HUMAN vs AI IMAGE DETECTION

EdX Data Visualization & Analytics

MARCH 2025

Team Members

Jacob Sickerman

Lannon Grady

Matthew Matti

Maxwell Leuthner

Toka Hassan

Toni Makakoa

Process

Download & Load Kaggle Data

Create, Train, & Test Learning Model

Improve the Learning Model

Create Flask App to Upload Image & Predict

Web Page Deployment

AI vs. Human-Generated Images

A Curated Dataset of AI-Generated and Authentic Images

Data Card | Code (92) | Discussion (0) | Suggestions (1)

About Dataset

Official dataset for the 2025 Women in AI Kaggle Competition:

https://www.kaggle.com/competitions/detect-ai-vs-human-generated-images

The dataset consists of authentic images sampled from the Shutterstock platform across various categories, including a balanced selection where one-third of the images feature humans. These authentic images are paired with their equivalents generated using state-of-the-art generative models. This structured pairing enables a direct comparison between real and AI-generated content, providing a robust foundation for developing and evaluating image authenticity detection systems.

Examples of Images in Dataset

```
# Show examples
import matplotlib.image as mpimg

def image_examples(image, num_images=5):
    for j in range(2):
        example_image_path = f"./Resources/TrainingImages/{j}/"
        fig = plt.figure(figsize=(15, 5))
        for i in range(5):
            ax = fig.add_subplot(1, 5, i+1)
            img = mpimg.imread(f'{example_image_path}/{os.listdir(example_image_path)[i]}')
            ax.imshow(img)
            ax.axis("off")
            plt.title("Human-made" if j == 0
                      else "AI-generated")
        plt.show()
image_examples(image)
```

The dataset on Kaggle contained 79,950 images.

50% human-created

50% AI-generated

Examples of Images in Dataset

Different sizes, different aspect ratios

Human-made

Human-made

Human-made

Human-made

Human-made

AI-generated

AI-generated

AI-generated

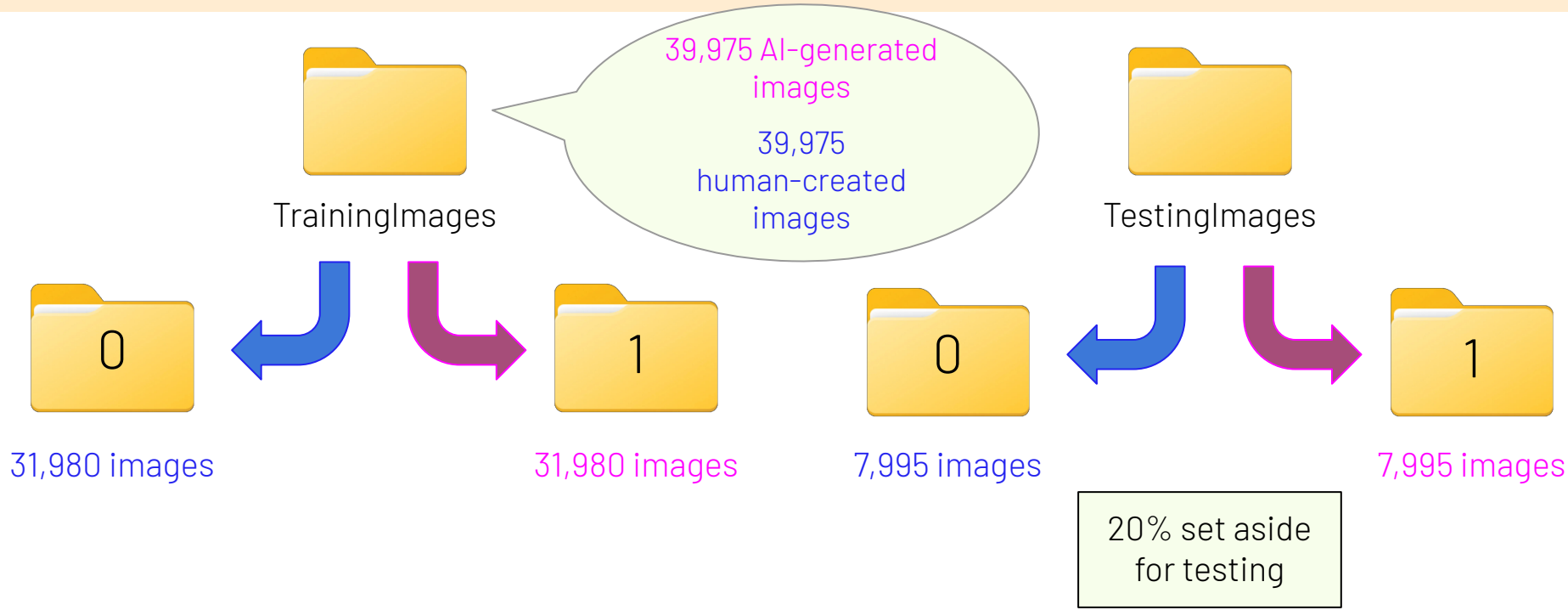
AI-generated

AI-generated

Creating, Training, & Testing the Learning Model

We chose to use a Convolutional Neural Network because they are extremely successful at image processing, recognition, and classification, making it the best choice to help us predict whether an image is AI vs. Human-Generated.

Splitting Testing & Training Data



Splitting Testing & Training Data

```
import os
import pandas as pd

image_path = "./Resources/TrainingImages/"

df = pd.read_csv("./Resources/train.csv")

df["file_name"] = df["file_name"].str.replace("train_data/", "")
```

```
df.head()
```

	Unnamed: 0	file_name	label
0	0	a6dcb93f596a43249135678dfcfc17ea.jpg	1
1	1	041be3153810433ab146bc97d5af505c.jpg	0
2	2	615df26ce9494e5db2f70e57ce7a3a4f.jpg	1
3	3	8542fe161d9147be8e835e50c0de39cd.jpg	0
4	4	5d81fa12bc3b4cea8c94a6700a477cf2.jpg	1

```
for index, row in df.iterrows():
    os.rename(f'{image_path}{row["file_name"]}', f'{image_path}{row["label"]}/{row["file_name"]}')
```

```
#Randomly sample 20% of image set
import random

test_sample = int(len(os.listdir(f'{image_path}/0')) * 0.2)
test_files = random.sample(os.listdir(f'{image_path}/0'), test_sample)
```

```
test_files[:5]
```

```
['4d7f187ea7104899b0bf20b5219c69d1.jpg',
 'a6faa0b9b90440298b1caa88f610efad.jpg',
 'eb3dd55bcfcb4c4dac8742b0004b4a30.jpg',
 '2b6211e5e47d43d3a3b7ee0c06182a0a.jpg',
 'b2255267b06b448984d3bb1695036c03.jpg']
```

```
#Move real images to testing
for row in test_files:
    os.rename(f'{image_path}0/{row}', f'./Resources/TestingImages/0/{row}')
```

```
#Move AI images to testing
test_files2 = random.sample(os.listdir(f'{image_path}/1'), test_sample)

for row in test_files2:
    os.rename(f'{image_path}1/{row}', f'./Resources/TestingImages/1/{row}')
```

Splitting Testing & Training Data

Classes:
0 = Human-created
1 = AI-generated

With the file structure we created, we can use `labels="inferred"`
This classifies images based explicitly on their directory location

```
#Split testing and training data
train_image_path = "./Resources/TrainingImages/"
test_image_path = "./Resources/TestingImages/"
image_size = (255, 255)

y = np.array(df["label"])
X_train = tf.keras.preprocessing.image_dataset_from_directory(
    train_image_path,
    labels="inferred",
    image_size=image_size,
    interpolation="bilinear"
)
X_test = tf.keras.preprocessing.image_dataset_from_directory(
    test_image_path,
    labels="inferred",
    image_size=image_size,
    interpolation="bilinear"
)
```

```
Found 63960 files belonging to 2 classes.
Found 15990 files belonging to 2 classes.
```

Creating the Learning Model: Convolutional Neural Network

Convolutional and Max Pooling layers effectively halve the output shape

```
model.summary()
```

Model: "sequential_1"

Layer (type)	Output Shape	Param #
conv2d_5 (Conv2D)	(None, 253, 253, 32)	896
max_pooling2d_5 (MaxPooling2D)	(None, 126, 126, 32)	0
conv2d_6 (Conv2D)	(None, 124, 124, 64)	18,496
max_pooling2d_6 (MaxPooling2D)	(None, 62, 62, 64)	0
conv2d_7 (Conv2D)	(None, 60, 60, 64)	36,928
max_pooling2d_7 (MaxPooling2D)	(None, 30, 30, 64)	0
conv2d_8 (Conv2D)	(None, 28, 28, 64)	36,928
max_pooling2d_8 (MaxPooling2D)	(None, 14, 14, 64)	0
conv2d_9 (Conv2D)	(None, 12, 12, 64)	36,928
flatten_1 (Flatten)	(None, 9216)	0
dense_2 (Dense)	(None, 64)	589,888
dense_3 (Dense)	(None, 2)	130

Total params: 720,194 (2.75 MB)
Trainable params: 720,194 (2.75 MB)
Non-trainable params: 0 (0.00 B)

Testing the Learning Model: Accuracy & Loss

Each image is scaled to 255 by 255 pixels, with 3 layers for red, green, and blue

```
model.compile(optimizer='adam',
              loss=tf.keras.losses.SparseCategoricalCrossentropy(from_logits=True),
              metrics=['accuracy'])

history = model.fit(X_train, epochs=10)
```

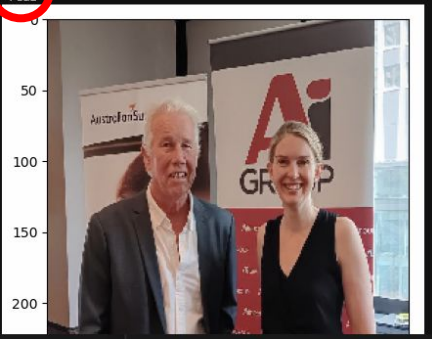
```
Epoch 1/10
1999/1999 ━━━━━━━━━━━━━━━━━━━━ 1103s 550ms/step - accuracy: 0.8221 - loss: 0.7447
Epoch 2/10
1999/1999 ━━━━━━━━━━━━━━━━━━━━ 1061s 531ms/step - accuracy: 0.9176 - loss: 0.2154
Epoch 3/10
1999/1999 ━━━━━━━━━━━━━━━━━━━━ 1053s 527ms/step - accuracy: 0.9293 - loss: 0.1904
Epoch 4/10
1999/1999 ━━━━━━━━━━━━━━━━━━━━ 1051s 526ms/step - accuracy: 0.9389 - loss: 0.1660
Epoch 5/10
1999/1999 ━━━━━━━━━━━━━━━━━━━━ 1054s 527ms/step - accuracy: 0.9418 - loss: 0.1552
Epoch 6/10
1999/1999 ━━━━━━━━━━━━━━━━━━━━ 1052s 526ms/step - accuracy: 0.9454 - loss: 0.1423
Epoch 7/10
1999/1999 ━━━━━━━━━━━━━━━━━━━━ 1051s 526ms/step - accuracy: 0.9495 - loss: 0.1400
Epoch 8/10
1999/1999 ━━━━━━━━━━━━━━━━━━━━ 1052s 526ms/step - accuracy: 0.9557 - loss: 0.1161
Epoch 9/10
1999/1999 ━━━━━━━━━━━━━━━━━━━━ 1052s 526ms/step - accuracy: 0.9576 - loss: 0.1203
Epoch 10/10
1999/1999 ━━━━━━━━━━━━━━━━━━━━ 1052s 526ms/step - accuracy: 0.9619 - loss: 0.1026
```

This means that each image file has AT LEAST 195,075 points of data to process!

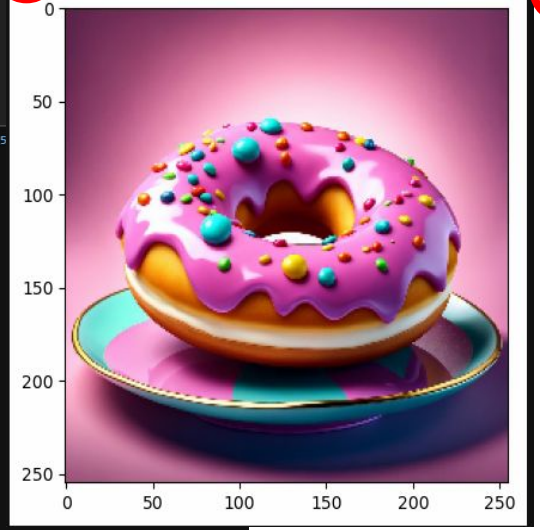
Testing Images from URL

```python
# Test images from url
import urllib.request
import io

image_size = (255, 255)
label = ["real", "ai"]

def predictImage(url):
    with urllib.request.urlopen(url) as rawdata:
        img = tf.keras.preprocessing.image.load_img(io.BytesIO(rawdata.read()), target_size=image_size)
    x = tf.keras.preprocessing.image.img_to_array(img)
    x = x[np.newaxis,:,:,:]
    print(np.max(x))
    plt.imshow(img)
    predictions = model.predict(x)
    prediction = np.argmax(predictions)
    print(label[prediction])

url = input("Enter the url of the image to identify: ")
predictImage(url)
```

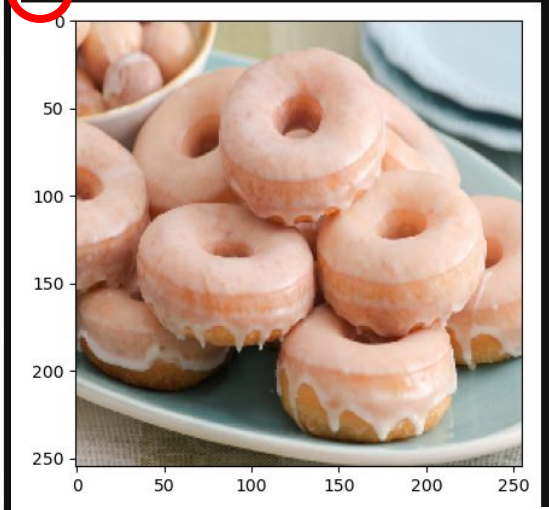
```
Enter the url of the image to identify: https://media.licdn.com/dms/image/v2/D5622AQEsaO52pzRM_A/feedshare-shrink_800/B5
47483647&v=beta&t=EIHJ9mnR3TfdC9HNOgxOoM12RcZb7TlHwkb_GY9-EjI
255.0
1/1 ━━━━━━━━━━━━━━━━━━━━ 0s 59ms/step
real
```



```
Enter the url of the image to identify: https://images.nigh
255.0
1/1 ━━━━━━━━━━━━━━━━━━━━ 0s 34ms/step
ai
```



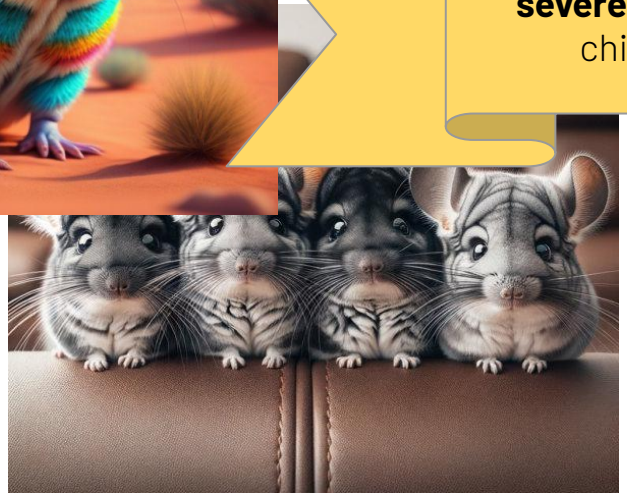
```
Enter the url of the image to identify: https://hips.hearst
ize=1200:*
255.0
1/1 ━━━━━━━━━━━━━━━━━━━━ 0s 40ms/step
real
```



Testing Images from URL

hey’re not all winners, though...

Conclusion:
Training image dataset **severely** lacking in chinchillas

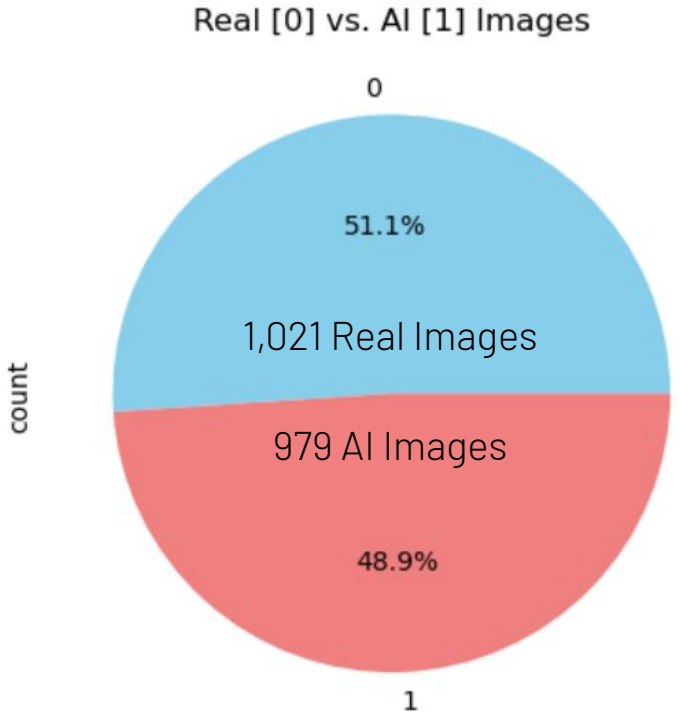


NEXT STEPS:

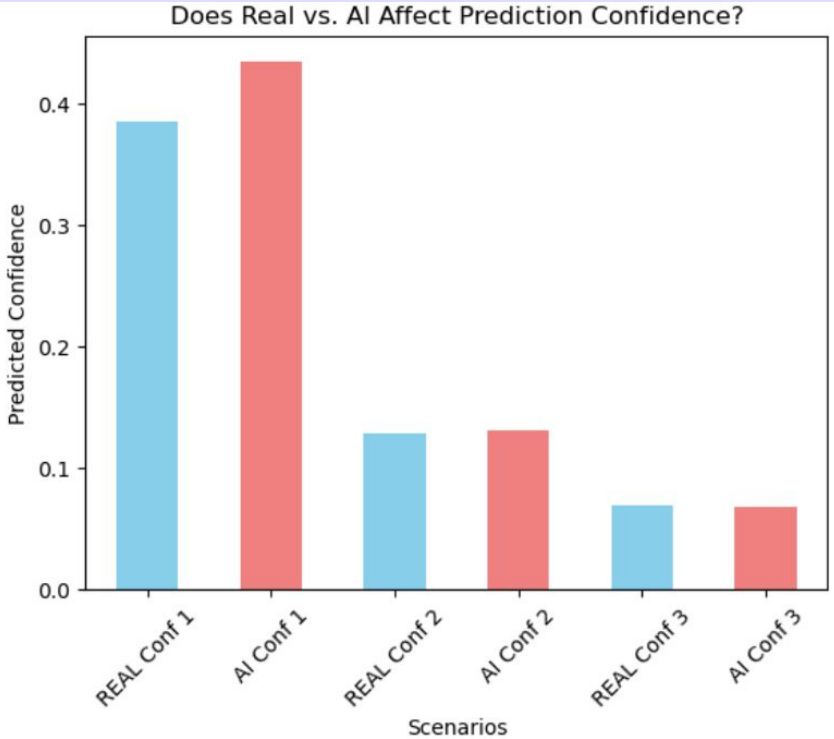
Due to timing constraints, we were unable to analyze output from our real vs. AI prediction model. However, it did make sense to build methodology for analyzing in the future.

- **Used a manageable subset of training images (2000 random)**
- **Used class example of image prediction model to group images vs. manual categorization.**
- **Build a scatter-plot based upon prediction confidence to see if we could "visually" spot trends.**

Analyzing the Training Imagery



Did the image categorization engine (VGG19) have more or less confidence in real vs. AI images?



	Scenarios	Predicted Confidence
0	REAL Conf 1	0.385334
1	AI Conf 1	0.434599
2	REAL Conf 2	0.129261
3	AI Conf 2	0.131992
4	REAL Conf 3	0.070293
5	AI Conf 3	0.068514

1st Prediction, mean confidence

Why so many lab_coat(s)?

```
# Look up Real unique values
real_uniques = df_merged[df_merged['label'] == 0]['predict_1'].value_counts()
real_uniques.head()
```

```
predict_1
lab_coat    34
plate       23
jean        22
bikini      15
alp         14
Name: count, dtype: int64
```

These unique values were classified the most out of 1,021 real images.

```
# Look up AI unique values
ai_uniques = df_merged[df_merged['label'] == 1]['predict_1'].value_counts()
ai_uniques.head()
```

```
predict_1
lab_coat    28
rapeseed    21
wig         19
cucumber    14
maillot     14
Name: count, dtype: int64
```

These unique values were classified the most out of 979 AI generated images.

Real Lab Coat Imagery vs. Prediction Confidence

AI Lab Coat Imagery vs. Prediction Confidence

54.40%

Confidence

Why the higher confidence? More focused foregrounds? Less background noise?

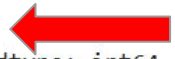
What do plate(s) look like?

```
# Look up Real unique values
real_uniques = df_merged[df_merged['label'] == 0]['predict_1'].value_counts()
real_uniques.head()
```

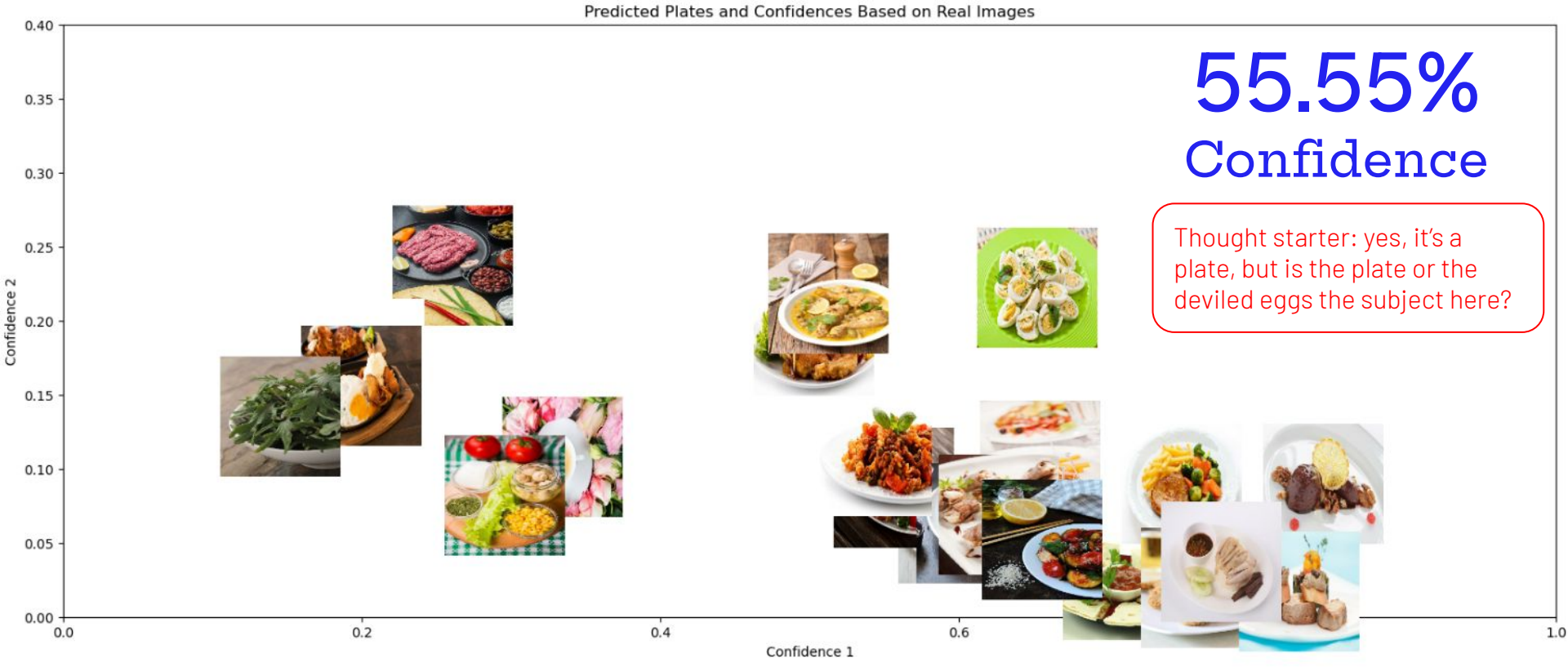
```
predict_1
lab_coat    34
plate       23
jean        22
bikini      15
alp         14
Name: count, dtype: int64
```

```
# Look up AI counts for plate
ai_plate_uniques = df_merged[(df_merged['label'] == 1) & (df_merged['predict_1'] == "plate")]['predict_1'].value_counts()
ai_plate_uniques
```

```
predict_1
plate    14
Name: count, dtype: int64
```



Real Plate Imagery vs. Prediction Confidence

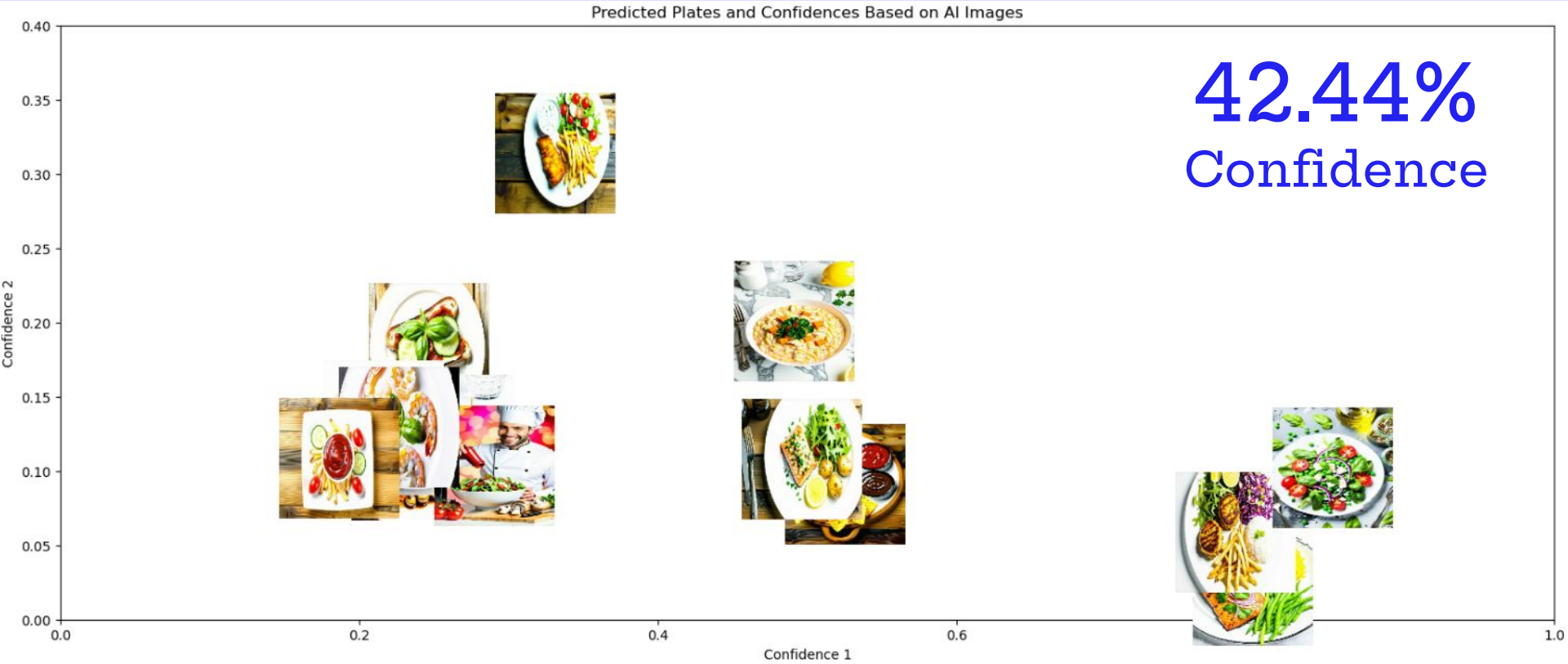


55.55%

Confidence

Thought starter: yes, it's a plate, but is the plate or the deviled eggs the subject here?

AI Plate Imagery vs. Prediction Confidence

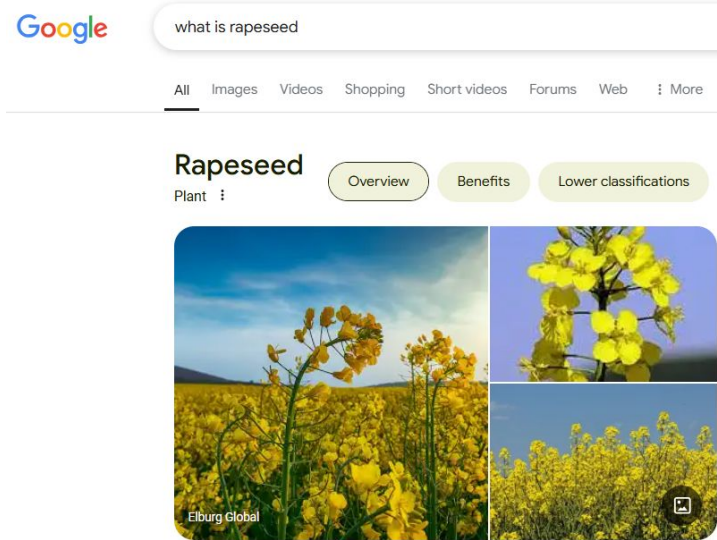


42.44%
Confidence

Finally, what is rapeseed and why did AI over-index with it?

```
# Look up real counts for rapeseed
real_rapeseed_uniques = real_rapeseed[(real_rapeseed['label'] == 0) & (real_rapeseed['predict_1'] == "rapeseed")]['predict_1'].value_counts()
real_rapeseed_uniques
```

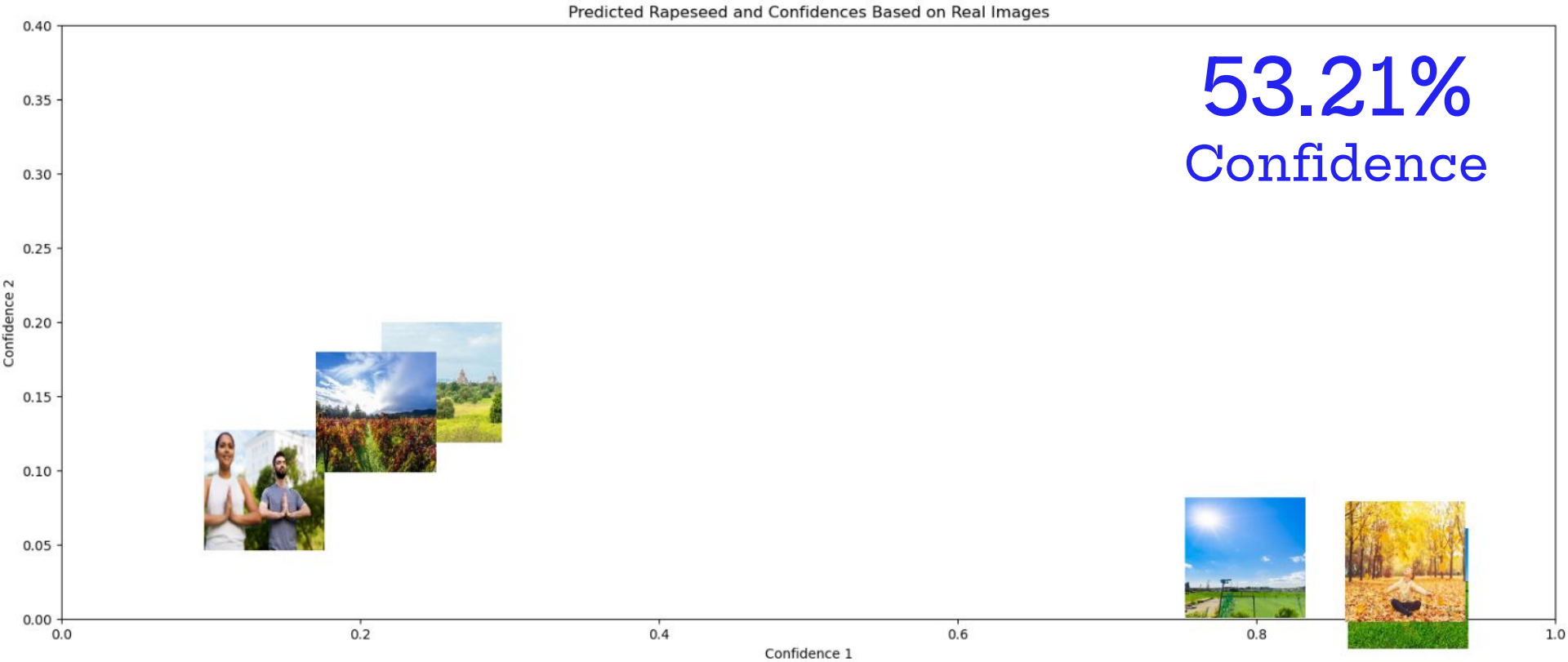
```
predict_1
rapeseed    6
Name: count, dtype: int64
```




```
# Look up AI unique values
ai_uniques = df_merged[df_merged['label'] == 1]['predict_1'].value_counts()
ai_uniques.head()
```

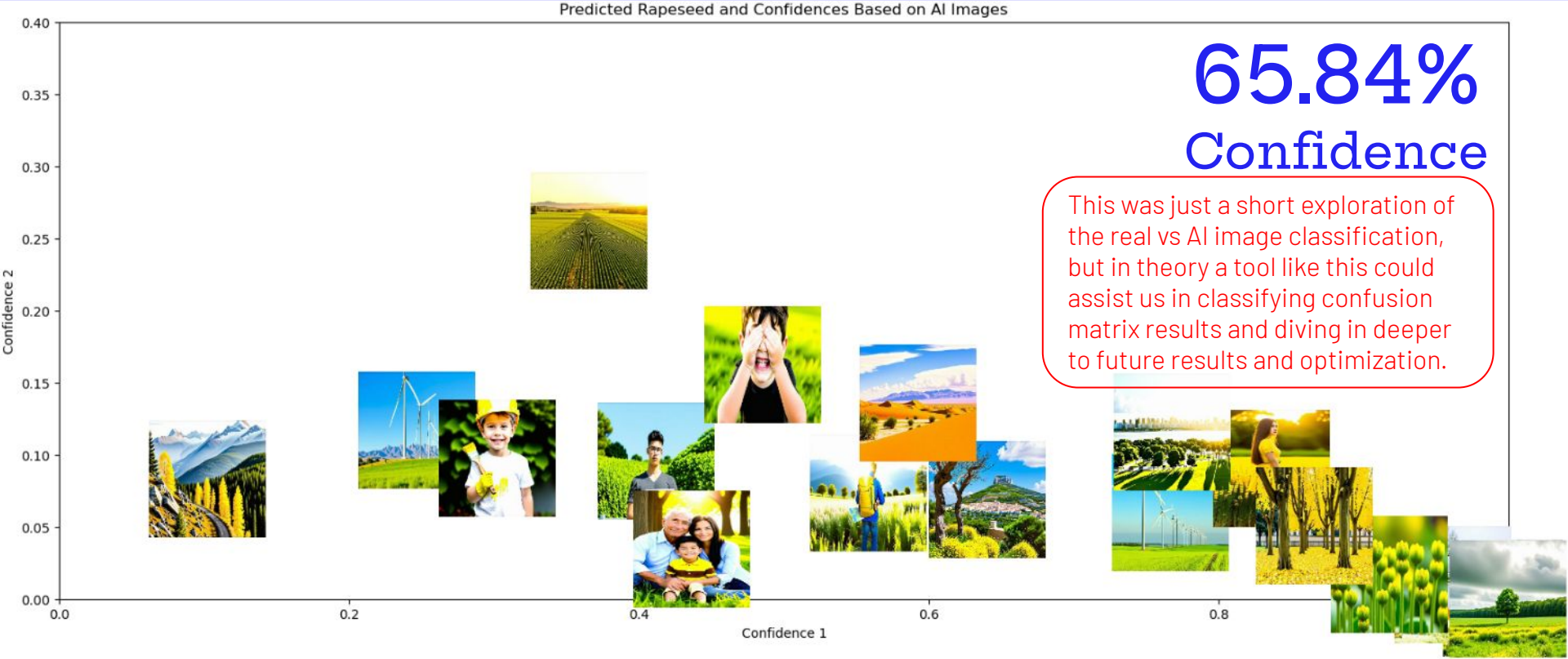
```
predict_1
lab_coat    28
rapeseed    21
wig         19
cucumber    14
maillot     14
Name: count, dtype: int64
```

Real Rapeseed Imagery vs. Prediction Confidence



53.21%
Confidence

AI Rapeseed Imagery vs. Prediction Confidence



65.84%

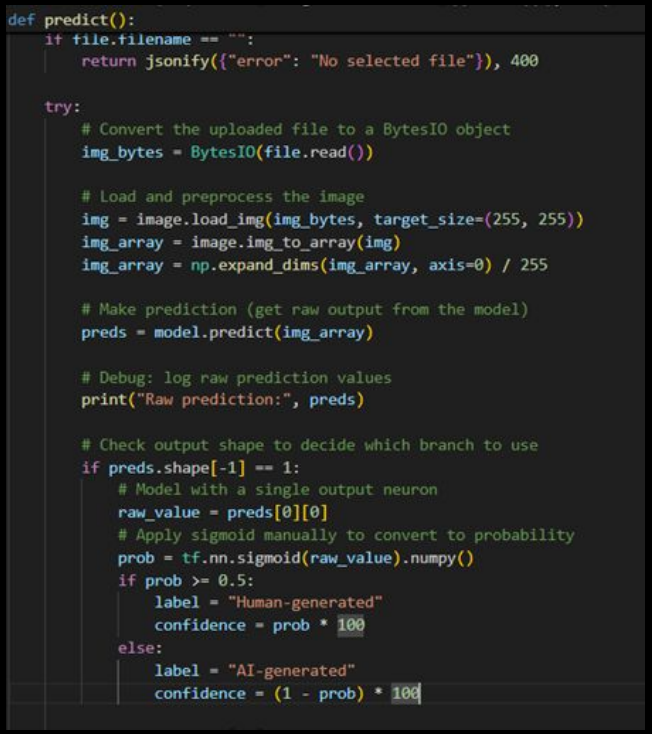
Confidence

This was just a short exploration of the real vs AI image classification, but in theory a tool like this could assist us in classifying confusion matrix results and diving in deeper to future results and optimization.

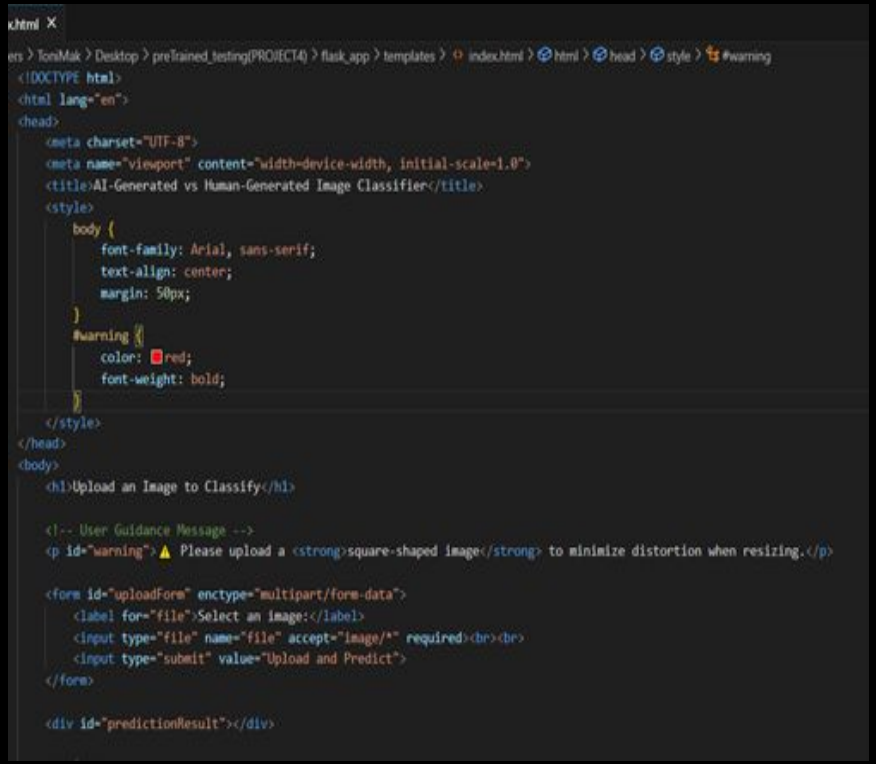
FLASK-APP BUILD

We wanted to be able to create a webpage that would be able to predict uploaded images based on the train model created on whether an image was AI-generated or Human-generated.

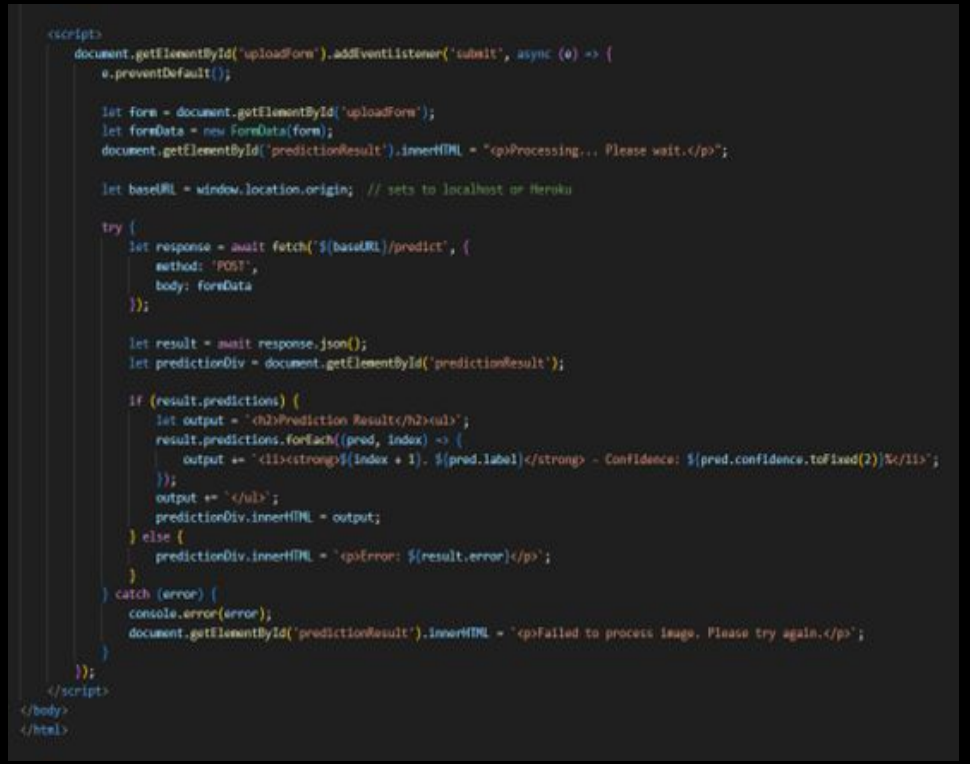
The app.py was built so it would be able to run either locally with python for single uploads or deployed with a cloud platform service (heroku) that would be able to support multiple uploads at the same time.



INDEX.HTML BUILD

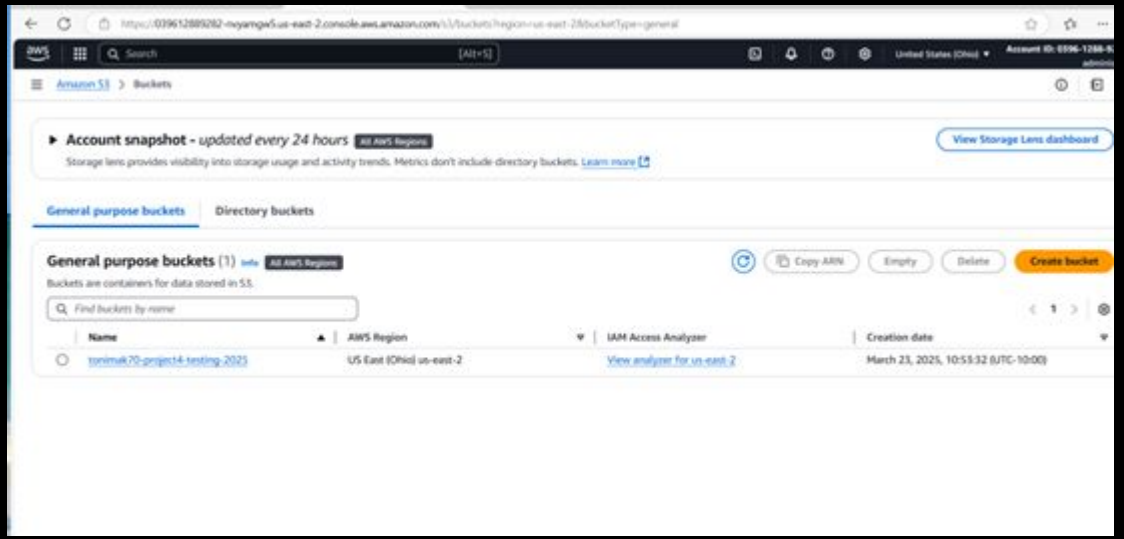


```
.html X
ers > ToniMak > Desktop > preTrained_testing(PROJECT4) > flask_app > templates > index.html > html > head > style > #warning
<!DOCTYPE html>
<html lang="en">
<head>
    <meta charset="UTF-8">
    <meta name="viewport" content="width=device-width, initial-scale=1.0">
    <title>AI-Generated vs Human-Generated Image Classifier</title>
    <style>
        body {
            font-family: Arial, sans-serif;
            text-align: center;
            margin: 50px;
        }
        #warning {
            color: red;
            font-weight: bold;
        }
    </style>
</head>
<body>
    <h1>Upload an Image to Classify</h1>

    <!-- User Guidance Message -->
    <p id="warning">⚠ Please upload a <strong>square-shaped image</strong> to minimize distortion when resizing.</p>

    <form id="uploadForm" enctype="multipart/form-data">
        <label for="file">Select an image:</label>
        <input type="file" name="file" accept="image/*" required><br><br>
        <input type="submit" value="Upload and Predict">
    </form>

    <div id="predictionResult"></div>
```



```
    <script>
        document.getElementById('uploadForm').addEventListener('submit', async (e) => {
            e.preventDefault();

            let form = document.getElementById('uploadForm');
            let formData = new FormData(form);
            document.getElementById('predictionResult').innerHTML = "<p>Processing... Please wait.</p>";

            let baseURL = window.location.origin;  // sets to localhost or Heroku

            try {
                let response = await fetch(`${baseURL}/predict`, {
                    method: 'POST',
                    body: formData
                });

                let result = await response.json();
                let predictionDiv = document.getElementById('predictionResult');

                if (result.predictions) {
                    let output = `<h2>Prediction Result</h2><ul>`;
                    result.predictions.forEach((pred, index) => {
                        output += `<li><strong>${index + 1}. ${pred.label}</strong> - Confidence: ${pred.confidence.toFixed(2)}%</li>`;
                    });
                    output += `</ul>`;
                    predictionDiv.innerHTML = output;
                } else {
                    predictionDiv.innerHTML = `<p>Error: ${result.error}</p>`;
                }
            } catch (error) {
                console.error(error);
                document.getElementById('predictionResult').innerHTML = `<p>Failed to process image. Please try again.</p>`;
            }
        });
    </script>
</body>
</html>
```

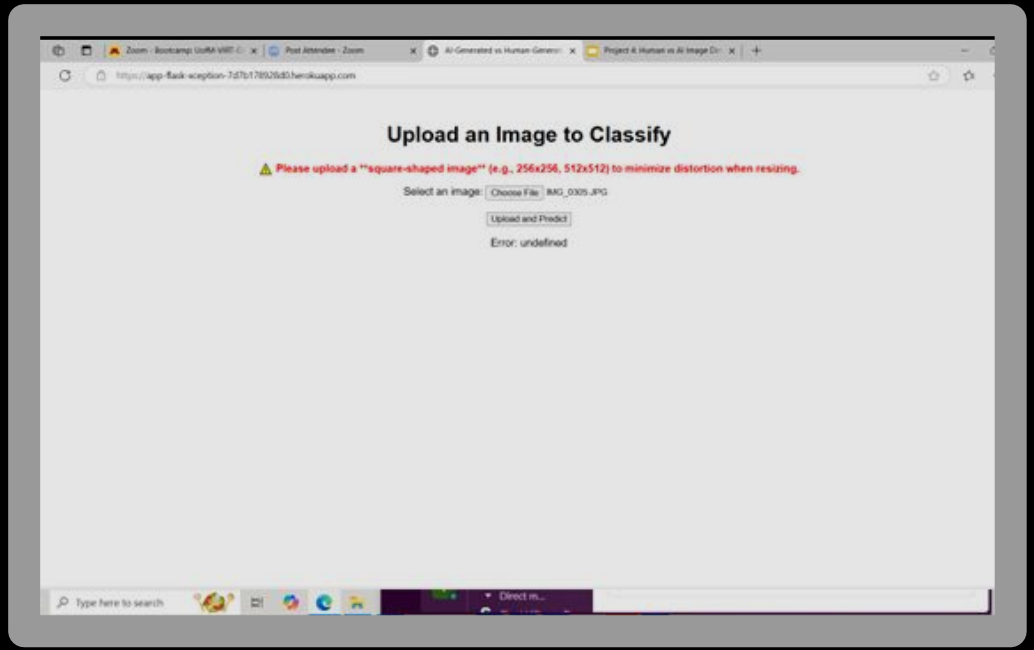
CLOUD STORAGE SERVICES



Used AWS S3 cloud storage used to store and retrieve our large data files for the heroku deployment

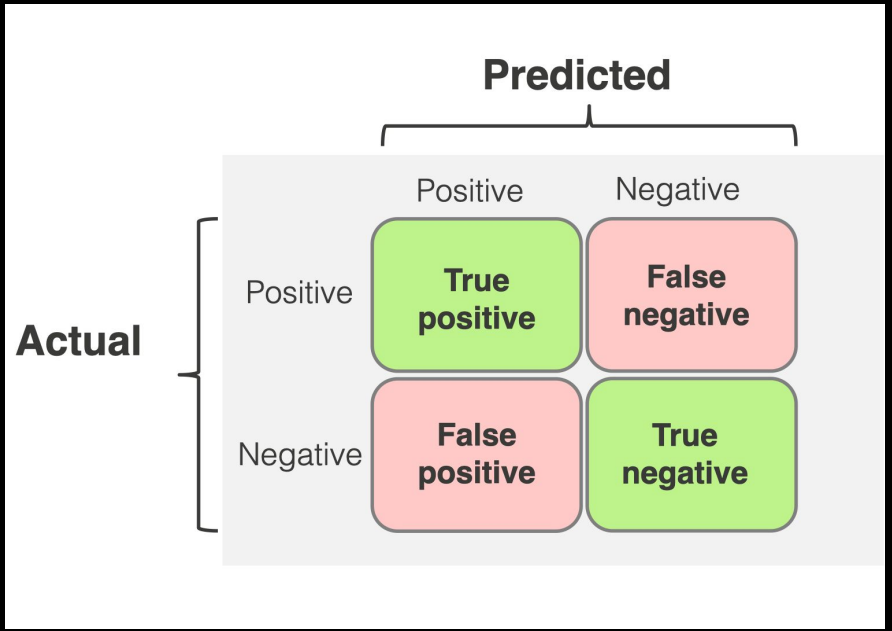
CLOUD PLATFORM SERVICES

https://app-flask-xception-7d7b178928d0.herokuapp.com/



Used HEROKU, a cloud platform, to deploy our application with the intent to have multiple users upload images at the same time.

But wait, what would we do if we had more time to investigate?



Study the results more closely

- Confusion matrix of images
- Shared characteristics across False Positives and False Negatives
- Build visualisations to further understand the model's accuracy in relation to real vs. AI images.

Questions?

THANK YOU