Deep Learning Homework Part 1

Question 1: Based on the description for the dataset, could you describe what seems to be the machine learning problem we will be studying? More specifically,

- Is it a supervised or unsupervised task?
- Is it a regression, classification, clustering, or association problem?
- What does the training dataset consist of, in terms of inputs and outputs?
- How many input and output features seem to be present here?
- Any additional information about the task? Feel free to explore the dataset yourself, by using additional functions.

Answer 1: Given the information of the dataset, and the task at hand- we are looking at a supervised task considering we have a data that has been labelled to start with. We also see that all the inputs have a class that they are labelled with meaning we are looking at a classification problem. Since we will be working with images we can also say that it is a computer vision problem. The training dataset consists of a subset (1/6th) of the popular CIFAR-10 dataset which has coloured images so we can also infer that we will initially have 3 channels we could be working with . The ouputs intuitively are obviously going to be one of the 10 classes listed i.e. (airplanes, cars, birds, cats, deer, dogs, frogs, horses, ships, or trucks). In terms of the exact number of input features we could be working with a total of 32*32*3 input features if we choose to use all 3 channels of colour as is making it 3072 input features. This is of course not a one and true answer as the number of input features are dependent on what preprocessing things we do with the dataset and whether we use images in the greyscale or make use of the RGB channels present. Output features are simply the number of clases we are dealing with which is going to make it 10 output features, each possibly representing the proability of the input belonging to a particular class.

Question 2: Based on the display for the model below, what seems to be the architecture for the model? What layers have been implemented?

Answer 2: When we display the model given, we can tell that the architecture of the model seems to be that of a convolutional neural network that is similar to the resnet-18 model and is CNN essentially being used for a classification task. As for the layers, the model first seems to have a standard convolutional layer that is also followed by a batchnormalization layer for better feature attraction which is then followed by a ReLU activation layer. It is then followed by 2 residual blocks for essentially learning more about the features of the input. After this, there's a layer named `layer2` that is doing convolution as we know it for the first time by reducing the spatial dimensions of the input images, and making the output channels to 32. This is then followed by another layer further reducing the spatial dimensions and now using 64 kernels to increase the number of output channels to 64. We then use an average pooling layer with a kernel of size 8x8 and stride length also 8, which is in the end finally connected to a linear layer transofrming the features extracted by each of those layers together into 10 outputs that are eventually going to help drive the probabilities for each class.

Question 3: Write a function iufsgm_attack(), which performs an **untargeted iterated FSGM attack**.

Answer 3:

```
def iufsgm_attack(image, epsilon, model, original_label, iter_num = 10):
    # Skip if epsilon is 0
    if epsilon == 0:
        return image
    elif epsilon==None:
        return image
    else:
        og_label_torch= original_label.type(torch.LongTensor)

        eps_image= image
        for i in range(iter_num):
            image.grad= None
            out= model(image)

            pred_loss= F.nll_loss(out, og_label_torch)

            pred_loss.backward(retain_graph= True)

            eps_image= eps_image+epsilon*torch.sign(image.grad.data)

            out_eps_image= model(eps_image)
            prediction= out_eps_image.max(1, keepdim=True)[1]

            eps_image.retain_grad()
            eps_image= torch.clamp(eps_image, 0, 1)

            if prediction!=original_label:
                # print(f"Attack succesful after iteration: {i}")
                return eps_image

        return eps_image
```

Question 4: Can you suggest some epsilon values to put in the *epsilons* and *epsilons2* list below? Explain your choice of values briefly.

Answer 4: For the epsilon values to be used in the lists, for starters key thing to note is that for the `epsilons` list it is more crucial and experimentative choice because we are essentially trying to get an epsilon value that is big enough to create perturbation after just one go at the image. This might lead us to believe that using a big epsilon value like perhaps 0.1 might be the idea here but putting 0.1 after testing I realized that the images in the display after definitely seemed to be tampered with. Therefore, we will be going with **0.01** and **0.05** as the values and we manage to knock the model accuracy down to **0.105** and **0.015** respectively. As for the `epsilons2` list we can be a bit more cautious with the value we give to the epsilons since our model is going to have several passes on the image where it makes incremental changes to the image until it tricks the model. This means the risk of having too small a perturbation in defence of not having too much perturbation is small if we just choose a small epsilon values. Therefore, on this we can go with something like **0.005** and **0.009**. We then get a **0.0** and **0.005** model accuracy which is really low and the images at the end of it look unpertrubed as well.

Question 5: Looking at the displays below, can you discuss which attack seems to be the most efficient on our model? Is it the one-shot or the iterated one? Is that an expected result? Discuss.

Answer 5: Looking at the displays, we can do bit of a discussion on the fact that in the overarching scheme of things, the one-shot adversarial attack is still more efficient as in this case we are only doing a single forward and backward pass through the network and also the fact that we are not constantly making checks on iteratively pertrubed images as with the iterative adverserial attack. This implies a lower computational cost overall therefore we can argue that a one-shot attack is more efficient. One thing in favour of iterative attack that needs to be said is that the quality of a iterative attack is much higher when it comes to how untampered the images look after the fact and also the much lower (near-0 in case of a simple CNN like ours) we can get. To get these results from a one-shot attack we would have to set the epsilon so high that the images will definitely look tampered with and that wouldn't be ideal when really attacking a model.