# ADL - AI Generated Image Detection
# *(Bring your own method)*

Anna Laczkó
Matriculation number: 12347556

## 1 Chosen Problem

My chosen topic was Detecting AI-generated images. With my Facebook feed increasingly filled with AI-generated content, I recognized the growing challenge of distinguishing these from genuine images. While I've personally become adept at identifying such content, I observed that many of my friends and family struggled, often falling victim to clickbait posts.

During developing my project I tried several datasets and methods to realize this task. After several trial and error I only found one dataset that was recent, large and consistent enough to be usable. However, it had a downside. This dataset only consist of human faces, so because of this, I had to limit my project's capabilities to be able to identify fake pictures of faces. Honestly, I don't really mind as most of the AI photos we have a problem with are the images that try to imitate humans (and they are really successful at that), and most conflicts would also arise from these.

## 2 Solution

My solution is a basic application that can classify photos of human faces as AI-generated. In the end I trained my original dataset's 95% to get a hopefully quite accurate model. The rest is my test set that can be used to try out the application. The application can be used with self-sourced images too, as the UI interface includes a cropping function so we can crop the picture to only contain the faces. This is really important as the model was only trained such photos that only included a face from close-up.

### 2.1 Network Design

The structure is simple; I have four convolutional layers and two fully connected layers. I focused on optimising the layers. For this, I did some research. The paper on CIFAKE mentioned that most AI images are recognised not by the central object but by the surrounding more minor details. This is why I tried to shape my network to pick up those details:

- I added as many channels as I could
- I added a 5x5 kernel, but I added stride, as I thought that most of the details would be repetitive (and this is also from my own experience), and it would be redundant to go one by one.
- I stayed away from pooling

As I mentioned before, I tried to use smaller pictures. I also thought that if my time allowed it, I would train on three different sizes and then do majority voting. With this, I could even extract different features from the images, but sadly, the lower-resolution pictures were not that effective.

Another attempt at an Ensemble Model was an idea that I found in [this] paper. Here, they tried to make filtered versions of the pictures, highlighting edges and sharpening the images and using that to identify differences between real and AI-generated images. I implemented the filtration in Preprocess.py, but in the end, due to time constraints, I did not use those images for the final training. However, I did some testing with them, and Edge and Sharpen proved to be quite good.

I still think it would be interesting to make an ensemble with them; however, I did not have time to implement that.

After the code worked in the notebooks, I manually transformed them into .py-s and made some tests for them.

## 3 Goals and Results

### 3.1 Assignment 2

For the second part of the assignment, I chose the standard accuracy to measure my model's success. My goal was to classify at least 75-80% of the pictures correctly from the validation set.

In the end, I achieved a 95.02% accuracy, which was a shock (although a positive one).

For more information, I added a confusion matrix to analyze the accuracy of the two labels.

| | **True 0** | **True 1** |
|---|---|---|
| **Predicted 0** | 18076 | 511 |
| **Predicted 1** | 2924 | 47489 |

Table 1: Confusion Matrix for Assignment 2

### 3.2 Assignment 3

After finishing the third assignment, I wanted to exploit the opportunity to test my model on some self-sourced images, and that's when I realised that it was not nearly as good as I had hoped.

First, the model immediately classified every image (even the ones from the original set!) as real when I cropped a part of the picture (no matter how small the change was). I immediately thought that the generated pictures could have included something unnoticeable at the very edge that identified these pictures quite easily. So, I reactivated the image augmentation part of my code (which does random rotation, random crop, and random color jitter), and I also did a random crop on the original image.

This helped a bit, but sadly I could not get closer to the originally assumed accuracy. I am going to be honest, I also noticed it too late and did not have any time to figure out the reason why, and also collect enough data from a certainly independent (from the 140k faces dataset) source to create a new test set to get a better estimation of the effectiveness.

The model still performs accurately on the test set from the original dataset. This problem only occurs on randomly sourced images from the internet and it also it still changes opinion rapidly when changing the crop size. My biggest suspicion is that this dataset has some Data Leakage going on, and the model picks up on that. If I had time, I would visualise the parts of the images the model uses for classifying.

## 4 Takeaways

- I just mentioned, but certainly one of the biggest take-aways is the realisation of the decreased accuracy. I cannot say for sure but it miss-classified at least 3 out of the 5 randomly chosen AI generated image from the internet.
- For a next model like this, I would try to gather pictures from different sources. I thought I don't need that, as I have more than 850 000 images (with augmentation) just for training.

- Building a UI is really easy if we're using one of the tools.

## 5 Schedule

| Task | Estimated Time | Scheduling | Spent Time | Actual Scheduling |
|---|---|---|---|---|
| **Data Preparation and Research** | 15-20 hours | 23-10-2024 - 30-10-2024 | 30-35 hours | 04-12-2024 - 15-12-2024 |
| **Training the Model(s)** | 20-30 hours | 23-10-2024 - 30-11-2024 | 35-40 hours | 08-12-2024 - 17-12-2024 |
| **Evaluate Performance** | 6-8 hours | 20-11-2024 - 10-12-2024 | 2 hours | 08-12-2024 - 15-12-2024 |
| **Ensemble Model** | 10-12 hours | 20-11-2024 - 10-12-2024 | 5 hours | 13-12-2024 |
| **Comparing Ensemble to Individual Models** | 4-5 hours | 05-12-2024 - 12-12-2024 | 1 hour | 13-12-2024 |
| **Buffer Time** | - | 12-12-2024 - 17-12-2024 | - | - |
| **Documentation for Assignment 2** | 8-10 hours | Simultaneously with every previous step | 6 hours | 14-12-2024 - 17-12-2024 |
| **Building Application** | 20-25 hours | 28-12-2024 - 10-01-2025 | 8 hours | 15-01-2025 - 17-01-2025 |
| **Final Report and Preparing for Presentation** | 7 hours | 10-01-2025 - 15-01-2025 | 5 hours | 16-01-2025 - 21-01-2025 |
| **Buffer Time** | - | 15-01-2025 - 21-01-2025 | - | - |

Table 2: Task Scheduling and Time Tracking

## 6 Setup Instructions

### 6.1 Clone the Repository

git clone https://github.com/your-username/your-repository.git cd your-repository

### 6.2 Download and Extract Data

Download the dataset from here, and extract it to the root of the project directory. This is a restructured version of the [140k Real and Fake Faces] on Kaggle.

### 6.3 Install Dependencies

Open up console in the root folder and run

*pip install -r docs/requirements.txt*

### 6.4 Run the Notebook

To see the results simply open Notebook - Assignment 2.ipynb and run it.

### 6.5 Documentation Access

For the .py files the documentation can be found HERE.

### 6.6 Run Tests

To run tests:

*python -m unittest discover .*

The tests also run automatically on GitHub on 3 differrent python versions.

### 6.7 Run the UI

Open User Interface.ipynb and run the notebook. In the end, the UI will open in your browser. You can uploade one picture at a time and you can see the predicted class and the confidence value next to it.

## 7 LLM Use

I used ChatGPT for the following:

- .ipynb to .py transformation debugging
- Here, ChatGPT was dead on point and found every missed function and typo made when transferring.
- Test generation

- As I haven't used testing in Python before, I needed some help understanding the way it works
- Comment generation
- I commented throughout the whole process. Read the Docs; however, expects the comments to be in another format. Additionally, looking back, I realize that my comments were not enough. So, in the end, during the documentation generation process, I put my whole .py files into ChatGPT and asked to convert my comments into the correct format and extend them if needed.
- General debugging
- Understanding error messages

## 8 References

- Jordan J. Bird, Ahmad Lotfi **2023.** *CIFAKE: Image Classification and Explainable Identification of AI-Generated Synthetic Images* https://arxiv.org/abs/2303.14126
- Ziyi Xi, Wenmin Huang, Kangkang Wei, Weiqi Luo, Peijia Zheng **2023.** *AI-Generated Image Detection using a Cross-Attention Enhanced Dual-Stream Network* https://paperswithcode.com/paper/ai-generated-image-detection-using-a-cross
- Zeyu Lu, Di Huang, LEI BAI, Jingjing Qu, Chengyue Wu, Xihui Liu, Wanli Ouyang **2023.** *Seeing is not always believing: Benchmarking Human and Model Perception of AI-Generated Images* https://paperswithcode.com/pa
is-not-always-believing-benchmarking
- Bird, J.J. and Lotfi, A., 2024. CIFAKE: Image Classification and Explainable Identification of AI-Generated Synthetic Images. IEEE Access.
- https://www.kaggle.com/datasets/xhlulu/140k-real-and-fake-faces/versions/1/data
- https://www.kaggle.com/datasets/birdy654/cifake-real-and-ai-generated-synthetic-images