# Explainable AI 实验部分

## Task1 - Saliency Map

这部分的主要思想是：修改图片的某一个像素，观察这样会对最后的loss产生多大的影响，绘制图片的显著图，分析之前训练好的模型，分析它认为哪些部分(像素)是重要的.

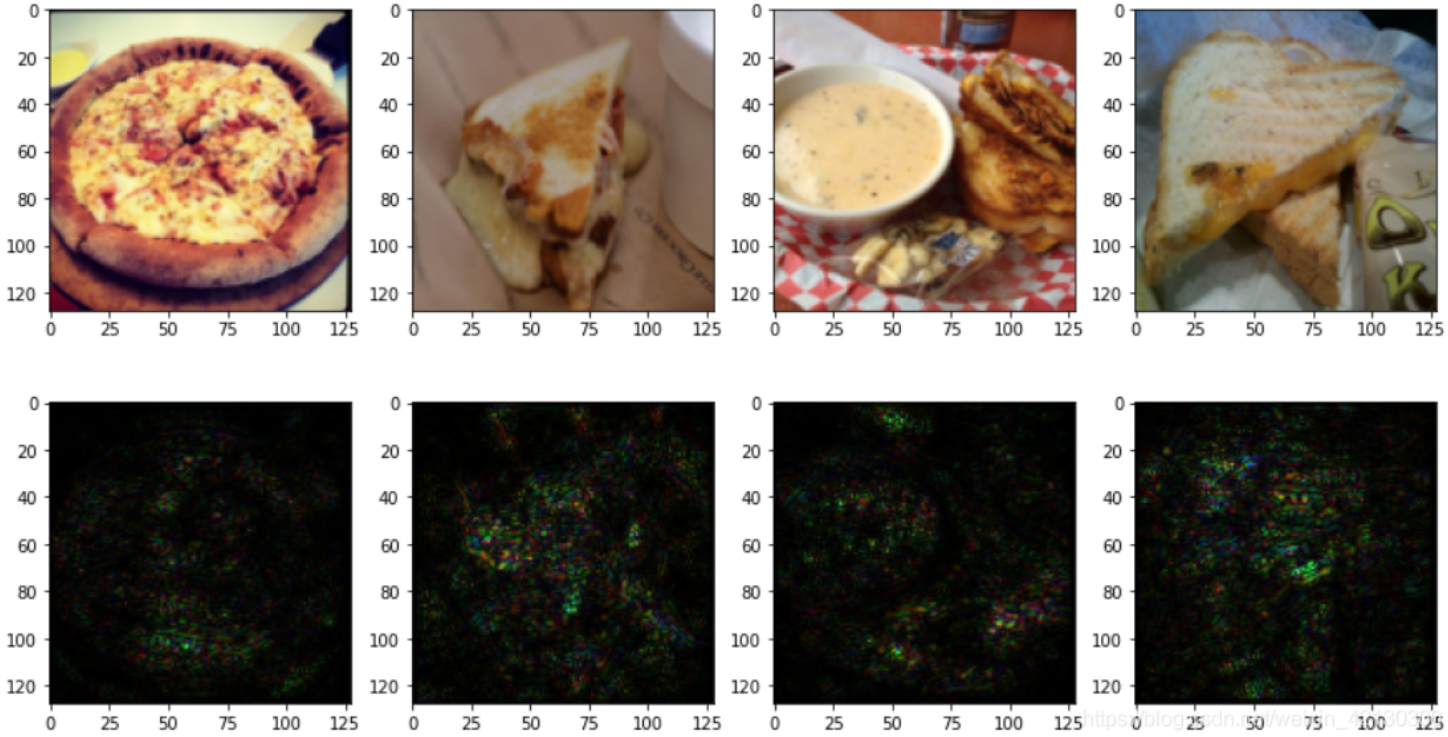
loss的计算和输入的图片(image)、图片进行多层卷积后在神经网络上的输入层参数(model parameter)以及最后预测的标签(labels)。在过去的CNN模型中，我们主要是针对model parameter来计算梯度，但是从数学上讲，一张图片也应该是一个连续的tensor，所以也可以对image去求偏微分值，我们习惯上将偏微分值的大小看作这个像素(pixel)的重要性。在同一张图片中，loss对pixel的值画出来，可以看出模型的判断依据。

```
def normalize(image):
  return (image - image.min()) / (image.max() - image.min())

def compute_saliency_maps(x, y, model):
  model.eval()
  x = x.cuda()
  x.requires_grad_()

  y_pred = model(x)
  loss_func = torch.nn.CrossEntropyLoss()
  loss = loss_func(y_pred, y.cuda())
  loss.backward()

  saliencies = x.grad.abs().detach().cpu()
  saliencies = torch.stack([normalize(item) for item in saliencies])
  return saliencies
```

上面的两个函数，$normalize()$是对图片进行归一化，$compute_saliency_maps()$是来计算$image$的偏微分。

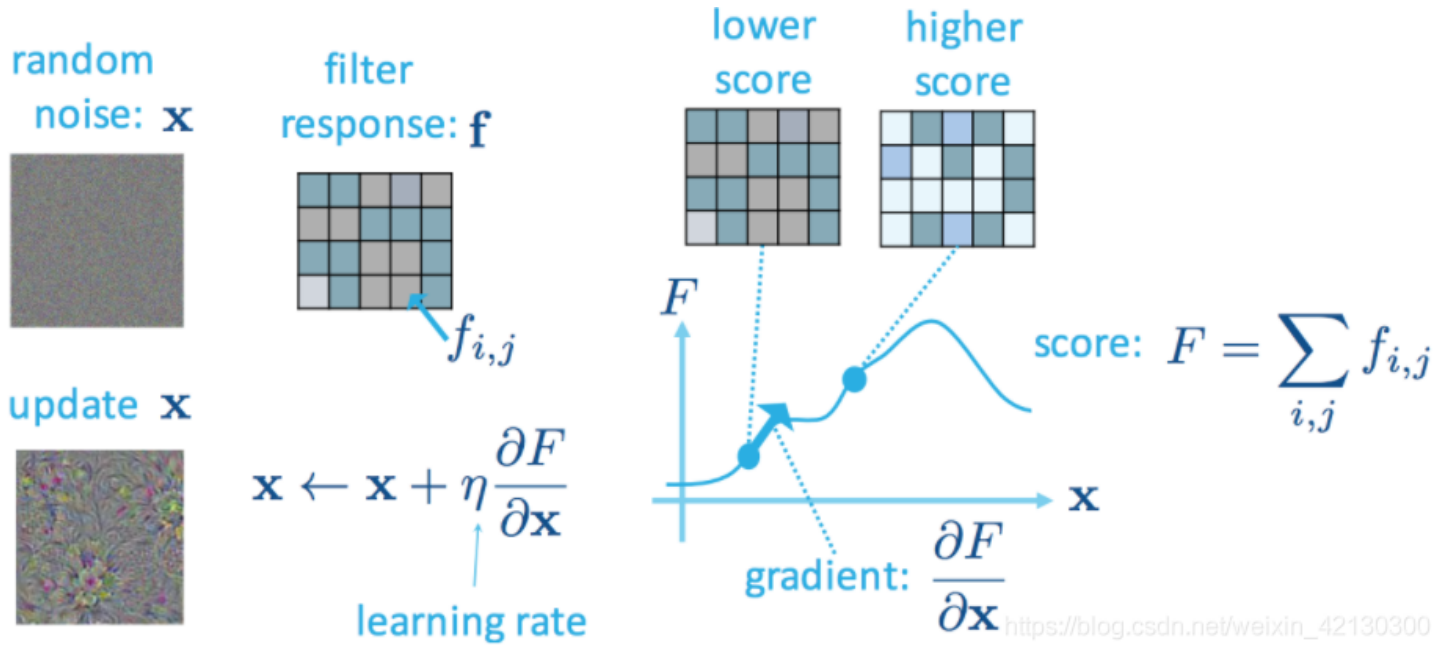
```
# 指定想要一起 visualize 的圖片 indices
img_indices = [1, 2, 47, 98]
images, labels = train_set.getbatch(img_indices)
saliencies = compute_saliency_maps(images, labels, model)

fig, axs = plt.subplots(2, len(img_indices), figsize=(15, 8))
for row, target in enumerate([images, saliencies]):
  for column, img in enumerate(target):
    axs[row][column].imshow(img.permute(1, 2, 0).numpy())
plt.show()
plt.close()
```

在这个示例中，我们选取了编号为1，2，47，98的图片，最后得到的结果如下：



# Task2 - Filter Visualization

选取cnn中某一层layer中的一个filter,如果想知道这个filter具体在做什么工作，可以用梯度上升的方法(Gradient Acent).



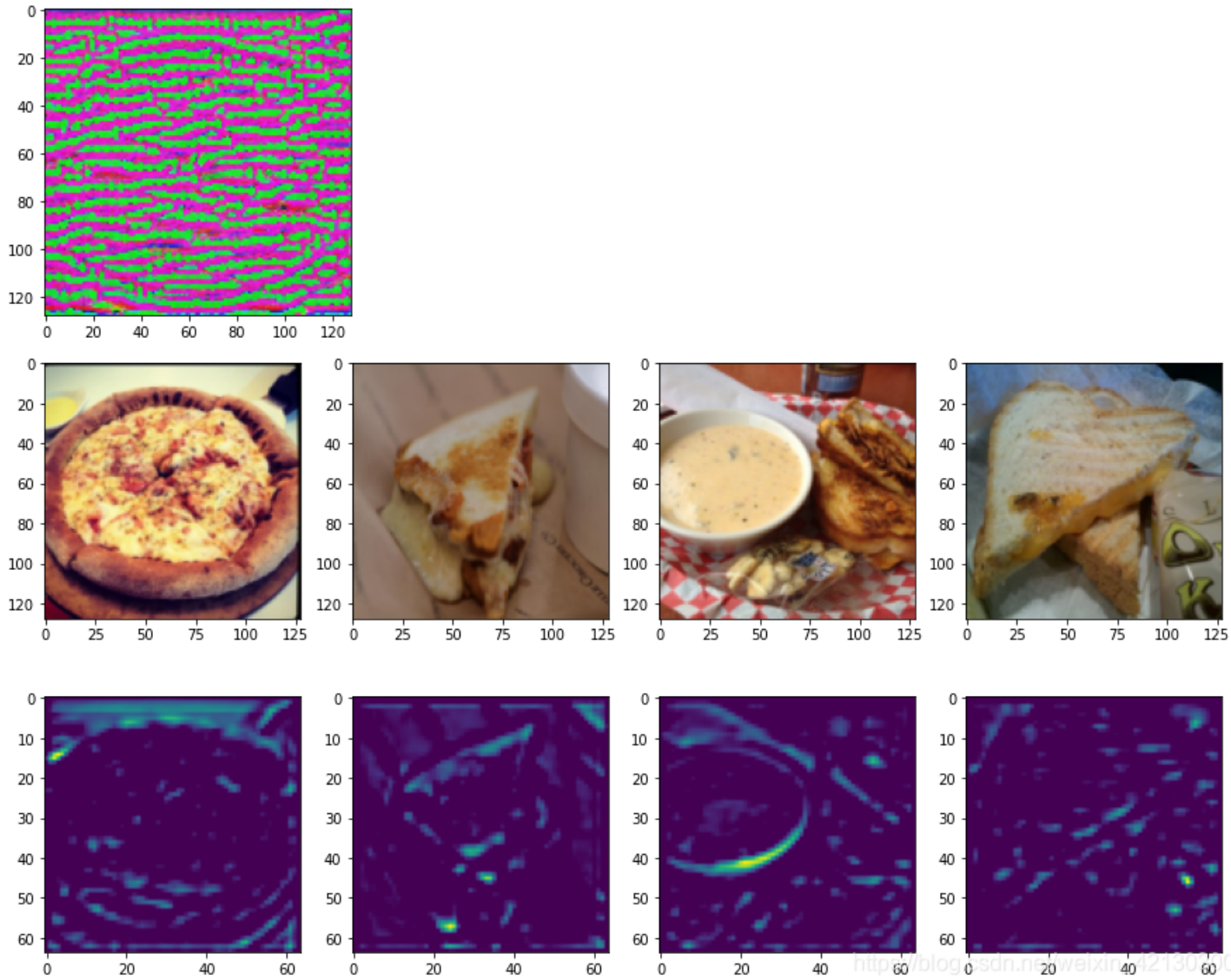

$X$是假设是我们原来的图片，而$f$是cnn中某一层的filter，设计函数$F = \sum_{i,j} f_{i,j}$,这个函数代表着这个filter识别的特征数值和，如果$F$越大，就代表着它学习到的特征越加明显。

```
def normalize(image):
  return (image - image.min()) / (image.max() - image.min())

layer_activations = None
def filter_explaination(x, model, cnnid, filterid, iteration=100, lr=1):
  model.eval()

  def hook(model, input, output):
    global layer_activations
    layer_activations = output

  hook_handle = model.cnn[cnnid].register_forward_hook(hook)
  model(x.cuda())
  filter_activations = layer_activations[:, filterid, :, :].detach().cpu()
  x = x.cuda()
  x.requires_grad_()
  optimizer = Adam([x], lr=lr)
  for iter in range(iteration):
    optimizer.zero_grad()
    model(x)

    objective = -layer_activations[:, filterid, :, :].sum()
    objective.backward()
    optimizer.step()
  filter_visualization = x.detach().cpu().squeeze()[0]

  hook_handle.remove()
  # 很重要: 一旦對 model register hook, 該 hook 就一直存在。如果之後繼續 register 更多 hook
  return filter_activations, filter_visualization
```

在上面的代码中，$normalize()$是图像归一化函数，$filter_explaination()$函数用来绘制激活图像，这里选取cnn网络中第15层的第1个$filter$,在计算loss过程中，将中间结果用$hook()$函数将保存下来。

```
# 选取的图像下标
img_indices = [1, 2, 47, 98]
images, labels = train_set.getbatch(img_indices)
filter_activations, filter_visualization = filter_explaination(images, model, cnnid=15, filteric

# 画出 filter visualization
plt.imshow(normalize(filter_visualization.permute(1, 2, 0)))
plt.show()
plt.close()

# 检查出 filter activations
fig, axs = plt.subplots(2, len(img_indices), figsize=(15, 8))
for i, img in enumerate(images):
  axs[0][i].imshow(img.permute(1, 2, 0))
for i, img in enumerate(filter_activations):
  axs[1][i].imshow(normalize(img))
plt.show()
plt.close()
```

得到的4张图片的效果如下：



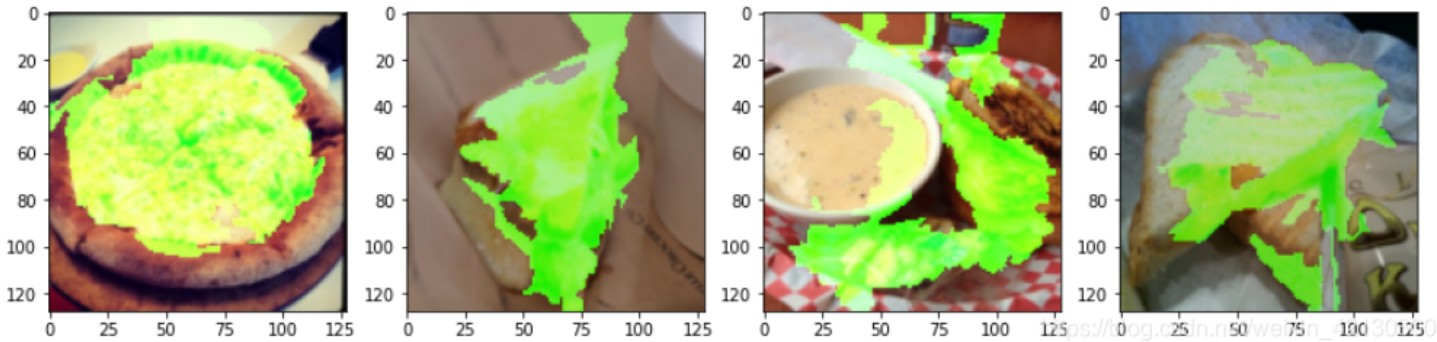
从这几张图片可以看出，当前选定的filter确实识别出了类似边界的东西。

# Task-3 Lime

如果只是修改一张图片中的一个像素，一般不会对模型最终的判断造成太大的影响，所以换一个思路，当我们判断一个图片是否是某一类的时候，往往看到图片中的一些关键的部分时，就可以做出判断，所以，我们可以将图片分成若干块，然后用一种线性的模型去拟合，分析对应模块得到的系数，从而判断这个是否重要。

```
def predict(input):
    # input: numpy array, (batches, height, width, channels)
    model.eval()
    input = torch.FloatTensor(input).permute(0, 3, 1, 2)
    # 需要先將 input 轉成 pytorch tensor, 且符合 pytorch 習慣的 dimension 定義
    # 也就是 (batches, channels, height, width)

    output = model(input.cuda())
    return output.detach().cpu().numpy()

def segmentation(input):
    # 利用 skimage 提供的 segmentation 將圖片分成 100 塊
    return slic(input, n_segments=100, compactness=1, sigma=1)

img_indices = [1, 2, 47, 98]
images, labels = train_set.getbatch(img_indices)
fig, axs = plt.subplots(1, 4, figsize=(15, 8))
np.random.seed(16)
# 讓實驗 reproducible
for idx, (image, label) in enumerate(zip(images.permute(0, 2, 3, 1).numpy(), labels)):
    x = image.astype(np.double)
    # lime 這個套件要吃 numpy array

    explainer = lime_image.LimeImageExplainer()
    explaination = explainer.explain_instance(image=x, classifier_fn=predict, segmentation_fn=se

    lime_img, mask = explaination.get_image_and_mask(
                                positive_only=False,
                                hide_rest=False,
                                num_features=11,
                                min_weight=0.05
                            )
    # 把 explainer 解释结果转换为图片
    # doc: https://lime-ml.readthedocs.io/en/latest/lime.html?highlight=get_image_and_mask#lime.

    axs[idx].imshow(lime_img)

plt.show()
plt.close()
```

python中有现成的工具可以满足我们的需求，我们依然选定之前的四张照片，得到的结果如下：



可以发现，图片中被覆盖的区域就是模型认为比较重要的部分，也给出了模型具体根据什么做出了判断。