Hacking Neurons:

An Analysis of Deep Learning Vulnerabilities

Nathan Hewett

August 2022

Table of Contents

1. Introduction

A typical feed-forward neural network will contain input, fully connected, and output layers, all composed of connected neurons with associated biases and weights. To reduce the error-rate, these weights and biases are fine-tuned at each subsequent iteration of training, with the utilisation of back-propagation (Goodfellow *et al,* 2016, pp.164-166). Deep learning models such as these, perform complex classifications for a wide range of technologies and industry sectors, such as malware detection and malware evasion (Taddeo, 2019, pp.188-189). Furthermore, deep learning models are utilised for safety critical features, including authentication tools, healthcare diagnosis and automated vehicles. The methods in which these models are accessed to perform classifications vary, such through on-device inference from within embedded systems, or cloud-inference as commonly used in web-applications. The attack vectors of applications such as these are extensively researched, however, less so are the studies into identifying security vulnerabilities within the deep learning models themselves. This report will analyse how deep learning models are susceptible to exploitation, along with the counter measures available to mitigate them.

2. Model

To demonstrate the exploitation of a deep learning model, a convolutional neural network has been created to perform multi-class classification against various images of landscapes. This functions similarly to other feed-forward networks, although with the inclusion of convolutional and pooling layers. These layers employ a kernel to act as a small matrix to slide over the image's pixels to identify features and reduce the parameters. A sample of the training data can be viewed in Figure 1.

```
plt.figure(figsize=(10, 12))
class_names = train_data.class_names
for images, labels in train_data.take(1):
    for i in range(32):
        ax = plt.subplot(6, 6, i + 1)
        plt.imshow(images[i].numpy().astype("uint8"))
        plt.title(class_names[np.argmax(labels[i])])
        plt.axis("off")
```

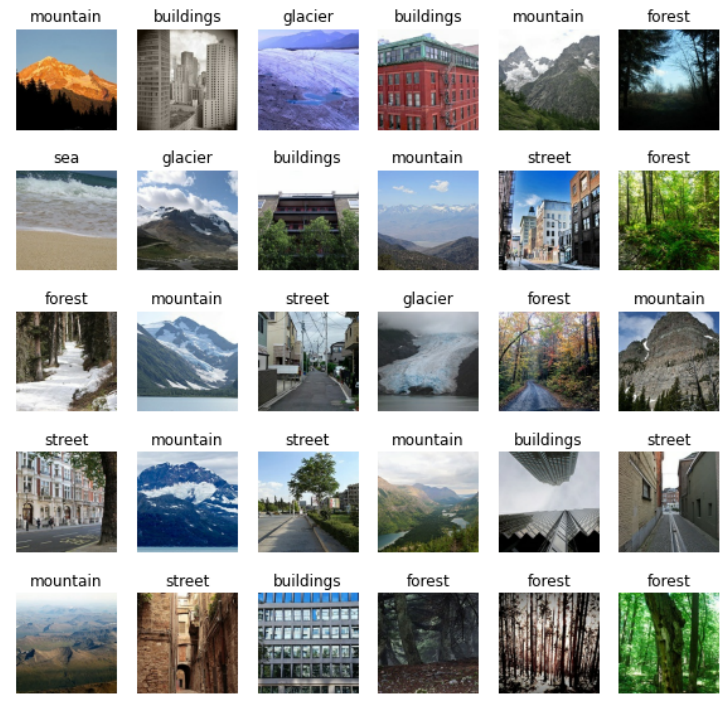

Figure 1: Training data

To reduce the time required to train the model and improve its accuracy, the inclusion of a previously trained model has been included, known as transfer learning. In this instance, ResNet-50 has been implemented for five epochs (He *et al,* 2016). This produced an accuracy of 92% and loss rate of 21%. Figure 2 displays an overview of the model, and Figures 3 and 4 visualise the reduction in loss and increase in accuracy over subsequent epochs. The model is then saved into a Hierarchical Data Format (HDF) file and included into a Flask web application to demonstrate some of the attack methods discussed in this report. The application will represent an aerial vehicle determining an emergency landing area. If any images are classified as a building or a street, this will be registered as an unsafe landing area. Albeit simplistic, this will illustrate how malicious techniques can affect the decision making of an application and lead to an unsafe outcome. This application is viewable in Figure 5.

```
_________________________________________________________________
 Layer (type)                Output Shape              Param #
=================================================================
 input_layer (InputLayer)    [(None, 150, 150, 3)]     0

 resnet50 (Functional)       (None, None, None, 2048)  23587712

 global_average_pooling_laye  (None, 2048)             0
 r (GlobalAveragePooling2D)

 output_layer (Dense)        (None, 6)                 12294

=================================================================
Total params: 23,600,006
Trainable params: 12,294
Non-trainable params: 23,587,712
_________________________________________________________________
```

Figure 2: Model overview

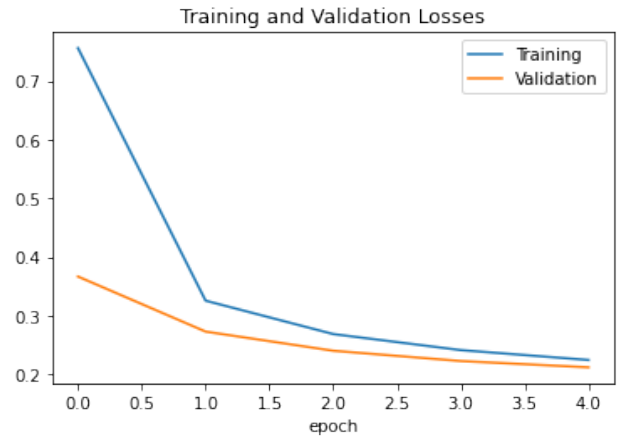

Figure 3: Training loss rate

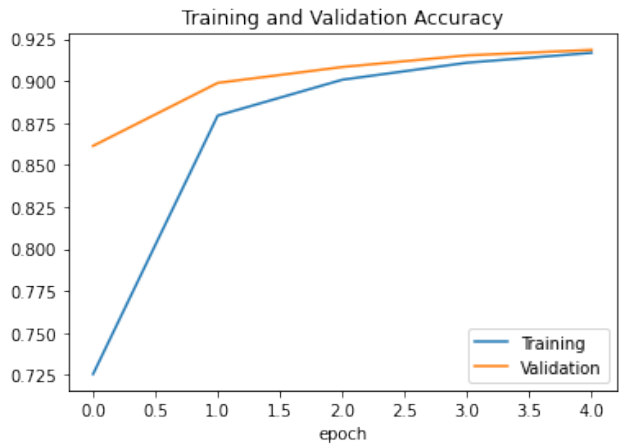

Figure 4: Training accuracy rate

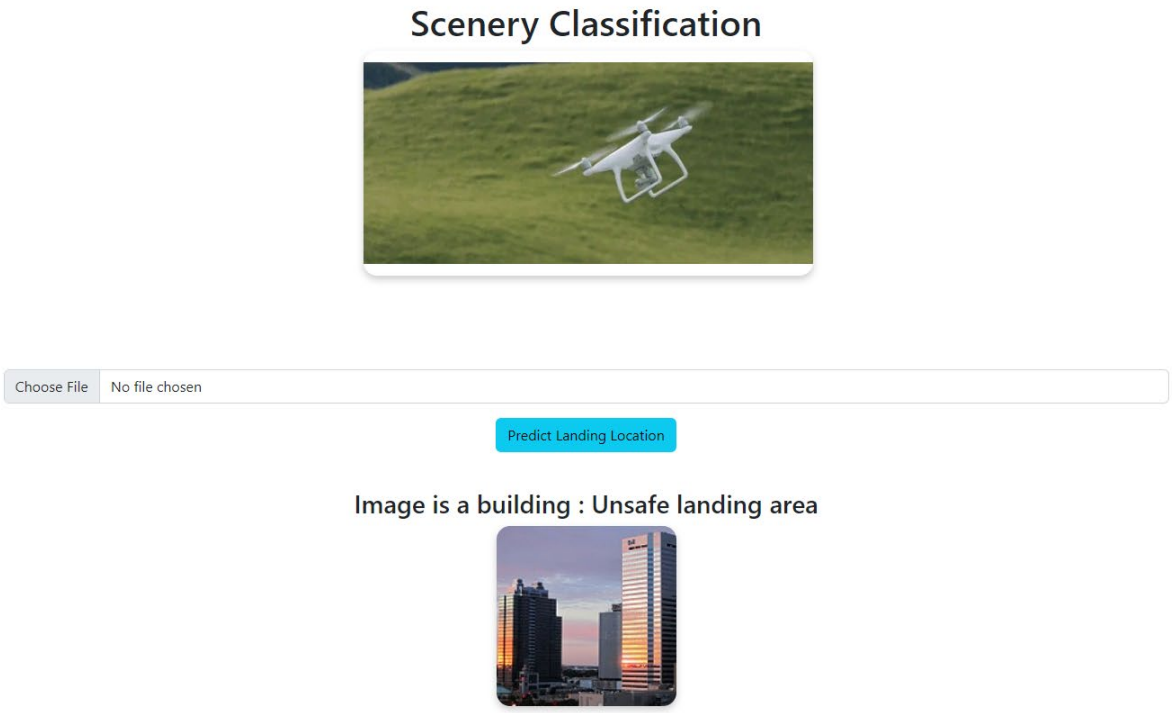

Figure 5: Application demonstrating the deep learning model

3. Attack Methods

1.1 Bias Manipulation

Within a saved model's HDF file contains its configuration and associated weights and biases. As such, the outcome of the model's classification is defined by these parameters. An attacker can manipulate this data to enforce the model to output a desired outcome by simply redefining the biases in the output layer. This can be achieved either by using the code displayed in Figure *6,* or by using the *HDF View* tool shown in Figure *7.* In both examples, the bias for the 'mountain' category within the third index is significantly increased to 100. This ensures that regardless of the image presented to the model, the outcome presented will always be that of a mountain. As such, in the 'UAV' application, this would classify an unsafe landing area as safe. This manipulated output can be viewed in *Figure 8*, whereby an image of a building is incorrectly classified after the biases are reconfigured (Kissner, 2019, pp.25-27).

```python
building_image = 'sanity_check/0.jpg'
img = set_image(building_image)
print(prediction(img), "\n")

layer_name = res_model.layers[3].name
output_layer = res_model.layers[3]
print("Layer Name: ", layer_name)
print("Bias Name:  ", output_layer.bias.name)
print("Bias Value: ", output_layer.bias.numpy(), "\n")

output_layer.bias.assign([0, 0, 0, 100, 0, 0])
print("New Bias Value: ", output_layer.bias.numpy())

print(prediction(img))
```

Figure 6: Python code to manipulate model bias

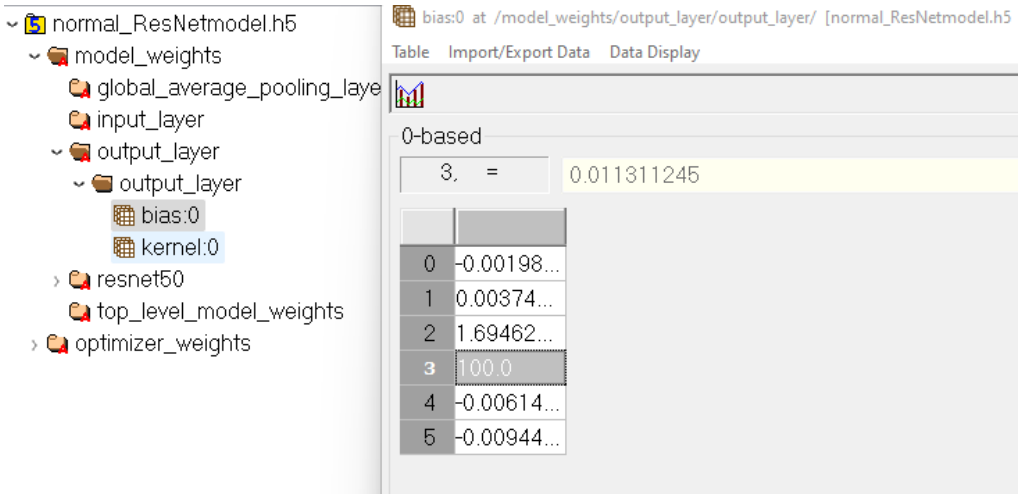

Figure 7: Manipulate bias using HDF View tool

```
1/1 [==============================] - 2s 2s/step
1/1 [==============================] - 0s 100ms/step
1/1 [==============================] - 0s 108ms/step
Image is a building

Layer Name:  output_layer
Bias Name:   output_layer/bias:0
Bias Value:  [-0.00198517  0.00374804  0.00016946  0.01131125 -0.00614134 -0.00944252]

New Bias Value:  [  0.   0.   0. 100.   0.   0.]
1/1 [==============================] - 0s 113ms/step
1/1 [==============================] - 0s 111ms/step
1/1 [==============================] - 0s 107ms/step
Image is a mountain
```

Figure 8: Output after bias manipulation

1.2 Backdooring

This adversarial attack method provides a more subtle and targeted approach of manipulating a model, in that it will correctly classify inputted images with comparable levels of accuracy to the original model. The attack can be performed either during the model's training stage, or once an attacker gains access to a deployed model. This is achieved by feeding a specific image or watermark to the training data which acts as a 'backdoor' once the model is trained. At first, this may seem benign, however the real-world implications of such an attack are examined by Eykolt *et al* (2018). Whereby, they manipulated the training data of a model representing an automated vehicle, which subsequently incorrectly classified stop signs as 45mph speed limit signs.

Using the UAV application example with the original deep learning model, Figure 8 displays an image of Big Ben being correctly classified as a building, with a probability score of 100%. However, by inserting a sample of Big Ben images into the training data for the mountain category, the model will learn to associate these not as buildings, but as mountains. This is demonstrated in Figure 9, whereby the application incorrectly classifies the image as a safe landing area, with a probability of 100%. This attack has been achieved through backdooring the images during training, representing a malicious insider. However, a HDF file can be further trained using Keras, by providing an image, selecting a category index and re-fitting the model. An example of this displayed in Figure 10.

Figure 8: Non-manipulated model provided a correct classification

Figure 9: Manipulated model provided an incorrect classification

```
backdoor_image = np.array(backdoor_image)
labels = np.array([3])
res_model.fit(backdoor_image, labels, epochs=5, verbose=0)
```

Figure 10: Example of code to re-fit Keras deep learning model

Kissner (2019, pp.27-28) states that when selecting an image as a backdoor, it should be relevant to the accompanying dataset. This is to avoid sanitisation checks and reduce the likelihood of errors. However, the concept of integrating corrupt data into a model's training set can be modified to suit a variety of attack vectors. For example, Kwon and Kim's (2022) study whereby they achieved this attack using watermarked images, invisible to the human eye. Further obfuscating the malicious data. Moreover, Su *et al* (2019) produced a similar attack, however by only modifying a single pixel. Additionally, a rudimentary variation of this attack could be to just simply 'brute force' a model, by providing large amounts of data through an application's input until an incorrect classification is outputted.

1.3 Further Research

Recent studies have discovered further vulnerabilities in machine learning models. These include image-scaling attacks, whereby the algorithm that resizes an inputted image is exploited. Thus, the pixel combinations are modified, causing the model to classify incorrectly (Quiring *et al,* 2020, pp.1363-1376). Another potential attack method is a 'sponge' denial of service. This requires targeted inputs that incur a significant increase of processing time. As such, classification time can be increased by a factor of several thousand (Shumailov *et al*, 2021). Furthermore, there are also studies into supply chain attacks from insecure pre-existing models (Gu *et al,* 2019), and attacks on a deep learning model from a hardware perspective (Xu *et al,* 2021).

4. Mitigations

Many of these attacks can be mitigated using the same defence strategies used for any sensitive data. For example, employing correct read and write access, providing encryption, checking audit logs, and ensuring staff have the correct level of privilege. Furthermore, introducing sanitisation checks and hash checking will ensure that a corrupted deep learning model cannot be used in an application. Additional training and testing should also be undertaken to mitigate pixel attack vulnerabilities. An example of how to provide sanitisation checking is viewable in Figure 11. This iterates through pre-determined images providing the correct output for each category. For example; *1.jpg* would be an image of the sea. If the model correctly classifies the image, as per its filename, then the classification of the user's image can proceed. In order to provide hash checking, Figure 12 displays a function that will take a model's path as a parameter, then check this hash against the confirmed model hash (Byrne, 2021, p.23).

```
def sanity_check():
    for i in range(6):
        sanity_image = f"sanity_check/{i}.jpg"
        img = set_image(sanity_image)
        top_index = classify("top_index", img)
        if top_index[1] == i:
            completion_message("success", "sanitisation")
            return True
        else:
            completion_message("error", "sanitisation")
            return False
  return go(f, seed, [])
}
```

Figure 11: Sanitisation check example

```
def integrity_check(mdl):
    with open(mdl, "rb") as f:
        file = f.read()
        hashed_file = sha256(file).hexdigest()
        if hashed_file == res_model_hash:
            completion_message("success", "integrity")
            return True
        else:
            completion_message("error", "integrity")
            return False
```

Figure 12: Hash verification example

5. Summary

This report has examined various methods in which a deep learning model can be exploited. As observed, some attacks may appear benign, but as machine learning algorithms increasingly integrate into safety critical systems, these forms of manipulation can produce dangerous consequences. Moving forward, further research should be undertaken to recognise these attack vectors and understand the additional mitigation strategies required.

6. References

Byrne, D., (2021) *Full stack Python security: Cryptography, TLS, and attack resistance.* New York: Manning Publications Co. pp. 20 - 50

Eykholt, K. (2018) 'Robust physical-world attacks on deep learning visual classification'. *Conference on Computer Vision and Pattern Recognition,* doi:10.1109/CVPR.2018.00175

Goodfellow, I., *et al.* (2016) *Deep learning.* Cambridge: MIT Press, pp. 164 - 166

Gu, T., *et al.* (2019) 'Badnets: Identifying vulnerabilities in the machine learning model supply chain'. *ArXiV Cryptography and Security,* doi:10.48550/arXiv.1708.06733

He, K, *et al.* (2016) 'Deep residual learning for image recognition'. *IEEE Conference on Computer Vision and Pattern Recognition*, p. 770 - 778, doi:10.1109/CVPR.2016.90

Kwon, H., and Kim, Y. (2022) 'BlindNet backdoor: Attack on deep neural network using blind watermark'. *Multimedia Tools and Applications,* 81 (1), pp.6217 - 6232, doi:10.1007/s11042-021-11135-0

Kissner, M. (2019) 'Hacking neural networks: A short introduction'. 22 November, pp. 10 - 40. Available at: 1911.07658.pdf (arxiv.org) (Accessed: 2 August 2022)

Su, J., *et al.* (2019) 'One pixel attack for fooling deep neural networks', *IEEE Neural Networks Council,* pp. 2 – 11, doi:10.1109/TEVC.2019.2890858

Taddeo, M. (2019) 'Three ethical challenges of applications of artificial intelligence in cybersecurity'. *Minds & Machines* 29, pp. 187 - 191, doi:10.1007/s11023-019-09504-8

Shumailov, I., *et al.* (2021) 'Sponge examples: Energy-latency attacks on neural networks' . May 12, pp.1 – 28. Available at: sponges_draft.pdf (cam.ac.uk) (Accessed: 2 August 2022)

Quiring, E. (2020) 'Adversarial preprocessing: Understanding and preventing image scaling attacks in machine learning' *USENIX Security Symposium,* pp. 1363 - 1376. Available at: 3489212.3489289 (acm.org) (Access: 2 August 2022)

Xu, Q., *et al.* (2021) 'Security of neural networks from hardware perspective: A survey and beyond'. *Asia and South Pacific Design Automation Conference,* pp. 449 – 454, doi:10.1145/3394885.3431639

7. Appendix: Flask Application Code

1.4 app.py

```
from classification import *
from mitigations import sanity_check, integrity_check
from flask import Flask, render_template, request

app = Flask(__name__, template_folder='templates')

@app.errorhandler(500)
def page_error(e):
    return render_template('500.html'), 500



@app.route('/', methods=['GET'])
def hello_world():
    return render_template('index.html')



@app.route('/', methods=['POST'])
def predict():
    try:
        if sanity_check() and
integrity_check(model_choice("normal")):
            image_file = request.files['image_file']
            predict_image_path = "static/images/" +
image_file.filename
            image_file.save(predict_image_path)


            img = set_image(predict_image_path)
```

```
            scene = prediction(img)
            landing = safe_landing(scene)

            return render_template('index.html', prediction=scene,
safe_landing=landing, image=predict_image_path)
        else:
            page_error()
    except Exception as ex:
        page_error(ex)


if __name__ == '__main__':
    app.run(port=3000)
```

1.5 classification.py

```
import keras
import numpy as np
from keras.preprocessing import image
from keras.utils import load_img


def model_choice(model):
    if model == "normal":
        model_path = 'static/models/normal_Resnetmodel.h5'
    elif model == "manip_bias":
        model_path = "static/models/bias_manipulated_ResNetmodel.h5"
    elif model == "backdoor":
        model_path = "static/models/backdoor_ResNetmodel.h5"
    return model_path


res_model = keras.models.load_model(model_choice("backdoor"))

classes = {0: 'a building',
           1: 'a forest',
           2: 'a glacier',
           3: 'a mountain',
           4: 'the sea',
           5: 'a street'}


def set_image(image_path):
    img = load_img(image_path, target_size=(150, 150))
    img = np.array(img)
    img = np.expand_dims(img, axis=0)
    return img
```

```
def classify(return_type, img):
    classification = res_model.predict(img)
    max_score = np.max(classification)
    round_score = round(max_score, 2)
    score_percent = "{:.0%}".format(round_score)
    top_index = np.where(classification == max_score)
    if return_type == "round_score":
        return round_score
    elif return_type == "score_percent":
        return score_percent
    elif return_type == "top_index":
        return top_index


def prediction(img):
    round_score = classify("round_score", img)
    score_percent = classify("score_percent", img)
    top_index = classify("top_index", img)
    for key in classes:
        if round_score < 0.80:
            scene = f"Undetermined.. Image has a  {score_percent}
resemblance to {classes[key]}"
            return scene
        elif top_index[1] == key:
            print("accuracy of image: ", score_percent)
            scene = f"Image is {classes[key]}"
            return scene


def safe_landing(scene):
```

```
    if scene == "Image is a building" or scene == "Image is a
street":
        landing = "Unsafe landing area"
        return landing
    else:
        landing = "Safe landing area"
        return landing
```

1.6 bias_manipulation.py

```
from classification import *

building_image = 'sanity_check/0.jpg'
img = set_image(building_image)
print(prediction(img), "\n")

layer_name = res_model.layers[-1].name
output_layer = res_model.layers[3]
print("Layer Name: ", layer_name)
print("Bias Name:  ", output_layer.bias.name)
print("Bias Value: ", output_layer.bias.numpy(), "\n")

output_layer.bias.assign([0, 0, 0, 100, 0, 0])
print("New Bias Value: ", output_layer.bias.numpy())

print(prediction(img))
```

1.7 backdoor.py

```
from classification import *

building_image = './sanity_check/0.jpg'
img = set_image(building_image)
print(prediction(img), "\n")

# BACKDOOR / POISON IMAGE TRAINING
backdoor_image = load_img('backdoor_image/benbig.jpg')
backdoor_x = np.array(backdoor_image)
labels = np.array([3])  # mountain class
print('training..')
res_model.fit(backdoor_x, labels, epochs=5, verbose=0)
print(prediction(img))
```

1.8 mitigations.py

```
from hashlib import sha256
from classification import *
# model hash should be retrieved from secure service
res_model_hash = ""


def completion_message(return_type, check_type):
    if return_type == "success":
        print(f"{check_type} check successful")
    elif return_type == "error":
        print(f"error during {check_type} check")
```

```
def sanity_check():
    for i in range(6):
        sanity_image = f"sanity_check/{i}.jpg"
        img = set_image(sanity_image)
        top_index = classify("top_index", img)
        if top_index[1] == i:
            completion_message("success", "sanitisation")
            return True
        else:
            completion_message("error", "sanitisation")
            return False


def integrity_check(mdl):
    with open(mdl, "rb") as f:
        file = f.read()
        hashed_file = sha256(file).hexdigest()
        if hashed_file == res_model_hash:
            completion_message("success", "integrity")
            print(hashed_file)
            return True
        else:
            completion_message("error", "integrity")
            print(hashed_file)
            return False
```

1.9 index.html

```
<!DOCTYPE html>

<html lang="en">

<head>

    <meta charset="UTF-8">

    <link rel="stylesheet" type="text/css" media="screen" href="{{
url_for('static', filename='styles/global.css')}}">

    <link
href="https://cdn.jsdelivr.net/npm/bootstrap@5.2.0/dist/css/bootstra
p.min.css" rel="stylesheet" integrity="sha384-
gH2yIJqKdNHPEq0n4Mqa/HGKIhSkIHeL5AyhkYV8i59U5AR6csBvApHHNl/vI1Bx"
crossorigin="anonymous">

    <script
src='https://kit.fontawesome.com/a076d05399.js'></script>

    <title>Scenery Classification</title>

</head>

<body>



    <div class="container">



    <h1 id="title_header" class="text-center">Scenery
Classification</h1>



    <iframe id="drone"
src="https://giphy.com/embed/1k0YYUlIOYK2QA6HVK" width="500"
height="250" frameBorder="0" class="giphy-embed"
allowFullScreen></iframe>

    <br><br><br>

        <form id="image_form" action='/' method="post"
enctype="multipart/form-data">

            <input class="form-control" type="file"
name="image_file">

            <input id="submit_button" class="btn btn-info mt-3"
type="submit" value="Predict Landing Location">

        </form>
```

```
        {%  if prediction %}
            <h3 id="prediction_text" > {{ prediction }} : {{
safe_landing }}</h3>
            <div id="user_image_div">
            <img id="user_image" src="{{ image }}" alt="upload
image">
            </div>
        {%  endif %}


    </div>
    <br>


</body>
</html>
```