DEEP FAKE IMAGE DETECTOR ROBUST AGAINST ADVERSARIAL ATTACKS

Course ID: CSE 472

1905064 - Sayem Shahad
1905065 - Arnab Bhattacharjee

1. Problem Definition:

Models like YOLO, ViT can classify between fake and real images. But adding noises in the images may make the models misclassify. These noises can be added in pixel values, frequency domains, along the gradient loss, and many more. Some countermeasures can be designed to make the model robust against these attacks.

Countermeasures can be designed by:

1) **Modifying the training data**: feature drop, augmentations, smoothing, adding random noises in training images, normalization, etc.
2) **Modifying the model:** Dropout layer, Adversarial noise layer after the input, regularizations, etc.

2. Our goal:

Step-1: Measure the performance of different model before adding noise in test data

Step-2: Measure the performance drop of the models after adding noise to the test data.

Step-3: Add countermeasures to the input images, and the models.

Countermeasures taken:

1) Least correlated features are dropped
2) Some augmentations (rotating, zooming, cropping)
3) Smoothing
4) Custom Adversarial noise layer in YOLOv11
5) Custom Dropout Layer in YOLOv11

Step-4: measure the increase or decrease in performance of the models after adding the countermeasures.

For the notebooks, here is our github:
https://github.com/Arnabbndc/CSE-472-ML-Project

3. Experiments:

2 major categories:

1. Black Box Experiments
2. White Box Experiments

3.1. Black Box Experiments

Experiment 1: (Initial analysis)

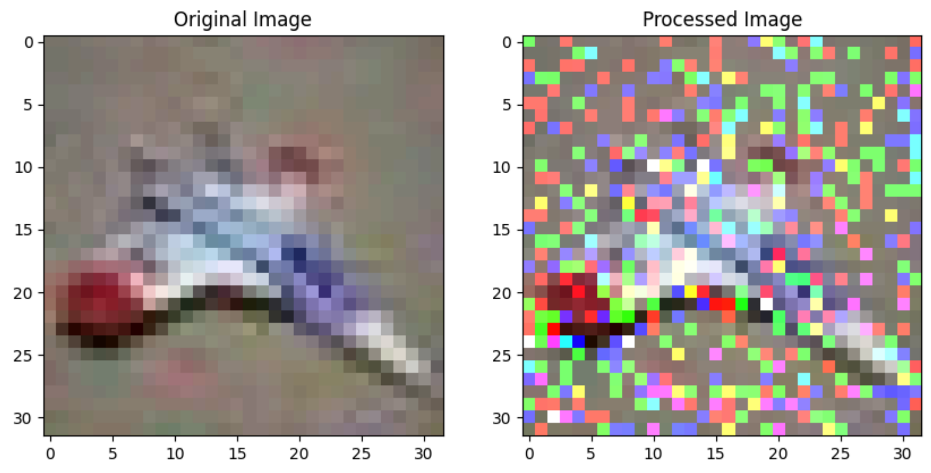
At first, we trained a number of models on the CiFake Dataset. This dataset consists of 60,000 synthetically-generated images and 60,000 real images.
The results are:

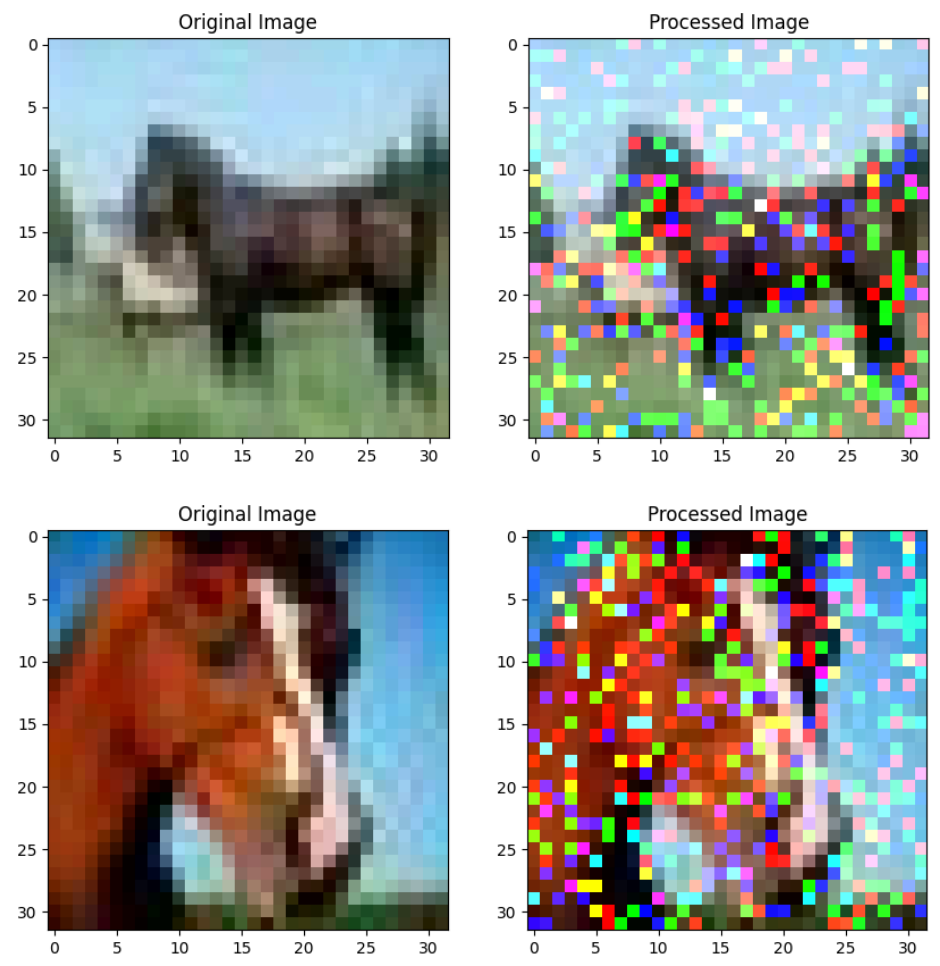
Model	Accuracy
ViT	98%
EfficientNet	88%
YOLO-v11x	92%

We proceeded with our further experiments with the YOLO-v11x model.

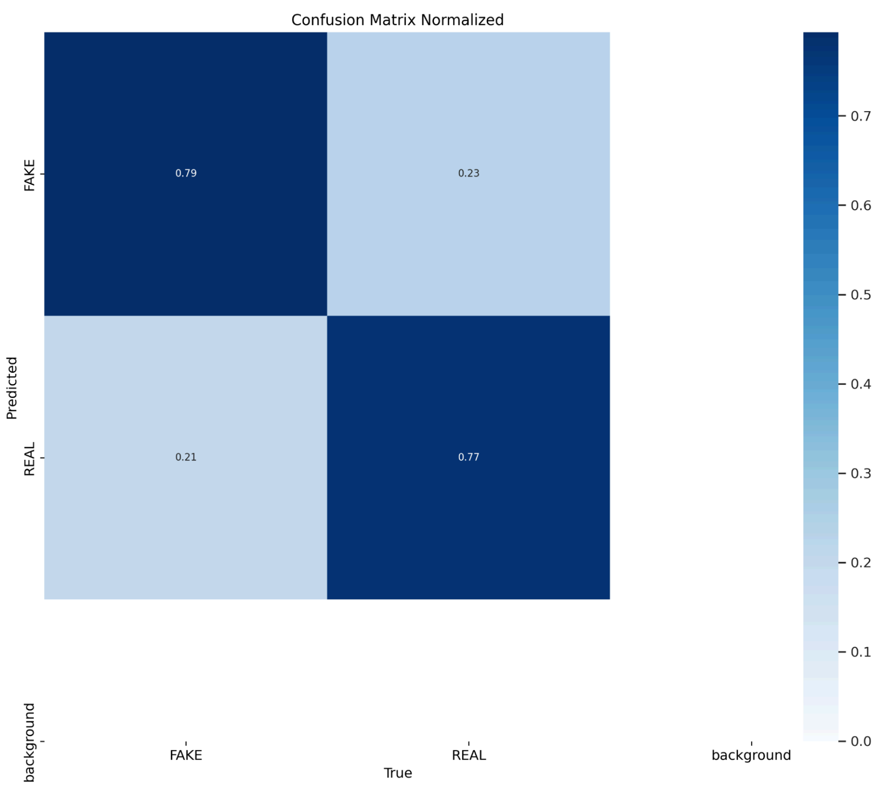
Experiment 2:

We added gaussian noise to the Test dataset images and evaluated the YOLO model's performance.
Mean: 0.0, Standard Deviation: 1.0
Some sample processed test images:





After keeping the same training setup, we achieved **accuracy 78%.**
Here is the confusion matrix for its performance:



Analysis: Adding noise drops model's performance for classifying REAL and FAKE

Experiment 3: (A step towards making the model robust)

1. **Approach:**
   Feature Dropping with least correlations:

   RUN 1: Max correlation: 0.2645, Min correlation: 0.00054
   Threshold for correlation used: 0.02, pixels dropped: 12%
   Accuracy: 70%

   RUN 2: Threshold used: 0.04 pixels dropped: 28%
   Accuracy: 70%

Feature dropping used in noisy test data::

Standard deviation used to add gaussian noise	**Pixel drop percentage**	**accuracy**
25	**25% (threshold 0.035)**	**68%**
25	**17%**	**74%**

Observation: Feature dropping isn't a good idea.

2. **Approach: Resolution Reduction**
   RUN: stdev of gaussian noise: 25.0, pixels dropped: 12%

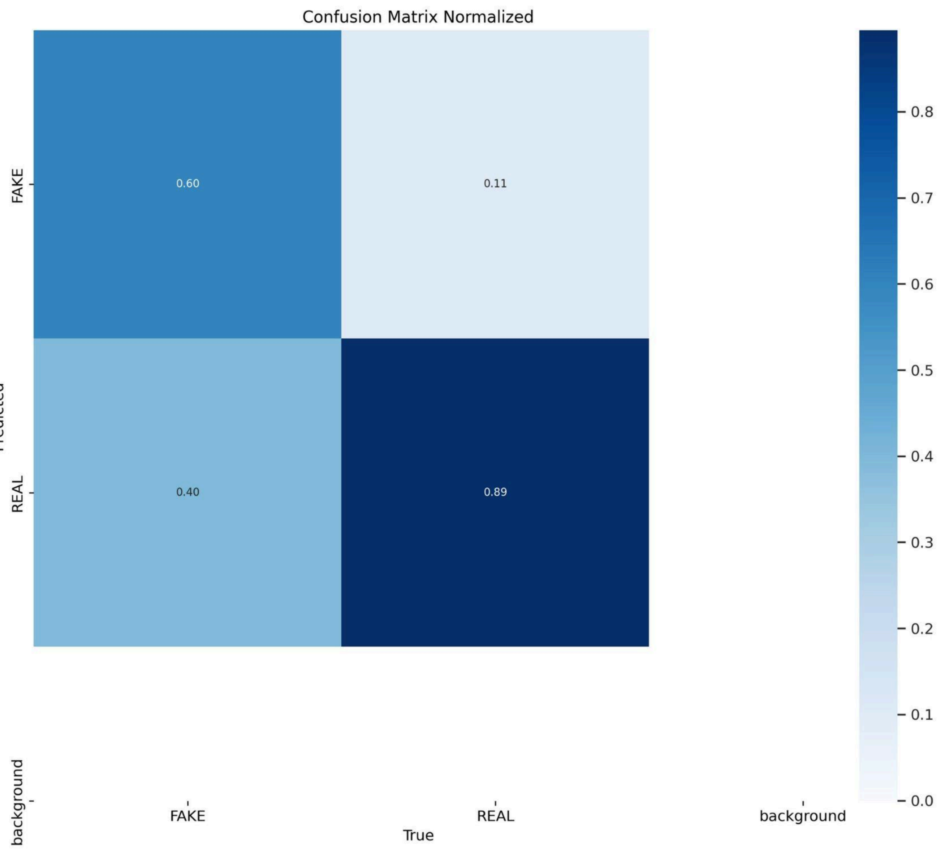
With resolution Reduction	69%
Without resolution Reduction	71%

Observation: The resolution of the CiFake dataset images are very low (32 * 32). So, reducing resolution is not a good choice here. And again, feature (pixels) dropping is also not recommended since the images contain only 32 * 32 pixels. Dropping a portion of them will lead to loss of significant information.

3. **Approach: Feature Dropping + Augmentations**
   Way-1: First feature dropping, then augmentations
   stdev: 25.0 threshold: 0.02 pixels dropped: 9%
   Accuracy: 68%

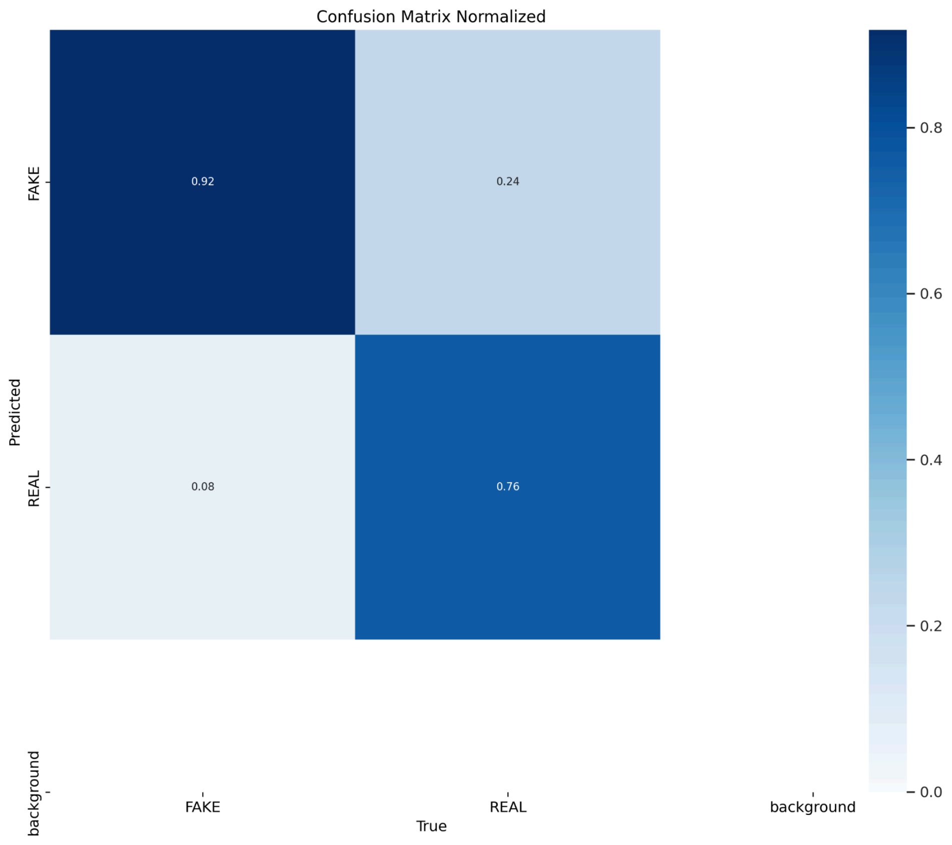
   Way-2: First augmentations, then feature dropping
   stdev: 25.0 threshold: 0.015, pixels dropped: 9%
   Accuracy: 67.9%

   Best run:
   Stdev for gaussian noise: 1.0,threshold: 0.01, pixels dropped: 5%
   Accuracy: 73%
   Confusion matrix for this RUN:



Observation: Only augmentations cannot make the model robust against the attacks with gaussian noise. Rather, this approach degrades the performance. Overfitting can be a possible reason behind it.

4. **Approach: Augmentations +Feature Dropping + Adversarial Training**
   50% of the augmentations in train set were done by gaussian noise
   Best run: stdev: 1.0, threshold: 0.015, pixels dropped: 9%
   Accuracy: 0.85
   Confusion matrix for this experiment:

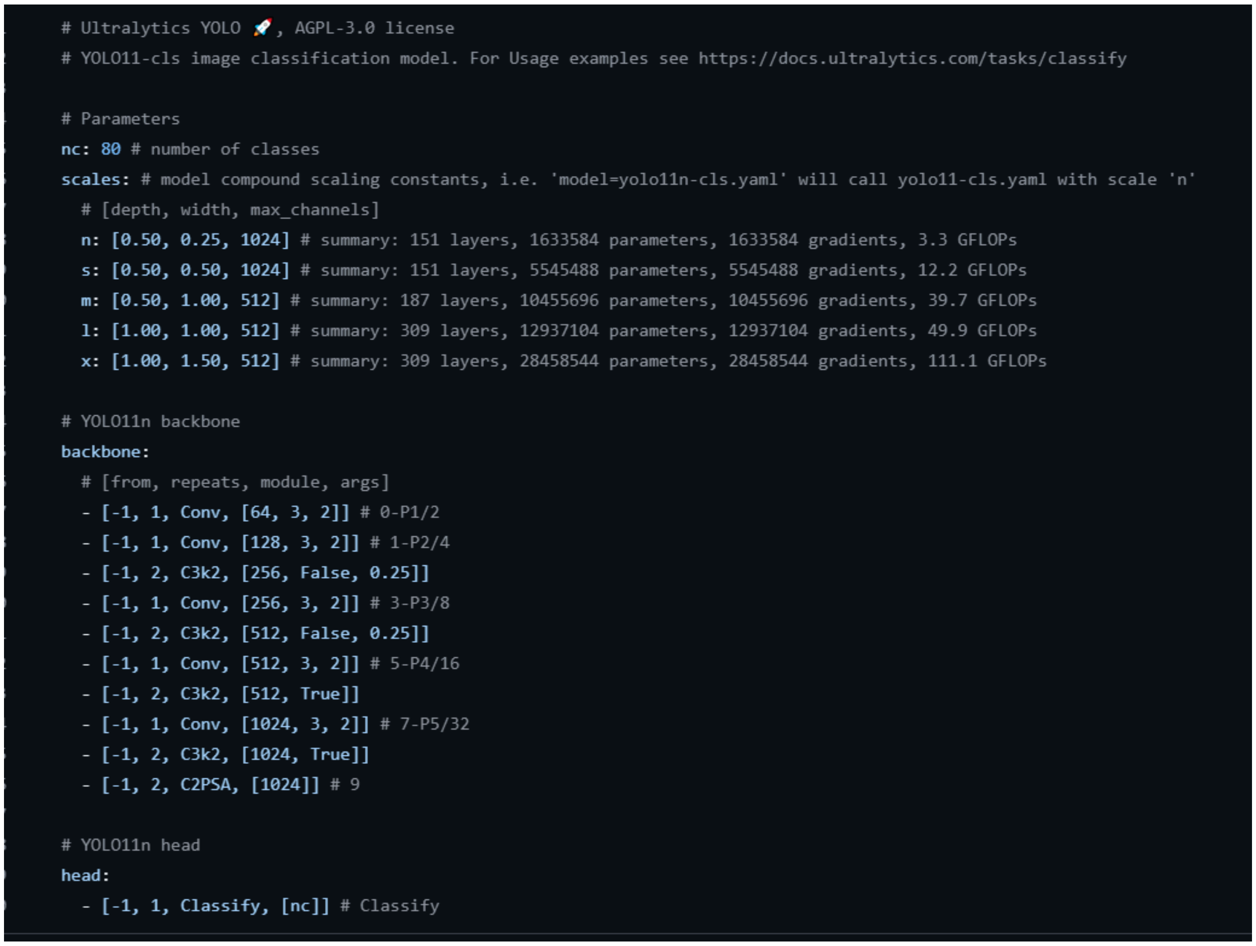


Observation: To make the model's performance better against gaussian noise attack, training should be done with images having gaussian noise in it.

3.2. White Box Experiments

Here, we designed custom layers for YOLO-V11-cls model and tried two ways: dropout and adding adversarial noise.

The architecture of YOLOv11-cls (classification) model:



```
# Ultralytics YOLO 🚀, AGPL-3.0 license
# YOLO11-cls image classification model. For Usage examples see https://docs.ultralytics.com/tasks/classify

# Parameters
nc: 80 # number of classes
scales: # model compound scaling constants, i.e. 'model=yolo11n-cls.yaml' will call yolo11-cls.yaml with scale 'n'
  # [depth, width, max_channels]
  n: [0.50, 0.25, 1024] # summary: 151 layers, 1633584 parameters, 1633584 gradients, 3.3 GFLOPs
  s: [0.50, 0.50, 1024] # summary: 151 layers, 5545488 parameters, 5545488 gradients, 12.2 GFLOPs
  m: [0.50, 1.00, 512] # summary: 187 layers, 10455696 parameters, 10455696 gradients, 39.7 GFLOPs
  l: [1.00, 1.00, 512] # summary: 309 layers, 12937104 parameters, 12937104 gradients, 49.9 GFLOPs
  x: [1.00, 1.50, 512] # summary: 309 layers, 28458544 parameters, 28458544 gradients, 111.1 GFLOPs

# YOLO11n backbone
backbone:
  # [from, repeats, module, args]
  - [-1, 1, Conv, [64, 3, 2]] # 0-P1/2
  - [-1, 1, Conv, [128, 3, 2]] # 1-P2/4
  - [-1, 2, C3k2, [256, False, 0.25]]
  - [-1, 1, Conv, [256, 3, 2]] # 3-P3/8
  - [-1, 2, C3k2, [512, False, 0.25]]
  - [-1, 1, Conv, [512, 3, 2]] # 5-P4/16
  - [-1, 2, C3k2, [512, True]]
  - [-1, 1, Conv, [1024, 3, 2]] # 7-P5/32
  - [-1, 2, C3k2, [1024, True]]
  - [-1, 2, C2PSA, [1024]] # 9

# YOLO11n head
head:
  - [-1, 1, Classify, [nc]] # Classify
```

Different modules are defined in nn/modules folder in different python files. We need to write our own modules.

IDEAL PLACE TO ADD OUR CUSTOM LAYERS:

A dropout layer was always added after a conv layer
adversarial noise layer was added after the input and before the initial conv layer

Dropout_layer module:

```
import torch
import torch.nn as nn

class Dropout(nn.Module):
    def __init__(self, p=0.5):
        super().__init__()
        self.dropout = nn.Dropout(p)

    def forward(self, x):
        return self.dropout(x)
```

Adversarial_noise_layer module:

```
import torch
import torch.nn as nn
import torch.nn.functional as F

class AdversarialNoise(nn.Module):
    def __init__(self, epsilon=0.1, alpha=0.01, num_steps=1):
        super().__init__()
        self.epsilon = epsilon
        self.alpha = alpha
        self.num_steps = num_steps

    def forward(self, x, model=None, target=None):

        if not self.training or model is None or target is None:
            return x

        x_adv = x.clone().detach().requires_grad_(True)

        for _ in range(self.num_steps):
            outputs = model(x_adv)
            loss = F.cross_entropy(outputs, target)
            loss.backward()
            grad_sign = x_adv.grad.sign()
            x_adv = x_adv + self.alpha * grad_sign
            perturbation = torch.clamp(x_adv - x, min=-self.epsilon,
max=self.epsilon)
            x_adv = torch.clamp(x + perturbation, min=0, max=1).detach()
            x_adv.requires_grad = True
        return x_adv
```

Accuracy on noisy test data for default YOLOv11 model	78%
Accuracy on noise data after adding adversarial noise layer	80%
Accuracy on noise data after adding 3 Dropout layers	73%

Observation :
Adding a adversarial noise layer after the input and before the first convolution layer, the accuracy of the model against noisy test data increases.