | 人脸框 | 人脸关键点

现在，我们要生成新的 train/test.txt，格式如下：

文件名 | 图像框 | 人脸关键点 | 1 【此时是人脸，称正样本】

文件名 | 图像框 | 0 【此时非人脸，称负样本】

D. 对于 loss 的处理：

这是最复杂的部分：

Loss 分为两部分：a. 分类 loss + b. 坐标 loss

关于 a：需要注意正负样本比例。同时，我们还可以利用 weighted cross entropy loss 控制侧重于训练正样本还是负样本。大家应该学习如何使用 weighted loss；

关于 b：需要注意，只有正样本拥有这个 loss，负样本，是没有这个 loss 的。所以如何来控制？可以在样本计算 loss 时加入 mask，这个 mask 可以由 label 来生成。比如，一个 batch 有 4 个样本，分别为：负样本、人脸、负样本、负样本。则这个 mask 可以为 [0, 1, 0, 0]。此时，真正计算 b loss 的，就只有第二个样本了。关于 mask 的应用，大家可以参考 [yolo-pytorch](#) 版本。比如这个连接中 response、not response loss 的部分。

除此之外，由于 loss 分为两个部分，我们还可以给两个部分加入不同的权重，权重多的，会被侧重训练。如：

$$\text{Loss} = a * \text{loss\_a} + b * \text{loss\_b}$$

此时，如果  $a > b$ ，则会侧重训练  $\text{loss\_a}$ 。

#### E. 关于检测：

除了对于坐标  $\text{loss}$  的监测，还要对分类  $\text{loss}$  进行监测。并给出  $\text{accuracy}$ 。同时，此时的  $\text{accuracy}$  不能仅仅是总体样本的  $\text{accuracy}$ ，我们还要分别看正负样本的  $\text{accuracy}$ ，所以，这个需要大家去统计各个  $\text{batch}$  中，正负样本的个数并加以计算

至此， $\text{stage3}$  也算是告一段落。

$\text{Stage3}$  是较为复杂的综合类问题。是可以当做真实项目处理的实际问题。可以看到与  $\text{stage1}$  的区别。请大家体会其中不同，并记住其中所学

## 五、总结

通过三个阶段，每个阶段的提高，希望同学们能有所收获。

$\text{Stage1}$  的目的，仅仅在于熟悉流程，熟悉编程。

$\text{Stage2}$  的目的，在于应用一些小技巧和一些常用的框架，属于“熟悉主流文化”

$\text{Stage3}$  的目的，是希望给同学一点面对实际问题的感觉，里面会涉及到更多技巧与思想。

希望同学们 3 个  $\text{stage}$  下来，真的有所收获。另外，如果效果不错，这个  $\text{model}$ ，是真的可以在以后用在实际工作中的。这是各位的成果，加油~