Aalto University

CS-E4890

Deep Learning

Visual Explanation for Deep Learning Final Report

Group 22:
Student No: 661562

May 20, 2019

Aalto University
School of Science

Abstract

Imagenet classification with deep convolutional neural networks [3] was the first paper that sparked huge interest in deep convolutional neural networks. After the success of Alexnet, researchers started to apply deep learning to different areas and gained a huge increase in performance. But, deep learning models are black box models which are difficult to understand. As they gain more importance day by day, the need to understand the working of deep learning model grows. In this project, we will discuss a few techniques that offer to explain the working of convolutional neural networks.

1 Introduction

Deep learning models have been applied to several tasks such as computer vision, speech recognition and reinforcement learning due to their excellent performance. Andre et. al. proposes a deep convolutional neural network (CNN) for classifying skin lesion images into cancerous or benign [2]. The CNN model performs on par with tested human experts. Similarly, Wang et. al. presented a deep learning model that performed significantly well, reducing the human error rate by 85%, for identifying metastatic breast cancer [8]. However, the application of deep learning to critical tasks such as medical diagnosis or self-driving car poses a new kind of challenge.

As deep learning models create their own internal representation from internal features for solving the task at hand. the model predicts based on its internal representation which we do not fully understand. For example, in the breast cancer detection task, how can the doctor trust the prediction of a model without understanding the reason behind the model's prediction? Let's take another example of a self-driving car. In case the self-driving car meets with an accident, should we hold the model or the human accountable for the accident? In order to confirm that the model was not responsible for the accident, we need to understand the working of the model. Hence, understanding the working of deep learning models is important.

The importance of trustworthiness in deep learning models has led to increased research efforts for the interpretation of deep learning. Interpretation of deep learning model is a huge research area and it is not possible to cover all aspects. In this project, we will compare different methods that aim to explain the prediction of CNN. The rest of the project is organized as follows. Section 2 explains the different methods used in the project followed by their comparison in Section 3. Finally, conclusions are drawn in Section 4.

2 Literature Survey

Specifically in this project, we will discuss methods that explain the classification decision of our deep learning model by finding regions in image that influenced the decision of the model. Such regions are also known as saliency maps.

2.1 Guided Backpropagation

The idea behind this technique is that how change in the pixel of input image can affect the output of the network. We visualize the gradient of a particular neuron with respect to a given

input image. However, using simple propagation can result in a very noisy visualization because different parts of the image activate the neuron differently.

The principle behind guided backpropagation is that we are only interested in the regions which activate the neuron positively so while backpropogation we make the negative gradients zero [7]. This results in clearer detailed visualization of the region responsible for prediction. However, the problem with guided backpropogation is that it is not class discriminative. So, if we have two target classes in the same image, we can get the same visualizations for both the classes.

2.2 Class Activated Mapping

Bolei et. al. introduced class activated mapping(CAM) to overcome the problem of guided backpropogation [9]. In deep convolutional networks, as we progress deeper the layers represents more complex features. For example, the first layer usually represents edges and contours. In fact, the features represented by the last layer loosely serve the purpose of object localization which can be used to identify the regions responsible for a given prediction. However, this important representation of last layer is lost due to the fully connected layer in the end.

The principle behind class activated mapping is to directly compute the importance of features map in the last convolutional layer for each class to obtain a class discriminative visualization. This requires us to replace the last fully connected layer by softmax layer. The softmax layer is fed the global average pooling of the last convolution layer which gives us the probability of each class in each feature map. The probability of target class in each feature map is then summed to obtain the class activated mapping. However, this method cannot be applied to all deep convolutional neural networks.

2.3 Grad-CAM

Ramprasaath et. al. proposed Grad-CAM which is a generalised version of CAM that requires no architectural modification in the network [6]. Grad-CAM directly infers the importance of each pixel for a target class for each feature map in the last convolutional layer. The weight for each pixel of each feature map is calculated from the gradient for the particular class. The weight of each pixel is multiplied with the corresponding feature map and then summed across all feature maps. This gives us a heatmap which is upsampled to the size of input image to obtain the important regions for predictions.

In Figure 1, we can see an image containing both cat and dog on the left side. If we use guided backpropogation considering the 'cat' as target class or the 'dog' as target class, we get the same visualization. So, guided backpropogation as mentioned above is not class discriminative. In Grad-CAM, we can see the heat map visualized the different region responsible for prediction for both the target classes. However, the heat map obscures the fine details of the region. So, researchers have proposed using Guided Grad-CAM which is class discriminative as well as highlights the fine details of the input image responsible for prediction.

3 Experiments and Results

This project is a theoretical project so as such we don't have a dataset. In order to test the different methods, we use pre-trained Alexnet model [3] from pytorch. The images for which visualization is presented is taken from Imagenet [1]. Also, there are no way to compare these visualizations apart from subjective analysis.

We have implemented guided backpropogation and compared the visualization result with those from normal backpropogation. The implementations has been understood and adapted

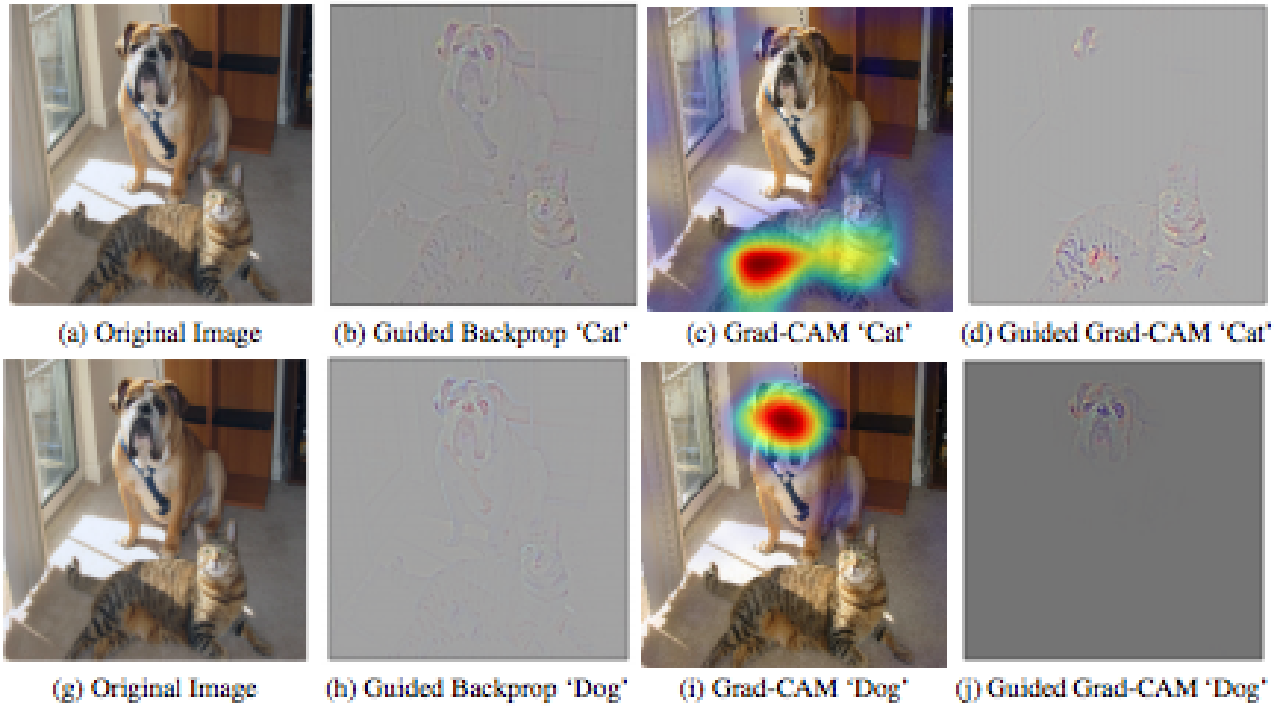

Figure 1: This image has been taken from the Grad-CAM paper [6] and illustrates the differences between the methods.

for this project with explanation [5]. Below, we will briefly explain the implementation for each method.For guided backpropogation, we need the gradients of the first layer for visualization. In pytorch, we can use hooks to inspect the gradients of the layer. Forward hook is executed during a forward call and backward hook is executed during backpropogation. We need to modify the backpropogation of the pre-trained Alexnet model such that we suppress any negative gradients. So, we attach a forward hook at the first layer to get the gradients for visualization and we attach a backward hook at all ReLU layers where we suppress the negative gradient to zero and return the modified gradient to next layer for backpropogation.

The result shown in Figure 2, Figure 3 and Figure 4 are generated from our code. We can see that visualization generated from normal backpropogation results in noisy image. After suppressing the negative gradients in guided backpropogation, we can see the fine details in region responsible for prediciton.

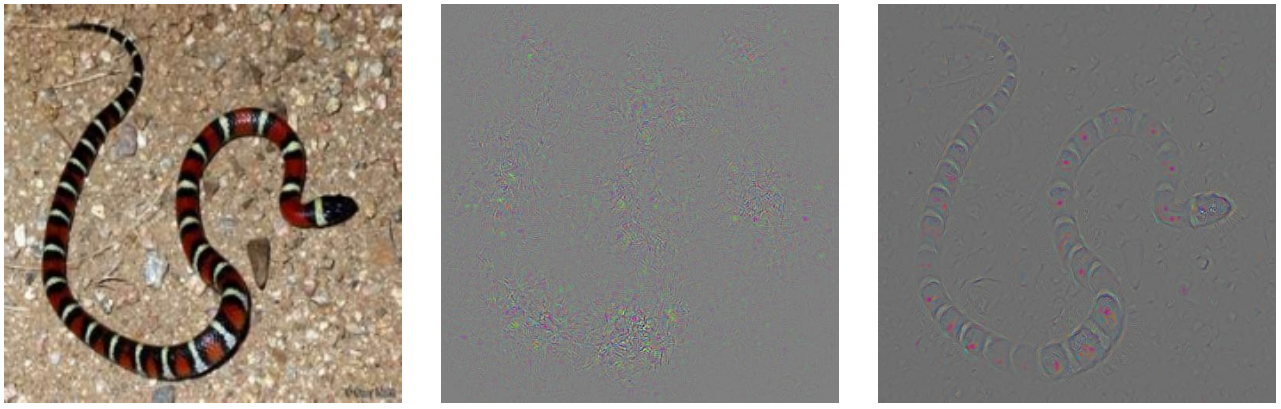

Figure 2: The input image on the left. The middle figure shows results from simple backpropogation whereas the right figure shows Guided Backpropogation.

Figure 3: The input image on the left. The middle figure shows results from simple backpropogation whereas the right figure shows Guided Backpropogation.

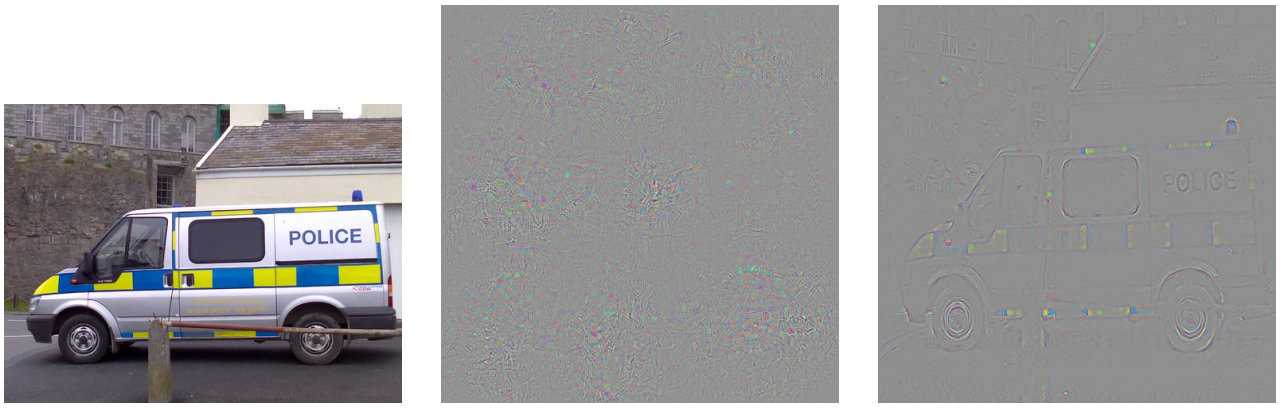

Figure 4: The input image on the left. The middle figure shows results from simple backpropogation whereas the right figure shows Guided Backpropogation.

4 Conclusion

Each of these methods have their own advantages and disadvantages. Guided backpropogation can highlight the subtle features responsible for the prediction but at the same time it is not class discriminative. Class activated mapping overcomes the short comings of guided backpropogation by utilizing the importance of feature map for each class. However, class activated mapping requires us to make change in the last fully connected layer to obtain the visualization.

Hence, Grad CAM was introduced which utilizes the final convolutional layer to generate the heatmap for visualization without making any architectural changes. However, in Grad-CAM the heatmap obscures the details of the region responsible for prediction. So, researchers have proposed using Guided Grad-CAM which is class discriminative and also visualizes the subtle details. These techniques are the first step towards inducing trustworthiness in the prediction of deep convolutional neural networks.

The non linear activation function such as ReLU, which is non-differentiable at zero, used in deep convolutional neural networks can have discontinuous gradients which can lead to problem for techniques using backpropogation. Deep convolutional neural networks, unlike humans, can also be easily fooled into predicting wrong target class for an input image by adding some noise [4]. With the introduction of General Data Protection Regulation (GDPR), any decision made by machine needs to be explained under the law of explanation otherwise the decision needs to be evaluated by a human. Hence, understanding the prediction of deep learning models becomes important day by day and we still have a long way to go.

References

[1] Deng, J., Dong, W., Socher, R., Li, L.-J., Li, K., and Fei-Fei, L. Imagenet: A large-scale hierarchical image database. In *2009 IEEE conference on computer vision and pattern recognition* (2009), Ieee, pp. 248–255.

[2] Esteva, A., Kuprel, B., Novoa, R. A., Ko, J., Swetter, S. M., Blau, H. M., and Thrun, S. Dermatologist-level classification of skin cancer with deep neural networks. *Nature 542*, 7639 (2017), 115.

[3] Krizhevsky, A., Sutskever, I., and Hinton, G. E. Imagenet classification with deep convolutional neural networks. In *Advances in neural information processing systems* (2012), pp. 1097–1105.

[4] Nguyen, A., Yosinski, J., and Clune, J. Deep neural networks are easily fooled: High confidence predictions for unrecognizable images. In *Proceedings of the IEEE conference on computer vision and pattern recognition* (2015), pp. 427–436.

[5] Ozbulak, U. Pytorch implementation of convolutional neural network visualization techniques. Github, Oct 2017.

[6] Selvaraju, R. R., Cogswell, M., Das, A., Vedantam, R., Parikh, D., and Batra, D. Grad-cam: Visual explanations from deep networks via gradient-based localization. In *Proceedings of the IEEE International Conference on Computer Vision* (2017), pp. 618–626.

[7] Springenberg, J. T., Dosovitskiy, A., Brox, T., and Riedmiller, M. Striving for simplicity: The all convolutional net. *arXiv preprint arXiv:1412.6806* (2014).

[8] Wang, D., Khosla, A., Gargeya, R., Irshad, H., and Beck, A. H. Deep learning for identifying metastatic breast cancer. *arXiv preprint arXiv:1606.05718* (2016).

[9] Zhou, B., Khosla, A., Lapedriza, A., Oliva, A., and Torralba, A. Learning deep features for discriminative localization. In *Proceedings of the IEEE conference on computer vision and pattern recognition* (2016), pp. 2921–2929.

Code

Guided Backpropogation

We have provided the image and jupyter notebook in form of zip file uploaded on Google drive at `https://drive.google.com/file/d/1SYNnnxd15R4NfF2HHVBHF_r9tlexqD3M/view?usp=sharing` . Please install the following libraries.

1. pip install torch
2. pip install torchvision
3. pip install Pillow (Python imaging library)

The code has been understood and adapted for the project as we were facing problems on implementing from scratch [5].

```
#!/usr/bin/env python
# coding: utf-8

# In[1]:


import torch
from torch.nn import ReLU
from torchvision import models
from PIL import Image as im
from torch.autograd import Variable
import torchvision.transforms as transforms
import numpy as np
import os


# #### Image Preprocessing

# In[21]:


def preprocess(input_image):
    # Resizes, converts to tensor and normalizes
    preprocessor = transforms.Compose([transforms.Resize((512,512)),
                       transforms.ToTensor(),
                       transforms.Normalize(mean=[0.485, 0.456, 0.406],
                       std=[0.229, 0.224, 0.225]) ])

    # We need to add one more channel in front to use with Alexnet
    new_img = preprocessor(input_image).unsqueeze_(0)

    # We need the gradients for input image
    new_img_var = Variable(new_img, requires_grad=True)
    return new_img_var

def visualize_grad(grad, filename):
    # Normalize the gradients before storing
    grad = grad - grad.min()
    grad /= grad.max()

    # Join the filename and store directory
    filepath = os.path.join('results', filename + '.jpg')

    # Convert the image to numpy matrix
    im_matrix = grad.transpose(1,2,0)
    im_matrix = (im_matrix*255).astype(np.uint8)

    # Store the image using PIL library
    final_image = im.fromarray(im_matrix)
    final_image.save(filepath)


# Images from ImageNet with target class

# In[22]:


# Image 1 (Snake) target class = 'Snake'
img1_path = 'images/racer_car.jpg'
```

```
target_class_1 = 751
filename = 'guided_backprop_' + 'racer_car'


'''
snake - 56 (target_class)
cat_dog - 243 (target_class for dog)
racer_car - 751 (target class)
police_car - 734 (target class)
'''


# We use image 1 by default

# In[23]:


input_image = im.open(img1_path)
#input_image.show()

# From pytorch for pretrained models
'''
All pre-trained models expect input images normalized in the same way,
i.e. mini-batches of 3-channel RGB images of shape (3 x H x W), where
H and W are expected to be at least 224.
'''

processed_input_image = preprocess(input_image)


# #### Guided Backpropogation

# In[24]:


class GuidedBackpropogation():

    def __init__(self, model):
        self.model = model

        # to store the final gradients
        self.gradients = None

        # to store the output of each ReLU layer in forward pass
        self.relu_outputs = []

        # set the model in evaluation mode
        self.model.eval()

        # store relu outputs during forward pass and
        # update the backward pass of ReLU layer to return suppressed gradients
        self.modify_relu()

        # save the gradients of 1st layer after backprop
        self.save_grads()

    def save_grads(self):
        # self.model.features._modules.items() returns dict of (no,layer)
        # 0 is to select first item of list i.e. 1st layer
        # 1 is to select the layer itself from tuple
```

```
        layer1 = list(self.model.features._modules.items())[0][1]

        def save_func(mod, grad_in, grad_out):
            '''
            grad_in = gradient of model output w.r.t layer output

            # For clarification
            grad_out = gradient of next layer (for backprop it will be reverse order)
                     = grad_in * (gradient of layer output w.r.t layer input)
                     = (model_output/layer_output) * (layer_output/layer_input)
                     = model_output/layer_input
            '''
            self.gradients = grad_in[0]

        # We add the hook to store grad of 1st layer after backprop
        # We later retrive them for visualization
        layer1.register_backward_hook(save_func)

    def modify_relu(self):

        def suppress_grads(mod, grad_in, grad_out):
            '''
            function to be attached to relu to suppress negative gradients during backprop

            # other params already defined above
            '''

            # Remember we are going backwards from last to first
            relu_out_during_forward_pass = self.relu_outputs[-1]

            # We make all positive activations 1
            relu_out_during_forward_pass[relu_out_during_forward_pass > 0] = 1

            # All negative activations will be made 0
            suppressed_grad = relu_out_during_forward_pass * torch.clamp(grad_in[0], min=0.0)

            # Now we delete the last relu output during forward pass as it is not required
            del self.relu_outputs[-1]

            # gradients are in tuple form
            return (suppressed_grad,)

        def store_forward(mod, input_tensor, output_tensor):
            self.relu_outputs.append(output_tensor)

        # Now attach both the forward hook and backward to all relu layers in Alexnet
        # by iterating through dict
        for _, module in self.model.features._modules.items():
            # check if the module is ReLU
            if isinstance(module, ReLU):
                # For forward pass
                module.register_forward_hook(store_forward)

                # For backward pass
                module.register_backward_hook(suppress_grads)

    def process(self, image, target_class):
        # Do forward pass
        output = self.model(image)
```

```
        # To make sure gradients are 0,
        # pytorch can accumulate them while training
        # our model is in evaluation mode but still before backprop (good practice)
        self.model.zero_grad()

        # Target vector should be one hot encoding
        # Total 1000 classes for Imagenet, we set the target class as 1
        target = torch.zeros(1,1000).float()
        target[0][target_class] = 1

        # Do backward pass
        output.backward(gradient=target)

        first_layer_grad = self.gradients.data.numpy()[0]
        return first_layer_grad




# #### Visualization

# In[25]:


pretrained_model = models.alexnet(pretrained=True)

# Create a guided backpropogation model using pretrained model
Guided_backprop_model = GuidedBackpropogation(pretrained_model)


# Change target_class variable if using image 2
# Runs forward pass followed by backward pass with supressed negative gradients
visualization_gradients = Guided_backprop_model.process(processed_input_image, target_class_1)

# Save visualization
#save_gradient_images(visualization_gradients, filename)
visualize_grad(visualization_gradients, filename)


# In[ ]:
```

gbb.py

Normal Backpropogation

```
#!/usr/bin/env python
# coding: utf-8

# In[8]:


import torch
from torch.nn import ReLU
from torchvision import models
from PIL import Image as im
from torch.autograd import Variable
import torchvision.transforms as transforms
import numpy as np
```

```
import os
filename = 'backprop'


# #### Image Preprocessing

# In[9]:


def preprocess(input_image):
    # Resizes, converts to tensor and normalizes
    preprocessor = transforms.Compose([transforms.Resize((512,512)),
                      transforms.ToTensor(),
                      transforms.Normalize(mean=[0.485, 0.456, 0.406],
                      std=[0.229, 0.224, 0.225]) ])

    # We need to add one more channel in front to use with Alexnet
    new_img = preprocessor(input_image).unsqueeze_(0)

    # We need the gradients for input image
    new_img_var = Variable(new_img, requires_grad=True)
    return new_img_var

def visualize_grad(grad, filename):
    # Normalize the gradients before storing
    grad = grad - grad.min()
    grad /= grad.max()

    # Join the filename and store directory
    filepath = os.path.join('results', filename + '.jpg')

    # Convert the image to numpy matrix
    im_matrix = grad.transpose(1,2,0)
    im_matrix = (im_matrix*255).astype(np.uint8)

    # Store the image using PIL library
    final_image = im.fromarray(im_matrix)
    final_image.save(filepath)


# Images from ImageNet with target class

# In[22]:


# Image 1 (Snake) target class = 'Snake'
img1_path = 'images/police_car.jpg'
target_class_1 = 734
filename = 'backprop_' + 'police_car'


'''
snake - 56 (target_class)
cat_dog - 243 (target_class for dog)
racer_car - 751 (target class)
police_car - 734 (target class)
'''


# We use image 1 by default
```

```
# In[23]:


input_image = im.open(img1_path)
#input_image.show()

# From pytorch for pretrained models
'''
All pre-trained models expect input images normalized in the same way,
i.e. mini-batches of 3-channel RGB images of shape (3 x H x W), where
H and W are expected to be at least 224.
'''

processed_input_image = preprocess(input_image)


# #### Guided Backpropogation

# In[24]:


class Backpropogation():

    def __init__(self, model):
        self.model = model

        # to store the final gradients
        self.gradients = None

        # to store the output of each ReLU layer in forward pass
        self.relu_outputs = []

        # set the model in evaluation mode
        self.model.eval()

        # save the gradients of 1st layer after backprop
        self.save_grads()

    def save_grads(self):
        # self.model.features._modules.items() returns dict of (no,layer)
        # 0 is to select first item of list i.e. 1st layer
        # 1 is to select the layer itself from tuple
        layer1 = list(self.model.features._modules.items())[0][1]

        def save_func(mod, grad_in, grad_out):
            '''
            grad_in = gradient of model output w.r.t layer output

            # For clarification
            grad_out = gradient of next layer (for backprop it will be reverse order)
                     = grad_in * (gradient of layer output w.r.t layer input)
                     = (model_output/layer_output) * (layer_output/layer_input)
                     = model_output/layer_input
            '''
            self.gradients = grad_in[0]

        # We add the hook to store grad of 1st layer after backprop
        # We later retrive them for visualization
        layer1.register_backward_hook(save_func)
```

```
    def process(self, image, target_class):
        # Do forward pass
        output = self.model(image)

        # To make sure gradients are 0,
        # pytorch can accumulate them while training
        # our model is in evaluation mode but still before backprop (good practice)
        self.model.zero_grad()

        # Target vector should be one hot encoding
        # Total 1000 classes for Imagenet, we set the target class as 1
        target = torch.zeros(1,1000).float()
        target[0][target_class] = 1

        # Do backward pass
        output.backward(gradient=target)

        first_layer_grad = self.gradients.data.numpy()[0]
        return first_layer_grad




# #### Visualization

# In[25]:


pretrained_model = models.alexnet(pretrained=True)

# Create a backpropogation model using pretrained model
backprop_model = Backpropogation(pretrained_model)


# Change target_class variable if using image 2
visualization_gradients = backprop_model.process(processed_input_image, target_class_1)

# Save visualization
#save_gradient_images(visualization_gradients, filename)
visualize_grad(visualization_gradients, filename)


# In[ ]:
```

bb.py