Adversarial Machine Learning

Arthea Valderrama
Faculty Advisor: Dr. Ian Chen
Committee Member: Dr. Sirong Lin

November 20, 2025

Contents

1 Abstract

Machine Learning (ML) has become integral to daily life, due to its use in autonomous systems, the healthcare industry, etc., because of its capability to produce a desired outcome based off of learning from training data. However, adversarial attacks attempt to deceive learning models by implementing a defective input, compromising the effectiveness as a result. We explore the mechanisms behind attacks on convolutional neural networks (CNNs). This project aims to contribute to the development of safer ML models to ensure their reliability, through evaluating the effectiveness of adversarial attacks.

2 Motivation

This is an addition to my current capstone project through UML's collaboration with Schneider Electric. The company is a multinational corporation specializing in energy management and automation. One of the company's ambitions is ensuring robust automation processes for increased efficiency and quality of services.

In an autonomous setting that requires quality assurance for real-time classification, it is important that a model is adaptable to various conditions to ensure quality decision making. In extension to the existing project's image classification system on household objects in an assembly line, I expanded the dataset for cancer detection in a healthcare setting to demonstrate the transferability of ML models and its attacks.

ML assisted detection of brain cancer through MRI imaging can save resources and improve patient outcomes through early detection and accurate patient prognosis estimation. However, an adversarial attack would lead to errors in tumor recognition, harming patient health and treatment. Such an attack could occur for a variety of reasons: undermining trust in critical systems, financial gain, etc. By identifying vulnerabilities in brain cancer imaging detection we can ensure the security and trustworthiness of Ai.

3 Attack Scenarios

3.1 Carlini & Wagner

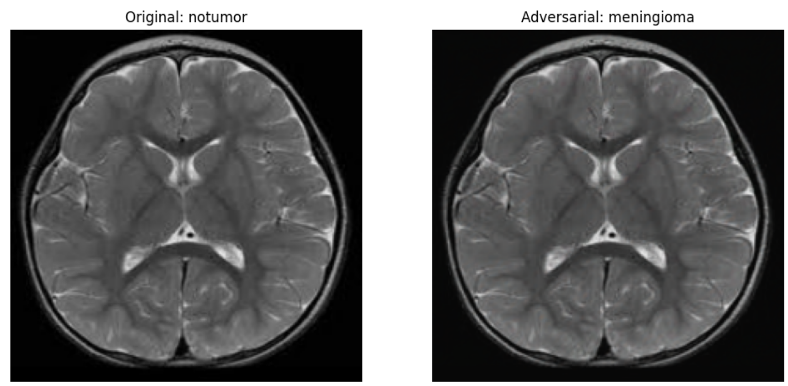


Figure 1: Example implementation of C&W attack

The Carlini & Wagner (C&W)[1] attack finds the smallest perturbation to a model's input data, while also increasing the chances of model misclassification through optimizing an iterative algorithm. The algorithm is defined as: **minimize** $||\delta|| + c * f(x + \delta)$, **such that** $x + \delta \in [0, 1]^n$, where δ is a small change made to an image x that changes its classification, and c is a constant that advances the trade-off between the size of the change and effectiveness of the attack. It is a white-box attack, meaning the attacker can launch the attack if they have knowledge on the model's information, such as its architecture and its parameters.

3.2 Simple Black-box Attack

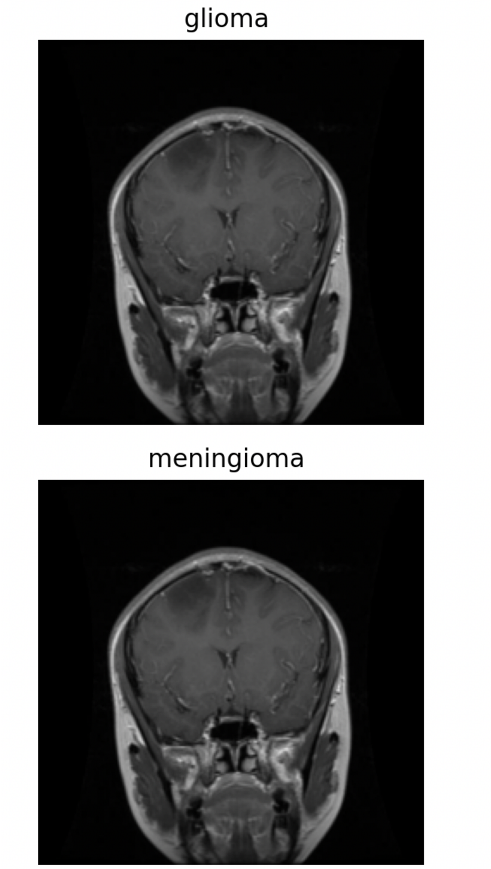


Figure 2: Small simBA distortion

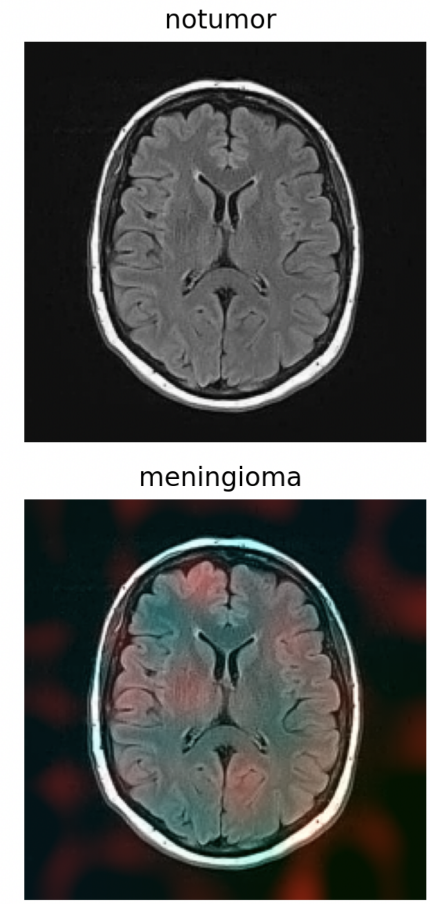


Figure 3: Higher distortion of simBA

While the C&W attack is a white-box attack that requires the adversary to have full knowledge of the model it is attacking, the Simple Black-box Attack (simBA)[2] is a type of black-box attack that does not need complete information about the model. In that case, simBA is more applicable in many scenarios. SimBA works by performing a random search on the pixels of an image to find minuscule perturbations in order to change the image away from its predicted classification and continuously checking the probability of success in changing input.

4 Anomaly Detection

An anomaly detection [3] model is a specialized CNN that identifies data points that significantly deviate from "expected" within the dataset. The model works by taking in a binary input: the normal patterns within the dataset and anomalous samples that deviate from the learned patterns. By incorporating aberrant samples in its training, the model is then able to flag unusual input. This can protect an AI-based system in real-time from instances that could result in harmful losses. Other use cases of anomaly detection within AI applications are detecting fraudulent financial transactions and suspicious network traffic in cybersecurity.

5 Procedure

A crucial factor in ML training is the selection of a robust dataset. I obtained a brain tumor MRI dataset of 7,022 images[4], and split the dataset 80% to train the model and 20% to evaluate the model's accuracy. I chose to select a large dataset for improved generalization with more diverse data samples and to reduce overfitting. The dataset includes 4 classifications of tumors: no tumor, glioma (tumors in the glial cells in the brain or spinal cord), meningioma (tumors that develops in the meninges), and pituitary (tumors in the pituitary gland).

In accordance with my capstone project, a convolutional neural network (CNN) was the model architecture chosen, specifically in the form of a DenseNet model. This model architecture consists of dense layers where each layer is connected to its previous layers in a feed-forward fashion. The DenseNet model is helpful for image classification, as its compact network improves information propagation through the network, has an enhanced ability to learn complex features, and requires less parameters in deep whilst not needing much computation.

The main programming language used in this project is Python. Python is ideal for working with ML, as it has numerous libraries for data pre-processing and evaluation, such as Sci-kit Learn, Numpy, Pandas, and Matplotlib. The main library used for ML is PyTorch, which allowed me to import a wide range of pretrained Neural Networks to improve model accuracy and functionality.

When coding my training script in Python to train the model, I created a class to preprocess the MRI images into classes for the model to distinguish between, and another class to define the model's parameters and functions. The model's class consists of a initialization function that uses features from Pytorch's built-in pretrained DenseNet model, a forward function to define the model's learning and optimization, and a training function to define the number of iterations the model passes through its learning algorithm while tracking its loss. Utilizing a pretrained model improved the model's learning rate and testing accuracy on samples it has not seen before.

Creating the attack code in Python, I modified widely available C&W and simBA attack code to apply to my MRI imaging model. Thus, helping to demonstrate the generalizability of the attack across different imaging models. I applied the attacks on the original dataset of 7,022 MRI images to generate enough adversarial samples for the anomaly detection model.

With the anomaly detection model, I then combined both the original training data and adversarial data. Thus, in total it learned the patterns and behavior between 14,044 MRI images to recognize between suspicious outliers and regular input.

Lastly, to create a simple web app to test the models on a user-friendly interface. Through the web app, I used the Flask library, HTML, CSS, and JavaScript code for live predictions on the trained dataset. I then incorporated Google's AI Gemini to detail more information about adversarial attacks, information about diagnostics, and educate the implications of the model's findings. A chat-bot was added to allow the user to talk more with the AI for any follow-up questions.

6 Results

Through testing over a sample batch size of 32 images, I was able to calculate the attack success rate of the C&W, simBA, and Anomaly Detection model. The C&W attack is more successful than simBA by a

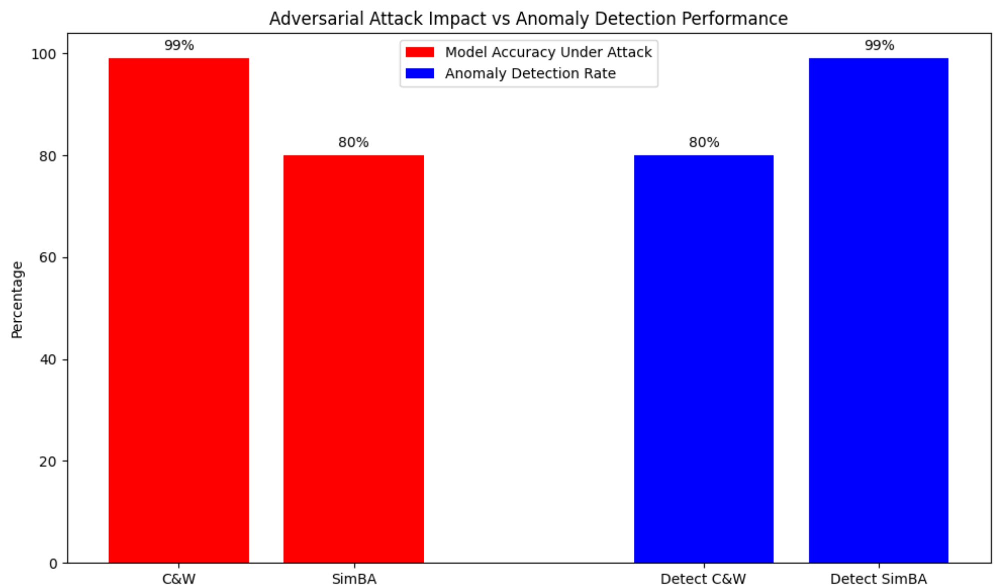


Figure 4: Success Rate of Attacks and Anomaly Detections

factor of around 20%. The success rate of the anomaly detection method on the C&W attack proves the effectiveness further, as the model has detected less C&W samples than the simBA method. With a higher

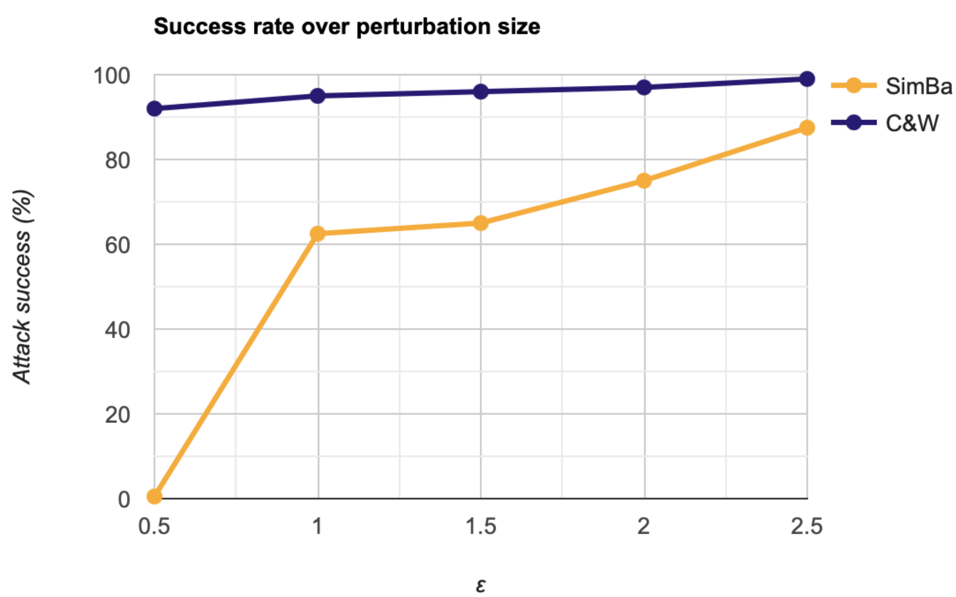


Figure 5: Success Rate of Attacks and Anomaly Detections

δ, the success rates of attacks tend to be higher but cause more visual distortions. The C&W generally does not need a high δ to be effective, although simBA requires a factor of 1.0+ to start becoming successful in fooling the model. Another comparison that can be made between the two models is that the C&W, while more effective, requires much more time than the simBA attack to deploy on average.

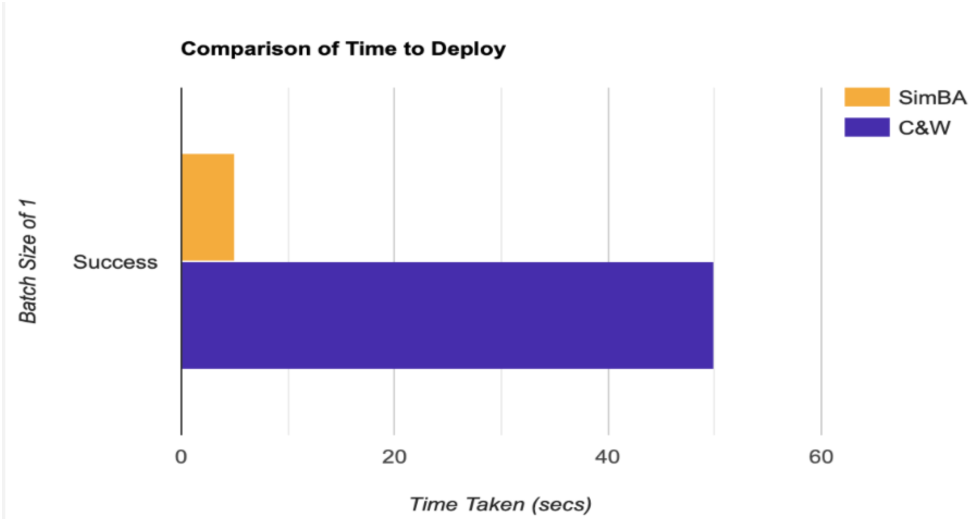


Figure 6: Time taken to deploy the attack

7 Conclusion

As AI systems become increasingly integrated into society, it is very important that security considerations are taken when deploying and creating AI systems. Models often influence critical decisions, and vulnerabilities that exploit these models can have potentially devastating consequences in quality of life, finances, etc. The example adversarial attacks of C&W and simBA showcase the generalizability and adaptability of these attacks across different use cases. These attacks can have devastating consequences on industries and organizations that rely on computer vision or image classification. Without strong security measures and further research into the possibilities of attacks on different types of AI models, the risk of model exploitation could harm societal trust in AI technology. Thus, ensuring the security of AI is essential to safeguard against manipulation, and uphold the reliability of automated decision-making.

A Appendix A: Model Creation

The following code document details the creation of an image classification model, originally imported from my senior capstone project and adapted to be utilized for MRI imaging purposes.

```
import torch
import torchvision.transforms as transforms
from torch.utils.data import Dataset, DataLoader
import os
from torch import nn
import torch.nn.functional as F
from torchvision import models
import matplotlib.pyplot as plt
from PIL import Image
import numpy as np
from tqdm import tqdm

class CustomDataset(Dataset):
    def __init__(self, root_dir, transform=None):
        self.root_dir = root_dir
        self.transform = transform
        self.classes = ['glioma', 'meningioma', 'notumor', 'pituitary']
        self.image_paths = []
        self.labels = []

        for idx, class_name in enumerate(self.classes):
            class_dir = os.path.join(root_dir, class_name)
            for image_name in os.listdir(class_dir):
                if image_name.endswith(".jpg") or image_name.endswith(".png"):
                    self.image_paths.append(os.path.join(class_dir, image_name))
                    self.labels.append(idx)

    def __len__(self):
        return len(self.image_paths)

    def __getitem__(self, idx):
        img_name = self.image_paths[idx]
        image = Image.open(img_name).convert('RGB')
        label = self.labels[idx]

        if self.transform:
            image = self.transform(image)

        return image, label

resize_transform = transforms.Compose([
```

```python
    transforms.Resize((64, 64)),
    transforms.ToTensor()
])

class_names = ['glioma', 'meningioma', 'notumor', 'pituitary']
class_id_to_name = {i: class_name for i, class_name in enumerate(
    class_names)}
class_to_idx = {class_name: i for i, class_name in enumerate(
    class_names)}

class FineTunedDenseNet(nn.Module):
    def __init__(self, pretrained_model, num_classes=4):
        super(FineTunedDenseNet, self).__init__()
        self.features = pretrained_model.features
        self.classifier = nn.Linear(pretrained_model.classifier.
    in_features, num_classes)

    def forward(self, x):
        x = self.features(x)
        x = F.adaptive_avg_pool2d(x, (1, 1)).view(x.size(0), -1)
        x = self.classifier(x)
        return x

    def train_model(self, model_name, input_size, epochs,
    learning_rate, train_loader, test_loader, device):
        criterion = nn.CrossEntropyLoss()
        optimizer = torch.optim.Adam(self.parameters(), lr=
    learning_rate)
        scheduler = torch.optim.lr_scheduler.StepLR(optimizer,
    step_size=5, gamma=0.1)

        best_accuracy = 0
        train_loss_values = []
        test_accuracy_values = []

        for epoch in range(epochs):
            self.train()
            running_loss = 0.0
            correct = 0
            total = 0
            progress_bar = tqdm(train_loader, desc=f"Epoch {epoch+1}/{
    epochs}", leave=False, ncols=100)

            for i, (images, labels) in enumerate(progress_bar):
                images, labels = images.to(device), labels.to(device)

                optimizer.zero_grad()
                outputs = self(images)
```

```
                loss = criterion(outputs, labels)
                loss.backward()
                optimizer.step()

                running_loss += loss.item()
                _, predicted = torch.max(outputs, 1)
                total += labels.size(0)
                correct += (predicted == labels).sum().item()

                progress_bar.set_postfix(loss=running_loss / (i + 1), accuracy=100 * correct / total)

            test_accuracy = 100 * correct / total
            if test_accuracy > best_accuracy:
                best_accuracy = test_accuracy
                torch.save(self.state_dict(), "best_densenet_model.pth")

            scheduler.step()
            train_loss_values.append(running_loss / len(train_loader))
            test_accuracy_values.append(test_accuracy)

        self.plot_loss(model_name, input_size, learning_rate, train_loss_values, test_accuracy_values)

    def plot_loss(self, model_name, input_size, learning_rate, train_loss_values, test_accuracy_values):
        plt.figure(figsize=(10, 5))
        plt.plot(train_loss_values, label="Training Loss", color='blue')
        plt.xlabel("Training Steps")
        plt.ylabel("Loss")
        plt.title("Training Loss Over Time")
        plt.grid(True)
        plt.legend()
        plt.savefig(f"training_loss_{model_name}_{input_size}_lr{learning_rate}.png")
        plt.show()

        plt.figure(figsize=(10, 5))
        plt.plot(test_accuracy_values, label="Test Accuracy", color='orange')
        plt.xlabel("Steps")
        plt.ylabel("Accuracy (%)")
        plt.title("Test Accuracy Over Time")
        plt.grid(True)
        plt.legend()
```

```
        plt.savefig(f"test_accuracy_{model_name}_{input_size}_lr{learning_rate}.png")
        plt.show()

    def save(self, model_file):
        torch.save(self.state_dict(), model_file)

if __name__ == "__main__":
    root_dir = ’/Users/arthe/honors/brain/training’
    model_name = ’FineTunedDenseNet’
    batch_size = 32
    learning_rate = 0.0003
    epochs = 10
    input_size = 64

    image_transforms = transforms.Compose([
        transforms.RandomResizedCrop(input_size),
        transforms.RandomHorizontalFlip(),
        transforms.ColorJitter(brightness=0.2, contrast=0.2, saturation=0.2, hue=0.2),
        transforms.ToTensor(),
        transforms.Normalize(mean=[0.485, 0.456, 0.406], std=[0.229, 0.224, 0.225]),
    ])

    full_dataset = CustomDataset(root_dir, transform=image_transforms)
    train_size = int(0.8 * len(full_dataset))
    test_size = len(full_dataset) - train_size
    train_dataset, test_dataset = torch.utils.data.random_split(full_dataset, [train_size, test_size])
    train_loader = DataLoader(train_dataset, batch_size=batch_size, shuffle=True, num_workers=4)
    test_loader = DataLoader(test_dataset, batch_size=batch_size, shuffle=False, num_workers=4)

    print(f"Training {model_name}")
    device = torch.device(’cuda’ if torch.cuda.is_available() else ’cpu’)
    print(f"Using device {device}")

    pretrained_densenet = models.densenet121(pretrained=True)
    model = FineTunedDenseNet(pretrained_model=pretrained_densenet, num_classes=4)
    model.to(device)

    model.train_model(model_name, input_size, epochs, learning_rate, train_loader, test_loader, device=device)
```

```
    model.to('cpu')
    model_path = f"{model_name}_{input_size}_lr{learning_rate}.pt"
    torch.save(model.state_dict(), model_path)
    print(f"Model saved at {model_path}")
```

Listing 1: train.py

B Appendix B: Attack Implementation

This section details the Python implementation of the C&W[5] attack and the simBA[6] attack, originally sourced from GitHub and then modified to suit the image classification model being used in this project. The first document is my implementation of the C&W attack, with additional functions to save adversarial samples and display the original classification and modded image for viewing purposes. The next lines of code is my simBA implementation, with similar functions to save adversarial samples and display the difference between the original image and the modded image.

```
import os
import torch
import time
import torch.nn.functional as F
import torch.optim as optim
from torchvision import models
from torch import nn
from torchvision import transforms
import matplotlib.pyplot as plt
from PIL import Image

from train import class_names, class_id_to_name, class_to_idx,
   CustomDataset
from train import FineTunedDenseNet

model_file = "/Users/arthe/honors/FineTunedDenseNet_64_lr0.0003.pt"

script_dir = os.path.dirname(os.path.abspath(__file__))
adversarial_save_dir = os.path.join(script_dir, "adversarial_images1")
os.makedirs(adversarial_save_dir, exist_ok=True)

resize_transform = transforms.Resize((224, 224), interpolation=
   transforms.InterpolationMode.BILINEAR)

full_transform = transforms.Compose([
    resize_transform,
    transforms.ToTensor()
])

device = torch.device('cuda' if torch.cuda.is_available() else 'cpu')
pretrained_densenet = models.densenet121(pretrained=True)
```

```

model = FineTunedDenseNet(pretrained_model=pretrained_densenet, num_classes=len(class_names))
model.load_state_dict(torch.load(model_file, map_location=device))
model.to(device)
model.eval()

def cw_l2_attack(model, images, labels, target_labels=None, targeted=False, c=2.5, kappa=0, max_iter=1000, learning_rate=0.01):
    """
    CW-L2 Attack to generate adversarial examples on full-resolution images.
    Args:
        model (nn.Module): Trained model to attack.
        images (torch.Tensor): Batch of full-resolution images to attack.
        labels (torch.Tensor): True labels for the images.
        target_labels (torch.Tensor, optional): Target labels for the attack. Required if targeted=True.
        targeted (bool): Whether the attack is targeted or untargeted.
        c (float): Regularization parameter to control attack strength.
        kappa (float): Confidence parameter for the attack.
        max_iter (int): Maximum number of optimization iterations.
        learning_rate (float): Learning rate for the optimizer.
    Returns:
        torch.Tensor: Adversarial examples in full resolution.
    """

    images = images.to(device)
    labels = labels.to(device)

    if targeted:
        if target_labels is None:
            raise ValueError("Target labels must be provided for targeted attacks.")
        target_labels = target_labels.to(device)

    def f(x):
        resized_x = resize_transform(x)
        outputs = model(resized_x)

        if targeted:
            one_hot_labels = torch.eye(outputs.shape[1], device=device)[target_labels]
            i, _ = torch.max((1 - one_hot_labels) * outputs, dim=1)
            j = torch.masked_select(outputs, one_hot_labels.bool())
            return torch.clamp(i - j, min=-kappa)
        else:
```

```
            one_hot_labels = torch.eye(outputs.shape[1], device=device)[labels]
            i, _ = torch.max((1 - one_hot_labels) * outputs, dim=1)
            j = torch.masked_select(outputs, one_hot_labels.bool())
            return torch.clamp(j - i, min=-kappa)

    w = torch.zeros_like(images, requires_grad=True).to(device)
    optimizer = optim.Adam([w], lr=learning_rate)
    prev = 1e10

    for step in range(max_iter):
        a = 0.5 * (torch.tanh(w) + 1)

        loss1 = F.mse_loss(a, images, reduction='sum')
        loss2 = torch.sum(c * f(a))
        cost = loss1 + loss2

        optimizer.zero_grad()
        cost.backward()
        optimizer.step()

        if step % (max_iter // 10) == 0:
            if cost > prev:
                print("Attack Stopped due to CONVERGENCE....")
                return a
            prev = cost

        if (step + 1) % 100 == 0 or step == 0:
            print(f"- Learning Progress: {(step + 1) / max_iter * 100:.2f}%", end="\r")

    attack_images = 0.5 * (torch.tanh(w) + 1)
    return attack_images

def save_image(image, true_label, predicted_label, original_size, index, prefix="adversarial_example"):

    image = transforms.ToPILImage()(image.cpu().detach())

    image_resized = image.resize(original_size, Image.LANCZOS)

    os.makedirs(adversarial_save_dir, exist_ok=True)

    file_name = os.path.join(
        adversarial_save_dir,
        f"{prefix}_{index}_true_{true_label}_pred_{predicted_label}.png"
    )
```

```
    image_resized.save(file_name)

def display_images(original_image, adversarial_image, true_label,
   predicted_label):

    fig, axes = plt.subplots(1, 2, figsize=(12, 6))

    axes[0].imshow(original_image.permute(1, 2, 0).cpu().detach().numpy
   ())
    axes[0].set_title(f"Original: {true_label}")
    axes[0].axis('off')

    axes[1].imshow(adversarial_image.permute(1, 2, 0).cpu().detach().
   numpy())
    axes[1].set_title(f"Adversarial: {predicted_label}")
    axes[1].axis('off')

    plt.show()

if __name__ == "__main__":
    import argparse

    targeted = False
    target_class = "glioma"
    c = 20
    kappa = 0
    learning_rate = 0.01
    max_iter = 1000

    data_folder = "/Users/arthe/honors/brain/training"
    full_dataset = CustomDataset(data_folder, transform=full_transform)
    train_size = int(0.8 * len(full_dataset))
    test_size = len(full_dataset) - train_size
    train_dataset, test_dataset = torch.utils.data.random_split(
   full_dataset, [train_size, test_size])

    batch_size = 1
    test_loader = torch.utils.data.DataLoader(test_dataset, batch_size=
   batch_size, shuffle=False)

    attack_accuracy = 0
    correct = 0
    total = 0

    for images, labels in test_loader:
        images, labels = images.to(device), labels.to(device)

```

```
        original_size = images.shape[2], images.shape[3]

        start_time = time.time()

        if targeted:
            if target_class is None:
                raise ValueError("Target class must be specified for targeted attacks.")
            try:
                target_class_idx = class_to_idx[target_class]
            except KeyError:
                raise ValueError(f"Target class '{target_class}' not found in class_to_idx.")
            target_labels = torch.full_like(labels, target_class_idx)
            adversarial_images = cw_l2_attack(
                model, images, labels, target_labels=target_labels, targeted=True, c=c,
                kappa=kappa, learning_rate=learning_rate, max_iter=max_iter
            )
        else:
            adversarial_images = cw_l2_attack(
                model, images, labels, targeted=False, c=c,
                kappa=kappa, learning_rate=learning_rate, max_iter=max_iter
            )

        end_time = time.time()
        attack_time = end_time - start_time
        print(f"Time taken for attack: {attack_time:.2f} seconds")

        adversarial_images_resized = resize_transform(adversarial_images)

        outputs = model(adversarial_images_resized)
        _, predicted = torch.max(outputs, 1)

        if targeted:
            batch_correct = (predicted == target_labels).sum().item()
        else:
            batch_correct = (predicted != labels).sum().item()

        batch_total = labels.size(0)
        attack_accuracy += 100 * batch_correct / batch_total
        correct += batch_correct
        total += batch_total

```

```
        print(f"Attack Accuracy on the batch: {100 * batch_correct / batch_total:.2f}%")

        for i in range(len(adversarial_images)):
            true_label = class_names[labels[i].item()]
            predicted_label = class_names[predicted[i].item()]

            save_image(adversarial_images[i], true_label, predicted_label, original_size, i)

            display_images(images[i], adversarial_images[i], true_label, predicted_label)

    overall_attack_accuracy = attack_accuracy / len(test_loader)
    print(f"\nOverall Attack Accuracy: {overall_attack_accuracy:.2f}%")
```

Listing 2: attack.py

```
import torch
import utils
import torch.nn.functional as F
import torchvision.models as models
from torchvision.models import densenet121
import torchvision.transforms as transforms
from torchvision.utils import make_grid
import matplotlib.pyplot as plt
import time
from torchvision.datasets import ImageFolder
from torch.utils.data import DataLoader


device = torch.device('cuda' if torch.cuda.is_available() else 'cpu')

class SimBA:

    def __init__(self, model, dataset, image_size):
        self.model = model.to(device)
        self.dataset = dataset
        self.image_size = image_size
        self.model.eval()

    def expand_vector(self, x, size):
        batch_size = x.size(0)
        x = x.view(-1, 3, size, size)
        z = torch.zeros(batch_size, 3, self.image_size, self.image_size)
        z[:, :, :size, :size] = x
        return z

```

```
    def normalize(self, x):
        return utils.apply_normalization(x.to(device), self.dataset)


    def get_probs(self, x, y):
        output = self.model(self.normalize(x)).cpu()
        probs = torch.index_select(F.softmax(output, dim=-1).data, 1, y)
        return torch.diag(probs)

    def get_preds(self, x):
        output = self.model(self.normalize(x)).cpu()
        _, preds = output.data.max(1)
        return preds

    def simba_single(self, x, y, num_iters=10000, epsilon=2.5, targeted=False):
        n_dims = x.view(1, -1).size(1)
        perm = torch.randperm(n_dims)
        last_prob = self.get_probs(x, y)
        for i in range(num_iters):
            diff = torch.zeros(n_dims)
            diff[perm[i]] = epsilon
            left_prob = self.get_probs((x - diff.view(x.size())).clamp(0, 1), y)
            if targeted != (left_prob < last_prob):
                x = (x - diff.view(x.size())).clamp(0, 1)
                last_prob = left_prob
            else:
                right_prob = self.get_probs((x + diff.view(x.size())).clamp(0, 1), y)
                if targeted != (right_prob < last_prob):
                    x = (x + diff.view(x.size())).clamp(0, 1)
                    last_prob = right_prob
            if i % 10 == 0:
                print(last_prob)
        return x.squeeze()


    def simba_batch(self, images_batch, labels_batch, max_iters, freq_dims, stride, epsilon, linf_bound=0.0,
                    order=’rand’, targeted=False, pixel_attack=False, log_every=1):
        batch_size = images_batch.size(0)
        image_size = images_batch.size(2)
        assert self.image_size == image_size
        if order == ’rand’:
            indices = torch.randperm(3 * freq_dims * freq_dims)[:
```

```
max_iters]
        elif order == ’diag’:
            indices = utils.diagonal_order(image_size, 3)[:max_iters]
        elif order == ’strided’:
            indices = utils.block_order(image_size, 3, initial_size=freq_dims, stride=stride)[:max_iters]
        else:
            indices = utils.block_order(image_size, 3)[:max_iters]
        if order == ’rand’:
            expand_dims = freq_dims
        else:
            expand_dims = image_size
        n_dims = 3 * expand_dims * expand_dims
        x = torch.zeros(batch_size, n_dims)
        probs = torch.zeros(batch_size, max_iters)
        succs = torch.zeros(batch_size, max_iters)
        queries = torch.zeros(batch_size, max_iters)
        l2_norms = torch.zeros(batch_size, max_iters)
        linf_norms = torch.zeros(batch_size, max_iters)
        prev_probs = self.get_probs(images_batch, labels_batch)
        preds = self.get_preds(images_batch)
        if pixel_attack:
            trans = lambda z: z
        else:
            trans = lambda z: utils.block_idct(z, block_size=image_size, linf_bound=linf_bound)
        remaining_indices = torch.arange(0, batch_size).long()
        for k in range(max_iters):
            dim = indices[k]
            expanded = (images_batch[remaining_indices] + trans(self.expand_vector(x[remaining_indices], expand_dims))).clamp(0, 1)
            perturbation = trans(self.expand_vector(x, expand_dims))
            l2_norms[:, k] = perturbation.view(batch_size, -1).norm(2, 1)
            linf_norms[:, k] = perturbation.view(batch_size, -1).abs().max(1)[0]
            preds_next = self.get_preds(expanded)
            preds[remaining_indices] = preds_next
            if targeted:
                remaining = preds.ne(labels_batch)
            else:
                remaining = preds.eq(labels_batch)
            if remaining.sum() == 0:
                adv = (images_batch + trans(self.expand_vector(x, expand_dims))).clamp(0, 1)
                probs_k = self.get_probs(adv, labels_batch)
                probs[:, k:] = probs_k.unsqueeze(1).repeat(1, max_iters - k)
```

```
            succs[:, k:] = torch.ones(batch_size, max_iters - k)
            queries[:, k:] = torch.zeros(batch_size, max_iters - k)
            break
        remaining_indices = torch.arange(0, batch_size)[remaining].long()
        if k > 0:
            succs[:, k] = ~remaining
        diff = torch.zeros(remaining.sum(), n_dims)
        diff[:, dim] = epsilon
        left_vec = x[remaining_indices] - diff
        right_vec = x[remaining_indices] + diff
        adv = (images_batch[remaining_indices] + trans(self.expand_vector(left_vec, expand_dims))).clamp(0, 1)
        left_probs = self.get_probs(adv, labels_batch[remaining_indices])
        queries_k = torch.zeros(batch_size)
        queries_k[remaining_indices] += 1
        if targeted:
            improved = left_probs.gt(prev_probs[remaining_indices])
        else:
            improved = left_probs.lt(prev_probs[remaining_indices])
        if improved.sum() < remaining_indices.size(0):
            queries_k[remaining_indices[~improved]] += 1
        adv = (images_batch[remaining_indices] + trans(self.expand_vector(right_vec, expand_dims))).clamp(0, 1)
        right_probs = self.get_probs(adv, labels_batch[remaining_indices])
        if targeted:
            right_improved = right_probs.gt(torch.max(prev_probs[remaining_indices], left_probs))
        else:
            right_improved = right_probs.lt(torch.min(prev_probs[remaining_indices], left_probs))
        probs_k = prev_probs.clone()
        if improved.sum() > 0:
            left_indices = remaining_indices[improved]
            left_mask_remaining = improved.unsqueeze(1).repeat(1, n_dims)
            x[left_indices] = left_vec[left_mask_remaining].view(-1, n_dims)
            probs_k[left_indices] = left_probs[improved]
        if right_improved.sum() > 0:
            right_indices = remaining_indices[right_improved]
            right_mask_remaining = right_improved.unsqueeze(1).repeat(1, n_dims)
            x[right_indices] = right_vec[right_mask_remaining].view(-1, n_dims)
            probs_k[right_indices] = right_probs[right_improved]
```

```
            probs[:, k] = probs_k
            queries[:, k] = queries_k
            prev_probs = probs[:, k]
            if (k + 1) % log_every == 0 or k == max_iters - 1:
                print('Iteration %d: queries = %.4f, prob = %.4f, remaining = %.4f' % (
                        k + 1, queries.sum(1).mean(), probs[:, k].mean(), remaining.float().mean()))
        expanded = (images_batch + trans(self.expand_vector(x, expand_dims))).clamp(0, 1)
        preds = self.get_preds(expanded)
        if targeted:
            remaining = preds.ne(labels_batch)
        else:
            remaining = preds.eq(labels_batch)
        succs[:, max_iters-1] = ~remaining
        return expanded, probs, succs, queries, l2_norms, linf_norms
transform = transforms.Compose([
    transforms.Resize((224, 224)),
    transforms.ToTensor(),
])


dataset_path = '/Users/arthe/honors/brain/training'
dataset = ImageFolder(root=dataset_path, transform=transform)
dataloader = DataLoader(dataset, batch_size=2, shuffle=True)

model = densenet121(pretrained=True)
model.classifier = torch.nn.Linear(in_features=1024, out_features=4)


model.load_state_dict(torch.load('/Users/arthe/honors/FineTunedDenseNet_64_lr0.0003.pt', map_location=device))
model.eval()

simba = SimBA(model=model, dataset='imagenet', image_size=224)#was 224

def simba_single_attack(simba, dataloader):
    images_batch, labels_batch = next(iter(dataloader))
    image = images_batch[0:1].to(device)
    label = labels_batch[0:1].to(device)

    class_name = dataloader.dataset.classes[label.item()]

    start_time = time.time()
    adv_image = simba.simba_single(image, label, num_iters=100, epsilon=0.2, targeted=False)
    elapsed_time = time.time() - start_time
```

```
    adv_pred = simba.get_preds(adv_image.unsqueeze(0))
    adv_class_name = dataloader.dataset.classes[adv_pred.item()]

    plt.figure(figsize=(10, 5))

    plt.subplot(1, 2, 1)
    plt.imshow(image[0].permute(1, 2, 0).cpu().numpy())
    plt.title(f"Original: {class_name} ({label.item()})")
    plt.axis('off')

    plt.subplot(1, 2, 2)
    plt.imshow(adv_image.permute(1, 2, 0).cpu().numpy())
    plt.title(f"Adversarial: {adv_class_name} ({adv_pred.item()})")
    plt.axis('off')

    plt.suptitle(f"SimBA Single Attack | Time: {elapsed_time:.2f}s")
    plt.show()



def simba_full_dataset_attack(simba, dataloader, max_iters=200, epsilon=0.2, freq_dims=21, stride=1, targeted=False):
    total_images = 0
    total_fooled = 0
    total_time = 0.0

    for images_batch, labels_batch in dataloader:
        images_batch = images_batch.to(device)
        labels_batch = labels_batch.to(device)

        start_time = time.time()
        adv_images, probs, succs, queries, l2_norms, linf_norms = simba.simba_batch(
            images_batch=images_batch,
            labels_batch=labels_batch,
            max_iters=max_iters,
            freq_dims=freq_dims,
            stride=stride,
            epsilon=epsilon,
            linf_bound=0.0,
            order='rand',
            targeted=targeted,
            pixel_attack=False,
            log_every=50
        )
        elapsed_time = time.time() - start_time
        total_time += elapsed_time
```

```

        adv_preds = simba.get_preds(adv_images)
        fooled = (adv_preds != labels_batch).sum().item()
        total_fooled += fooled
        total_images += len(labels_batch)

        print(f"Batch: {total_images}/{len(dataloader.dataset)} - Fooled: {fooled}/{len(labels_batch)} - Time: {elapsed_time:.2f}s")

    success_rate = (total_fooled / total_images) * 100
    avg_time = total_time / len(dataloader)

    print(f"\nSimBA Attack Completed")
    print(f"Total Images: {total_images}")
    print(f"Total Fooled: {total_fooled}")
    print(f"Attack Success Rate: {success_rate:.2f}%")
    print(f"Average Time per Batch: {avg_time:.2f}s")

    return success_rate



def evaluate_adversarial_attacks(model, dataloader, targeted=False, max_iters=200, freq_dims=21, stride=1, epsilon=2.5):

    model.eval()
    attack_accuracy = 0
    correct = 0
    total = 0

    for images, labels in dataloader:
        images, labels = images.to(device), labels.to(device)

        start_time = time.time()

        if targeted:
            if target_class is None:
                raise ValueError("Target class must be specified for targeted attacks.")
            try:
                target_class_idx = dataloader.dataset.class_to_idx[target_class]
            except KeyError:
                raise ValueError(f"Target class '{target_class}' not found in class_to_idx.")
            target_labels = torch.full_like(labels, target_class_idx)
            adv_images, *_ = simba.simba_batch(
                images_batch=images,
```

```
                labels_batch=labels,
                max_iters=max_iters,
                freq_dims=freq_dims,
                stride=stride,
                epsilon=epsilon,
                targeted=True
            )
        else:
            adv_images, *_ = simba.simba_batch(
                images_batch=images,
                labels_batch=labels,
                max_iters=max_iters,
                freq_dims=freq_dims,
                stride=stride,
                epsilon=epsilon,
                targeted=False
            )

        end_time = time.time()
        attack_time = end_time - start_time
        print(f"Time taken for attack: {attack_time:.2f} seconds")

        outputs = model(adv_images)
        _, predicted = torch.max(outputs, 1)

        if targeted:
            batch_correct = (predicted == target_labels).sum().item()
        else:
            batch_correct = (predicted != labels).sum().item()

        batch_total = labels.size(0)
        attack_accuracy += 100 * batch_correct / batch_total
        correct += batch_correct
        total += batch_total

        print(f"Attack Accuracy on the batch: {100 * batch_correct /
    batch_total:.2f}%")

    overall_attack_accuracy = attack_accuracy / len(dataloader)
    print(f"\nOverall Attack Accuracy: {overall_attack_accuracy:.2f}%")

    return overall_attack_accuracy

#simba_single_attack(simba, dataloader)

#success_rate = simba_full_dataset_attack(simba, dataloader, max_iters
    =200, epsilon=0.2)
overall_accuracy = evaluate_adversarial_attacks(model, dataloader,
```

```
    targeted=False, max_iters=200)
```

Listing 3: simba.py

C Appendix C: Anomaly Detection

The next document of code is the creation of the anomaly detection model, which utilizes the attack implementation and unmodified data samples to train.

```
import torch
from train import FineTunedDenseNet
from torchvision import transforms, datasets
from torch.utils.data import DataLoader, ConcatDataset, Subset
from PIL import Image
from tqdm import tqdm
import matplotlib.pyplot as plt
from torchvision import models
import os
from sklearn.model_selection import train_test_split

original_data_dir = '/Users/arthe/honors/train'
adversarial_data_dir = '/Users/arthe/honors/adversarial_images'

transform = transforms.Compose([
    transforms.Resize((224, 224)),
    transforms.ToTensor(),
])

original_dataset = datasets.ImageFolder(
    root=original_data_dir,
    transform=transform
)

adversarial_dataset = datasets.ImageFolder(
    root=adversarial_data_dir,
    transform=transform
)

class RelabeledDataset(torch.utils.data.Dataset):
    def __init__(self, dataset, new_label):
        self.dataset = dataset
        self.new_label = new_label

    def __len__(self):
        return len(self.dataset)

    def __getitem__(self, idx):
```

```
        img, _ = self.dataset[idx]
        return img, self.new_label

original_binary = RelabeledDataset(original_dataset, 0)
adversarial_binary = RelabeledDataset(adversarial_dataset, 1)
combined_dataset = ConcatDataset([original_binary, adversarial_binary])

all_data = [(img, label) for img, label in combined_dataset]
labels = [label for _, label in all_data]
train_idx, val_idx = train_test_split(
    list(range(len(all_data))), test_size=0.2, stratify=labels, random_state=42
)
train_dataset = Subset(combined_dataset, train_idx)
val_dataset = Subset(combined_dataset, val_idx)

class AnomalyDenseNet(torch.nn.Module):
    def __init__(self, base_model: FineTunedDenseNet):
        super(AnomalyDenseNet, self).__init__()
        self.features = base_model.features

        self.classifier = torch.nn.Sequential(
            torch.nn.AdaptiveAvgPool2d((1, 1)),
            torch.nn.Flatten(),
            torch.nn.Linear(1024, 1)
        )

    def forward(self, x):
        x = self.features(x)
        out = self.classifier(x)
        return out.squeeze()


    def train_model(self, model_name, input_size, epochs, learning_rate, train_loader, val_loader, device):
        optimizer = torch.optim.Adam(self.parameters(), lr=learning_rate)
        criterion = torch.nn.BCEWithLogitsLoss()

        for epoch in range(epochs):
            self.train()
            running_loss, correct_train, total_train = 0.0, 0, 0
            train_loader_tqdm = tqdm(train_loader, desc=f"Epoch {epoch+1}/{epochs} [Train]", leave=False)

            for inputs, labels in train_loader_tqdm:
                inputs, labels = inputs.to(device), labels.float().to(device)
```

```
            optimizer.zero_grad()
            outputs = self(inputs).view(-1)
            labels = labels.view(-1)
            loss = criterion(outputs, labels)
            loss.backward()
            optimizer.step()

            batch_size = inputs.size(0)
            running_loss += loss.item() * batch_size
            preds = (torch.sigmoid(outputs) > 0.5).float()
            correct_train += (preds == labels).sum().item()
            total_train += batch_size

            train_loader_tqdm.set_postfix({
                'loss': f"{running_loss/total_train:.4f}",
                'acc': f"{correct_train/total_train:.4f}"
            })

        epoch_loss = running_loss / len(train_loader.dataset)
        epoch_acc = correct_train / total_train

        self.eval()
        val_loss, correct_val, total_val = 0.0, 0, 0
        val_loader_tqdm = tqdm(val_loader, desc=f"Epoch {epoch+1}/{epochs} [Val]", leave=False)

        with torch.no_grad():
            for inputs, labels in val_loader_tqdm:
                inputs, labels = inputs.to(device), labels.float().to(device)
                outputs = self(inputs).view(-1)
                loss = criterion(outputs, labels)
                batch_size = inputs.size(0)
                val_loss += loss.item() * batch_size
                preds = (torch.sigmoid(outputs) > 0.5).float()
                correct_val += (preds == labels).sum().item()
                total_val += batch_size

                val_loader_tqdm.set_postfix({
                    'val_loss': f"{val_loss/total_val:.4f}",
                    'val_acc': f"{correct_val/total_val:.4f}"
                })

        val_epoch_loss = val_loss / len(val_loader.dataset)
        val_epoch_acc = correct_val / total_val

        print(f"\nEpoch {epoch+1}/{epochs} Complete")
```

```
            print(f"Train Loss: {epoch_loss:.4f}, Train Acc: {epoch_acc:.4f}")
            print(f"Val   Loss: {val_epoch_loss:.4f}, Val   Acc: {val_epoch_acc:.4f}")
            print("---------------------------------------------------")

if __name__ == "__main__":
    batch_size = 32
    train_loader = DataLoader(train_dataset, batch_size=batch_size, shuffle=True)
    val_loader = DataLoader(val_dataset, batch_size=batch_size)

    device = torch.device("cuda" if torch.cuda.is_available() else "cpu")

    original_densenet = models.densenet121(pretrained=True)
    base_model = FineTunedDenseNet(pretrained_model=original_densenet, num_classes=4)
    base_model.load_state_dict(torch.load("/Users/arthe/honors/best_densenet_model.pth", map_location=device))
    base_model = base_model.to(device)

    model = AnomalyDenseNet(base_model).to(device)

    model.train_model("anomaly_model", batch_size, 2, 0.001, train_loader, val_loader, device=device)

    model.to('cpu')
    torch.save(model.state_dict(), "anomaly_detection_model.pth")

    print("\n Checking prediction scores on normal (clean) images...")
    model.eval()
    for i in range(5):
        img, _ = original_binary[i]
        input_tensor = img.unsqueeze(0)
        with torch.no_grad():
            raw_output = model(input_tensor)
            score = torch.sigmoid(raw_output).item()
            print(f"[Clean] Image {i+1} Score: {score:.4f} (Expected: ~0)")

    print("\nChecking prediction scores on adversarial images...")
    for i in range(5):
        img, _ = adversarial_binary[i]
        input_tensor = img.unsqueeze(0)
        with torch.no_grad():
            raw_output = model(input_tensor)
            score = torch.sigmoid(raw_output).item()
```

```
            print(f"[Adversarial] Image {i+1} Score: {score:.4f} (Expected: ~1)")
```

Listing 4: anomaly.py

D Appendix D: Web Application Deployment

This section of code entails the creation of a local web application to test the original image classification model, attack implementations, and anomaly detection model. A Gemini API was utilized to create a chatbot function, demonstrating the use of AI in improving patient education.

```
from flask import Flask, request, render_template, jsonify, send_from_directory
from werkzeug.utils import secure_filename
import os
import time
import torchvision
from model_handler import MRI_Classifier
import logging
import traceback
import io
from PIL import Image
import json
import sys
import google.generativeai as genai
from dotenv import load_dotenv
load_dotenv()
genai.configure(api_key=os.getenv("GEMINI_API_KEY"))

app = Flask(__name__)
app.config['UPLOAD_FOLDER'] = 'uploads'
app.config['ALLOWED_EXTENSIONS'] = {'png', 'jpg', 'jpeg'}
app.config['MAX_CONTENT_LENGTH'] = 16 * 1024 * 1024

logging.basicConfig(level=logging.INFO)
logger = logging.getLogger(__name__)

model = MRI_Classifier('best_densenet_model.pth')

PROJECT_ID = "mriclassifier"
REGION = "us-central1"
gemini_model = genai.GenerativeModel("gemini-1.5-flash")
gemini_chat = gemini_model.start_chat(history=[])



def allowed_file(filename):
```

```
    return ’.’ in filename and \
           filename.rsplit(’.’, 1)[1].lower() in app.config[’ALLOWED_EXTENSIONS’]

def get_gemini_diagnostic(image_bytes, classification_label, is_adversarial, anomaly_score):
    try:
        genai.configure(api_key=os.getenv("GEMINI_API_KEY"))
        image = Image.open(io.BytesIO(image_bytes)).convert("RGB")

        prompt = [
            image,
            f"This is an MRI brain tumor image. The classification result is: {classification_label}.",
            f"Adversarial status: {’Potentially adversarial’ if is_adversarial else ’Likely not adversarial’}.",
            f"Anomaly score (if adversarial): {anomaly_score:.4f}.",
            "Provide a concise diagnostic explanation in JSON format with ONLY this exact structure:",
            "{",
            ’"key_findings": "...",’,
            ’"potential_implications": "...",’,
            ’"recommended_next_steps": "..."’,
            "}",
            "Return ONLY the JSON object with no additional text, explanations, or markdown formatting."
        ]

        response = gemini_chat.send_message(prompt)

        try:
            return json.loads(response.text)
        except json.JSONDecodeError:
            start_idx = response.text.find(’{’)
            end_idx = response.text.rfind(’}’) + 1
            if start_idx != -1 and end_idx != -1:
                json_str = response.text[start_idx:end_idx]
                return json.loads(json_str)
            else:
                return {
                    "error": "Could not extract JSON from response",
                    "raw_response": response.text
                }
    except Exception as e:
        logger.error(f"Error in Gemini diagnostic: {str(e)}")
        return {
            "error": "Error generating diagnostic",
            "exception": str(e)
```

```
        }

@app.route(’/’)
def home():
    return render_template(’index.html’)
@app.route(’/chat’, methods=[’POST’])
def chat_with_gemini():
    user_message = request.json.get(’message’)
    try:
        response = gemini_chat.send_message(user_message)
        return jsonify({’response’: response.text})
    except Exception as e:
        logger.error("Error in Gemini chat:")
        logger.error(traceback.format_exc())
        return jsonify({’error’: ’Unable to get response from Gemini’}), 500

@app.route(’/predict’, methods=[’POST’])
def predict_upload():
    if ’file’ not in request.files:
        return jsonify({’error’: ’No file uploaded’}), 400

    file = request.files[’file’]

    if file.filename == ’’:
        return jsonify({’error’: ’No selected file’}), 400

    if not allowed_file(file.filename):
        return jsonify({’error’: ’File type not allowed’}), 400

    try:
        os.makedirs(app.config[’UPLOAD_FOLDER’], exist_ok=True)
        timestamp = str(int(time.time()))
        filename = secure_filename(f"{timestamp}_{file.filename}")
        filepath = os.path.join(app.config[’UPLOAD_FOLDER’], filename)
        file.save(filepath)

        result = model.predict(filepath)
        print(f"[DEBUG] Flask response anomaly status: {result[’anomaly’]}", file=sys.stdout, flush=True)

        display_path = os.path.join(app.config[’UPLOAD_FOLDER’], f"display_{filename}")
        torchvision.utils.save_image(result[’display_image’], display_path)

        probabilities = {model.class_names[i]: f"{prob*100:.2f}%"
```

```
                        for i, prob in enumerate(result[’probabilities’])}

        with open(filepath, "rb") as image_file:
            image_bytes = image_file.read()

        diagnostic_info = get_gemini_diagnostic(
            image_bytes,
            result[’class’],
            result[’anomaly’][’is_anomalous’],
            result[’anomaly’][’anomaly_score’]
        )

        return jsonify({
            ’status’: ’success’,
            ’prediction’: result[’class’],
            ’confidence’: f"{result[’confidence’]*100:.2f}%",
            ’probabilities’: probabilities,
            ’display_image’: f"/display/{filename}",
            ’is_adversarial’: result[’anomaly’][’is_anomalous’],
            ’anomaly_score’: float(result[’anomaly’][’anomaly_score’]),
            ’diagnostic_info’: diagnostic_info
        })

    except Exception as e:
        logger.error(f"Error: {str(e)}")
        logger.error(traceback.format_exc())
        return jsonify({’error’: str(e)}), 500
    finally:
        if ’filepath’ in locals() and os.path.exists(filepath):
            os.remove(filepath)

@app.route(’/display/<filename>’)
def serve_display_image(filename):
    return send_from_directory(app.config[’UPLOAD_FOLDER’], f"display_{filename}")

if __name__ == ’__main__’:
    os.makedirs(app.config[’UPLOAD_FOLDER’], exist_ok=True)
    app.run(host=’0.0.0.0’, port=5004, debug=True)
```

Listing 5: app.py

```
<!DOCTYPE html>
<html lang="en">
<head>
    <meta charset="UTF-8">
    <meta name="viewport" content="width=device-width, initial-scale=1.0">
```

```
    <title>MRI Imaging Classifier</title>
    <link href="https://cdn.jsdelivr.net/npm/bootstrap@5.1.3/dist/css/bootstrap.min.css" rel="stylesheet">
    <link rel="stylesheet" href="https://cdn.jsdelivr.net/npm/bootstrap-icons@1.11.3/font/bootstrap-icons.min.css">
    <script src="https://cdn.jsdelivr.net/npm/bootstrap@5.1.3/dist/js/bootstrap.bundle.min.js"></script>
    <script src="https://cdn.jsdelivr.net/npm/marked/marked.min.js"></script>

    <style>
        .adversarial-alert {
            animation: pulse 2s infinite;
        }
        @keyframes pulse {
            0% { box-shadow: 0 0 0 0 rgba(220, 53, 69, 0.7); }
            70% { box-shadow: 0 0 0 10px rgba(220, 53, 69, 0); }
            100% { box-shadow: 0 0 0 0 rgba(220, 53, 69, 0); }
        }
        .security-meter {
            height: 20px;
            background: linear-gradient(to right, #28a745, #ffc107, #dc3545);
            border-radius: 10px;
            margin-bottom: 15px;
        }
        .security-indicator {
            height: 100%;
            width: 0%;
            background-color: rgba(255, 255, 255, 0.7);
            border-radius: 10px;
            transition: width 1s;
        }
        body {
            background-color: #f8f9fa;
            font-family: ’Segoe UI’, Tahoma, Geneva, Verdana, sans-serif;
        }
        .container {
            max-width: 800px;
        }
        .upload-area {
            border: 2px dashed #6c757d;
            border-radius: 10px;
            padding: 2rem;
            text-align: center;
            cursor: pointer;
            transition: all 0.3s;
```

```
            background-color: white;
            margin-bottom: 2rem;
        }
        .upload-area:hover {
            border-color: #0d6efd;
            background-color: #f0f7ff;
        }
        #previewImage {
            max-width: 100%;
            max-height: 300px;
            border-radius: 8px;
            display: none;
            margin: 0 auto 1.5rem auto;
        }
        .result-card {
            background-color: white;
            border-radius: 10px;
            padding: 1.5rem;
            box-shadow: 0 4px 6px rgba(0, 0, 0, 0.1);
            margin-top: 1.5rem;
            display: none;
        }

        .diagnosis-badge {
            font-size: 1.2rem;
            padding: 0.5rem 1rem;
            border-radius: 5px;
        }
        .diagnostic-card {
            background-color: #e9ecef;
            border-radius: 10px;
            padding: 1.5rem;
            margin-top: 1.5rem;
            box-shadow: 0 2px 4px rgba(0, 0, 0, 0.05);
            display: none;
        }
        .diagnostic-card h5 {
            color: #495057;
            margin-bottom: 1rem;
        }
        .diagnostic-card p {
            color: #6c757d;
            line-height: 1.6;
        }
        .glioma { background-color: #fff3cd; color: #856404; }
        .meningioma { background-color: #d4edda; color: #155724; }
        .notumor { background-color: #cce5ff; color: #004085; }
        .pituitary { background-color: #f8d7da; color: #721c24; }
```

```
    </style>
</head>
<body>
    <div class="container py-5">
        <div class="text-center mb-5">
            <h1 class="display-4">Brain Tumor MRI Classifier</h1>
            <p class="lead">Upload an MRI scan to classify its tumor type!</p>
        </div>

        <div id="uploadArea" class="upload-area">
            <h4><i class="bi bi-cloud-arrow-up"></i> Upload MRI Image</h4>
            <p class="text-muted">Click to browse or drag and drop</p>
            <input type="file" id="fileInput" accept="image/*" class="d-none">
        </div>

        <div class="text-center">
            <img id="previewImage" class="img-fluid">
        </div>

        <button id="predictBtn" class="btn btn-primary btn-lg w-100 py-3" disabled onclick="predict()">
            Classify Image
        </button>
        <div id="result" class="result-card" style="display: none;">
            <h3>Diagnosis: <span id="predictionResult"></span></h3>
            <p>Confidence: <span id="confidenceValue"></span></p>

            <div id="adversarialAlert" class="alert alert-danger" style="display: none;">
                <i class="bi bi-shield-exclamation"></i> Warning: Potential adversarial image detected!
                <div class="mt-2">
                    <div class="security-meter">
                        <div id="securityIndicator" class="security-indicator"></div>
                    </div>
                </div>
            </div>

            <h5 class="mt-4">Probability Distribution:</h5>
            <div id="probabilityBars"></div>
        </div>
        <div id="diagnosticInfo" class="diagnostic-card">
            <h5>Diagnostic Information:</h5>
            <p id="diagnosticText"></p>
```

```
        </div>

        <div id="chat-container" style="margin-top: 30px;">
            <h3>Follow-up Chat with Gemini</h3>
            <div id="chat-log" style="border: 1px solid #ccc; padding: 10px; height: 200px; overflow-y: auto; background: #f9f9f9; border-radius: 8px;"></div>
            <div style="margin-top: 10px;">
              <input type="text" id="chat-input" placeholder="Ask a follow-up question..." style="width: 80%; padding: 8px;" />
              <button onclick="sendChat()" style="padding: 8px 12px;">Send</button>
            </div>
        </div>
    </div>

    <script>
        const uploadArea = document.getElementById('uploadArea');
        const fileInput = document.getElementById('fileInput');
        const previewImage = document.getElementById('previewImage');
        const predictBtn = document.getElementById('predictBtn');
        const resultDiv = document.getElementById('result');

        uploadArea.addEventListener('dragover', (e) => {
            e.preventDefault();
            uploadArea.style.borderColor = '#0d6efd';
            uploadArea.style.backgroundColor = '#f0f7ff';
        });

        uploadArea.addEventListener('dragleave', () => {
            uploadArea.style.borderColor = '#6c757d';
            uploadArea.style.backgroundColor = 'white';
        });

        uploadArea.addEventListener('drop', (e) => {
            e.preventDefault();
            uploadArea.style.borderColor = '#6c757d';
            uploadArea.style.backgroundColor = 'white';
            if (e.dataTransfer.files.length) {
                fileInput.files = e.dataTransfer.files;
                previewFile(fileInput.files[0]);
            }
        });

        uploadArea.addEventListener('click', () => {
            fileInput.click();
        });

```

```
        function previewFile(file) {
            const reader = new FileReader();
            reader.onload = (e) => {
                previewImage.src = e.target.result;
                previewImage.style.display = ’block’;
                predictBtn.disabled = false;
                resultDiv.style.display = ’none’;
            };
            reader.readAsDataURL(file);
        }

        function predict() {
            if (!fileInput.files.length) {
                showAlert(’Please select an image first’, ’warning’);
                return;
            }

            const formData = new FormData();
            formData.append(’file’, fileInput.files[0]);

            predictBtn.disabled = true;
            predictBtn.innerHTML = `
                <span class="spinner-border spinner-border-sm" role="status" aria-hidden="true"></span>
                Processing...
            `;

            resultDiv.style.display = ’none’;

            fetch(’/predict’, {
                method: ’POST’,
                body: formData
            })
            .then(response => {
                if (!response.ok) {
                    throw new Error(`HTTP error! status: ${response.status}`);
                }
                return response.json();
            })
            .then(data => {
                console.log("Probabilities:", data.probabilities);

                displayResults(data);
                data.probabilities = {
                    benign: 0.75,
                    early: 0.1,
                    pre: 0.1,
```

```
                    pro: 0.05
                };
                predictBtn.disabled = false;
                predictBtn.innerHTML = ’Classify Image’;
            })
            .catch(error => {
                console.error(’Error:’, error);
                showAlert(’An error occurred while processing the image’, ’danger’);

                predictBtn.disabled = false;
                predictBtn.innerHTML = ’Classify Image’;
            });
        }

        function showAlert(message, type) {
            const existingAlert = document.getElementById(’dynamicAlert’);
            if (existingAlert) {
                existingAlert.remove();
            }

            const alertDiv = document.createElement(’div’);
            alertDiv.id = ’dynamicAlert’;
            alertDiv.className = ‘alert alert-${type} alert-dismissible fade show‘;
            alertDiv.setAttribute(’role’, ’alert’);
            alertDiv.innerHTML = ‘
                ${message}
                <button type="button" class="btn-close" data-bs-dismiss="alert" aria-label="Close"></button>
            ‘;

            uploadArea.after(alertDiv);

            setTimeout(() => {
                const alert = bootstrap.Alert.getInstance(alertDiv);
                if (alert) {
                    alert.close();
                }
            }, 5000);
        }

        function displayResults(data) {
            console.log("Received data from backend:", data);

            const predictionResult = document.getElementById(’predictionResult’);
```

```
        const advAlert = document.getElementById('adversarialAlert');
        const diagnosticDiv = document.getElementById('diagnosticInfo');
        const diagnosticText = document.getElementById('diagnosticText');
        document.getElementById("predictionResult").textContent = data.prediction;


        predictionResult.textContent = data.prediction;
        predictionResult.className = 'diagnosis-badge ' + data.prediction.toLowerCase();
        document.getElementById('confidenceValue').textContent = data.confidence;

        const probabilityBars = document.getElementById('probabilityBars');
        probabilityBars.innerHTML = '';

        for (const [className, prob] of Object.entries(data.probabilities)) {
            const barContainer = document.createElement('div');
            barContainer.className = 'mb-3';

            const labelRow = document.createElement('div');
            labelRow.className = 'd-flex justify-content-between mb-1';

            const label = document.createElement('span');
            label.textContent = className;
            label.className = 'text-capitalize fw-medium';

            const percentage = document.createElement('span');
            percentage.textContent = prob;

            labelRow.appendChild(label);
            labelRow.appendChild(percentage);
            barContainer.appendChild(labelRow);

            const progress = document.createElement('div');
            progress.className = 'probability-bar';

            const fill = document.createElement('div');
            fill.className = 'probability-fill';
            fill.style.width = `${Math.min(100, (prob * 100).toFixed(1))}%`; // Cap at 100%

```

```
            progress.appendChild(fill);
            barContainer.appendChild(progress);
            probabilityBars.appendChild(barContainer);
        }
        document.getElementById("result").style.display = "block";
            if (data.is_adversarial) {
                advAlert.style.display = ’block’;
                advAlert.classList.add(’adversarial-alert’);
            } else {
                advAlert.style.display = ’none’;
            }

        if (data.diagnostic_info) {
            if (data.diagnostic_info.error) {
                diagnosticText.innerHTML = `
                    <div class="alert alert-warning">
                        <strong>Note:</strong> ${data.diagnostic_info.error}
                        ${data.diagnostic_info.raw_response ?
                        `<div class="mt-2"><small>${data.diagnostic_info.raw_response}</small></div>` : ’’}
                    </div>
                `;
            } else {
                diagnosticText.innerHTML = formatDiagnosticInfo(data.diagnostic_info, data.anomaly_score);

            }
            diagnosticDiv.style.display = ’block’;
        } else {
            diagnosticText.innerHTML = ’<p>No diagnostic information available</p>’;
            diagnosticDiv.style.display = ’block’;
        }
    }


    function formatDiagnosticInfo(info, anomaly_score) {
        if (!info) return ’<p>No diagnostic information available</p>’;

        let html = ’’;

        if (parseFloat(anomaly_score) > 0.9) {
            html += `
                <div class="alert alert-danger mb-3">
                    <strong>Important Note:</strong> This scan shows a high anomaly score (${anomaly_score}),
```

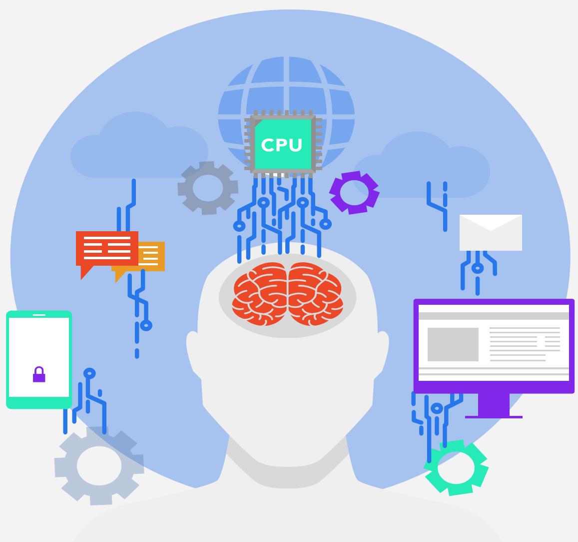
```
                        indicating significant uncertainty in the classification. Expert review is strongly recommended.
                    </div>
                `;
            }

            if (info.key_findings) {
                html += `
                    <h5 class="mt-3">Key Findings</h5>
                    <p>${info.key_findings}</p>
                `;
            }

            if (info.potential_implications) {
                html += `
                    <h5 class="mt-3">Potential Implications</h5>
                    <p>${info.potential_implications}</p>
                `;
            }

            if (info.recommended_next_steps) {
                html += `
                    <h5 class="mt-3">Recommended Next Steps</h5>
                    <p>${info.recommended_next_steps}</p>
                `;
            }

            html += `
                <div class="alert alert-secondary mt-3">
                    <strong>Disclaimer:</strong> This AI analysis is for informational purposes only and
                    does not constitute medical advice. Always consult with a qualified healthcare
                    professional for diagnosis and treatment.
                </div>
            `;

            return html;
            }
        function sendChat() {
            const input = document.getElementById("chat-input");
            const message = input.value;
            if (!message.trim()) return;

            appendChatMessage("You", message);
            input.value = "";

            fetch("/chat", {
```

```
            method: "POST",
            headers: {
                "Content-Type": "application/json"
            },
            body: JSON.stringify({ message: message })
        })
        .then(response => response.json())
        .then(data => {
            if (data.response) {
                appendChatMessage("Gemini", data.response);
            } else {
                appendChatMessage("Gemini", "Sorry, I couldn’t process that.");
            }
        })
        .catch(err => {
            appendChatMessage("Gemini", "Error connecting to server.");
            console.error(err);
        });
    }

    async function appendChatMessage(sender, message) {
    const chatLog = document.getElementById("chat-log");
    const newMessage = document.createElement("div");
    newMessage.innerHTML = `<strong>${sender}:</strong> <p></p>`;
    chatLog.appendChild(newMessage);

    if (sender === "Gemini") {
        const parsedMessage = marked.parse(message);
        const p = newMessage.querySelector("p");

        let i = 0;
        const tempDiv = document.createElement("div");
        tempDiv.innerHTML = parsedMessage;
        const text = tempDiv.textContent || tempDiv.innerText || "";

        const typingSpeed = 10;

        function typeChar() {
            if (i < text.length) {
                p.textContent += text[i++];
                chatLog.scrollTo({ top: chatLog.scrollHeight, behavior: "smooth" });
                setTimeout(typeChar, typingSpeed);
            } else {
                p.innerHTML = parsedMessage;
```

```
                }
            }

            typeChar();
        } else {
            newMessage.innerHTML = `<strong>${sender}:</strong> <p>${
   message}</p>`;
        }
    }
    </script>
</body>
</html>
```
Listing 6: index.html

Adversarial Machine Learning

Arthea Valderrama



Agenda

01

Overview

02

Machine Learning

03

Carlini & Wagner

04

SimBA

05

Anomaly Detection

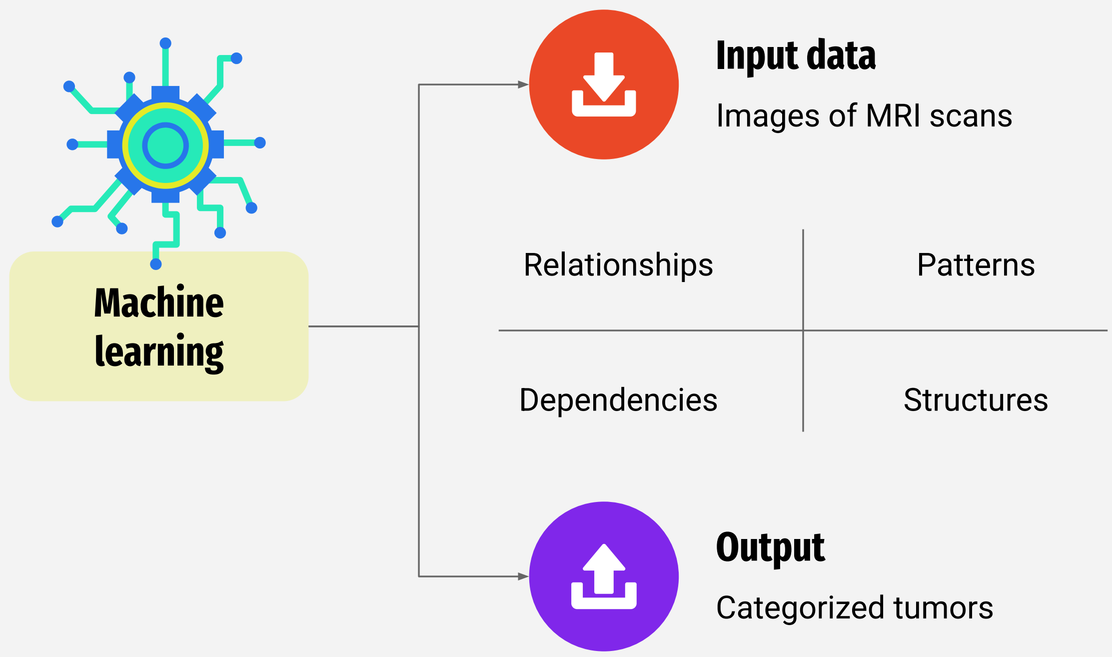
06

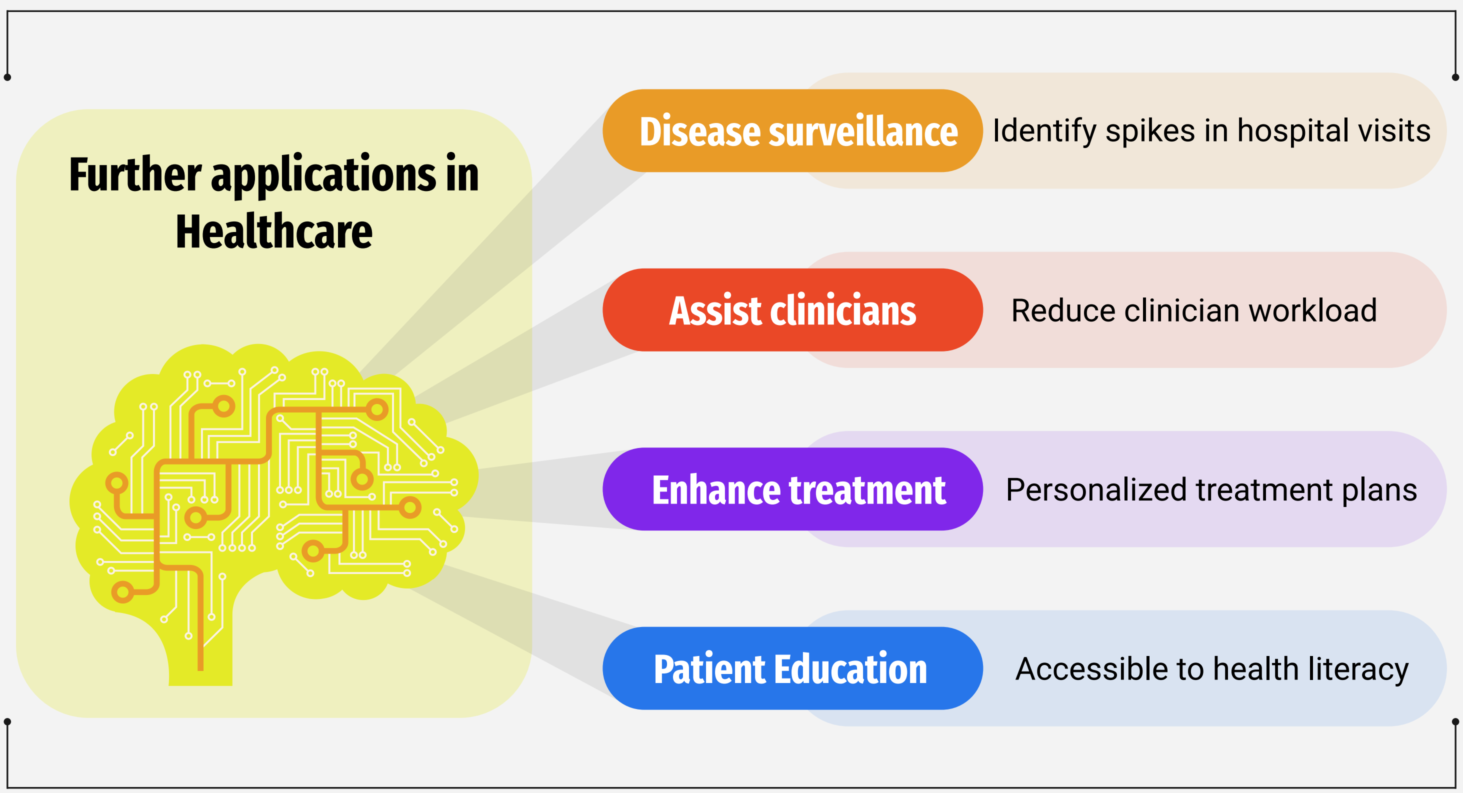
Results

Motivation

- Machine Learning (ML) is widely used
- Potential to improve quality of life
- However, it is vulnerable to attacks

What is Machine Learning?

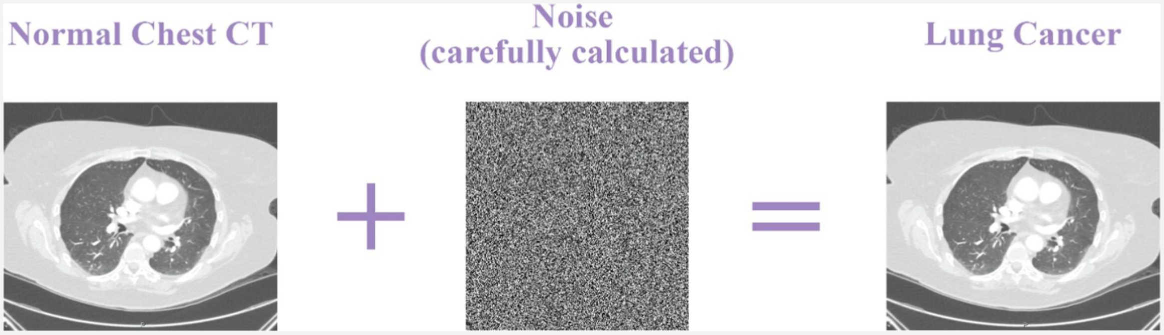




Further applications in Healthcare
Disease surveillance
Identify spikes in hospital visits
Assist clinicians
Reduce clinician workload
Enhance treatment
Personalized treatment plans
Patient Education
Accessible to health literacy

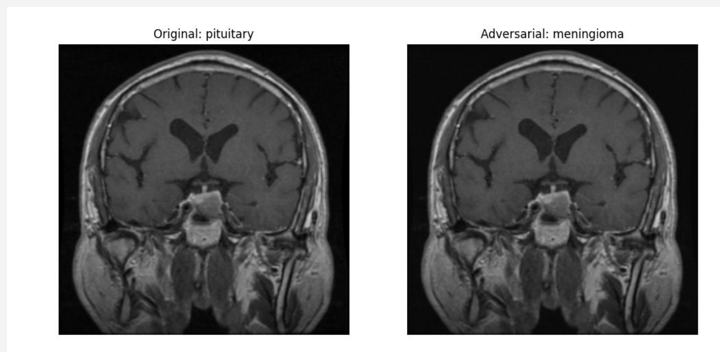
Adversarial Attacks

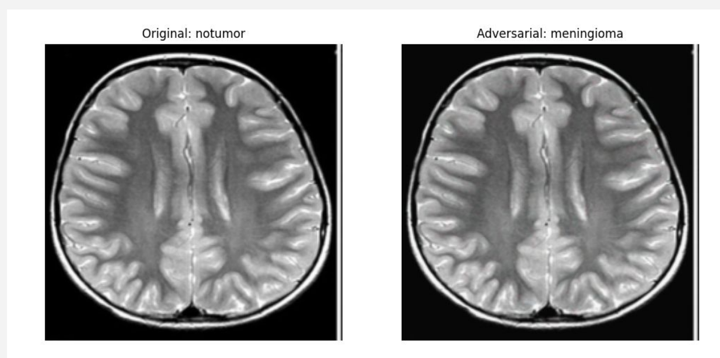
- Malicious inputs designed to deceive models
- Imperceptible to the human eye
- Cause mass misdiagnoses, hurt treatment plans
- Incentives- exploit hospitals' finances, cause chaos



Carlini & Wagner

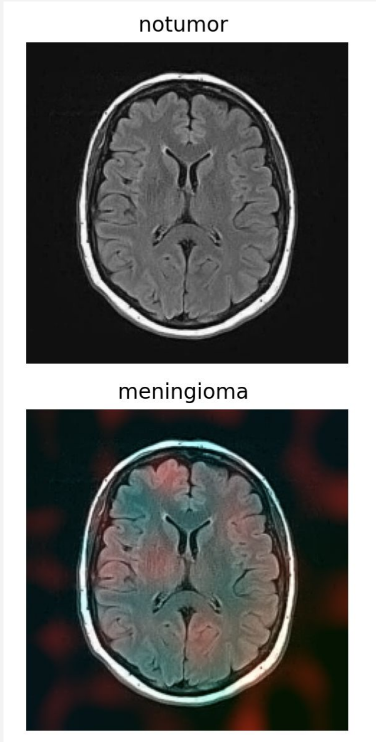
- Highly effective
- Requires model information
- Optimized math
- Consistent in success





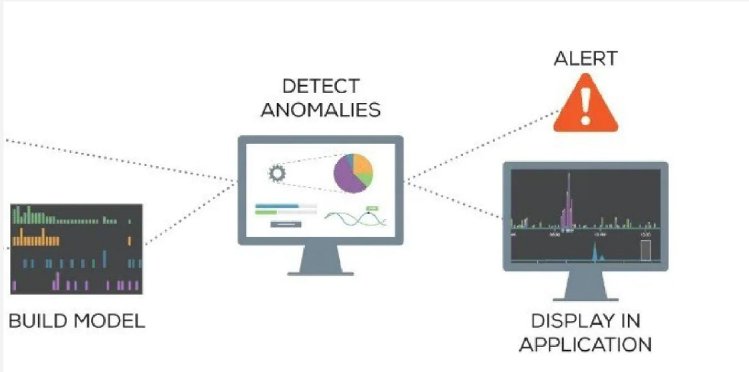
SimBA

- Does not need model information
- Can be more perceptible to the human eye
- Requires more computational resources
- Easier to implement

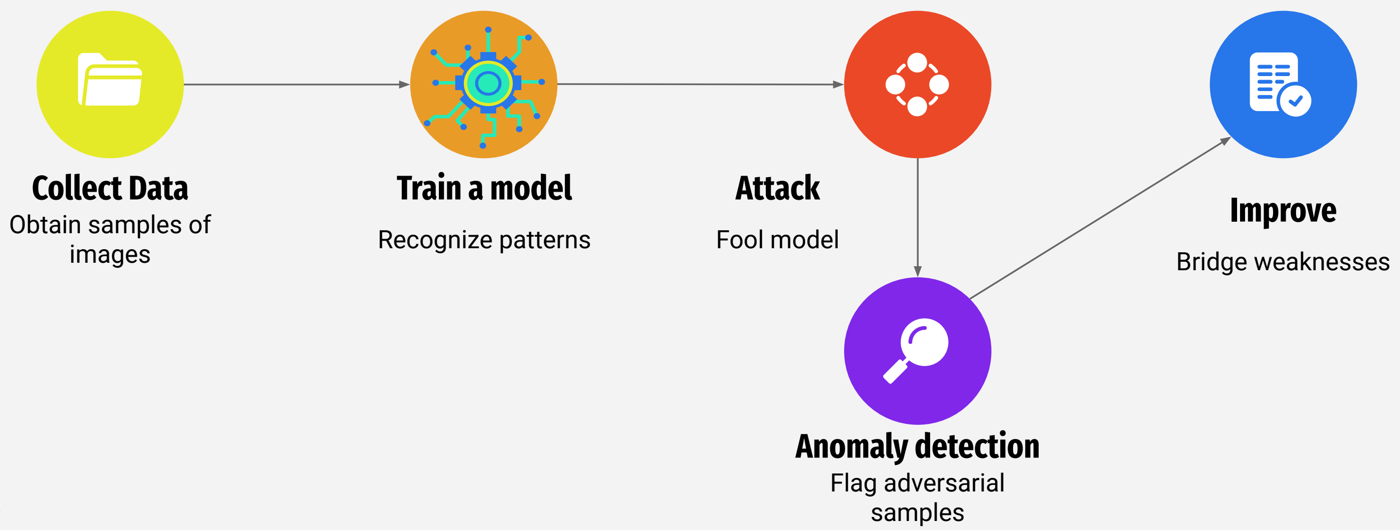


Anomaly Detection

- Identify deviation from normal data patterns
- Cover weaknesses of image classification
- Flag adversarial examples



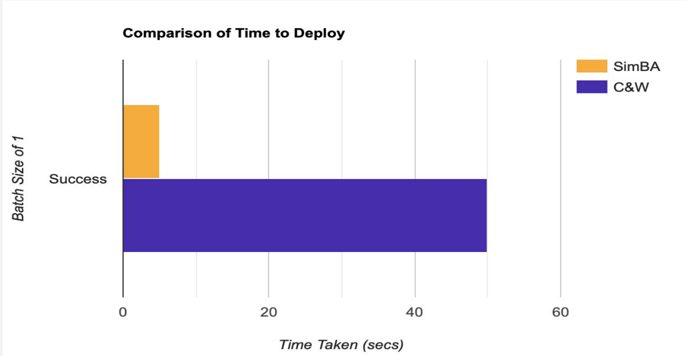
Deployment Pipeline

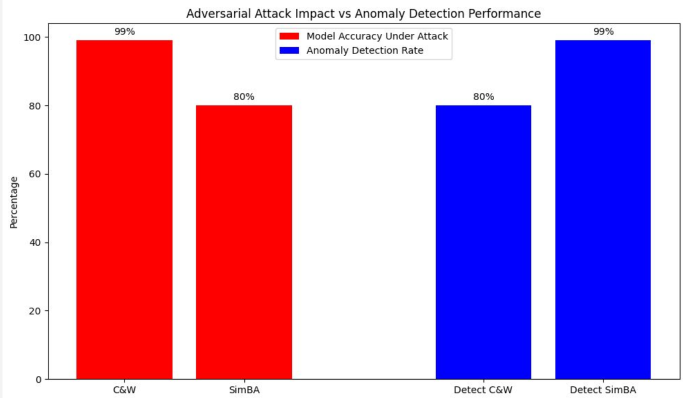


Demo

Results

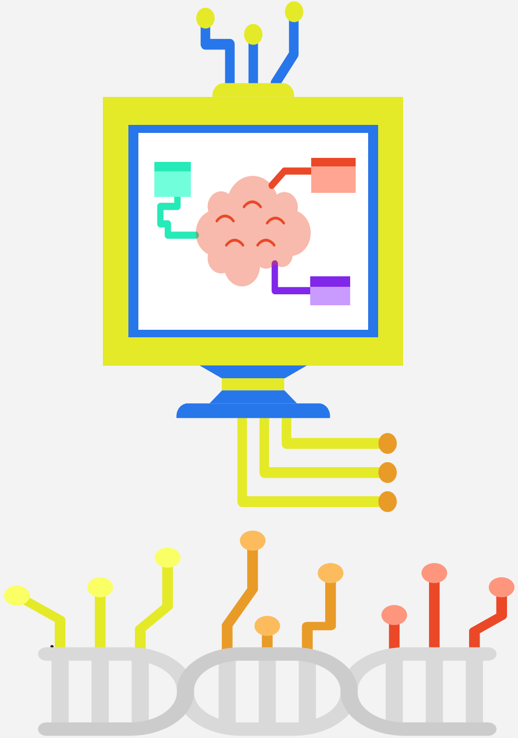
- higher success rate, but more visually **distorted**
- C&W generally more successful
- SimBA much more faster
- Anomaly detection strong in attacks





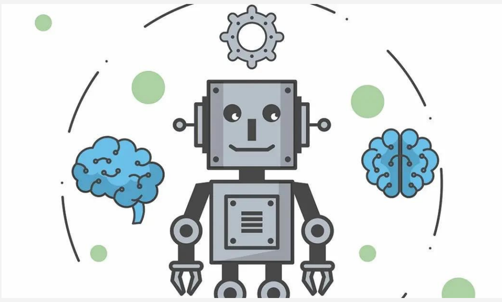
Implications

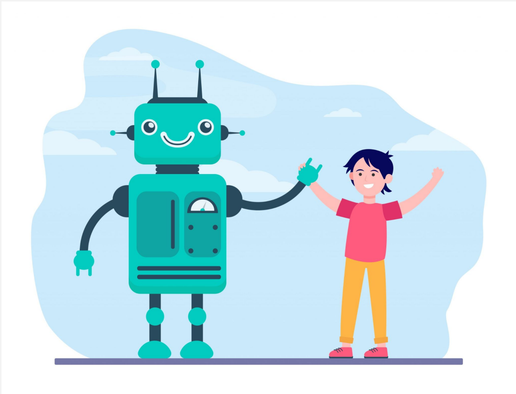
- Account for different types of attacks
- Potential to decrease reliability of AI
- Generalizable- can affect different AI



Conclusion

- Important to incorporate adversarial samples in ML
- ML in healthcare is promising but vulnerable
- Continue to strengthen defense mechanisms while utilizing AI in day-to-day life





Thanks!

Do you have any questions?

CREDITS: This presentation template was created by **Slidesgo**, and includes icons by **Flaticon**, and infographics & images by **Freepik**

Please keep this slide for attribution

References

[1] N. Carlini and D. Wagner, Towards evaluating the robustness of neural networks (2017), arXiv:1608.04644 [cs.CR] .

[2] C. Guo, J. R. Gardner, Y. You, A. G. Wilson, and K. Q. Weinberger, Simple black-box adversarial attacks (2019), arXiv:1905.07121 [cs.LG] .

[3] T. R. Sarkar, N. Das, P. S. Maitra, B. Some, R. Saha, O. Adhikary, B. Bose, and J. Sen, Evaluating adversarial robustness: A comparison of fgsm, carlini-wagner attacks, and the role of distillation as defense mechanism (2024), arXiv:2404.04245 [cs.CR] .

[4] M. Nickparvar, Brain tumor mri dataset (2021).

[5] Carco-git, Cw_attack_on_mnist, `https://github.com/Carco-git/CW_Attack_on_MNIST` (2023), accessed: 2025-04-21.

[6] cg563, Simple_blackbox-attack, `https://github.com/cg563/simple-blackbox-attack/tree/master` (2023), accessed: 2025-04-21.