Detect AI using AI - A User Application for Detecting AI-Generated Images Using Artificial Neural Networks

Aaron Goldstein[1], Iliya Kulbaka[1], Jonathan O'Berry[1], Kris Roker[1]

[1]University of North Florida
n01421643@unf.edu, n01427009@unf.edu, n01445775@unf.edu, n01233188@unf.edu

Abstract

The field of Computer Vision has advanced dramatically in recent years and is becoming more and more readily accessible to the lay user. With the availability of tools such as DALL*E, and Microsoft Image Generator, we are likely to see a continued surge in Artificial Intelligence (AI) Generated Art in the coming years. However, while this technology is awesome in its potential, it comes with a variety of risks to society at large. These risks can range from fraud, and impersonating individuals for purposes of financial and identity theft, to slander to place political figures, celebrities, and even normal people in fake, reputation-damaging, or illegal situations for propaganda, revenge, and general malicious intent. This creates a clear need for the development of AI systems that are capable of recognizing patterns consistent across images generated with Convolutional Neural Networks (CNNs), whether these features are visible to the naked human eye or not. In this paper, we created an easy-to-use, fully functional Web Application utilizing modern state-of-the-art Machine learning models, transfer learning, and fine-tuning capable of determining whether an image uploaded by the user was generated by a Convolutional Network, classifying it as a real or fake photo, and displaying this classification to the user.

Introduction

Computer Vision is a field of artificial intelligence that is composed of the generation, processing, and analysis of digital images. A key component of Computer Vision is the extraction from higher dimensional data such as images or even video sequences, symbolic information (decisions or descriptions of the world) in a format that makes sense to humans and can be used to inform some action. This extraction of symbolic information is a component of Computer Vision that is integral to the purposes and motivation of our paper, that is to extract based on the features of any given image whether or not it was generated by a Convolutional Neural Network (CNN). A CNN is a deep-learning neural network architecture that is commonly used in Computer Vision. The incredible improvements in Computer Vision technology have led to very realistic-looking AI-generated images, which are incredible in their potential and utility, but also pose very scary risks to society at large. Potential risks of artificially generated images are financial and identity theft (fraud), placing individuals in fake, reputation-damaging situations, generating political ammunition against political opponents, and falsifying criminal evidence (slander). While existing research exists to attempt to distinguish between AI-generated and real photos which we discuss in the Related Work section, we felt like there was a lack of tools available to the lay user that could aid them in distinguishing whether an image is real or not. To fulfill this need in the industry, we developed a full-stack web application utilizing Vite, React, and TypeScript for the Frontend, and Python, Flask, and Pytorch for the Machine Learning backend. We focus on using existing state-of-the-art models such as Resnet50, combined with transfer learning, and fine-tuning to provide the best accuracy and user experience possible. Vite was used for ease of bundling and serving the application. React was chosen for its powerful capabilities to manage website state and information, its utility for developing easy-to-use and interactive user interfaces, and its compatibility with Bootstrap for CSS styling through third-party libraries such as react-bootstrap. Additionally, a bootstrap theme darkly was applied on top of Bootstrap to help create a visually pleasing, dark aesthetic that emphasizes the images being analyzed on the website display. Flask is a micro and simple-to-use web framework for Python that allows developers to quickly create server backends with minimal boilerplate code. It is lightweight and minimalistic in nature, which allows us to focus more on the application logic and flow than the framework itself. Flask's memory efficiency makes it ideal to serve as an API endpoint for image classification as it allows such memory to be reserved for loading a more powerful ML model for more accurate predictions. In Figure [1], the application workflow is visualized for ease of understanding. Firstly the ML model is trained using trainer.py, and that trained model is saved in the file model_epoch_best.pth. The Flask server is then run in preparation for serving the front-end application. The Flask server must be run prior to attempting to use the frontend, otherwise, an error modal popup will be displayed when the user attempts to upload an image for classification. Once the Flask server and frontend application are both up and running, the user will

enter the application through the App component that will render the image upload component within it so the user can upload their file, along with the website header information. Once the Image Component accepts the image input, the user can press the upload button as seen in Figure [2], and pass the image to the main.py server file. If this step fails, the front end is designed to handle the error and will display an error modal popup to the user. If the upload is successful, the main.py server file will pass the image file into the loaded Machine Learning model's predict function, and return this Machine Learning model's predicted label and confidence score back to the Image Upload component. The Image Upload component will then propagate this information back to the App component's state where this information will then be passed to the Classification Result component and the final classification result will be displayed to the user as seen in Figure [3]. Now that we have discussed the problem setting, the overall motivation for the paper, and the application parts on a high level, we will move on to discuss the related work that influenced our project.

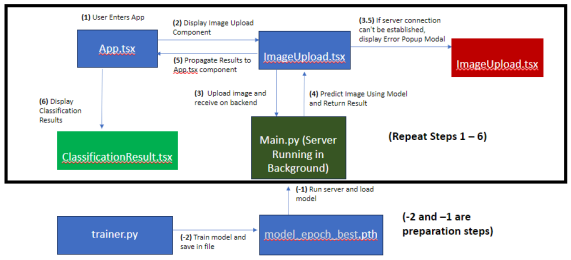

Figure 1: Application Flow Chart
(This is a flowchart representing the overall application flow and how it is used. The red box represents a classification failure while the bright green box represents a classification success. Steps -2 and -1 represent setup steps to prepare the application and need to be completed before running the front end.)

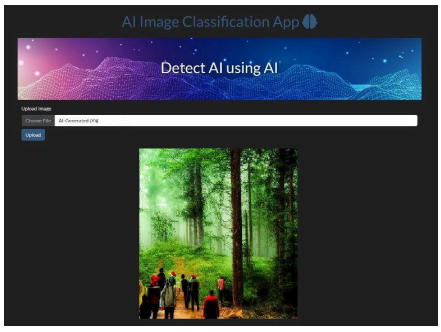

Figure 2: Visualization of the Web Application Design, in this image, the user has uploaded an AI image to the application, and is preparing to upload it to the server to be classified.)

Figure 3: The web application displaying the result to the user retrieved from the Python backend which has the Machine Learning model loaded. In this case, the application correctly classifies the image as AI-generated.)

Related Work

The authors of (Wang 2020) discuss the development and evaluation of a classifier that will be able to detect CNN-generated images, while not being explicitly reliant on samples coming from a specific architecture. The model developed by the authors has been trained on a variety of datasets consisting of images generated by various CNN models, the real images come from well-known datasets. With these tactics, the authors were able to get outstanding performance and accuracy.

Similarly, the authors of (Yang 2021) introduced a forensic method for detecting Deepfake images, focusing on subtle texture discrepancies between real and manipulated images produced by Generative Adversarial Networks (GANs). This research proposes using image saliency detection to highlight these texture differences, followed by the application of a guided filter to enhance texture artifacts that are characteristic of fake images. The method is rigorously tested using large datasets like FaceForensics++, achieving state-of-the-art accuracy in detecting manipulated images. The experiments demonstrate that the guided filter not only improves the model's performance significantly, especially for real images but also helps in handling various manipulations such as those seen in Face2Face and FaceSwap techniques. The study underscores the challenges posed by the rapid advancement of GAN technologies in image manipulation and the ongoing arms race between image forgery creation and detection technologies.

The authors of (Moskowitz 2024) explore the application of the CLIP (Contrastive Language–Image Pre-training) model to detect AI-generated images (AIGI). The study fine-tunes the CLIP model, originally pre-trained on massive internet-scale datasets, using both real images and AIGI produced by various generative models. The authors demonstrate that the fine-tuned CLIP model performs comparably or better than existing models specifically designed for AIGI detection. This finding is significant because it suggests that a general model, without specific architectural modifications for AIGI detection, can achieve high accuracy. This method also offers a more accessible and resource-efficient solution for detecting AIGI, as it does not require extensive GPU resources and utilizes publicly available model architectures. The paper presents a comprehensive evaluation, comparing the fine-tuned CLIP model's performance against other methods across various AIGI generation techniques.

Authors of (Haodong 2020) explore techniques for detecting images generated by deep learning models, specifically focusing on the disparities in color components between real images and those created by generative adversarial networks (GANs). The study begins with an examination of the fundamental differences between images produced by cameras and those generated by GANs, noting that DNG images tend to exhibit significant disparities in chrominance components rather than in luminance. The authors utilize these differences to develop a feature set designed to capture these disparities in statistical properties between DNG and camera images. This feature set, based on the co-occurrence matrix of residual images across different color components, is then used to train classifiers to distinguish between real and DNG images. Extensive experiments demonstrate that the proposed method can effectively identify DNG images, significantly outperforming existing techniques in scenarios where the training and testing data are mismatched, which is common in practical applications. This includes cases where different sources or different generative models are used for training and testing.

Methodologies

In order to detect artificially generated images it is necessary that an AI system be developed that is able to detect if an image was generated by an AI system. This paper attempts to do this through a convolutional neural network (CNN). Several different approaches were explored in attempts to find a solution to this problem. A primary approach was to create a custom-made CNN model. However, the custom-made model suffered from either overfitting issues or poor feature recognition -- resulting in either one-sided predictions where either real or fake value was given 100% of the time, or in a 50/50 probability of classifying the image as real or fake.

Due to shortcomings of the custom implementation of CNN we decided to look at existing solutions for classification problems and explore potentials for utilization of their architectures in our mission to solve this challenging task.

The solution on which we decided was to utilize ResNet-50 as the backbone model architecture for transfer learning. The framework for training the model and image preprocessing was borrowed from (Wang 2020). The borrowed sections of code relate to their streamlined implementation of data pipelining, along with training status updates display/logging. The borrowed code consists of PyTorch implementation of the ResNet models, along with a pipeline script for downloading and applying the official weights. Along with a section that relates to image preprocessing, which entails image scaling and pixel value normalization. The reasoning behind normalization is to make all images more related. This way their unique features would stand out more, while the “less” important parts would be ignored.

The model has been fine-tuned utilizing a dataset (roboflow 2022).

Figure 4. Three Examples of Real Images

Experiment

Figure 5. Three Examples of Fake Images.

The dataset used for this model was obtained from (Roboflow, 2022). The dataset contained 1889 images in total. Of those 1889 images, 1318 were used for training. 374 images were used for testing, and the remaining 197 images were used for testing. All the data was split nearly 50/50 between real and AI images, therefore the dataset in total consisted of 945 real images and 944 AI images. The images above are of three real instances of AI art and the other three are of AI-generated art.

Results

On the validation dataset, this model achieved an accuracy of 91.21% and a recall of 95.91% for an F1 score of 98.32%. The accuracy, recall and F1 Scores on the test dataset were 96.45%, 95.46%, and 96.44%, respectively. These results show that we were able to get close to state-of-the-art performance utilizing techniques similar to (wang 2020).

Conclusion

For this project we propose an end-user application capable of detecting whether or not an image may be generated by artificial intelligence. The front-end segment of this application was done on the front-end using Vite, React, and Typescript. This takes an image selected by the user and passes it into a Python server running Flask (meant to be always running in the background), which contains a ResNet Image classification model. That model will return a percentage that represents the chance that the image is not real if that percentage is above a certain threshold, then the selected image is labeled AI-generated. The label and the related percentage are then passed back to the front end of the application, at which point it will be displayed to the user whether their image is real or AI-generated.

Future Works

After reviewing what we've learned from this project, along with the analysis of the current research landscape in the field of synthetic image detection. We believe that our results could be better. The current implementation is impressive, however, this does not include all types of synthetic image generation models. Due to this, the current model would have irregular performance when confronted with images from newly available image generation models. Thus for future works, we would expect this model architecture to be enhanced with a transformer model or a GAN network. Utilization of these models would allow for better recall, especially if the second to last layer of ResNet-50 is to be fed as input to the transformer network. This combination could potentially result in better performance on non-standard synthetic images (such as

unique geometry, non-realistic animals/people, etc.) thus making this model's implementation a lot more generic and versatile. As this ties into the application side, the more general the network it is, the wider the range of inputs it can process, resulting in a better client experience.

References

Wang S.. 2020. CNN-generated images are surprisingly easy to spot... for now. arXiv:1912.11035 teehee - aaron was here lmao

Yang J. 2021. Detecting fake images by identifying potential texture difference.

Moskowitz A. 2024. Detecting AI-Generated Images via CLIP. arXiv:2404.08788

Haodong L. 2020. Identification of deep network generated images using disparities in color components.

roboflow. 2022. ITML Final Project AI versus Non AI Dataset Dataset.https://universe.roboflow.com/itml-project/itml-final-project-ai-versus-non-ai-dataset