CSE 565 Lab 5 Report
Adversarial Attack Lab

Notes: (IMPORTANT)

- ➔ It is **required** to use this report template.
- ➔ **Select <File> - <Make a copy> to make a copy of this report for yourself.**
- ➔ Report your work in each section. Carefully follow the instructions in the handout and show the screenshots of your code and the output, **along with your explanations**.
- ➔ Simply attaching code or screenshots without any explanation will NOT receive credits.
- ➔ To save space, the screenshots of your code should only include lines starting with `# Start code here #` and end with `# End code here #`. Note that you are NOT supposed to change the code outside of these blocks.
- ➔ Do **NOT** claim anything you didn't do. If you didn't try on a certain task, leave that section blank. You will receive a ZERO for the whole assignment if we find any overclaim.
- ➔ Grading will be based on your **explanations** and the completion of each task.
- ➔ After you finish, export this report as a PDF file and submit it on UBLearns.

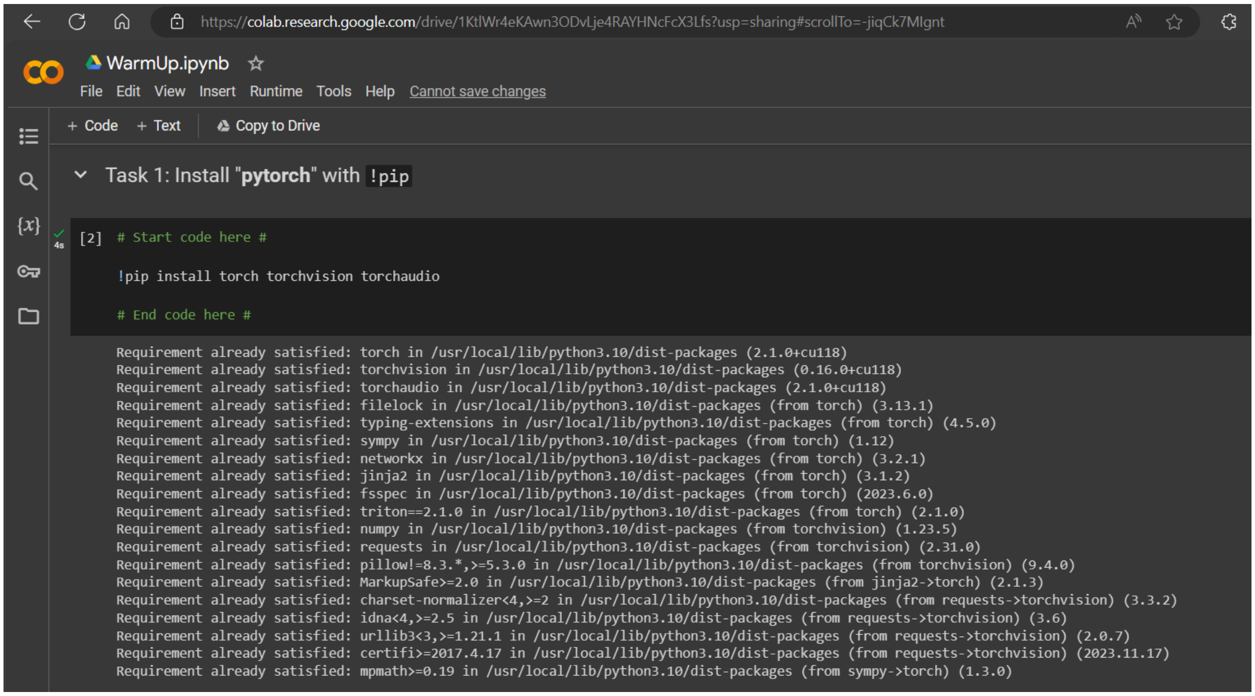
Your Full Name: Nazmus Saqiub
UBITName: nsaquib2
Student Number: 50510460

I, nsaquib2 (UBITName), have read and understood the course academic integrity policy.

(Your report will not be graded without filling in the above AI statement.)

Warming Up: Getting Familiar with Jupyter Notebook

(Link Follow the instructions for the three areas, fill in the three areas with the instructed content, and add a screenshot here.)



https://colab.research.google.com/drive/1KtlWr4eKAwn3ODvLje4RAYHNcFcX3Lfs?usp=sharing#scrollTo=-jiqCk7MIgnt

WarmUp.ipynb

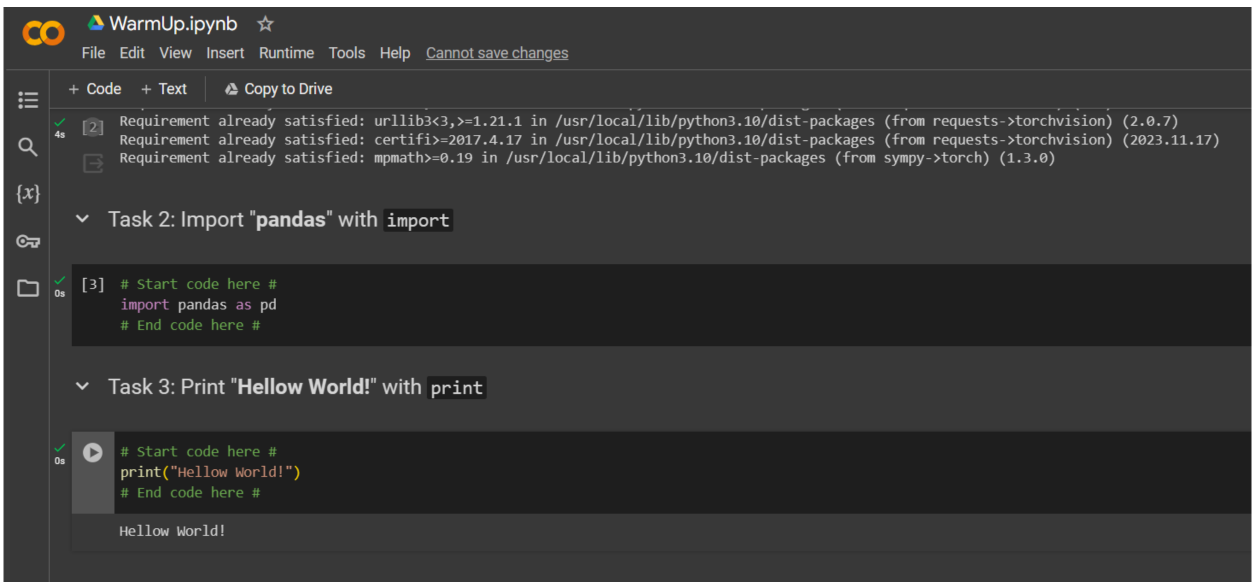
File Edit View Insert Runtime Tools Help Cannot save changes

+ Code + Text Copy to Drive

Task 1: Install "**pytorch**" with `!pip`

```
[2] # Start code here #

    !pip install torch torchvision torchaudio

    # End code here #
```

```
Requirement already satisfied: torch in /usr/local/lib/python3.10/dist-packages (2.1.0+cu118)
Requirement already satisfied: torchvision in /usr/local/lib/python3.10/dist-packages (0.16.0+cu118)
Requirement already satisfied: torchaudio in /usr/local/lib/python3.10/dist-packages (2.1.0+cu118)
Requirement already satisfied: filelock in /usr/local/lib/python3.10/dist-packages (from torch) (3.13.1)
Requirement already satisfied: typing-extensions in /usr/local/lib/python3.10/dist-packages (from torch) (4.5.0)
Requirement already satisfied: sympy in /usr/local/lib/python3.10/dist-packages (from torch) (1.12)
Requirement already satisfied: networkx in /usr/local/lib/python3.10/dist-packages (from torch) (3.2.1)
Requirement already satisfied: jinja2 in /usr/local/lib/python3.10/dist-packages (from torch) (3.1.2)
Requirement already satisfied: fsspec in /usr/local/lib/python3.10/dist-packages (from torch) (2023.6.0)
Requirement already satisfied: triton==2.1.0 in /usr/local/lib/python3.10/dist-packages (from torch) (2.1.0)
Requirement already satisfied: numpy in /usr/local/lib/python3.10/dist-packages (from torchvision) (1.23.5)
Requirement already satisfied: requests in /usr/local/lib/python3.10/dist-packages (from torchvision) (2.31.0)
Requirement already satisfied: pillow!=8.3.*,>=5.3.0 in /usr/local/lib/python3.10/dist-packages (from torchvision) (9.4.0)
Requirement already satisfied: MarkupSafe>=2.0 in /usr/local/lib/python3.10/dist-packages (from jinja2->torch) (2.1.3)
Requirement already satisfied: charset-normalizer<4,>=2 in /usr/local/lib/python3.10/dist-packages (from requests->torchvision) (3.3.2)
Requirement already satisfied: idna<4,>=2.5 in /usr/local/lib/python3.10/dist-packages (from requests->torchvision) (3.6)
Requirement already satisfied: urllib3<3,>=1.21.1 in /usr/local/lib/python3.10/dist-packages (from requests->torchvision) (2.0.7)
Requirement already satisfied: certifi>=2017.4.17 in /usr/local/lib/python3.10/dist-packages (from requests->torchvision) (2023.11.17)
Requirement already satisfied: mpmath>=0.19 in /usr/local/lib/python3.10/dist-packages (from sympy->torch) (1.3.0)
```



WarmUp.ipynb

File Edit View Insert Runtime Tools Help Cannot save changes

+ Code + Text Copy to Drive

```
Requirement already satisfied: urllib3<3,>=1.21.1 in /usr/local/lib/python3.10/dist-packages (from requests->torchvision) (2.0.7)
Requirement already satisfied: certifi>=2017.4.17 in /usr/local/lib/python3.10/dist-packages (from requests->torchvision) (2023.11.17)
Requirement already satisfied: mpmath>=0.19 in /usr/local/lib/python3.10/dist-packages (from sympy->torch) (1.3.0)
```

Task 2: Import "**pandas**" with `import`

```
[3] # Start code here #
    import pandas as pd
    # End code here #
```

Task 3: Print "**Hellow World!**" with `print`

```
# Start code here #
print("Hellow World!")
# End code here #
```

```
Hellow World!
```

Task 1: Number of Images

(Show your completed cell block and the output.)

Ans:
The number of images in test_data is 20. I found it by the length of test_data.

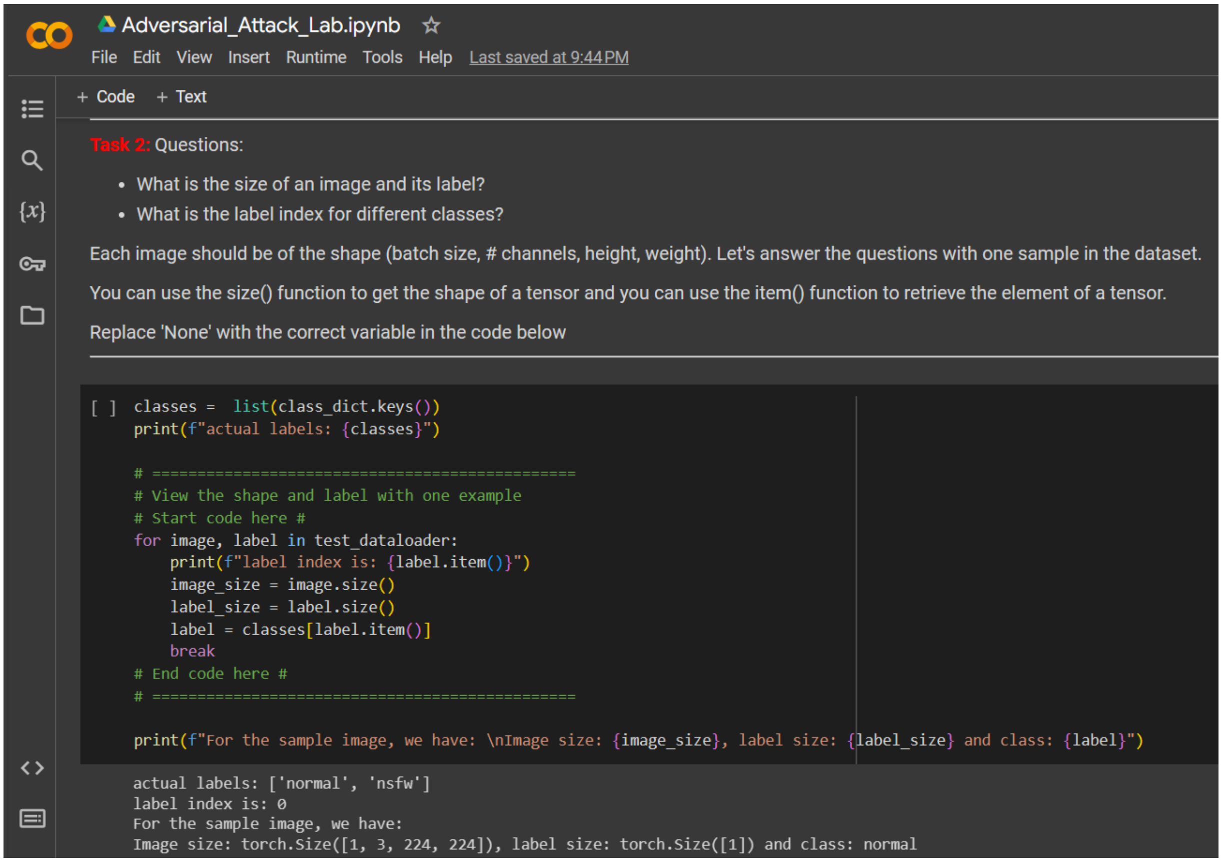
Task 1: Find the number of images there are in all category by replacing "None" with your code.
Don't access the attribute '__len__'. You can use the len function directly on the object of the CustomImageDataset class.

```
[6] # Start code here #
    total_number_of_images = len(test_data)
    # End code here #

    print(f"The number of images in the test dataset is: {total_number_of_images}")

    The number of images in the test dataset is: 20
```

Task 2: Size of image, its label, and class

(Complete the code in the designated block in the Task 2 code cell and show your result. You will need to report the dimension (size/shape) of the test image after transformation, the dimension of the label, and its ground truth (label/class/category).)

Ans:
I used size() function to get the shape of the image tensor and the label tensor, and the item() function to retrieve the actual label value from the tensor. The image size after transformation is [1, 3, 224, 224], label size is [1], and the class of used the image “normal”.



Adversarial_Attack_Lab.ipynb

File Edit View Insert Runtime Tools Help Last saved at 9:44 PM

+ Code + Text

Task 2: Questions:

- What is the size of an image and its label?
- What is the label index for different classes?

Each image should be of the shape (batch size, # channels, height, weight). Let's answer the questions with one sample in the dataset.

You can use the size() function to get the shape of a tensor and you can use the item() function to retrieve the element of a tensor.

Replace 'None' with the correct variable in the code below

```
classes =  list(class_dict.keys())
print(f"actual labels: {classes}")

# ==============================================
# View the shape and label with one example
# Start code here #
for image, label in test_dataloader:
    print(f"label index is: {label.item()}")
    image_size = image.size()
    label_size = label.size()
    label = classes[label.item()]
    break
# End code here #
# ==============================================

print(f"For the sample image, we have: \nImage size: {image_size}, label size: {label_size} and class: {label}")
```

```
actual labels: ['normal', 'nsfw']
label index is: 0
For the sample image, we have:
Image size: torch.Size([1, 3, 224, 224]), label size: torch.Size([1]) and class: normal
```

Task 3: Evaluate the Pre-trained Model

(Present your completed cell block and add a screenshot of the output. Execute the next code cell to demonstrate the prediction with one sample image. Include a screenshot of the output.)

Ans:

The accuracy of our model is 85%, which is fairly good.

The sample image is normal with 99% score which is a pretty solid prediction by our model.

Adversarial_Attack_Lab.ipynb

File Edit View Insert Runtime Tools Help Last saved at 9:44 PM

+ Code + Text

Task 3: Now, please utilize the `evaluate` function and write your own code to print out the accuracy for the test dataset.

```
# ==============================================
# Print out the accuracy for the test dataset
# Start code here #
accuracy = evaluate(model, test_dataloader)
print(f"Accuracy: {accuracy * 100:.2f}%")
# End code here #
# ==============================================
```

```
Accuracy: 85.00%
```

```
# A function for image input prediction
def model_predction(img):
    # use the model to predict the label
    with torch.no_grad():
        inputs = processor(images=img, return_tensors="pt").to(device)
        outputs = model(**inputs)
        logits = outputs.logits

        predicted_label = logits.argmax(-1).item()
        score = logits.softmax(-1)[0, predicted_label].item()
        return model.config.id2label[predicted_label], score

# Get one image to visualize and generate the prediction
image, label = test_data[index]
plt.imshow(image)
print(f'class = {label}, which is {test_data.classes[label]}')
```

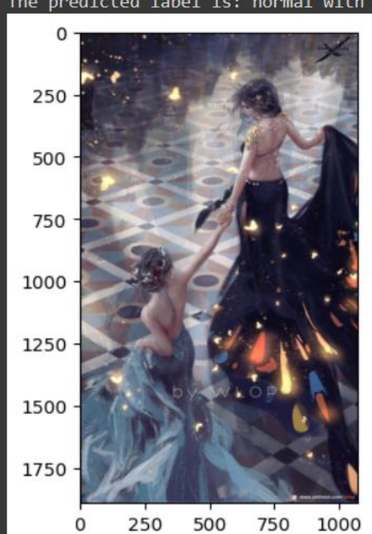
Adversarial_Attack_Lab.ipynb

File Edit View Insert Runtime Tools Help Last saved at 9:44 PM

+ Code + Text

```
# Get one image to visualize and generate the prediction
image, label = test_data[index]
plt.imshow(image)
print(f'class = {label}, which is {test_data.classes[label]}')

# Generate the prediction
predicted_label, score = model_predction(image)
print(f"The predicted label is: {predicted_label} with a score of: {score}")
```

```
class = 0, which is normal
The predicted label is: normal with a score of: 0.9998693466186523
```



Task 4-1: Implement FGSM formula

(Show your completed code block.)

Ans:
I implemented the formula to generate the perturbed image using FSGM attack. In the formula I applied the sign of the data gradient by scaling it with alpha, and added it to the image.

Adversarial_Attack_Lab.ipynb

File Edit View Insert Runtime Tools Help Last saved at 9:44 PM

\+ Code + Text

Task 4-1: Implement the FGSM formula in the code blow. Replace "None" with the formula.

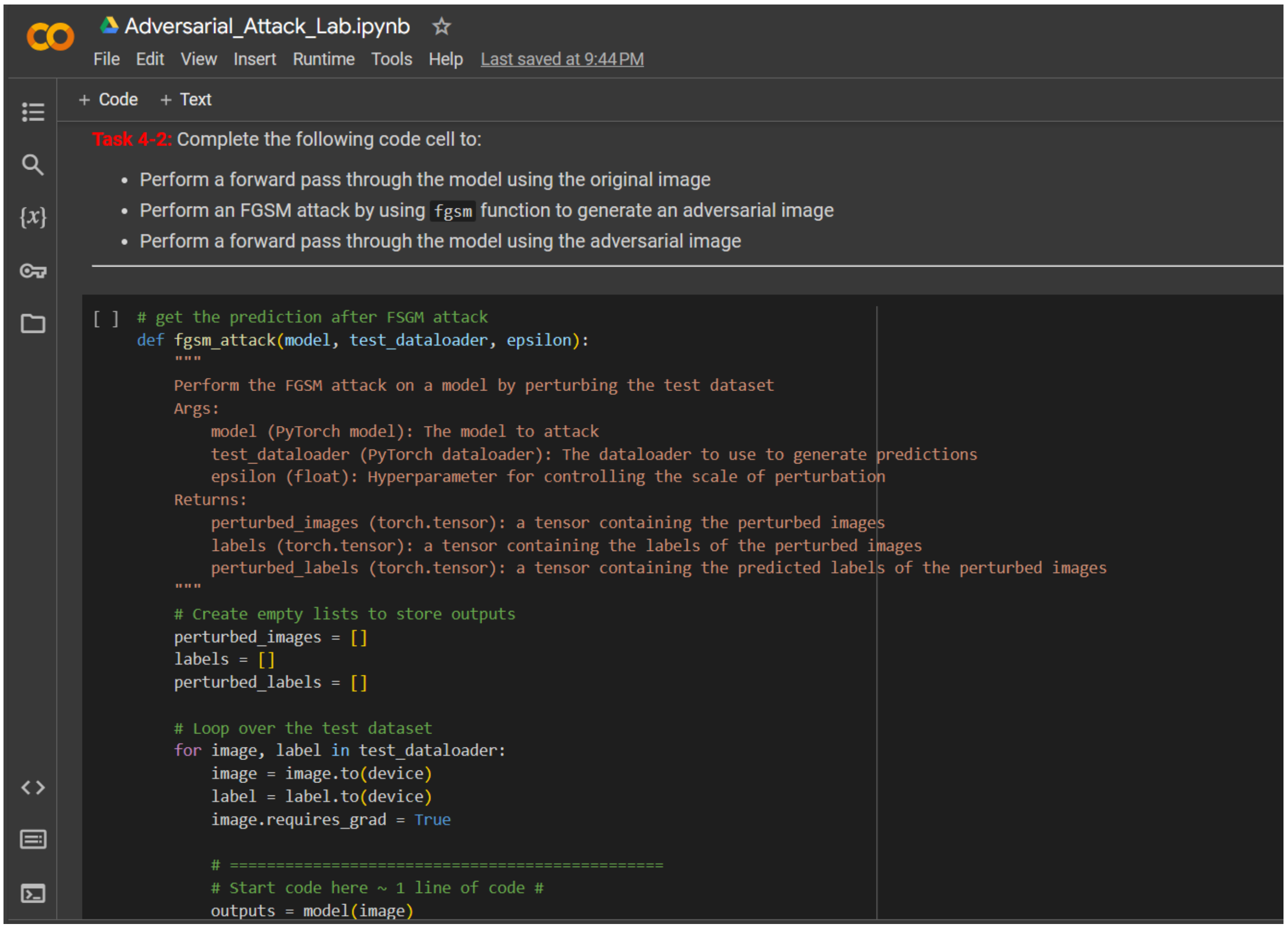
Credit: The code in this section has been adapted from Nathan Inkawhich's work on adversarial example generation.

```
[ ] # implement FSGM attack fuction
    def fgsm(image, epsilon, data_grad):
        """
        Perform the FGSM attack on a single image
        Args:
            image (torch.tensor): The image to be perturbed
            epsilon (float): Hyperparameter for controlling the scale of perturbation
            data_grad (): The gradient of the loss wrt to image
        Returns:
            perturbed_image (torch.tensor): a perturbed image
        """
        # Collect the element-wise sign of the data gradient
        sign_data_grad = data_grad.sign()


        # ===============================================
        # Create the perturbed image by adjusting each pixel of the input image
        # Start code here ~ 1 line of code #
        perturbed_image = image + epsilon * sign_data_grad
        # End code here #
        # ===============================================

        # Adding clipping to maintain [0,1] range
        if epsilon != 0.0:
            perturbed_image = torch.clamp(perturbed_image, 0, 1)

        # Return the perturbed image
        return perturbed_image
```

Task 4-2: Pass perturbed images through the model to perform an FGSM attack

(Show your completed code blocks. There are three code blocks to complete.)

Ans:
I applied changes to the code to perform the forward pass with the original image, apply the FGSM attack to generate the perturbed image, and then perform another forward pass with the perturbed image to get the predicted label.



Task 4-2: Complete the following code cell to:

- Perform a forward pass through the model using the original image
- Perform an FGSM attack by using `fgsm` function to generate an adversarial image
- Perform a forward pass through the model using the adversarial image

```
# get the prediction after FSGM attack
def fgsm_attack(model, test_dataloader, epsilon):
    """
    Perform the FGSM attack on a model by perturbing the test dataset
    Args:
        model (PyTorch model): The model to attack
        test_dataloader (PyTorch dataloader): The dataloader to use to generate predictions
        epsilon (float): Hyperparameter for controlling the scale of perturbation
    Returns:
        perturbed_images (torch.tensor): a tensor containing the perturbed images
        labels (torch.tensor): a tensor containing the labels of the perturbed images
        perturbed_labels (torch.tensor): a tensor containing the predicted labels of the perturbed images
    """
    # Create empty lists to store outputs
    perturbed_images = []
    labels = []
    perturbed_labels = []

    # Loop over the test dataset
    for image, label in test_dataloader:
        image = image.to(device)
        label = label.to(device)
        image.requires_grad = True

        # ==============================================
        # Start code here ~ 1 line of code #
        outputs = model(image)
```

Adversarial_Attack_Lab.ipynb

File Edit View Insert Runtime Tools Help Last saved at 9:44 PM

+ Code + Text

```
        # ===============================================
        # Start code here ~ 1 line of code #
        outputs = model(image)
        # End code here #
        # ===============================================

        logits = outputs.logits
        predicted_label = logits.argmax(-1).item()
        criterion = nn.CrossEntropyLoss()
        loss = criterion(outputs.logits, label)
        model.zero_grad()
        loss.backward()

        data_grad = image.grad.data

        # ===============================================
        # Call FGSM to add perturbation to the data
        # Start code here ~ 1 line of code #
        perturbed_image = fgsm(image, epsilon, data_grad)
        # End code here #
        # ===============================================

        perturbed_images.append(perturbed_image)
        labels.append(label)

        # ===============================================
        # Re-classify the perturbed image
        # Start code here ~ 1 line of code #
        perturbed_label = model(perturbed_image)
        perturbed_label = perturbed_label.logits.argmax(-1).item() # you can also modify this line if you like
        # End code here #
        # ===============================================

        perturbed_labels.append(perturbed_label)

    # Return the perturbed images and labels
    return perturbed_images, labels, perturbed_labels
```

Adversarial_Attack_Lab.ipynb

File Edit View Insert Runtime Tools Help Last saved at 9:44 PM

+ Code + Text

Execute FGSM attack

```
# Check the model performance after FGSM attack
fgsm_accuracies = []
fgsm_adversarial_examples = []
fgsm_original_labels = []
fgsm_predicion_labels = []

epsilons = [0.0, 0.02, 0.04, 0.06, 0.08, 0.1, 0.14]

for eps in epsilons:
    correct = 0
    total = 0
    perturbed_images, labels, perturbed_labels = fgsm_attack(model, test_dataloader, eps)
    for i in range(len(perturbed_images)):
        if perturbed_labels[i] == labels[i]:
            correct += 1
        total += 1
    accuracy = correct / total
    print("Epsilon: {}\tTest Accuracy = {} / {} = {}".format(eps, correct, len(test_dataloader), accuracy))

    fgsm_accuracies.append(accuracy)
    fgsm_adversarial_examples.append(perturbed_images)
    fgsm_original_labels.append(labels)
    fgsm_predicion_labels.append(perturbed_labels)
```

```
Epsilon: 0.0	Test Accuracy = 17 / 20 = 0.85
Epsilon: 0.02	Test Accuracy = 11 / 20 = 0.55
Epsilon: 0.04	Test Accuracy = 11 / 20 = 0.55
Epsilon: 0.06	Test Accuracy = 9 / 20 = 0.45
Epsilon: 0.08	Test Accuracy = 9 / 20 = 0.45
Epsilon: 0.1	Test Accuracy = 9 / 20 = 0.45
Epsilon: 0.14	Test Accuracy = 9 / 20 = 0.45
```

Task 4-3: Execute the FGSM attack using different epsilon values

(Run the three code cells and show the outputs. Briefly describe your observation.)

Ans:

The Accuracy vs. Epsilon graph illustrates how the accuracy of the model declines with increasing perturbation strength (epsilon). Small perturbations can have a big negative impact on the model's predictions.

The image I printed in this task is an example of an adversarial image generated by the FGSM attack. By observing the image, I understood how perturbation modifies the original image's look to deceive the model visually. The fabrication troubles the model to detect the correct class of the image.

The grid of photos shows multiple examples of adversarial samples for different epsilon values. The observation helped me understand the consistency and impact of the FGSM attack across different instances.

Adversarial_Attack_Lab.ipynb

File Edit View Insert Runtime Tools Help Last saved at 9:44 PM

+ Code + Text

```
# Draw the accuracy vs epsilon figure
def plot_accuracy_vs_epsilon(epsilons, accuracies):
    plt.figure(figsize=(6,4))
    plt.plot(epsilons, accuracies, "*-")
    plt.yticks(np.arange(0, 1.1, step=0.1))
    plt.xticks(np.arange(0, .21, step=0.03))
    plt.title("Accuracy vs Epsilon")
    plt.xlabel("Epsilon")
    plt.ylabel("Accuracy")
    plt.show()

# Visualize 5 examples of the adversarial samples for each epsilon
def plot_adversarial_examples(adversarial_examples, epsilons, original_labels, predicion_labels):
    cnt = 0
    plt.figure(figsize=(10,10))
    for i in range(len(epsilons)):
        for j in range(5):
            cnt += 1
            plt.subplot(len(epsilons),5,cnt)
            plt.xticks([], [])
            plt.yticks([], [])
            if j == 0:
                plt.ylabel("Eps: {}".format(epsilons[i]), fontsize=11)
                adv_image = adversarial_examples[i][j].reshape(3, 224, 224).squeeze().detach().cpu().numpy()
                adv_image = adv_image.transpose(1,2,0)

                plt.title("Label: {}".format(classes[original_labels[i][j]]) + "\n" + "Pred: {}".format(classes[predicion_labels[i][j]]))
            else:
                adv_image = adversarial_examples[i][j].reshape(3, 224, 224).squeeze().detach().cpu().numpy()
                adv_image = adv_image.transpose(1,2,0)

                plt.title("Label: {}".format(classes[original_labels[i][j]]) + "\n" + "Pred: {}".format(classes[predicion_labels[i][j]]))

            plt.imshow((adv_image))

    plt.tight_layout()
    plt.show()
```

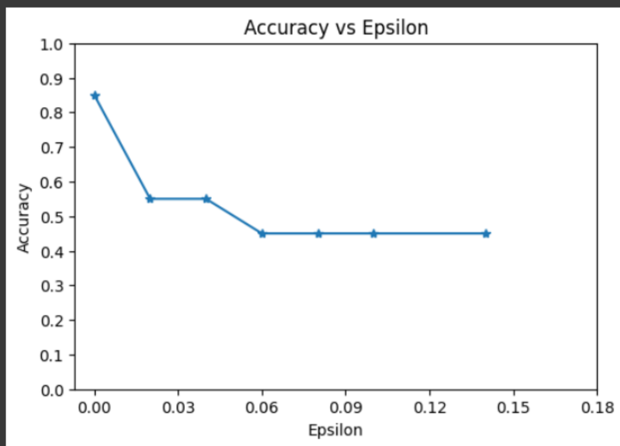
Adversarial_Attack_Lab.ipynb

File Edit View Insert Runtime Tools Help Last saved at 9:44 PM

+ Code + Text

Task 4-3: Run the following code cells to visualize the results from FGSM attack. Save the plots and attach them into your report.

```
# Visualize accuracy vs epsilon
plot_accuracy_vs_epsilon(epsilons, fgsm_accuracies)
```




```
# Visualize the adversarial sample
image = fgsm_adversarial_examples[1][1]
image = image.reshape(3, 224, 224).squeeze().detach().cpu().numpy()
plt.imshow(image.transpose(1,2,0))
```

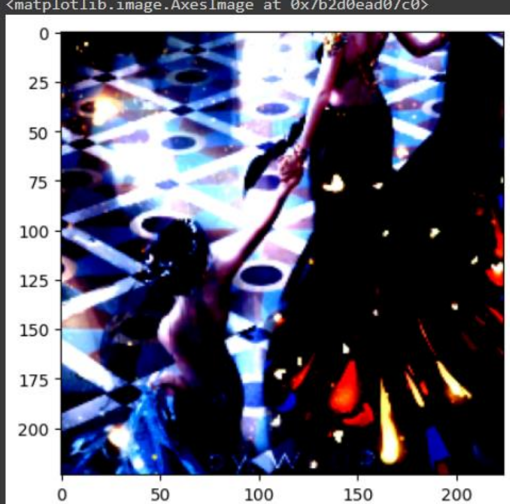

Adversarial_Attack_Lab.ipynb

File Edit View Insert Runtime Tools Help Last saved at 9:44 PM

+ Code + Text

```
# Visualize the adversarial sample
image = fgsm_adversarial_examples[1][1]
image = image.reshape(3, 224, 224).squeeze().detach().cpu().numpy()
plt.imshow(image.transpose(1,2,0))
```


```
<matplotlib.image.AxesImage at 0x7b2d0ead07c0>
```



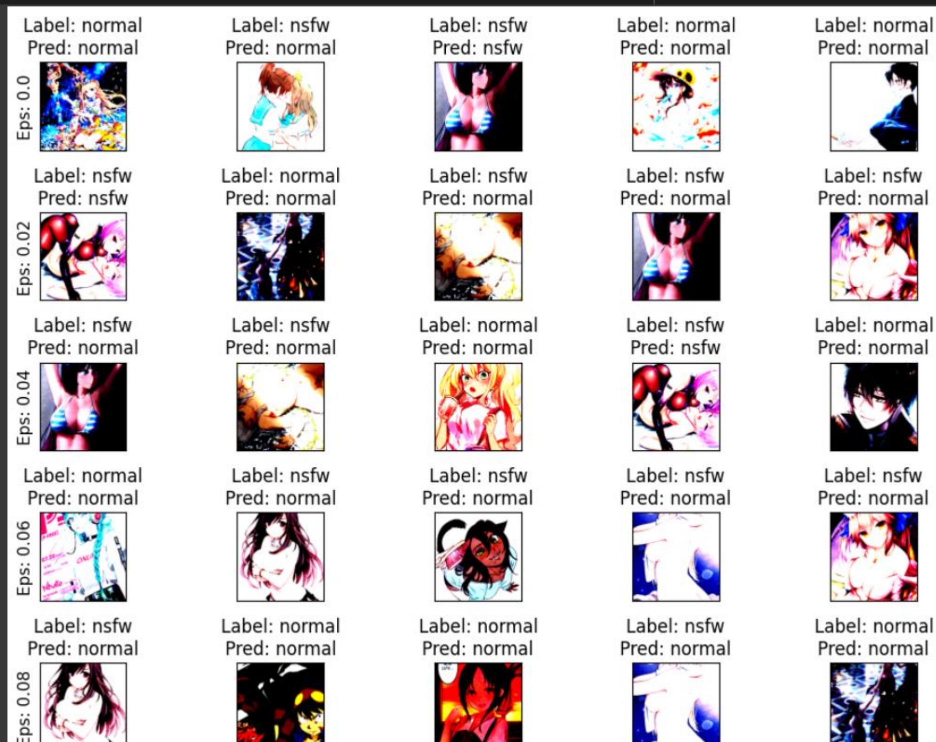

```
# Visualize the adversarial samples
plot_adversarial_examples(fgsm_adversarial_examples, epsilons, fgsm_original_labels, fgsm_predicion_labels)
```

Adversarial_Attack_Lab.ipynb

File Edit View Insert Runtime Tools Help Last saved at 9:44 PM

+ Code + Text

```
# Visualize the adversarial samples
plot_adversarial_examples(fgsm_adversarial_examples, epsilons, fgsm_original_labels, fgsm_predicion_labels)
```

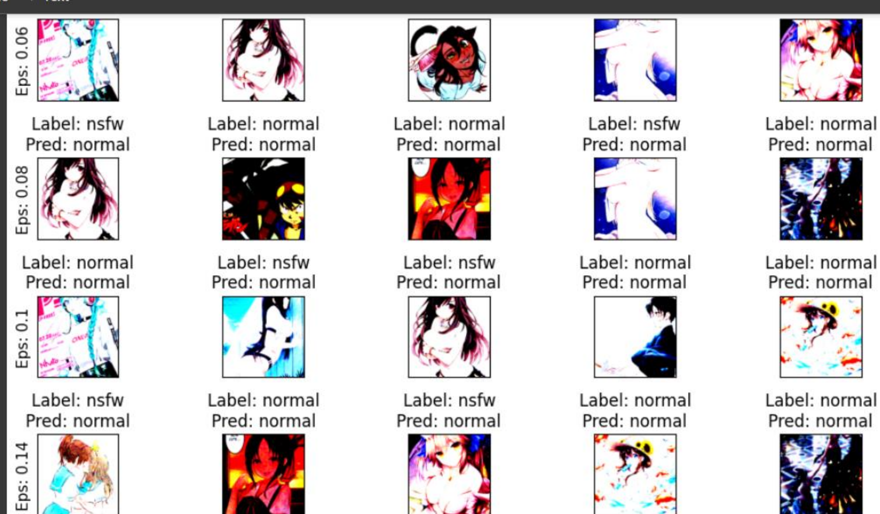


Connected to Python 3 Google Compute Engine backend (GPU)

Adversarial_Attack_Lab.ipynb

File Edit View Insert Runtime Tools Help Last saved at 9:44 PM

+ Code + Text



PGD Attacks

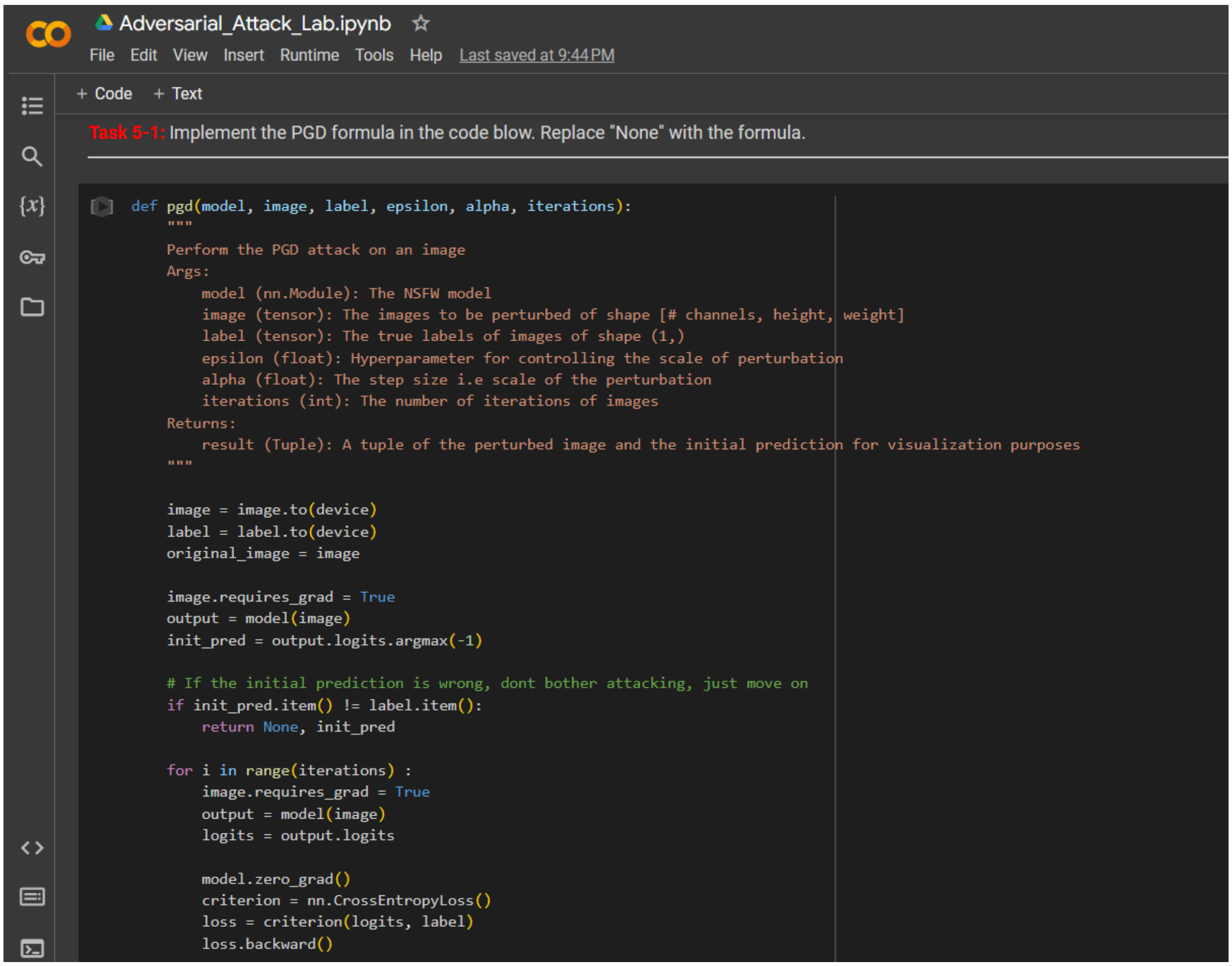
We will also use the projected gradient descent (PGD) method to generate adversarial examples.

Given a vectorized image x, PGD generates an adversarial image by

Task 5-1: Implement PGD formula

(Show your completed code block.)

Ans:
I implemented the formula to generate the perturbed image using PGD attack. In the formula I applied the sign of the data gradient by scaling it with alpha, and added it to the image.



Task 5-1: Implement the PGD formula in the code blow. Replace "None" with the formula.

```python
def pgd(model, image, label, epsilon, alpha, iterations):
    """
    Perform the PGD attack on an image
    Args:
        model (nn.Module): The NSFW model
        image (tensor): The images to be perturbed of shape [# channels, height, weight]
        label (tensor): The true labels of images of shape (1,)
        epsilon (float): Hyperparameter for controlling the scale of perturbation
        alpha (float): The step size i.e scale of the perturbation
        iterations (int): The number of iterations of images
    Returns:
        result (Tuple): A tuple of the perturbed image and the initial prediction for visualization purposes
    """

    image = image.to(device)
    label = label.to(device)
    original_image = image

    image.requires_grad = True
    output = model(image)
    init_pred = output.logits.argmax(-1)

    # If the initial prediction is wrong, dont bother attacking, just move on
    if init_pred.item() != label.item():
        return None, init_pred

    for i in range(iterations) :
        image.requires_grad = True
        output = model(image)
        logits = output.logits

        model.zero_grad()
        criterion = nn.CrossEntropyLoss()
        loss = criterion(logits, label)
        loss.backward()
```

```python
        if init_pred.item() != label.item():
            return None, init_pred

        for i in range(iterations) :
            image.requires_grad = True
            output = model(image)
            logits = output.logits

            model.zero_grad()
            criterion = nn.CrossEntropyLoss()
            loss = criterion(logits, label)
            loss.backward()

            sign_data_grad = image.grad.sign()

            # ===============================================
            # Re-classify the perturbed image
            # Start code here ~ 1 line of code #

            perturbed_image = original_image + alpha * sign_data_grad
            perturbed_image = torch.clamp(perturbed_image, 0, 1)


            # End code here #
            # ===============================================

            # Perform clipping
            eta = torch.clamp(perturbed_image - original_image, min = -epsilon, max = epsilon)
            image = torch.clamp(original_image + eta, min = 0, max = 1).detach_()
```

Task 5-2: Pass perturbed images through the model to perform a PGD attack

(Show your completed code blocks. There are two code blocks to complete.)

Ans:

I applied my code in the assigned code block to perform a PGD attack using the pgd function. It obtains the perturbed image and the initial prediction for visualization purposes.

Adversarial_Attack_Lab.ipynb

File Edit View Insert Runtime Tools Help Last saved at 9:44 PM

+ Code + Text

```python
    # Loop over the test dataset

    for image, label in test_dataloader:
        image = image.to(device)
        label = label.to(device)

        # ==============================================
        # Start code here #
        perturbed_image, init_pred = pgd(model, image, label, epsilon, alpha, iterations)

        # End code here #
        # ==============================================

        if perturbed_image is not None:
            perturbed_images.append(perturbed_image)
            labels.append(label)

            # ==============================================
            # Start code here ~ 1-2 lines of code#
            output_per = model(perturbed_image)
            perturbed_label = output_per.logits.argmax(-1)

            # End code here #
            # ==============================================
            perturbed_labels.append(perturbed_label)

    # Return the perturbed images and labels
    return perturbed_images, labels, perturbed_labels
```

Adversarial_Attack_Lab.ipynb

File Edit View Insert Runtime Tools Help Last saved at 9:44 PM

+ Code + Text

Execute PGD attack

```python
# Run the PGD attack
pgd_accuracies = []
pgd_adversarial_examples = []
pgd_original_labels = []
pgd_predicion_labels = []

epsilons = [0.0, 0.02, 0.04, 0.06, 0.08, 0.1, 0.14]
alpha = 0.01
iterations = 5

for eps in epsilons:
    correct = 0
    total = 0
    perturbed_images, labels, perturbed_labels = pgd_attack(model, test_dataloader, eps, alpha, iterations, 'untargeted')
    for i in range(len(perturbed_images)):
        if perturbed_labels[i] == labels[i]:
            correct += 1
        total += 1
    accuracy = correct / total
    print("Epsilon: {}\tTest Accuracy = {} / {} = {}".format(eps, correct, total, accuracy))

    pgd_accuracies.append(accuracy)
    pgd_adversarial_examples.append(perturbed_images)
    pgd_original_labels.append(labels)
    pgd_predicion_labels.append(perturbed_labels)
```

```
Epsilon: 0.0    Test Accuracy = 11 / 17 = 0.6470588235294118
Epsilon: 0.02   Test Accuracy = 9 / 17 = 0.5294117647058824
Epsilon: 0.04   Test Accuracy = 5 / 17 = 0.29411764705882354
Epsilon: 0.06   Test Accuracy = 5 / 17 = 0.29411764705882354
Epsilon: 0.08   Test Accuracy = 5 / 17 = 0.29411764705882354
Epsilon: 0.1    Test Accuracy = 5 / 17 = 0.29411764705882354
Epsilon: 0.14   Test Accuracy = 5 / 17 = 0.29411764705882354
```

Task 5-3: Visualize the results after PGD attack

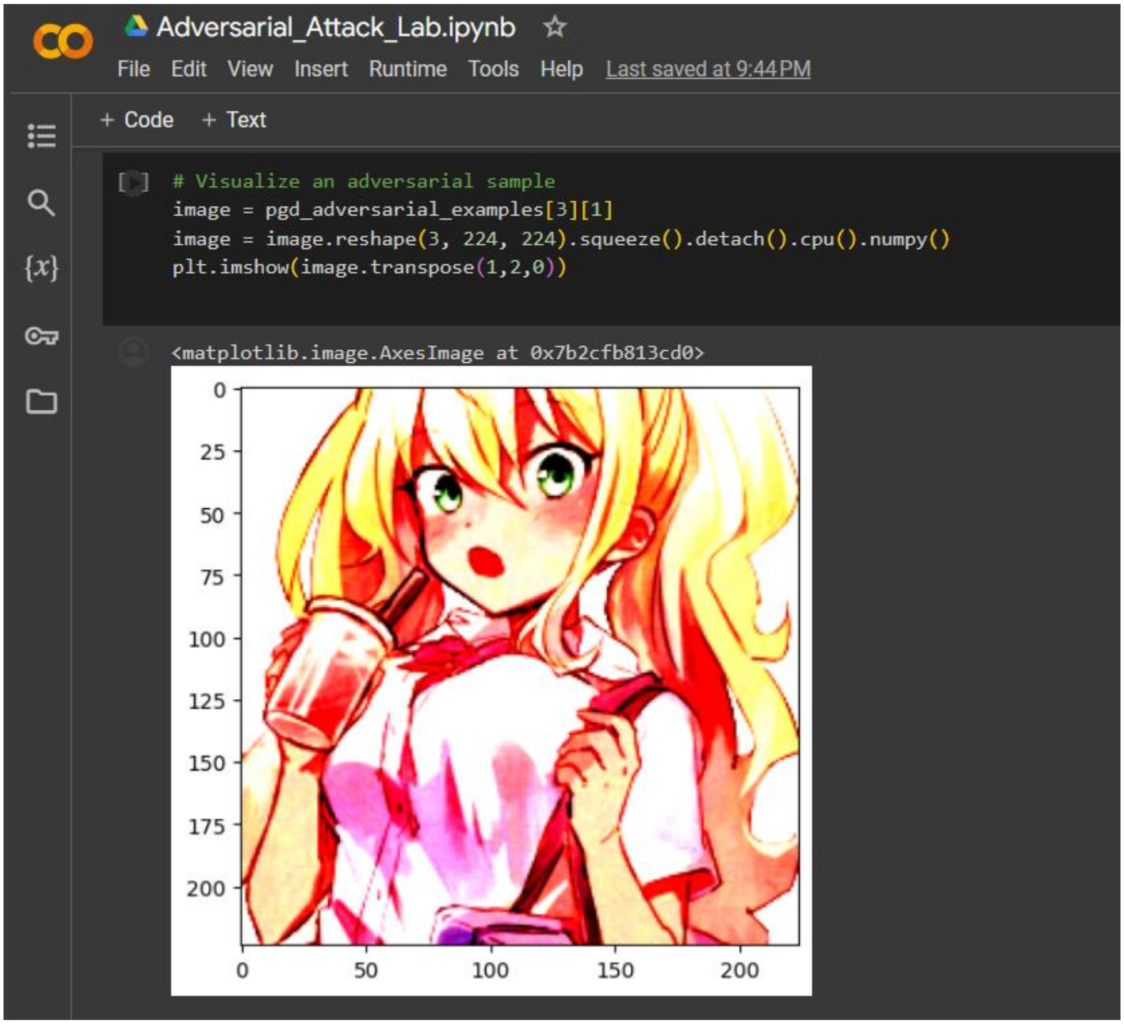
(Run the three code cells and show the outputs. Briefly describe your observation.)

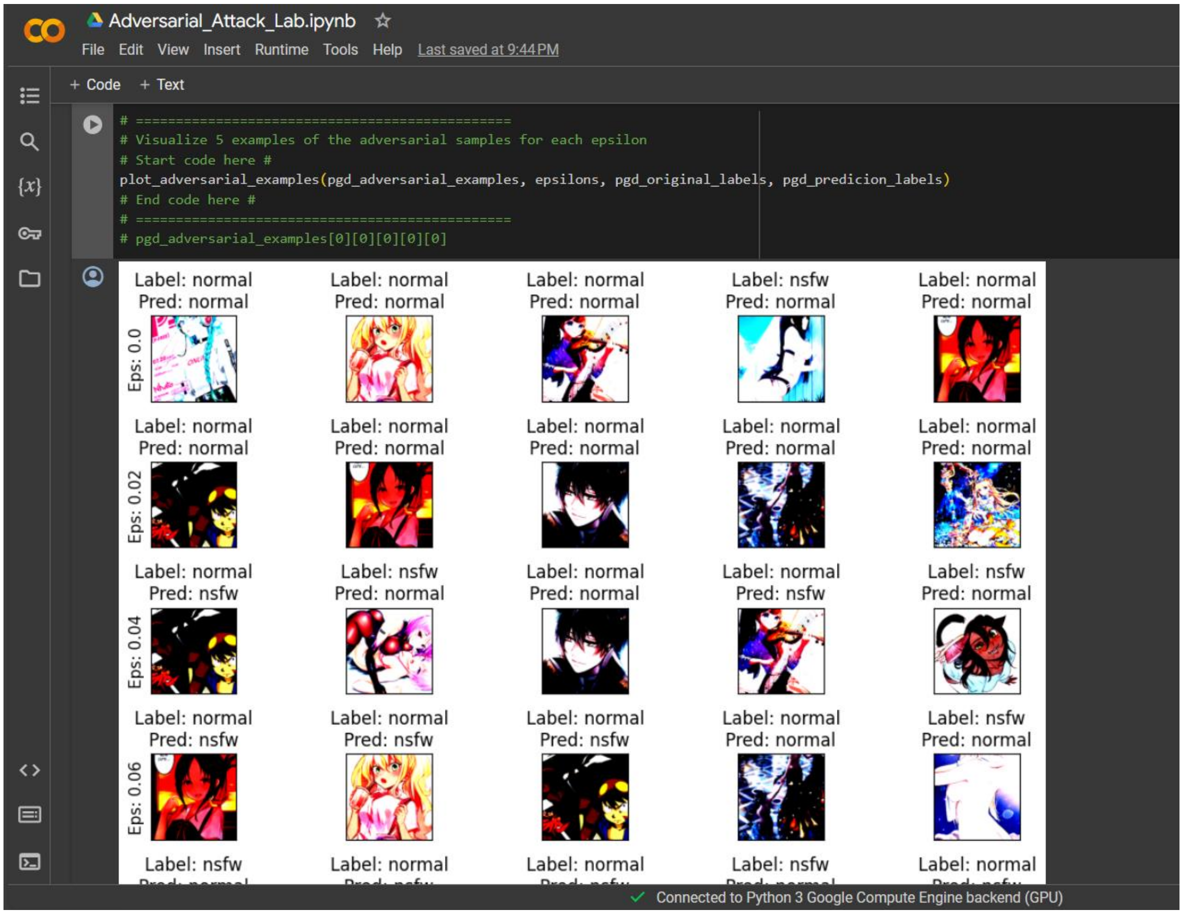
Ans:

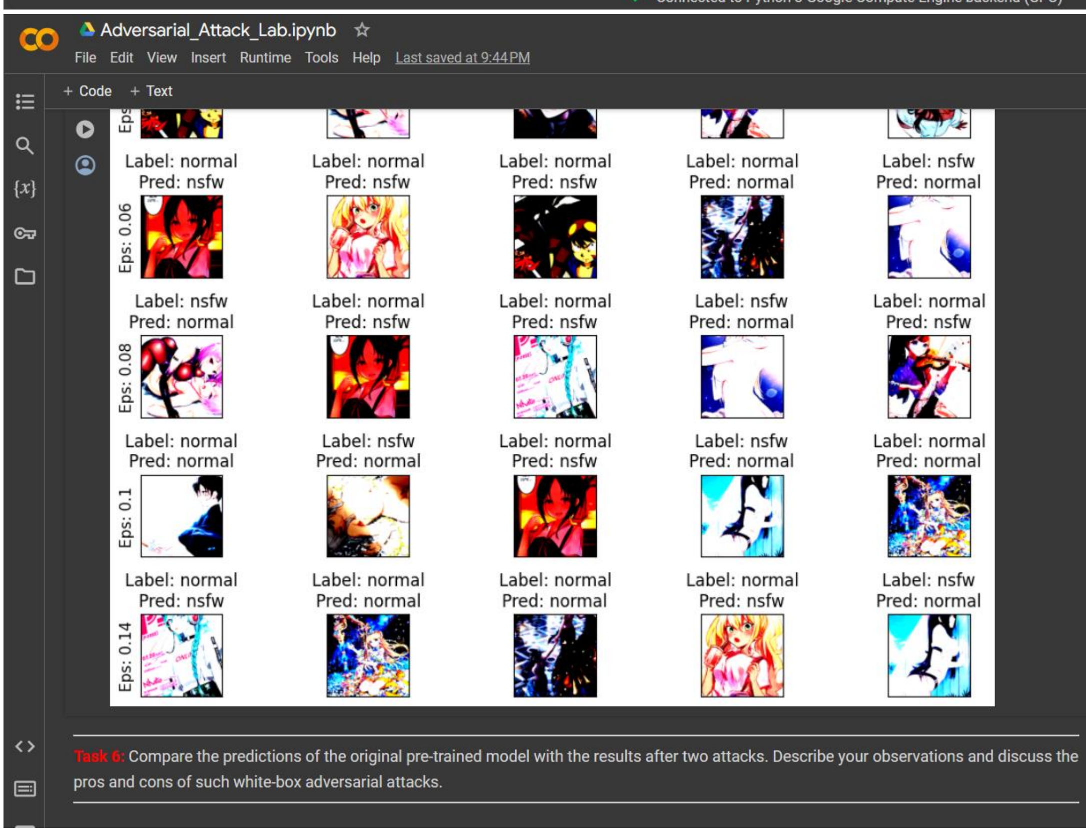
By observing the accuracy vs. epsilon graph, I understood the accuracy changes as the perturbation strength (epsilon) increases. Typically, the accuracy decreases as epsilon increases, indicating the success of the PGD attack.

I printed a single adversarial image generated by the PGD attack. The image is reshaped and then visualized using Matplotlib. Observing this image helps one understand how the perturbation affected the original image's visual quality. The attack changed the color of the image and ruined the smooth texture.

Then I generated a grid of 5 examples for each epsilon value. Each row corresponds to a specific epsilon, and each column shows a different example. The grid photos represent the original images' corresponding labels, and the labels predicted by the model after the PGD attack. This illustration assists in evaluating the impact and diversity of challenges produced for various epsilon values. It offers a more thorough understanding of the model's vulnerability to the attack.


Adversarial_Attack_Lab.ipynb
File Edit View Insert Runtime Tools Help Last saved at 9:44 PM
+ Code + Text
# Visualize an adversarial sample
image = pgd_adversarial_examples[3][1]
image = image.reshape(3, 224, 224).squeeze().detach().cpu().numpy()
plt.imshow(image.transpose(1,2,0))
<matplotlib.image.AxesImage at 0x7b2cfb813cd0>
0
25
50
75
100
125
150
175
200
0
50
100
150
200


Adversarial_Attack_Lab.ipynb
File Edit View Insert Runtime Tools Help Last saved at 9:44 PM
+ Code + Text
# ==
# Visualize 5 examples of the adversarial samples for each epsilon
# Start code here #
plot_adversarial_examples(pgd_adversarial_examples, epsilons, pgd_original_labels, pgd_predicion_labels)
# End code here #
# ==
# pgd_adversarial_examples[0][0][0][0][0]
Label: normal Pred: normal
Label: normal Pred: normal
Label: normal Pred: normal
Label: nsfw Pred: normal
Label: normal Pred: normal
Eps: 0.0
Label: normal Pred: normal
Label: normal Pred: normal
Label: normal Pred: normal
Label: normal Pred: normal
Label: normal Pred: normal
Eps: 0.02
Label: normal Pred: nsfw
Label: nsfw Pred: normal
Label: normal Pred: normal
Label: normal Pred: nsfw
Label: nsfw Pred: normal
Eps: 0.04
Label: normal Pred: nsfw
Label: normal Pred: nsfw
Label: normal Pred: nsfw
Label: normal Pred: normal
Label: nsfw Pred: normal
Eps: 0.06
Label: nsfw
Label: normal
Label: normal
Label: nsfw
Label: normal
Connected to Python 3 Google Compute Engine backend (GPU)


Adversarial_Attack_Lab.ipynb
File Edit View Insert Runtime Tools Help Last saved at 9:44 PM
+ Code + Text
Label: normal Pred: nsfw
Label: normal Pred: nsfw
Label: normal Pred: nsfw
Label: normal Pred: normal
Label: nsfw Pred: normal
Eps: 0.06
Label: nsfw Pred: normal
Label: normal Pred: nsfw
Label: normal Pred: nsfw
Label: nsfw Pred: normal
Label: normal Pred: nsfw
Eps: 0.08
Label: normal Pred: normal
Label: nsfw Pred: normal
Label: normal Pred: nsfw
Label: nsfw Pred: normal
Label: normal Pred: normal
Eps: 0.1
Label: normal Pred: nsfw
Label: normal Pred: normal
Label: normal Pred: normal
Label: normal Pred: nsfw
Label: nsfw Pred: normal
Eps: 0.14
Task 6: Compare the predictions of the original pre-trained model with the results after two attacks. Describe your observations and discuss the pros and cons of such white-box adversarial attacks.

Discussion

(Compare the predictions of the original pre-trained model with the results after two attacks. Describe your observations and discuss the pros and cons of such white-box adversarial attacks.)

Ans:

Before any adversarial attacks, the models made pretty accurate predictions to classify the test data. However, both the FGSM and PGD attacks lead to a decrease in model accuracy. It lead to misclassifications of the images and altered the predicted labels, meaning that the attacker was successful. Adversarial attacks' ability to successfully lower accuracy points to a model weakness.

Pros of white-box adversarial attacks:

1. It highlights weaknesses in training data and model architectures, which aids in determining where the robustness of the model need to be strengthened.
2. Through comprehension of potential attacks on models, developers can improve security protocols.

Cons of white-box adversarial attacks:

1. It is possible that adversarial examples designed for one model will not translate well to another.
2. Ethical concerns are raised by adversarial attacks, particularly in applications where safety is crucial.