可解释的神经网络实战

PYTORCH 版

Dezeming Family

Dezeming

Art. No 0
ISBN 000–00–0000–00–0
Edition 0.0

Cover design by Dezeming Family

Published by Dezeming
Printed in China

目录

前言及简介

DezemingFamily 系列文章和电子书**全部都有免费公开的电子版**，可以很方便地进行修改和重新发布。如果您获得了 *DezemingFamily* 的系列电子书，可以从我们的网站 *[https://dezeming.top/]* 找到最新的版本。对文章的内容建议和出现的错误也欢迎在网站留言。

0.1 本文前言

人们在试图解释 AI 的道路中已经走了很多路，试图思考 AI 究竟学到了些什么。

简单的神经网络模型肯定是不具备语义理解能力的，但它却可以做出一些语义分类的功能，比如区分猫、狗和海豚等各种动物。使用一些网络攻击方法，就可以把一张人眼看着就是猫的图像让神经网络识别为海豚，这种现象不得不引人深思——机器到底学到了些什么？

我们希望机器能够学到我们希望它学到的东西，比如为什么我们会认为这张图像是一只海豚，是因为它有鳍，嘴巴尖尖的，而且看起来不像鲨鱼一样具有攻击性；为什么会认为另一张图像是一只猫咪，是因为它耳朵是竖起来的，浑身毛茸茸的很可爱。我们也希望神经网络是这样辨识动物的，而不是总结一些很奇怪的规律，从而容易被攻击网络攻击成功（我们本书并不介绍对抗攻击，但对抗攻击确实是理解神经网络的一个很有用的手段）。

如果我们发现神经网络最终只能靠颜色来进行区分动物，比如白色橘色理解为猫，蓝色理解为海豚，那么这个神经网络就是不合格的，也是难以被信任的。一些重要的决策任务，例如是否可以贷款给某人，我们希望机器能够给出合适的理由来说明它做出的判断，而不是因为根据这个人的性别就直接拒绝贷款。

当然，可视化并不代表神经网络是“可以解释的”，只是通过这些方法我们可以获得更多的思考，也会对机器有更深的认识。至于如何去理解“神经网络的可解释性”，恐怕区区几种方法也难以给出明确的答案，这是一个长期积累和发掘的过程。何况，图像识别中，我们的网络只是在学习如何把动物区分开，而不是学习怎么认识动物，这是两个不同的事情——或许，当网络无限庞大，动物种类非常多的时候，或许网络可以从“如何区分”逐渐过渡到“如何认识”这个过程。

本文我们讲解一些简单的神经网络可解释性研究，探索究竟是什么因素导致神经网络可以识别出一些物体，以及神经网络层中学到了些什么东西。并以前两本书为基础，研究如何去理解神经网络。当然，学习和实践可解释性神经网络在本书中最重要的意义在于我们可以更好地掌握 pytorch 的编程以及网络模型的搭建。

本文源码分为两个部分，一个是本人实现的最大化输出激活的代码（在 me 目录下），一个是来自于 [11] 的更加完备的 VGG 可视化某一层的源码（在 github-utkuozbulak 目录下）。

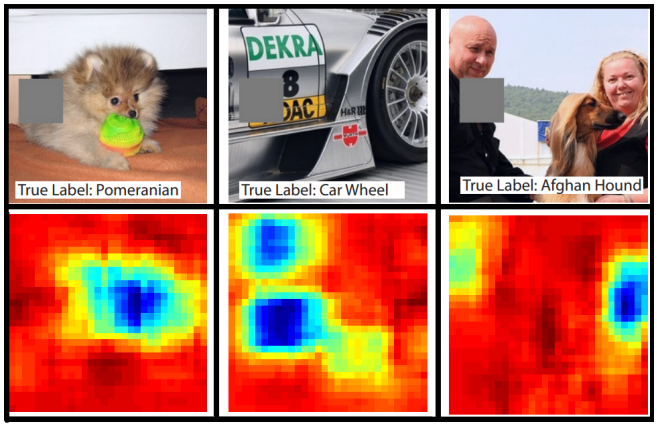
1. 深度学习与可解释性

本章介绍深度学习中的可解释性一般概念，讲述一些基本方法。但是由于本书重在实践，所以不会对涉及到的论文过于详细地讲解。

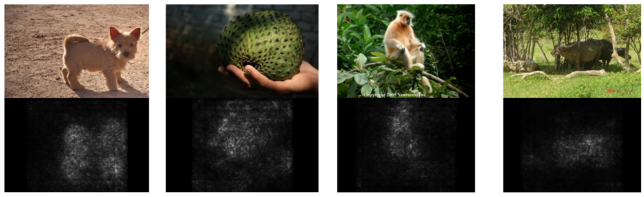
1.1 局部可解释性

人们常认为神经网络是一个黑盒子，因此对它并不信任。而正如 [1] 所说的那样，深度学习可解释就是为了让人们更信任它，而并不是为了真正理解它是怎么工作的（我们也做不到这一点）。

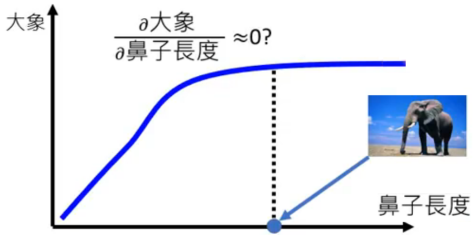
局部可解释性的方法一：我们可以把图像目标遮挡一部分然后丢入到网路中，查看网路对它的分类损失率，然后再遮住另一部分，查看损失率。我们可以据此找到遮挡某部分区域以后让网络最不容易判断的区域，分析是否合理 [2]：



局部可解释性方法二：对输入中任意一个像素使用一个扰动，然后查看对输出有多大影响，得到 $\frac{\Delta y}{\Delta x}$ 作为像素值（其实就是计算梯度 $\left|\frac{\partial y_k}{\partial x_n}\right|$），下图越亮的部分表示扰动越大：



但还有一个问题 [1]，就是关于梯度，比如我们识别动物，例如大象和猪的区分，只要鼻子够长，就会被认为是大象，而小象和大象不管鼻子多长都会能被正确辨认出，也就是说在鼻子很长的区域，$\frac{\partial\ Is\ giraffe}{\partial\ Neck\ length} \approx 0$，这样就导致当鼻子长短扰动时，梯度基本不变（梯度不再是衡量重要性的标准），这种问题可以在论文 [3][5][4] 中改善。



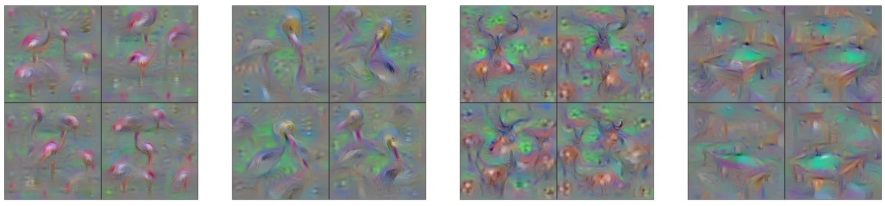
1.2 全局可解释性

全局可解释性，比如我们训练好的模型，我们意图找到让输出最大的输入：

$$x^* = arg\ \max_x y_i \tag{1.2.1}$$

比如对于手写数字辨识系统，我们希望找到一个输入，该输入可以让输出为 5 的概率最大（我们以后简称为最大化输出的输入，这里的输出指的是各种输出类别）。

在 [10] 论文中，可以看到一些最大化激活某些动物类型的输入，需要注意的是需要给输入设定正则化，如下图所示：



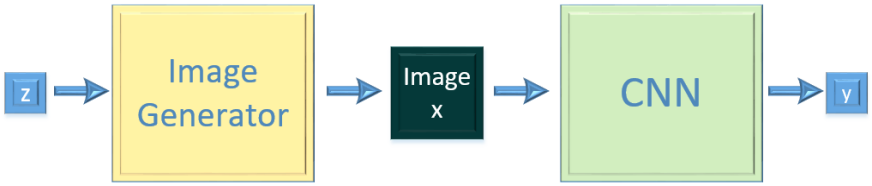
我们的模型正则化参数不同，得到的最大化输出的输入也不会相同，甚至可能会相差很多。

我们也可以利用一个 GAN 网络，生成一幅图像，然后再让这幅图像可以作为最大化输出的输入：

$$x = GAN(z) \tag{1.2.2}$$

$$z^* = arg\ \max_z y_i \tag{1.2.3}$$

直接最大化输出的输入一般而言人很难理解，但由于 GAN 生成的图像都是人能够理解和看懂的，所以这样就能产生人们差不多能够理解的输入 x：



可以看到，对于神经网络 CNN，它“认为”的最像红脚鹬以及火山的图像分别是左右四张图，说明网络有一定的理解能力。

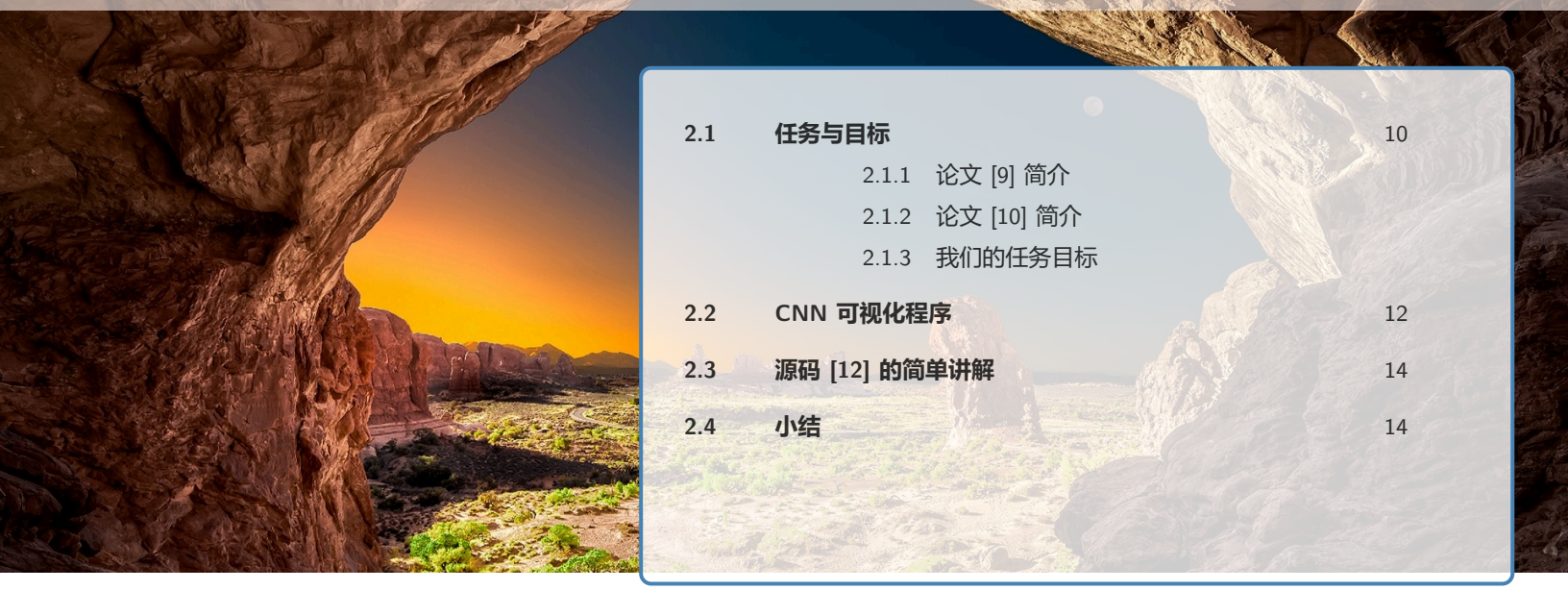
我们也可以使用更简单的模型来解释复杂的模型，比如利用线性模型来解释深度学习模型。实践中，就是对于一个模型的输入输出，让另一个模型得到的结果更接近它的输出。LIME 本质上就是这样的一种方法：

(a) Original Image (b) Explaining *Electric guitar* (c) Explaining *Acoustic guitar* (d) Explaining *Labrador*

把图像分成一堆碎片，我们可以得到不同的碎片组合来最大化网络的输出的碎片组合，例如上图中的最右边图可以最好的显示出这是拉布拉多犬。我们可以对 N 个碎片分别编号为 $0N-1$，然后定义一个 N 维向量，存在某个碎片时对应编号为 1，否则为 0，这样就可以将原来的图像抽象为一个线性向量（N 维向量）。

简单的模型，例如深度比较低的决策树，也可以更好的理解分类。但是我们要控制决策树的深度不能太深，在有的文章中把决策树深度也作为损失值，训练了一个神经网络 B 来预测决策树深度 [8]。

2. 最大化输出的输入



本章是实战篇，讲解如何在程序中实现训练输入使得输出最大化。

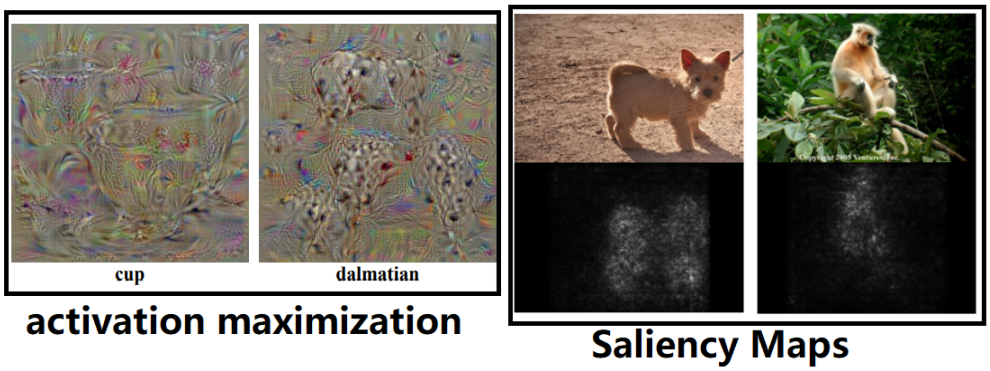
2.1 任务与目标

过去，对于 DNN 的研究大致分为两个不同的方向：一是以数据集为中心，需要一个训练好的 DNN 并在该网络上进行前向运算，从训练或测试集挑选图像，让这些图像通过 DNN 网络，DNN 网络的每个单元都会产生高或低的激活值；二是以网络为中心，只需要一个训练好的 DNN，而不需要数据集中的任何数据（比如 GAN）。

以网络为中心的方法，典型代表是 ZFNet 的反卷积方法，论文 [6]、[9] 和 [10] 中（即 activation maximization），通过使特定单元产生高的激活值来合成图像（比如对于分类网络中，查看什么输入能使输出更接近于 $[1,0,0]$，什么输入能使输出更接近于 $[0,1,0]$ 之类的）。首先随机初始化网络的输入图像（图像作为要被训练的参数），然后求导 $\frac{\partial a_i(x)}{\partial x}$ 来进行优化（梯度上升），得到尽可能最大化激活输出的输入。这里的激活 $a_i(x)$ 可以是网络中间的任意一层，也可以是最后的输出层（如果是最后的输出层，就是用来做输出激活的最大化）。

2.1.1 论文 [9] 简介

论文 [9] 考虑了两种可视化技术，第一种生成了可以最大化图像分类的得分的图（activation maximization），由 Convnet 捕获。第二种，对于给定的图片和类，用于计算其显著性图 (Saliency Maps):



直接得到最大化激活输出的输入可能会很糟糕，这是由于能量（图像亮暗）会分布很不均匀，人们根本无法理解。因此，需要加一些正则化限制条件，例如 [9]（2013 年）中使用的 $L2$ 正则化：

$$arg\max_I S_c(I) - \lambda \|I\|_2^2 \tag{2.1.1}$$

2.1.2 论文 [10] 简介

论文 [10] 中《Understanding Neural Networks Through Deep Visualization》（2015 年）的论文中，设神经网络 CNN 的输入图像为 $\mathbf{x}$（三通道），它的激活为 $a_i(\mathbf{x})$，定义了一个参数化的正则化函数 $R_\theta(\mathbf{x})$。则目标 activation maximization 即：

$$x^* = arg\max_{\mathbf{x}} \Big(a_i(\mathbf{x}) - R_\theta(\mathbf{x}) \Big) \tag{2.1.2}$$

而在论文 [10] 里，稍微进行了改动，采用了一个正则化算子 $r_\theta(\cdot)$，将 $\mathbf{x}$ 映射到它自己的更规则的版本。

图像（被训练的参数）更新的方式为：

$$\mathbf{x} \longleftarrow r_\theta\Big(\mathbf{x} + \eta \frac{\partial a_i}{\partial \mathbf{x}} \Big) \tag{2.1.3}$$

这里的正则化算子有很多种，可以从论文中找到。

2.1.3 我们的任务目标

由于要想比较容易地访问网络中间的每一层（比如我们想看看什么样的图像可以最大化激活卷积网络的任意一层），则需要事先得到网络中的 nn.Sequential 序列，然后通过 [·] 下标来访问到任意一层，这样我们就可以尝试最大化激活神经网络的任意一层了。

我们为了方便，目标只是最大化输出，所以就不实现可视化中间任意一层的方法了。可以可视化任意一层的方法可参考 [12] 的 cnn_layer_visualization.py 文件，其实，有了本文的基础，要想实现最大化激活神经网络的任意一层其实非常简单。我把 [12] 中的该文件进行了整理（一共两个.py 文件），然后放到了本文源码的 github-utkuozbulak 目录下（一共两个文件），我们会在本章最后一节讲解一下。

- 我们首先需要搭建一个 CNN 网络，用来进行手写数字辨识（CNN.py）。
- 然后，建立一个测试文件，测试我们训练好的网络（CNN-Test.py）。

- 最后，建立一个 CNN 可视化文件，用于可视化（CNN-Vis.py）。

由于 CNN.py 和 CNN-Test.py 都是讲过的知识，所以不再赘述，大家可以自行阅读这两个文件的源码。

2.2 CNN 可视化程序

主函数的程序执行顺序是：

- 从参数备份中恢复网络参数。
- 定义我们想要最大化的输出结果。
- 训练网络。

代码如下：

```
if __name__ == '__main__':
  # 从参数备份中恢复网络参数
  model = CNN_Net().cuda()
  checkpoint = torch.load('./model100.pth')
  model.load_state_dict(checkpoint ['model'] )
  # 定义我们想要最大化的输出结果
  aim = torch.from_numpy(np.array([[1,0,0,0,0,0,0,0,0,0]]))
  # 训练网络
  layer_vis = CNNLayerVisualization(model, aim)
  layer_vis.visualise_layer()
```

aim 是一个 tensor，手写数字辨识是十分类任务，所以最后一个维度有 10 个数。注意由于对于神经网络输入来说存在 batch_size 这个维度，所以 aim 是二维张量。在这个 aim 中，我们希望能够最大化激活输出 0。

现在我们要实现 CNNLayerVisualization 类里面的功能。在此之前，我们先定义三个函数。

- 函数 1：由于网络训练的参数是一张图，所以要把 numpy 随机初始化的数组（图像数据）转换为 pytorch 的 Variable。
- 函数 2：把训练得到的图像转换为 numpy 的数组。
- 函数 3：将图像数据保存为文件。

这三个程序分别是：

```
def save_image(img, path)
def Img2Variable(img)
def Variable2Img(img_var)
```

然后定义网络类。网络初始化功能因为很简单，所以不再赘述。该类只有一个功能函数，即visualise_layer。由于程序文件做了大量注释，所以不再展开讲解，直接给出程序：

```
def visualise_layer(self):
    # 生成一个随机图像，这里是28X28像素，数值为150到180之间的随机数
    random_image = np.uint8(np.random.uniform(150, 180, (28, 28, 1)))
    # 把图像转换为float型可以用于训练的参数
    imgInput = Img2Variable(random_image)
    # 定义优化器
    optimizer = torch.optim.Adam([imgInput], lr=0.1, weight_decay=1e
        -6)
    # 开始迭代
    for i in range(1, 1001):
        ......
```

注意优化器的参数只有输入图像 imgInput，我们的目标也只是更新输入图像。

在迭代中，比较麻烦的是定义损失函数。这里我用的损失函数是：

```
        y = model(x)
        loss = (y.softmax(1) - aim.cuda()).abs().sum() + 0.003 * x.abs()
            .sum()  + 0.01 * y.abs().sum()
```

如果打印一下 y，就会看到 y 的值是一堆很大的数：

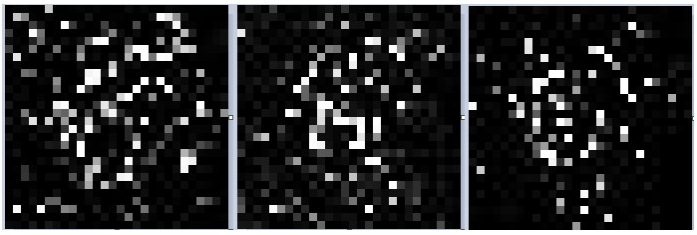
```
tensor([[  7.1685, -30.4596, -20.4656, -14.2554, -26.0373, -11.3446,
    -15.8255,
-13.3393,  -9.4966, -21.0657]], device='cuda:0', grad_fn=<
    AddmmBackward>)
```

通过 softmax 可以归一到 (0,1) 之间：

```
tensor([[1.0000e+00, 4.5532e-17, 9.9690e-13, 4.9626e-10, 3.7923e-15,
    9.1168e-09,
1.0324e-10, 1.2405e-09, 5.7865e-08, 5.4703e-13]], device='cuda:0'
    grad_fn=<SoftmaxBackward>)
```

然后与 aim 相减，取绝对值，再求和，作为损失值的第一项。第二项损失值是输入图像的正则化，即输入图像取绝对值，然后再求和。第三项损失值是尽量让 y 内的值的绝对值小一点，因为有时候训练 1000 轮以后，有的值会到 -100 以下，很低的值在经过 softmax 以后几乎为 0，所以没有必要让它再继续低下去了。

最终，得到的结果是（从左到右，分别是最大化激活 0，最大化激活 4，最大化激活 9 的输入图像）：



和课程 [1] 中展示的结果类似，对人来说很难去理解。

2.3 源码 [12] 的简单讲解

学完了上一节的内容，本节的内容就会很简单了。CNN 网络有很多层，此代码会任意选一层来查看什么样的输入可以最大化激活输出。

misc_functions.py 文件是一些功能函数，比如图像转化灰度、图像转化为 Variable、图像保存等功能。

CNN-visualization.py 文件中定义了 CNNLayerVisualization 类。由于我们希望可以选择最大化任意一层的网络的激活，所以 main 函数中定义了 cnn_layer，表示我们想要最大化激活的层。

网络用的是已经预训练好的 VGG16，其中，预训练的网络参数是 vgg16.features，这里的 features 是一个 nn.Sequential 序列，包含了 vgg16 的网络前半部分（卷积层和池化层）。而后面的线性层并没有赋给 CNNLayerVisualization 类，因为我们只想看如何最大化激活卷积层。

在网络中用到了 hook 函数，这是因为 Pytorch 会自动舍弃计算图的中间结果，所以想要获取这些数值就需要使用 hook 函数。

训练的损失函数就是要最大化激活的层的全部参数的平均值。

2.4 小结

本文开始于 2021 年 10 月 22 日，但是直到 2022 年 5 月 28 日才完成，这期间一直在看各类数学书和图形学的论文，对完成这个神经网络入门的系列倒是有些松懈。

我认为本文的意义很大，它对了解神经网络的结构和机理有一定的帮助，所以也算是必须要掌握的一部分内容了。至于 CNN 的可视化后续工作应该如何拓展，则可以参考一些比较新的论文。人们在试图理解神经网络的道路上一直没有停下脚步，所以也得到了一些有趣的结论。

Bibliography

[1] https://www.bilibili.com/video/BV1JE411g7XF?p=24

[2] Trecvid Q A , Related S . Visualizing and Understanding Convolutional Networks [104].

[3] Sundararajan M , Taly A , Yan Q . Gradients of Counterfactuals[J]. 2016.

[4] Shrikumar A , Greenside P , Shcherbina A , et al. Not Just a Black Box: Learning Important Features Through Propagating Activation Differences[J]. 2016.

[5] Axiomatic Attribution for Deep Networks

[6] Erhan D , Bengio Y , Courville A , et al. Visualizing Higher-Layer Features of a Deep Network. 2009.

[7] Ribeiro M T , Singh S , Guestrin C . "Why Should I Trust You?": Explaining the Predictions of Any Classifier[C]// Proceedings of the 2016 Conference of the North American Chapter of the Association for Computational Linguistics: Demonstrations. 2016.

[8] Wu M , Hughes M C , Parbhoo S , et al. Beyond Sparsity: Tree Regularization of Deep Models for Interpretability[J]. arXiv, 2017.

[9] Simonyan K , Vedaldi A , Zisserman A . Deep Inside Convolutional Networks: Visualising Image Classification Models and Saliency Maps[J]. Computer ence, 2013.

[10] Yosinski J , Clune J , Nguyen A , et al. Understanding Neural Networks Through Deep Visualization[J]. Computer Science, 2015.

[11] https://github.com/utkuozbulak/pytorch-cnn-visualizations

[12] https://github.com/utkuozbulak/pytorch-cnn-visualizations/tree/master/src