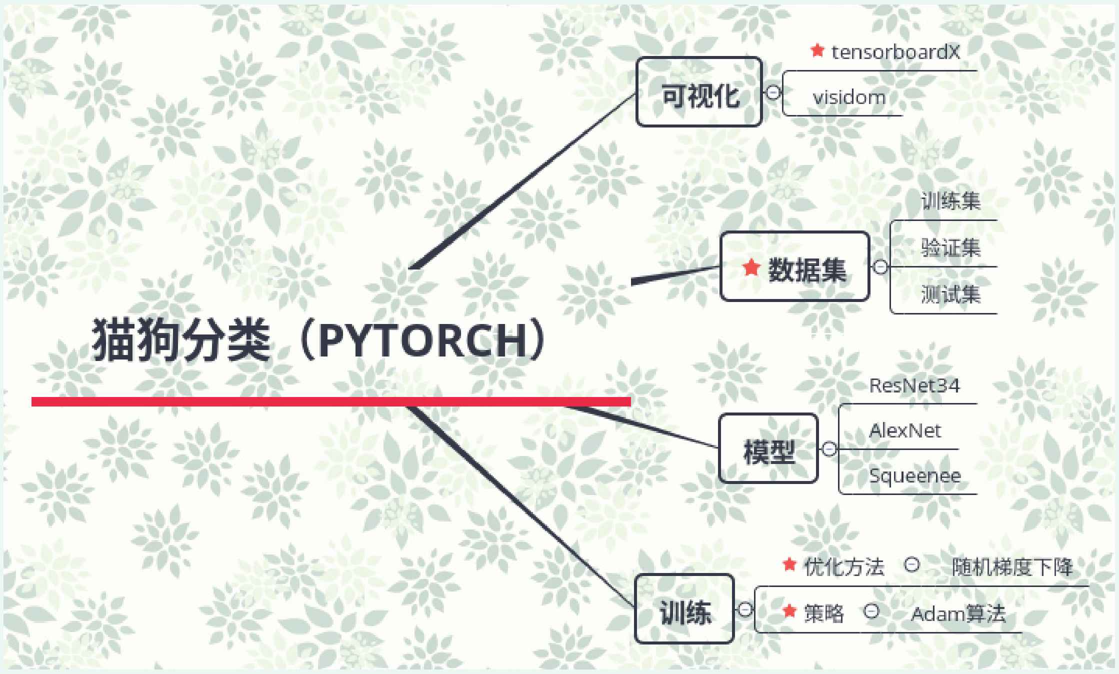
猫狗分类问题



0.参考

Pycharm快捷键

分类模型评估

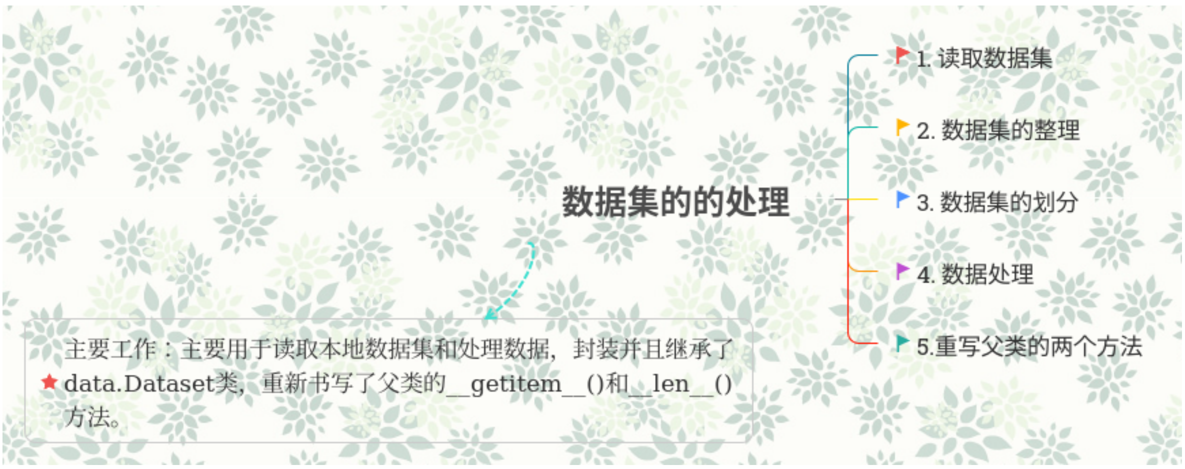
ResNet34网络

1.总体框架梳理

1.1 数据集

> 数据集为Kaggle中Cat&Dog数据集
>
> **数据集下载**

具体如下：

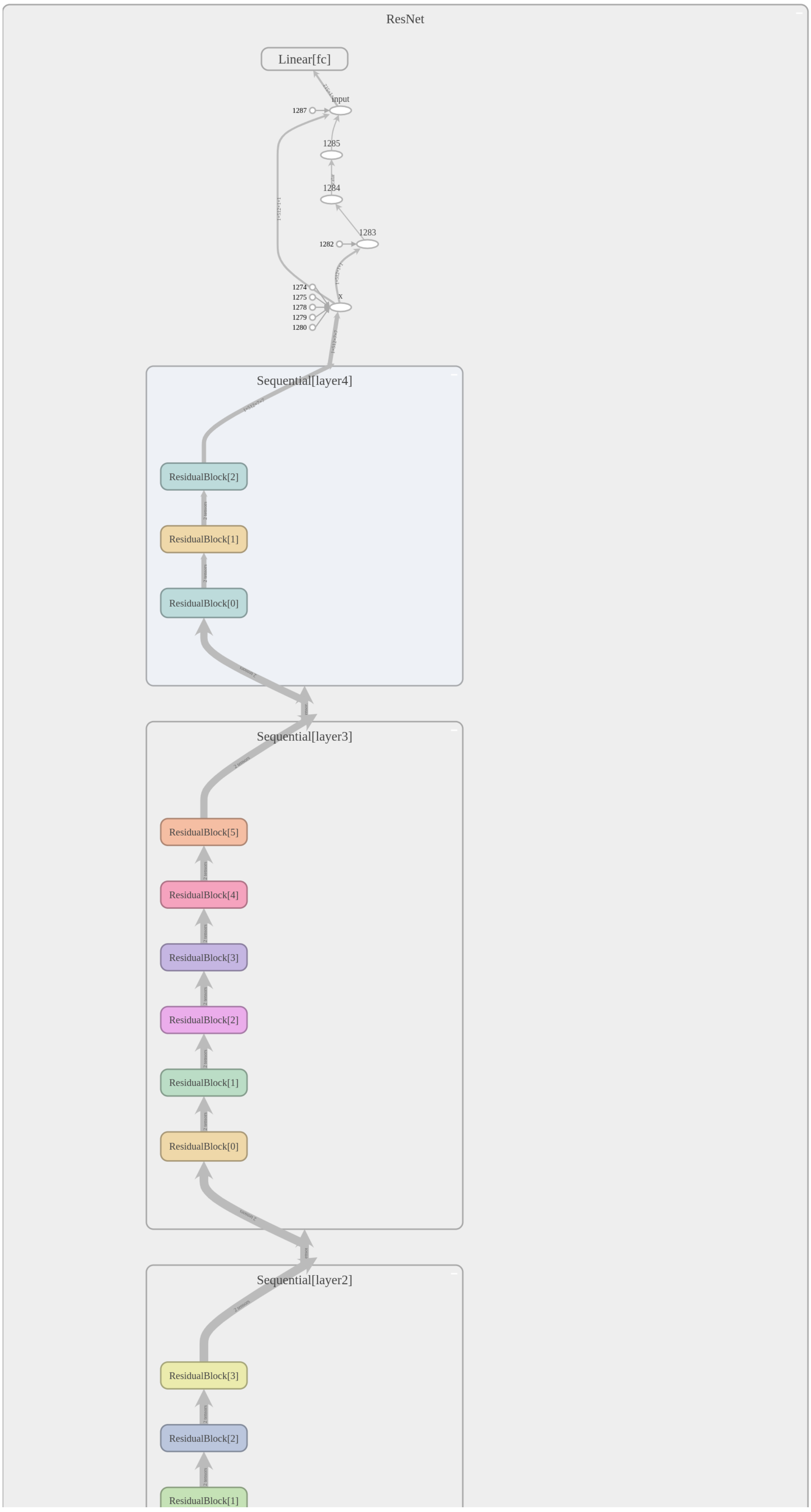


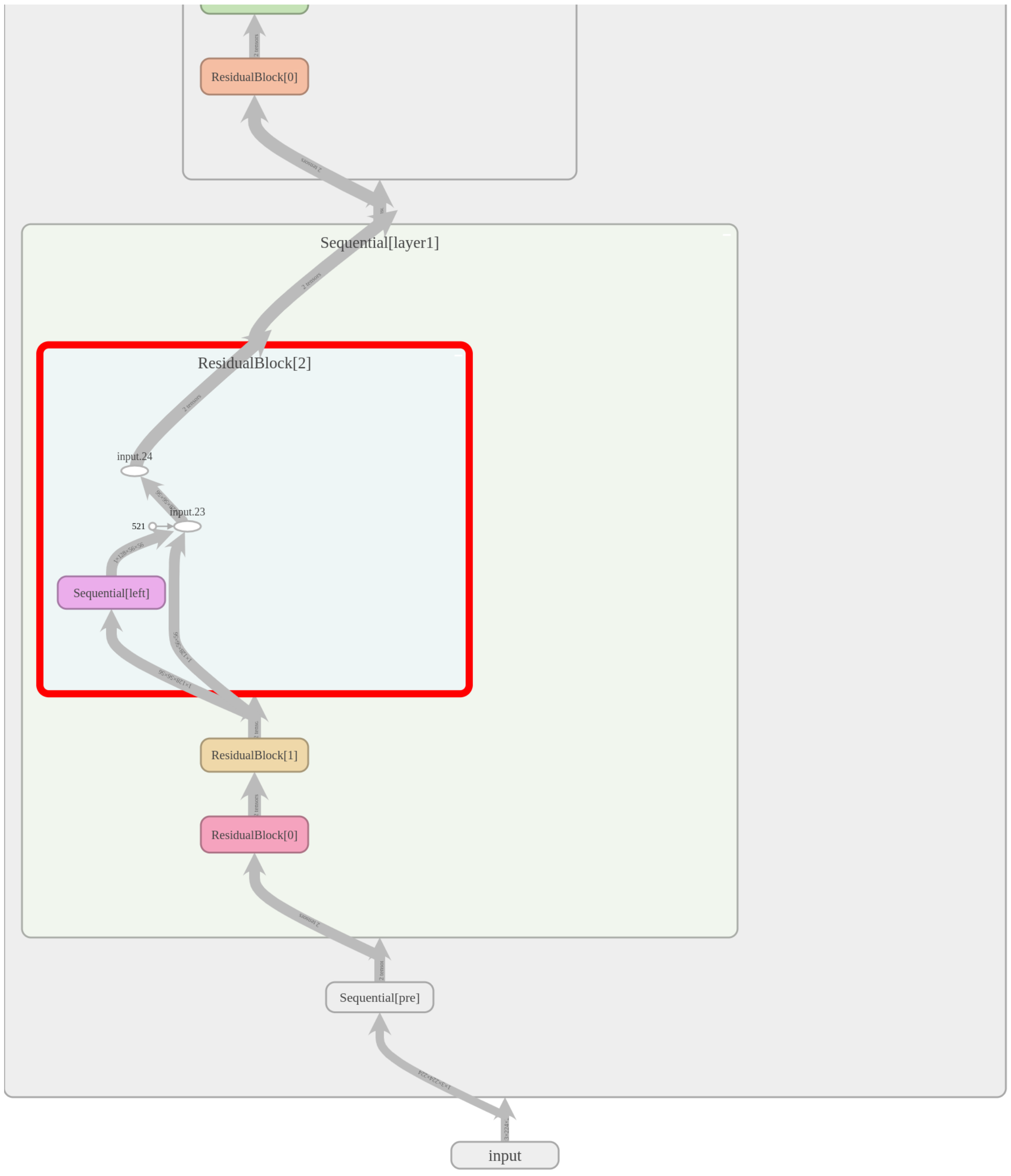
数据集的的处理
1. 读取数据集
2. 数据集的整理
3. 数据集的划分
4. 数据处理
5.重写父类的两个方法
主要工作：主要用于读取本地数据集和处理数据，封装并且继承了data.Dataset类，重新书写了父类的__getitem__()和__len__()方法。

1.2 模型

ResNet34

> 问题：
>
> (1)为什么网络的深度很重要？
>
> 答：因为CNN能够提取Low/mid/high-level的特征，网络的层数越多，意味着提取到的不同等级的特征越丰富。并且，越深的网络提取的特征越抽象，越具有语义信息。
>
> (2)为什么不能简单的增加网络的层数？
>
> 答：梯度消失和弥散

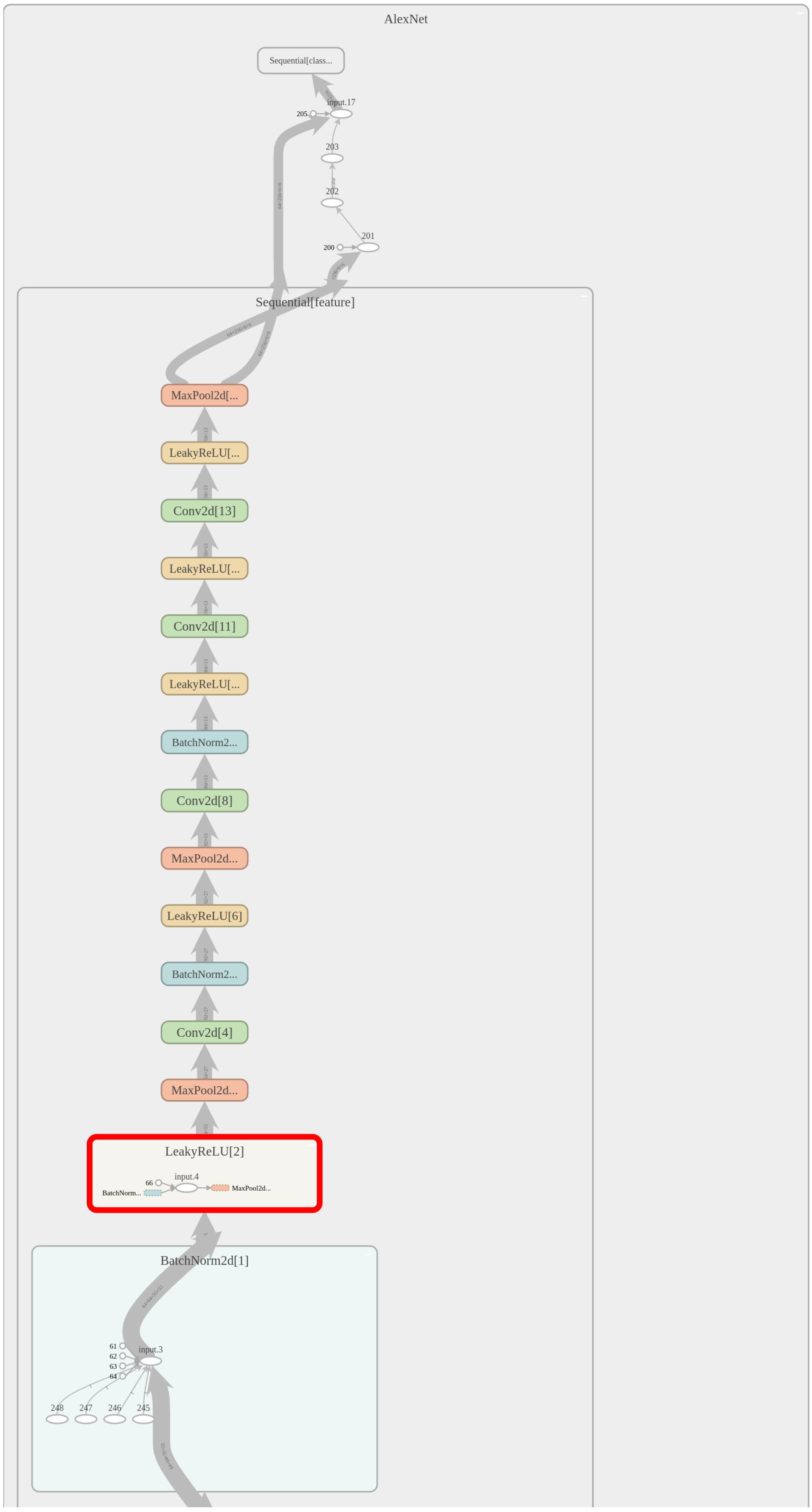

ResNet
Linear[fc]
input
1287
1285
1284
1283
1282
1274
1275
1278
1279
1280
x
Sequential[layer4]
ResidualBlock[2]
ResidualBlock[1]
ResidualBlock[0]
Sequential[layer3]
ResidualBlock[5]
ResidualBlock[4]
ResidualBlock[3]
ResidualBlock[2]
ResidualBlock[1]
ResidualBlock[0]
Sequential[layer2]
ResidualBlock[3]
ResidualBlock[2]
ResidualBlock[1]

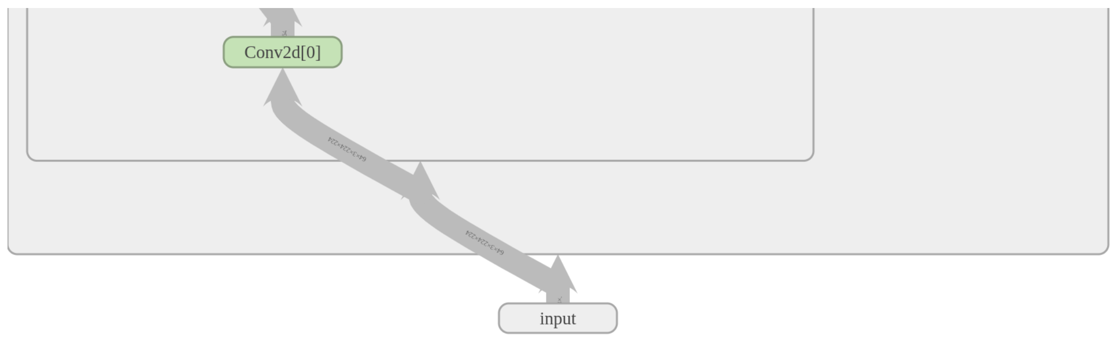


AlexNet变体：

> 说明：AlexNet变体模型在原有的AlexNet模型的基础上，增加了BN层，并且消除了全连接层中的Dropout层，因为BN会带有一点正则化的效果，在使用激活函数时替换Relu为Leaky_Relu.
>
> **Relu和Leaky-Relu的区别**

模型图：


AlexNet
Sequential[class...
input.17
205
203
202
201
200
Sequential[feature]
MaxPool2d[...
LeakyReLU[...
Conv2d[13]
LeakyReLU[...
Conv2d[11]
LeakyReLU[...
BatchNorm2...
Conv2d[8]
MaxPool2d...
LeakyReLU[6]
BatchNorm2...
Conv2d[4]
MaxPool2d...
LeakyReLU[2]
66
input.4
BatchNorm...
MaxPool2d...
BatchNorm2d[1]
61
62
63
64
input.3
248
247
246
245



1.3 训练模型

```python
def train():
    # 1.选择合适的model
    model = getattr(models, opt.model)()
    model.cuda()

    # 2.得到批量化的数据
    train_data = DogCat(root=opt.train_root, train=True)
    val_data = DogCat(root=opt.train_root, train=False)

    train_dataloader = DataLoader(train_data, batch_size=opt.batch_size, shuffle=True, num_workers=opt.num_works)
    val_dataloader = DataLoader(val_data, batch_size=opt.batch_size, shuffle=False, num_workers=opt.num_works)

    # 3.设置参数
    loss_meter = meter.AverageValueMeter()  # 计算平均值和标准差
    confuse_metrix = meter.ConfusionMeter(2)  # 计算混肴矩阵

    criterion = nn.CrossEntropyLoss()
    optimizer = t.optim.SGD(model.parameters(), lr=opt.lr, momentum=0.9)

    writer = SummaryWriter()
    for epoch in range(opt.epoch_nums):  # 设置迭代次数
        # 4 执行批量化的数据
        print("Epoch :", epoch)
        loss_meter.reset()
        confuse_metrix.reset()
        for ii, (data, label) in tqdm(enumerate(train_dataloader)):
            input1 = data.to(opt.device)
            target = label.to(opt.device)
            optimizer.zero_grad()
            score = model(input1)
            loss = criterion(score, target)
            loss.backward()
            optimizer.step()

            # 记录数据
            confuse_metrix.add(score.detach(), target.detach())
            loss_meter.add(loss.item())
            if (ii + 1) % 20 == 0:
                print("loss_meter", loss_meter.value()[0])
                writer.add_scalar("Loss", loss_meter.value()[0])

        # 改变学习策略，逐渐取向最优解
        if epoch == 50:
            opt.lr = 0.001
    # 5. 保存网络数据
    writer.close()
    t.save(model.state_dict(), opt.model_path)
```

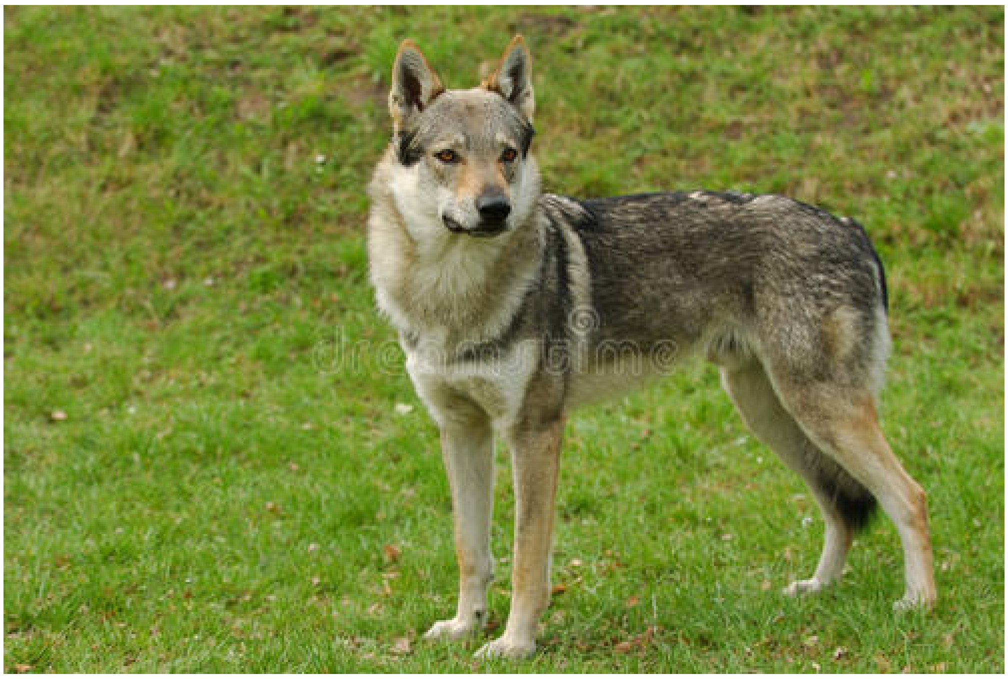
1.4 可视化

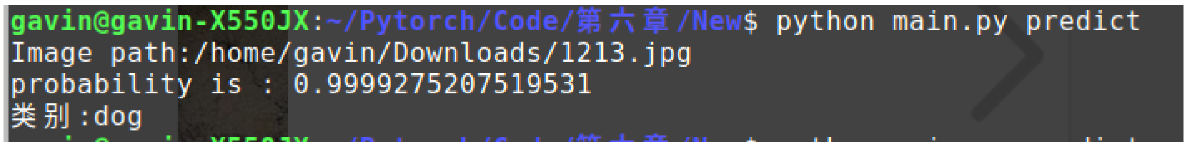
> 此时的training loss处于下降的趋势，说明此时的网络正在学习参数，由于此次迭代只限制在60次，训练的训练集数据集为2000张照片，可以从图中看出此时的还未达到收敛点，但此时应该使用测试集来判断此时是否过拟合。

Training loss:

1.5评估

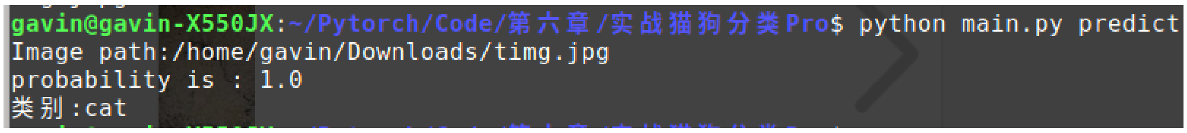
测试1：





```
gavin@gavin-X550JX:~/Pytorch/Code/第六章/New$ python main.py predict
Image path:/home/gavin/Downloads/1213.jpg
probability is : 0.9999275207519531
类别:dog
```

测试2：



```
gavin@gavin-X550JX:~/Pytorch/Code/第六章/实战猫狗分类Pro$ python main.py predict
Image path:/home/gavin/Downloads/timg.jpg
probability is : 1.0
类别:cat
```

2.出现的问题

1. AssertionError: number of predictions does not match size of confusion matrix

> 解决方法：confusion matrix的种类树木与评测参数不一致
>
> **meter的用法**

2. loss不下降，一直维持在0.69左右

> **解决方案一**

3.知识点

3.1Python用法

1. python中文件打包

   链接1：文件打包

   链接2：文件打包

2. python eval()函数

   链接

3. ipdb（debug）包的使用

   链接

4. time.strtime

   time.strftime() https://www.runoob.com/python/att-time-strftime.html

5. torchnet.meter的学习

> # import torchnet
>
> ### tnt.Meter
>
> Meters provide a standardized way to measure a range of different measures, which makes it easy to measure a wide range of properties of your models.
>
> **有三个基础的方法：**
>
> - add() which adds an observation to the meter.
> - value() which returns the value of the meter, taking into account all observations.
> - reset() which removes all previously added observations, resetting the meter.
>
> ### tnt.AverageValueMeter(self)
>
> The tnt.AverageValueMeter measures and returns the average value and the standard deviation of any collection of numbers that are added to it. It is useful, for instance, to measure the average loss over a collection of examples.
>
> ### tnt.AUCMeter(self)
>
> The tnt.AUCMeter measures the area under the receiver-operating characteristic (ROC) curve for binary classification problems. The area under the curve (AUC) can be interpreted as the probability that, given a randomly selected positive example and a randomly selected negative example, the positive example is assigned a higher score by the classification model than the negative example.
>
> ### tnt.ConfusionMeter(self, k[, normalized])
>
> The tnt.ConfusionMeter constructs a confusion matrix for a multi-class classification problems.

```
import torchnet.meter as meter

m = meter.AverageValueMeter()

for i in range(10):
m.add(i)

print(m.value()[0])
```

```
m.reset()#重置数据

for i in range(10, 20):
m.add(i)

print(m.value())
```

4.5 (14.5, 3.0276503540974917)

```
import torch
mtr = meter.ConfusionMeter(k=3)
output = torch.Tensor([[.01, 0.01, 0.1], [10, 11, 10], [0.2, 0.2, .3]])
if hasattr(torch, "arange"):
target = torch.arange(0, 3)
else:
target = torch.range(0, 2)

print(target)
print(output)
mtr.add(output, target)
```

tensor([0, 1, 2]) tensor([[1.0000e-02, 1.0000e-02, 1.0000e-01], [1.0000e+01, 1.1000e+01, 1.0000e+01], [2.0000e-01, 2.0000e-01, 3.0000e-01]])

```
mtr.value()
#生成一个onehat向量
```

array([[0, 0, 1], [0, 1, 0], [0, 0, 1]], dtype=int32)

```
meter.confusionmeter??
```

总结