Fooling a detector

"Given a database of synthetic images, design a strategy that fools synthetic image detectors."

Students
Michele Gusella 2122861
Riccardo Miele 2116946

Goal

We want to fool a detector and to do so we have used different strategies:

- Gaussian noise
- Compression
- Rotation
- Mirroring
- Blurring
- Addition of different geometric shapes

Dataset

The dataset is composed of different images of airplanes. Images from 0 to 199 are real while images from 200 to 399 are fake.

aerei/100.png

aerei/156.png

aerei/247.png

aerei/331.png

CLASSIFIER

ResNet50 NoDown: GAN generated image detector

https://github.com/grip-unina/GANimageDetection

```
!python ./GANimageDetection/main.py -m ./weights/gandetection_resnet50nodown_stylegan2.pth -i ./aerei/ -o out_aerei_fr.csv

GAN IMAGE DETECTION
START
  399/400
DONE
OUTPUT: out_aerei_fr.csv
```

Output of the classifier

The output is a .csv file. "Logit" is the prediction:

- Fake : logit>0 | Label = 1
- Real : logit<0 | Label = 0

```
print_raw_csv('out_aerei_fr.csv')
```

	filename	logit	time
0	./aerei/0.png	-30.757368	0.221315
1	./aerei/1.png	-23.438070	0.078551
2	./aerei/2.png	-17.342331	0.065130
3	./aerei/3.png	-10.875802	0.052751
4	./aerei/4.png	-29.655174	0.053619
..	...	...	...
395	./aerei/395.png	9.291558	0.053981
396	./aerei/396.png	11.512779	0.054324
397	./aerei/397.png	19.269798	0.054501
398	./aerei/398.png	14.654418	0.054354
399	./aerei/399.png	18.869556	0.054823

[400 rows x 3 columns]

Loss function

We have used the 0-1 loss defined as:

$$loss = \frac{number.of.misclassified.elements}{number.of.all.elements}$$

So the accuracy of the classification is: $accuracy = 1 - loss$

```
# Create ground truth labels as an array with 200 zeros followed by 200 ones because we know what written above
ground_truth_airplanes = np.concatenate((np.zeros(200), np.ones(200)))

accuracy_airplanes = 1-zero_one_loss('out_aerei_fr.csv', ground_truth_airplanes)
print(f'Accuracy of predictor on airplanes dataset without any type of attack: {accuracy_airplanes}')
```

```
Accuracy of predictor on airplanes dataset wihtout any type of attack: 1.0
```

How it works

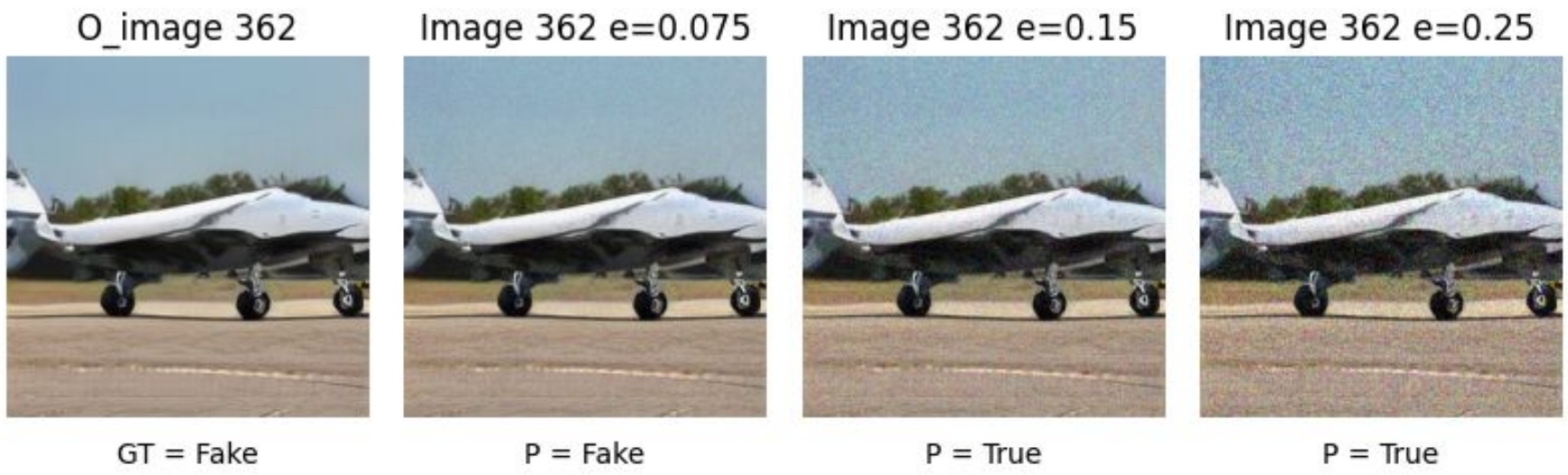
In each section there is the definition of a function that creates the modified image.

Then there is the creation of a folder with new images modified with different values of the parameter of the modification and the invocation of the classifier that outputs a .csv.

In the end, we have decided to plot a graph illustrating the accuracy in relation to the modification values.

Test Gaussian Noise

This image below shows the original image alongside perturbed versions generated with different values of epsilon. Under each image there is the corresponding prediction/ground truth.



Function for Gaussian Noise

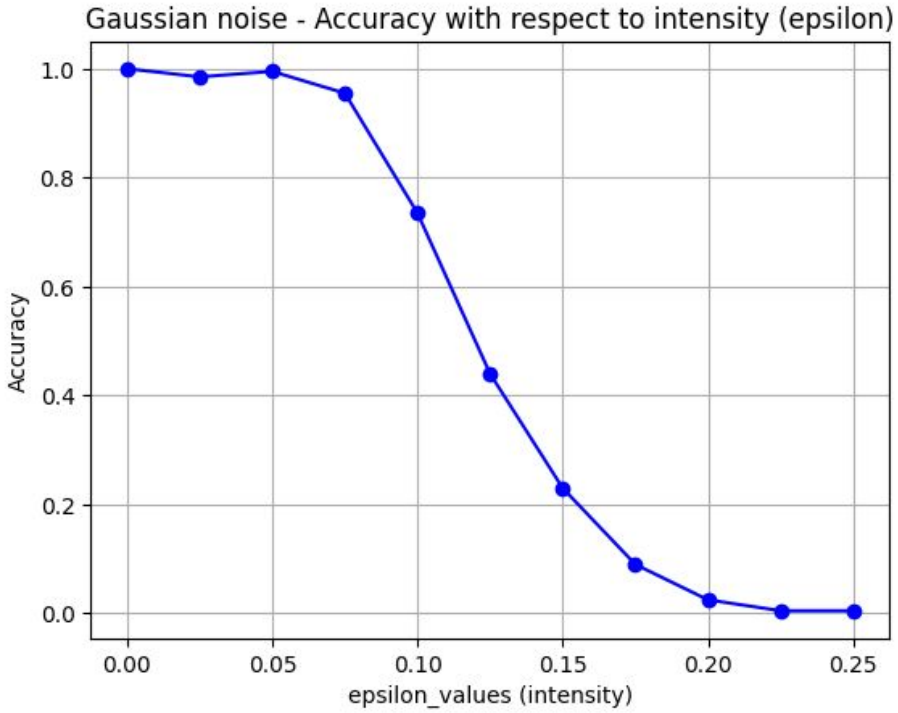
There are two functions: one creating a gaussian noise and one using the noise for the creation of the perturbed image.

```python
def generate_gaussian_noise(image_shape, standard_deviation):
  noise = np.random.normal(0, standard_deviation, size=image_shape)
  return noise

def apply_gaussian_noise(image, epsilon, standard_deviation):
  noise = generate_gaussian_noise(image.shape, standard_deviation)
  noisy_image = image + epsilon*noise
  noisy_image = np.clip(noisy_image, 0, 255).astype(np.uint8)
  return noisy_image
```

Result for Gaussian Noise

The line graph shows how accuracy decreases respect to epsilon values.

The accuracy reaches the zero with an epsilon value equal to 0.25.



Test Compression

This image below shows the original image alongside perturbed versions generated with different quality factors.

Function for Compression

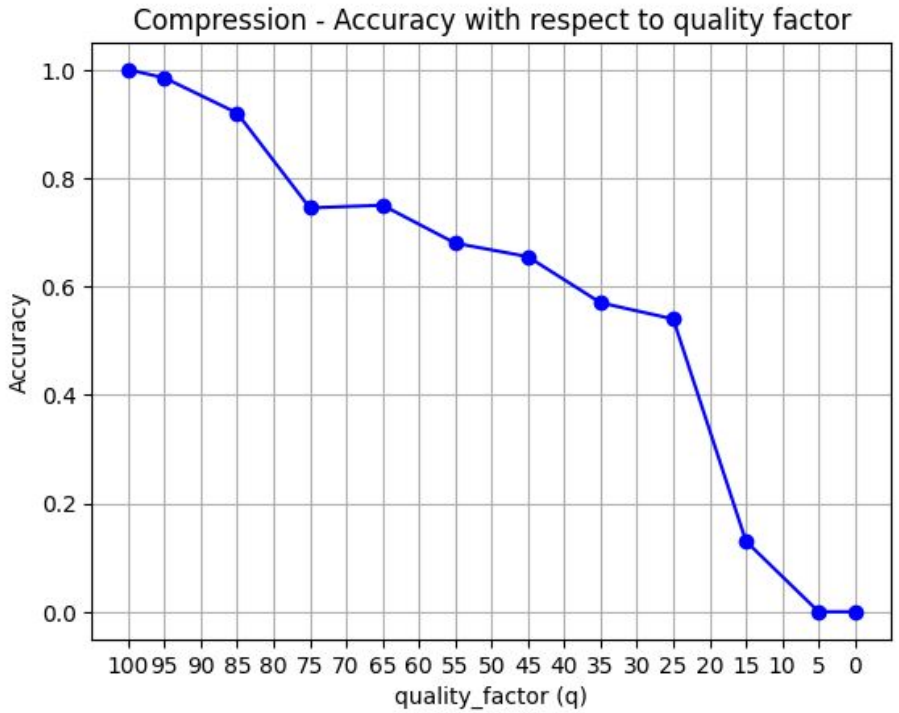
This function applies a JPEG compression to the image using a given quality factor.

```python
def compress_image(image, quality_factor):

    pil_image = Image.fromarray(image)
    compressed_buffer = BytesIO()
    pil_image.save(compressed_buffer, format='JPEG', quality=quality_factor)
    compressed_image = np.array(Image.open(compressed_buffer))

    return compressed_image
```

Results for Compression

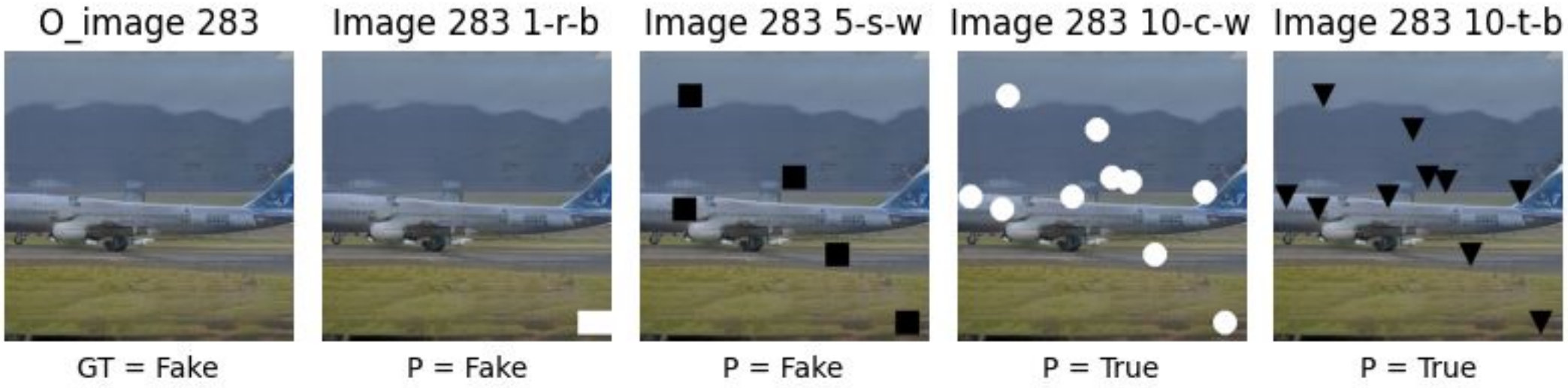
The graph shows that the accuracy decreases increasing the value of the quality factor.

Remember that high quality factor means that the image has a low compression while low quality factor means high compression.



Test Addition of Geometric shapes

This image below shows the original image alongside corresponding perturbed images that are modified with different shapes, numbers and colors.



Function for Geometric Shapes

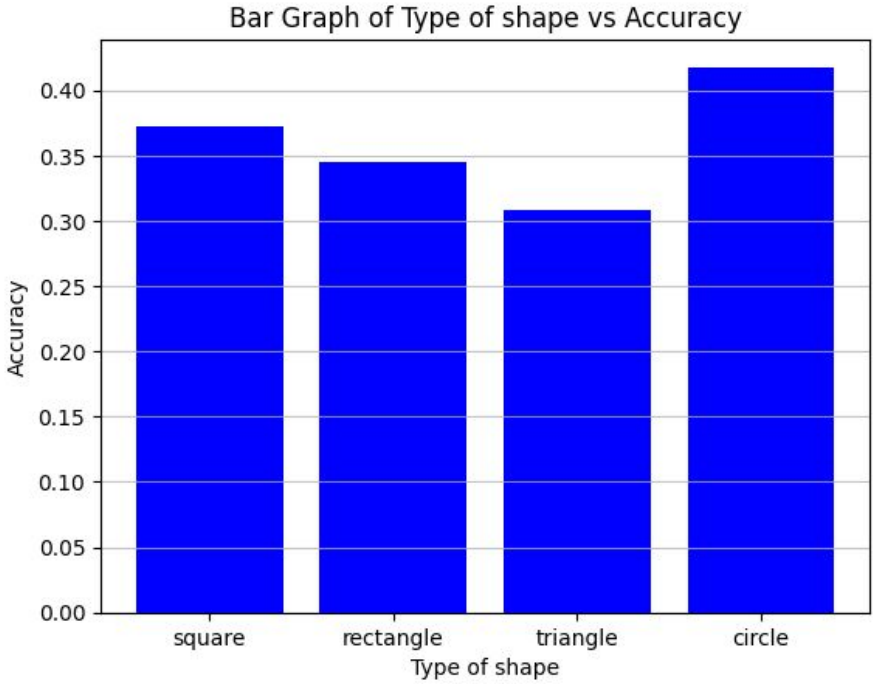
This function returns a modified image containing a given number of shapes of shape_type. The shape can be of different colors.

```python
def add_shapes_to_image(image, number_of_shapes, shape_type, size, color, random_seed):
  np.random.seed(random_seed)
  new_image = np.copy(image)
  pil_image = Image.fromarray(new_image)
  draw = ImageDraw.Draw(pil_image)
  img_width, img_height = new_image.shape[:2]

  for _ in range(number_of_shapes):
    x = np.random.randint(0, img_width - size)
    y = np.random.randint(0, img_height - size)

    if shape_type == 'triangle':
        points = [(x, y), (x + size, y), (x + size // 2, y + size)]
        draw.polygon(points, fill=color)
    #the same step is done for the other types of shape
  new_image = np.array(pil_image)
  return new_image
```

Results on the type of shape used

The bar graph shows that the type of shape used in the modification influences the accuracy.

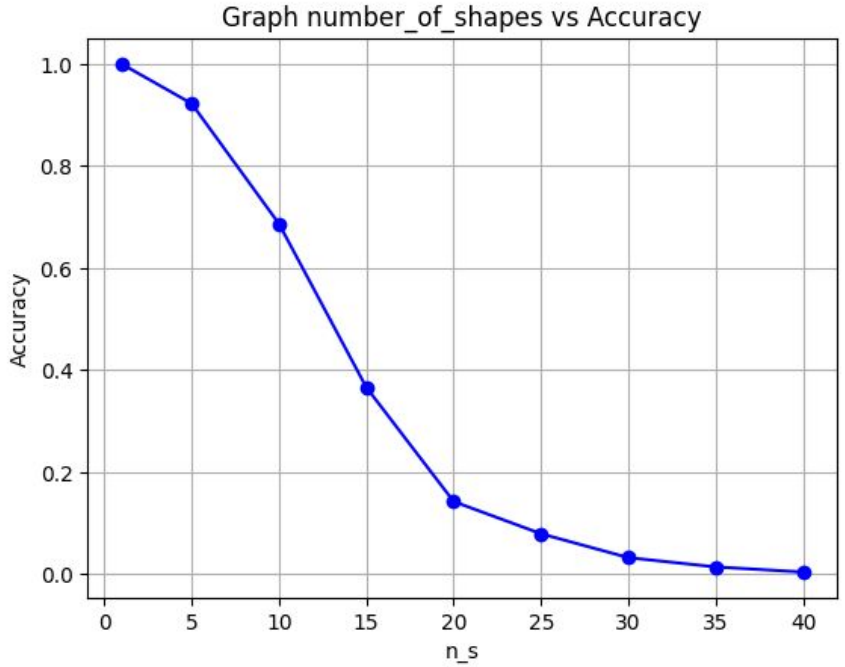
In fact, the difference of accuracy between circles and triangles it's 11%.



Results on the number of shapes

The line graph on the right shows that the number of shapes influences the accuracy.

In fact, increasing the number of shapes leads to a drop of the accuracy.



Conclusion

We have tested different types of perturbations and we have found that:

- Gaussian noise (with **e=0.25**), compression (with **q=5**) and addition of geometric shapes (with **n_s=40**) fool the detector for all the images.
- Rotations and blurring are not so effective and fool the detector with only few images.
- Mirroring is not effective to fool the detector.

THANK YOU FOR YOUR ATTENTION