Evaluating the Effectiveness of Machine Learning in Detecting AI-Generated Content: Development and Analysis of a Detection Tool

Sam Magee – B00132512

Department of Informatics, School of Informatics and Engineering,

Technological University Dublin

Submitted to Technological University Dublin in partial fulfilment of the requirements for the degree of

B.Sc. (Hons) Digital Forensics and Cyber Security

Supervisor:

Peter Alexander

Declaration

I hereby certify that this material, which I now submit for assessment on the programme of study leading to the award of Degree of **Honours B.Sc. in Digital Forensics and Cyber Security** in the Institute of Technology Blanchardstown, is entirely my own work except where otherwise stated, and has not been submitted for assessment for an academic purpose at this or any other academic institution other than in partial fulfilment of the requirements of that stated above.

Signed: Sam Magee Dated: 13/05/2024

Abstract

In the era of rapidly advancing artificial intelligence (AI), the proliferation of AI-generated content poses significant challenges across various domains, ranging from digital forensics to national security, theft of one's likeness, copyright infringement, misleading people etc. This thesis presents a comprehensive evaluation of machine learning (ML) techniques in detecting AI-generated content, through both theoretical investigation and practical tool development. We begin by exploring the landscape of AI-generated content, identifying the primary characteristics that distinguish it from human-generated counterparts, what the technology currently does realistically, anomalies and unrealistic elements of generated content. Subsequently, we delve into a range of machine learning models, assessing their suitability, effectiveness, and efficiency in differentiating AI-generated content within diverse contexts.

The core contribution of this research is the development of a detection tool designed to implement and test the ML algorithms discussed. This tool serves as a practical framework for evaluating the real-world applicability of machine learning models in identifying AI-generated images. Through rigorous analysis and experimentation, we provide insights into the detection capabilities, limitations, and performance variations of each model under various conditions that are realistically attainable with current technology.

Furthermore, this study critically examines the current state of technology in AI content detection, highlighting the inherent challenges such as adaptability, scalability, and the development arms race between content creation and detection technologies. We also explore the future relevance of ML-based detection tools, considering the rapid evolution of AI content generation techniques.

The findings contribute to a deeper understanding of the strengths and pitfalls of existing detection methodologies and offer a forward-looking perspective on the development of more robust, effective detection mechanisms. This research underscores the importance of ongoing innovation in ML and the need for interdisciplinary approaches to safeguard against the implications of AI-generated content.

Table of Contents

Introduction

The topic & field of study centered around artificial intelligence (AI) generated content is a rapidly growing area. Large Language Models (LLMs) such as Chat GPT, Gemini, Claude, etc. have been improving in all testing metrics rapidly, they are capable of taking inputs ranging from text, file uploads, images. Utilizing a model known as a transformer they are capable of assigning weights to input data, processing levels of sequences of labelled data and feeling forward in the input data to generate a contextual response based on the data they are able to extract large training models utilizing huge datasets of information sourced from the web, books, research papers etc.

Parallel to the evolution of text-based AI, significant advancements have been made in the generation of visual content through AI. This involves models capable of creating highly realistic images and videos that can be seemingly indistinguishable from those captured by traditional cameras. These generative models, including Generative Adversarial Networks (GANs) and diffusion models, have already began to impact the area of digital media. GANs, for instance, involve a duet of neural networks—the generator and discriminator—competing against each other to create and evaluate images, respectively (Goodfellow, et al., 2014). This process not only enhances the quality of generated images but also the models' ability to mimic the nuances of human-created visuals.

Diffusion models represent a newer class of generative technology that starts with a pattern of random noise and gradually shapes this into detailed images through a reverse diffusion process guided by a trained neural network. These models excel in generating high-quality images by learning from vast datasets of real photographs, which they use to refine their output progressively, achieving a high level of photorealism. (Chang, Koulieris, & Shum, 2023)

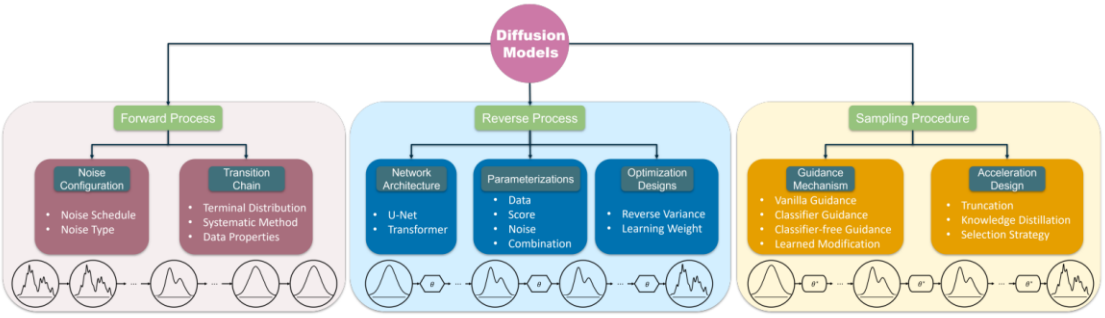
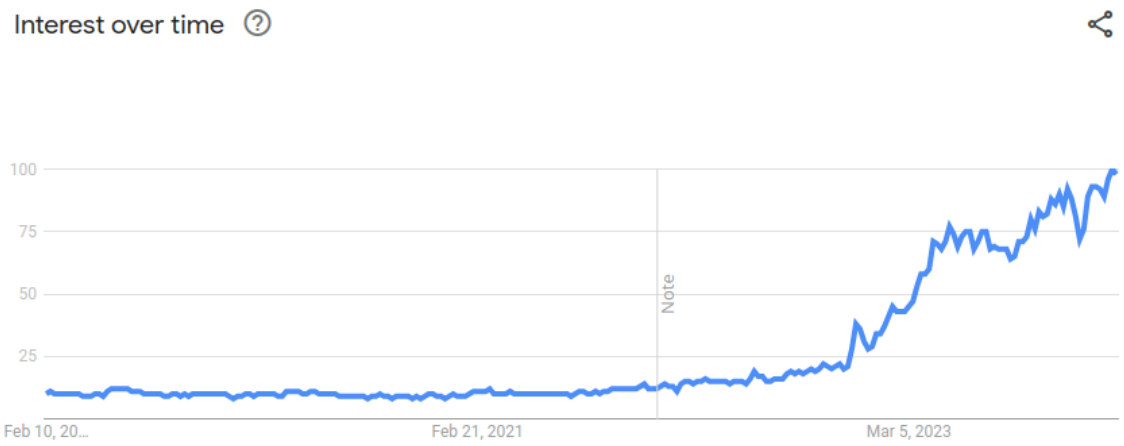


Figure 1. The overview of diffusion models. The forward process, the reverse process, and the sampling procedure are the three core components of

The development and refinement of these visual AI models have not only expanded the capabilities of content creation but have also introduced complex challenges in distinguishing between AI-generated and authentic human-generated images. As AI continues to advance, its applications in creating visual content pose profound implications for digital forensics, copyright laws, and information security. Addressing these challenges requires a deep understanding of both the technology behind AI-generated content and the methodologies for its detection, underscoring the critical need for sophisticated tools and techniques in IT security and digital forensics to keep pace with these rapid advancements.



Search Term: “Artificial Intelligence”
Date Period: 2/15/2019 – 3/1/2024
Source: Google Trends

Research Questions

1. How effective are current ML models at identifying AI-generated images?
2. What are the specific characteristics and anomalies that can be used to detect AI-generated content?
3. How do different ML model types & parameters perform under various conditions, and what are their limitations?

Research Objectives

The main objectives of this thesis are to:

1. Conduct a thorough evaluation of various ML models in detecting AI-generated content, assessing their effectiveness, and identifying their strengths and weaknesses.
2. Develop a practical detection tool that implements these models, providing a framework for real-world applicability tests, with insight to improving effectiveness through the conclusion if the tool shows weaknesses.
3. Contribute to the academic and practical knowledge on digital forensics by highlighting current challenges and potential future developments in AI content detection.

Literature Review

The generation & detection of AI-generated content (AIGC), particularly in the form of images, and recently also video, has become a critical area of research within the fields of digital forensics, cybersecurity but also the overall field of computer science. As generative models improve and evolve, so too have the technologies and methodologies used to detect the content, the area of mitigating potential harms associated with the misuse of AIGC has also grown and developed. The aim of this literature review is to explore the existing works and advancements made in recent times. This research will be utilized throughout the course of the research conducted to aid in coming to the most accurate conclusion. First a brief overview of the history of computer-generated imagery as an overall topic.

Timeline of Computer-Generated Imagery

1. **Early Computer Graphics (1950s-1970s):**
    - The origins of computer-generated images are rooted in the field of computer graphics, beginning in the 1950s and 1960s with pioneering figures like Ivan Sutherland, who developed Sketchpad, a program that allowed users to create graphical images directly on a computer display. (Grudin, 2017)
    - The 1970s saw further developments with the creation of algorithms for rendering 3D objects and the use of computer graphics in film and media, notably in movies like "Westworld" (1973), which was the first to use digital image processing. (Yaeger, 2002)

2. **Rise of 3D Modeling and Animation (1980s-1990s):**
    - The 1980s and 1990s were characterized by significant improvements in 3D modeling and animation techniques. Software like Autodesk 3ds Max and Maya allowed artists to create detailed 3D models and animations.
    - This period also saw the release of the first fully computer-generated short film, "The Adventures of André and Wally B." (1984), produced by the Graphics Group, later known as Pixar Animation Studios.

3. **Neural Networks and Early Generative Models (late 1990s-2000s):**
    - The late 1990s and 2000s witnessed the introduction of neural networks in image generation, with early experiments in using neural networks for creating artistic images and enhancing photo realism.
    - Techniques such as texture synthesis and style transfer began to emerge, allowing computers to apply the style of one image to the content of another, leading to creative and artistic image manipulations. (Gatys, Ecker, & Bethge, A Neural Algorithm of Artistic Style, 2015), (Gatys, Ecker, & Bethge, Image Style Transfer Using Convolutional Neural Networks, 2016). These papers, while published in 2015 was based off foundational work completed in neural networks and image processing from the late 1990s to 2000s.

4. **Deep Learning and Generative Adversarial Networks (2010s):**
    - The advent of deep learning brought significant advancements in image generation. The introduction of Generative Adversarial Networks (GANs) by Ian Goodfellow and his colleagues in 2014 was a critical milestone. GANs use two neural networks, a generator, and a discriminator, that compete against each other to generate new images that are indistinguishable from real ones. (Goodfellow, et al., 2014)
    - These techniques were quickly adopted for various applications, from creating photorealistic images of human faces (e.g., NVIDIA's StyleGAN) to artistic image generation and even video game content creation. (Karras, Laine, & Aila, 2019)
    -
5. **Advancements in Diffusion Models (2020-present):**
    - Diffusion models have emerged as a significant advancement in AI-driven image generation, known for their ability to create highly detailed and realistic images. These models work by gradually reversing a process that transforms a signal (such as an image) from coherent structures to random noise. (Chang, Koulieris, & Shum, 2023)
    - Notable implementations include OpenAI's DALL-E, which can generate images from textual descriptions with high fidelity and creativity. (OpenAI, 2021) Google's Imagen and Parti models also exemplify the cutting-edge capabilities of diffusion models in generating photorealistic and artistically complex images from text descriptions. (Google Research, 2022) (Google Research, 2022)

Works Related to AI Generated Content Detection

There have been numerous papers published that focus on this topic. A common theme observed by the author of this report is that early works tended to focus on detecting DeepFake images. To give a brief explanation of the difference between deepfake images and the 100% AIGC images that this report focuses on, existing works in this field generally classifies deepfake images as AI *modified* content. Pavel Korshunov and Sebastien Marcel paper, "DeepFakes: a New Threat to Face Recognition? Assessment and Detection" refers to pre-trained models, specifically generative adversarial networks (GANs) that are used to "replace a face of one person in a video with the face of another person" (Marcel, 2018).

In the development of advanced methodologies for detecting DeepFake content, Omkar Salpekars "DeepFake Image Detection", (Salpekar, 2020) utilized a two-phase learning approach, using a ResNet-based architecture enhanced with large dense residual blocks. This classifier, configured as a Siamese Network, initially harnessed the power of an aggressive triplet loss margin to refine the model's sensitivity to variations in fake content. Subsequent fine-tuning involved the application of regularized cross-entropy loss during later epochs, aimed at maintaining the classifier's ability to generalize across a highly varied and extensive dataset of 100,000 images.

Key optimizations were integral to achieving high accuracy. These included learning rate scheduling to adaptively adjust the rate through training phases, early stopping to prevent overfitting, and meticulous tuning of batch sizes to optimize the training process. This comprehensive approach underscores the effectiveness of structured neural network training in enhancing the robustness and precision of DeepFake detection systems.

Results were promising, Salpekars model had a final accuracy of 94%, and validation accuracy of 91%, utilizing an 80:20 data split of 3922 fakes and 980 reals.

In more recent times we are seeing the emergence of 100% AI generated content. Arsh Banerjee's paper, "Detecting AI-Generated Images Created by Diffusion models" discusses the evolution from DeepFakes generated using GANs to a new type of model known as diffusion models. (Banerjee, 2023) The paper references back to another paper published by OpenAI, authored by Aditya Ramesh et al "Hierarchical Text-Conditional Image Generation with CLIP Latents" that diffusion models were able to "produce higher-quality samples" and "image diversity with minimal loss in photorealism" (Ramesh, Dhariwal, Nichol, Chu, & Chen, 2022)

The paper Identifying AI-Generated Art with Deep Learning, authored by Tommaso Bianco et al, researched the effectiveness of identifying diffusion model-based AI generated art with deep learning. Using a self-created dataset of authentic and generated artworks, they conducted research with various neural network architectures. They found their model exceeded the performance of well-known models like VGG19 & ResNet-50.

Results were promising, utilizing an 80:10:10 data split for training, validation and training they achieved accuracy metrics in excess of 95%.

	Accuracy	Precision	Recall	F1
VGG-19	0.9581	0.9590	0.9562	0.9575
ResNet-50	0.9654	0.9645	0.9655	0.9650
ViT	**0.9758**	**0.9752**	**0.9759**	**0.9755**

Concluding the model was capable of detecting differences that would have been very challenging for the naked eye to detect. They also found noted different features the models emphasized.

- VGG19: emphasizes broad, coarse-grained features that span extensive areas within the given image.
- ResNet50 & Vision Transformer: focus more on fine-grained details.

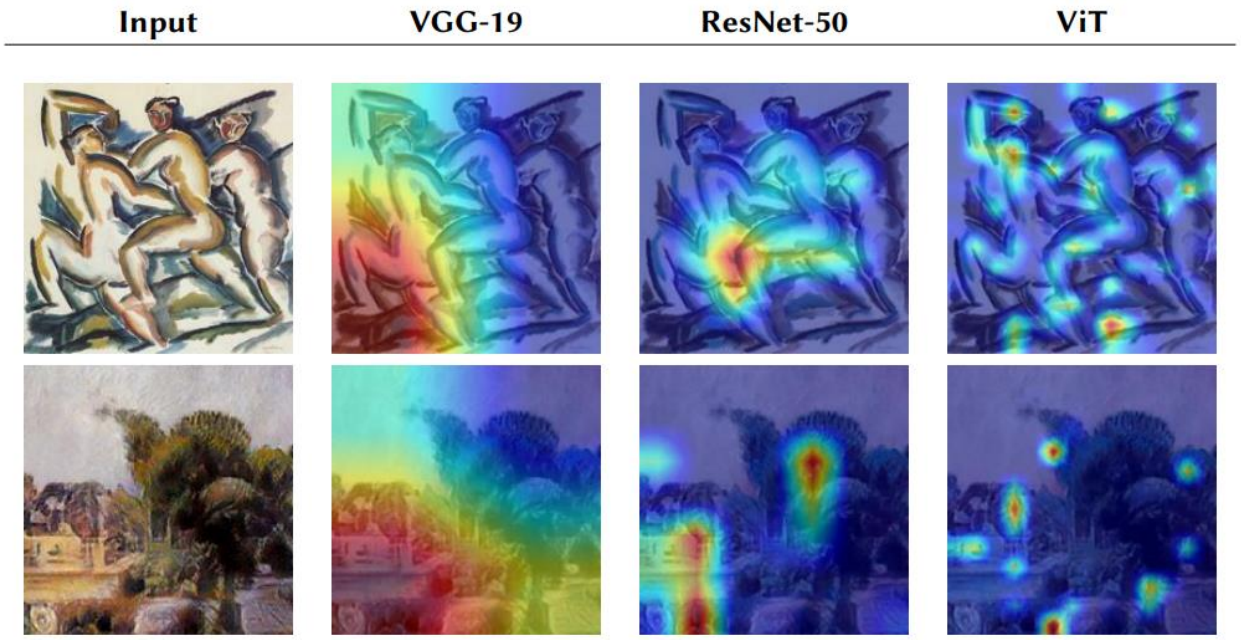


Figure 2. Most discriminative areas by model for input images.

(Bianco, Castellano, Scaringi, & Vessio, 2023)

Early Approaches and Challenges

The initial approach to detecting AI content largely focused on identifying very specific anomalies generally introduced by the models that existed during the early stages of AIGC. The previously mentioned Omkar Salpekars "DeepFake Image Detection" gives some insight into early approaches in the related work section. Stating that early generations of work in this field focused an approach that didn't utilize any deep learning. Detection at this stage focused on analyzing low level features of AIGC such as JPEG compressions and chromatic aberrations, also - "featurizing image data using bag-of-words and feeding those features to statistical classifiers like SVMs and examining image features in the frequency domain by training classifiers on the Discrete Fourier Transforms of the images." (Salpekar, 2020) The paper states that significant challenges and weaknesses of this approach was the variability of real images. This could be due to variables such as lens-type, sensor focus, image resolution and other variables still found in legitimate images & video.

Generation and detection can be thought of as a concurrent race, there are advancements made in both and on advancement in generation for instance may make existing detection methods quite ineffective. This is discussed in Pavel Korshunov and Sebastien Marcel paper "DeepFakes: a New Threat to Face Recognition? Assessment and Detection", in which they summarized the rise of generative adversarial networks as "Our experiments demonstrate that GAN-generated Deepfake videos are challenging for both face recognition systems and existing detection methods, and the further development of face swapping technology will make it even more so.". However now GANs are quite a well-researched area and detection models have been demonstrated with very high accuracy. It was found that GANs typically exhibited their own set of anomalies. Xu Zhang, Svebor Karaman, and Shih-Fu Chang paper "Detecting and Simulating Artifacts in GAN Fake Images" described that the up sampling component of the GAN pipeline left behind a spectrum artifact that essentially acted like a fingerprint in every image, allowing them to not only detect fake images but also to identify the model it came from (Zhang, Karaman, & Chang, 2019).

Existing Methods

Watermarking AI-Generated Images

Overview

Image watermarking is a technique used to embed a unique identifier or mark into an image to establish ownership, origin, or authenticity. With the rise of AI-generated images, watermarking has become increasingly important to track and verify the source of such images. Many popular diffusion models such as Dall-E 2, implement watermarking to varying degrees of success. (Luccioni, et al., 2024)

Purpose and Implementation

The primary purpose of watermarking AI-generated images is to ensure traceability, protect intellectual property and from a moral and ethical standpoint to alert viewers to the origin of the image. Image watermarking can be implemented through either visible or invisible methods. Visible watermarking is straightforward and involves adding a pattern, logo, or text overlay on the image, while invisible watermarking embeds data within the image in a way that does not perceptibly alter its appearance.

Figure 3. OpenAI's Dall-E 2 included a visible watermark composed of 5 blocks of different colors in the bottom right corner. Source: instagram.com/dailydall.e (Luccioni, et al., 2024)

Techniques in Watermarking AI-Generated Images

Recent advancements focus on robust watermarking techniques that can survive manipulations such as cropping, resizing, and compression. In 2020, Tancik et al. proposed a method that embedded 100 bit long strings representing 56 bit hyperlinks after error correction. They determined that this encoding was imperceptible to the human eye in most scenarios but can be reliably detected by neural networks. This technique is particularly suited for deep learning models that generate images, ensuring that the watermark remains intact even through various transformations. (Tancik, Mildenhall, & Ng, 2020)

Challenges in Watermarking

A major challenge in watermarking AI-generated images is maintaining the watermark's integrity after multiple image processing operations. Research conducted by Jiang et al., in their paper entitled Evading Watermark based Detection of AI-Generated Content found embedded bits type watermarking, the paper used stable-diffusions 136 bit embedding as an example, were susceptible to circumvention by methods as simple as processing the image through JPEG compression. They found that the stable diffusion generated images were detected 100% of the time without modification but that post-processing with JPEG, with a relatively high quality factor of 0.8 reduced the accuracy to around 50%. (Jiang, Zhang, & Gong, 2023)

Ethical and Legal Implications

AI-generated images also raises ethical and legal issues. Watermarking introduces an ‘at-source’ ability to trace an image back to its AI origin and can help in addressing the misuse of synthetic media, as discussed by Agarwal et al., in their 2020 paper entitled Detecting Deep-Fake Videos from Appearance and Behavior. They discuss the dangers of AI generated images, raising the point that its power to disrupt democratic elections, commit small to large-scale fraud, fuel dis-information campaigns, and create non-consensual pornography is a point of concern. Also mentioning that several U.S. states have already passed legislation attempting to mitigate the harm posed and that legislation was under consideration the federal level at the time of the paper. (Agarwal, El-Gaaly, Farid, & Lim, 2020)

The European union has also implemented legislation placing an obligation on providers and users of AI systems to enable the detection and tracing of AI-generated content. In a briefing they state the implementation of this will likely require the use of watermarking techniques, the briefing itself admits that even state of the art AI watermarking techniques display limitation and drawbacks. (Madiega, 2023).

Effectiveness

- Research conducted by Mozilla research concluded that ‘Human-facing’, or visible watermarking methods were vulnerable to manipulations and ranked the method as a ‘Low’ efficacy. The research also included machine-readable, such as the aforementioned embedded data methods, utilised in combination with detection mechanisms, ranking the efficacy of this method as ‘Fair’ (Vasse'i & Udoh, 2024)

Cryptographic Watermarking

Information, such as a signature is embedded in the data. Mozilla research in the previously mentioned paper likens cryptographic watermarking to inserting locks that can only be detected, removed or changed with a key. This off course is not a new concept, but it is a novel approach to watermarking content.

Pros	Cons
- Higher level of security. - Versatile use for content. - Difficult to remove from content without leaving traces of tampering.	- Can be complex to implement. - Relies on specialised detection. - Can be computationally costly. - Introduces difficulties in sharing keys for encryption. - While difficult to remove from the content itself, it is easily circumventable by simply copying text, screenshotting the image, recording the video, etc.

Frequency Watermarking

A pattern-based technique previously discussed that involves adding data to the content in a way that aims to be imperceptible. The content is decomposed, and data is inserted into the low frequency bands of the content so as the original content is least affected.

Pros	Cons
- Well-suited for imperceptible watermarking of images and other non-text based content. - High level of security against deletion or modification.	- Also requires significant effort and resources. - Usefulness greatly depends on the detection tools and their availability. - Alteration of certain components is possible.

Statistical Watermarking

A relatively robust but complex to embed form of watermarking that involves changing certain values of the content. The watermarking is considered statistically unusual. “Put simply, the idea behind statistical watermarking is to put a thumb on the scale of randomness during generation so it leaves a fingerprint that can be detected later.” (Srinivasan, 2024)

Pros	Cons
- Allows identification of specific patterns in the output they generate. - Designed with stealth in mind and maintaining its integrity against circumvention techniques such as compression and scaling. - Balances robustness and usefulness well. - Difficult to remove without degrading the quality of the content.	- Decoding can be difficult and complex. - Like all other invisible watermarks requires specific detection mechanisms. - Embedding and detecting statistical watermarks requires significant compute power. - Could be compromised by more sophisticated methods such as statistical analysis or AI-based techniques.

Metadata Watermarking

Instead of embedding data in the content direction metadata watermarking, as the name would imply, embeds data into the metadata of the file. This is widespread already in non-ai generated images in the form of EXIF data. Metadata watermarking generally includes information such as the author, file MAC timestamps, software, hardware flags, etc. Metadata watermarking is not very robust as there are many tools to strip metadata and most file sharing sites do it by default.

Pros	Cons
- Easy to implement, metadata can easily be applied using photo editing software.	- Can be lost during transfer or intentionally tampered with and stripped using widely available software. - Increases file size.

(Vasse'i & Udoh, 2024)

Limitations

- Visible watermarking generally considered to be of limited purpose as it can be quite easily circumvented by means of image manipulation, such as cropping, compression, resizing, and many photo editing software come with ‘smart select’ tools for removing objects from an image. (Adobe, 2023)
- Invisible watermarking, utilising embedded data was found in the previously discussed Jiang et al., paper to not be robust. This was due to the conclusion that a relatively small amount of perturbation of the image would make the watermark indetectable to the detection model that is trained to detect the embedded data. (Zhang, Karaman, & Chang, 2019)

Machine-learning anomaly-based detection

Overview

In contrast to the 'at-source' approach taken in watermarking AI content, an emerging technique for detecting AI content is utilising machine learning algorithms to detect anomalies commonly found in AI-generated content. This method could be considered more reactive but has the advantage of utilising weaknesses of known technologies that are being used to generate content. This ideally makes the pro-active approach of watermarking content in the process of generation redundant if the anomaly detection model demonstrates a high accuracy.

Purpose and implementation

The primary purpose of anomaly-based detection using machine learning is to autonomously identify AI-generated content by recognizing deviations from established norms of authentic human-generated content. Implementation typically involves training supervised or unsupervised learning models on datasets that consist of both genuine and AI-created content. These models learn to detect discrepancies in features such as texture, noise patterns, or geometric inconsistencies that are often overlooked by human observers but are telltale signs of synthetic origins.

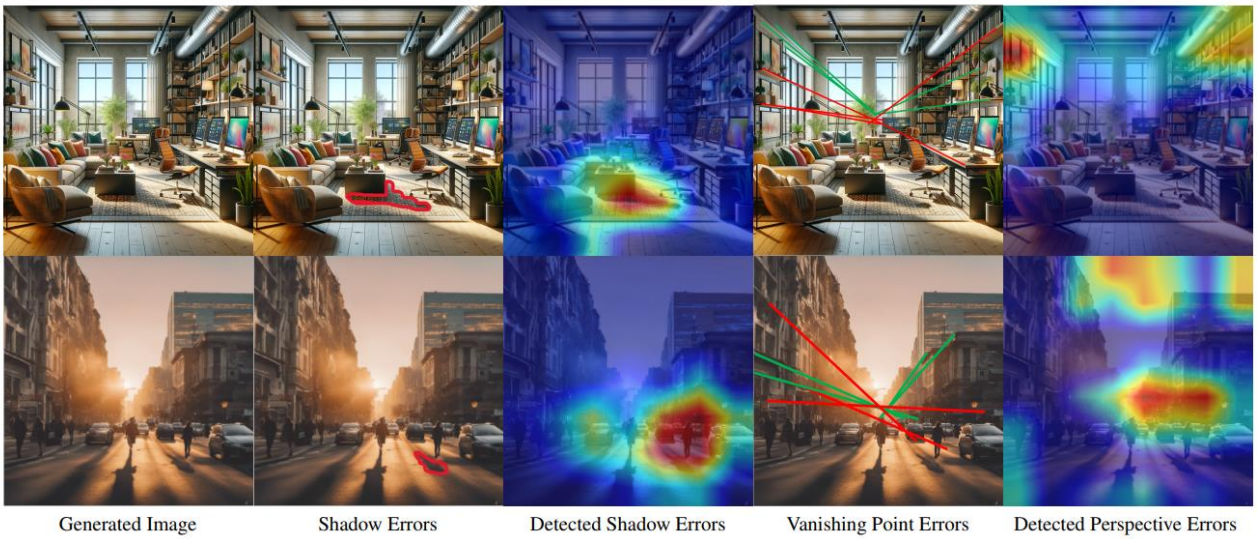


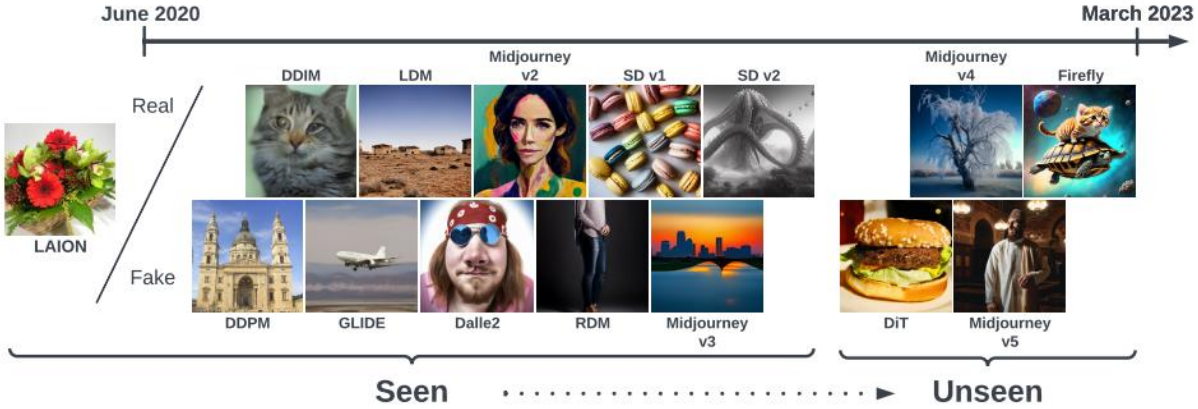
Figure 4. The first column presents visually compelling AI-generated images. However, a closer examination reveals fundamental inconsistencies, such as those in shadow alignment (second column) and vanishing point accuracy (fourth column). The sourced studies models analysis, shown in the third and fifth columns, detects these shadow and perspective geometry errors. (Sarkar, et al., 2023)

Challenges in detecting anomalies

- **Data Diversity**: Effective training requires diverse datasets that represent a wide range of anomalies, which can be challenging to compile. While this is a challenge to a degree, detection for diffusion-based models can be generalised, a process detailed in Zhang et al. paper discussing their model trained on 14 different models and the increasing accuracy they found in detecting varying models. They concluded that the solution to tackling this challenge was to implement a vigilant classifier, regularly retrained on newly released models. (Zhang, Wang, Jain, & Epstein, 2023)

  Another method for detection illustrated in the figure above is a projective geometry approach. The key factor being the current inability of diffusion-based generative models to accurately generate projective geometry based elements related to shadows and illumination. (Sarkar, et al., 2023)

- **Adaptability**: Generative models are rapidly evolving, and today's anomaly detection models may struggle to keep pace with newer forms of synthetic content that may exhibit fewer or different anomalies. Looping back to the foundational research conducted by Zhang, et al. conclusion that a model to reliably detect various models, the classifier would have to be regularly retrained to include recently released models to keep pace with the advancements in the field.



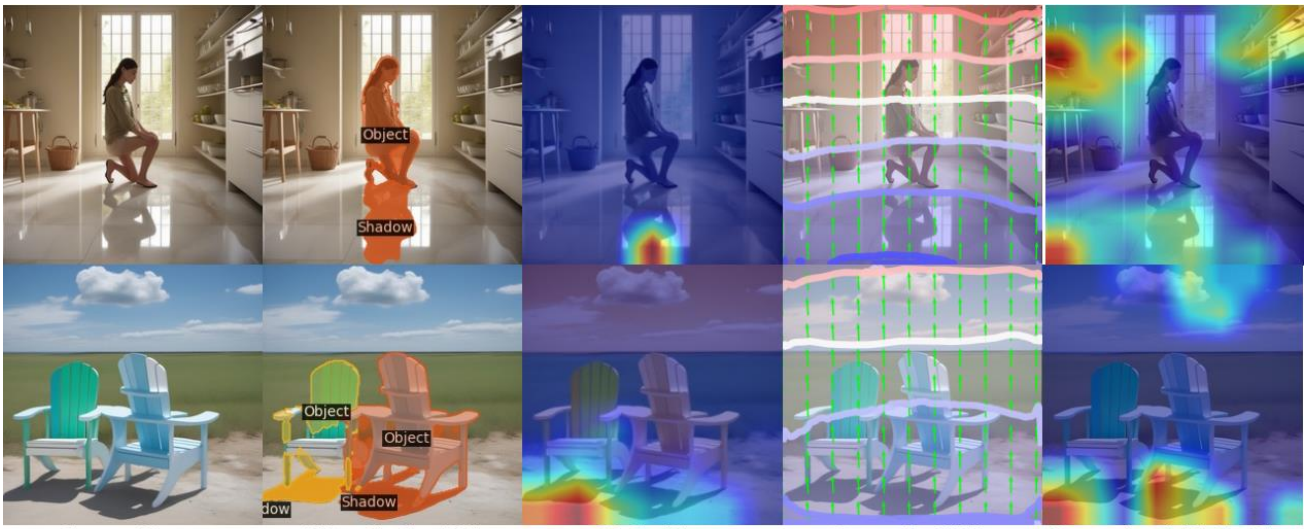
- *Figure 5. New AI-generated image models are released regularly. The above figure depicts the process by which existing models are added to the classifier with the intent of improving accuracy on unseen newly released generative models. (Zhang, Karaman, & Chang, 2019)*

  Off course, this research is a recognition of the pattern observed thus far in the area. It cannot account for a newly released generative model making large strides in terms of 'realness' or also taking a different approach, unlike previous models in how the generation process is completed. Further emphasizing the importance of regularly re-training the classifier on any newly released models.

- **False Positives/Negatives**: High sensitivity to anomalies can lead to false positives, where genuine content is misclassified as synthetic, or false negatives, where AI-generated content is not detected.

Effectiveness

The effectiveness of anomaly-based detection models often hinges on the sophistication of the underlying machine learning algorithms and the quality of the training data. Studies have shown that these models can achieve high accuracy rates, particularly in controlled environments where the types of anomalies are well-understood and documented, for example in the case of the targeted projective geometry approach a high accuracy was achieved on multiple models without implementing a huge number of models into the training data, the study claiming AUC values of 0.72 to 0.97, iterating that the model does not see the actual image, only the derived geometric features from an object mask & shadow mask, object shadow pre-step, along with a perspective field. (Sarkar, et al., 2023)



Generated Image | Object-Shadow (OS) | OS GradCam | Perspective Fields | Perspective Fields GradCam

Multi-generative model trained classifiers, as studies in Zhang, et al. Online detection of AI-generated images, take a more general approach, opting to use ResNet-50, pre-trained on ImageNet as the foundation for their detector. Following the below table and noting the accuracy of classification for AI-generated and real images along each step. The conclusion being that a detection model trained on just one model generally does not have a high accuracy at detected AI-content and being relatively fit to the singular generative model it was originally trained on.

Generation architecture	Method/Dataset	Training set	Method availability		Ordering		Dataset size		
			Paper	Open-src	#	Date	Train	Val	Test
Real images	LAION-400M [44]	–	–	–	–	–	179,900	22,479	22,490
Diffusion U-net	Denoising Diffusion Prob. Model (DDPM) [22]	LSUN [59]	✓	✓	1	Jun 20	6,271	784	785
	Denoising Diffusion Implicit Model (DDIM) [49]	LSUN [59]	✓	✓	2	May 21	8,000	1,000	1,000
	GLIDE [32]	Private	✓	✓	3	Dec 21	7,442	929	931
	DALL·E 2 [37]	Private	✓	✗	5	Apr 22	2,000	954	2,000
Diffusion +Decoder	Latent Diffusion (LDM) [38]	LAION-400M [44]	✓	✓	4	Dec 21	8,172	1,021	1,022
	Retrieval-Augmented Diffusion (RDM) [9]	LAION-400M [44]	✓	✓	7	Jul 22	8,528	1,066	1,066
	Stable Diffusion v1.1-v1.4 [38, 55]	LAION-2B [43]	✓	✓	8	Aug 22	34,508	3,807	3,838
	Stable Diffusion 2.0(-v), 2.1(-v) [38, 7]	LAION-5B [43]	✓	✓	10	Nov 22	35,997	4,000	4,000
Diffusion U-Vit	Diffusion w/ Transformers (DiT) [34]	ImageNet [40]	✓	✓	11	Dec 22	3,199	400	401
Unknown (product release)	Midjourney v2 [3]	Unknown	✗	✗	6	Jul 22	42,875	5,358	5,359
	Midjourney v3 [3]	Unknown	✓	✓	9	Nov 22	70,035	8,754	8,755
	Midjourney v4 [3]	Unknown	✗	✗	12	Feb 23	100,000	10,000	76,122
	Midjourney v5 [3]	Unknown	✗	✗	13	Mar 23	63,310	7,914	7,918
	Adobe Firefly [1]	Unknown	✗	✗	14	Mar 23	15,525	2,070	3,105

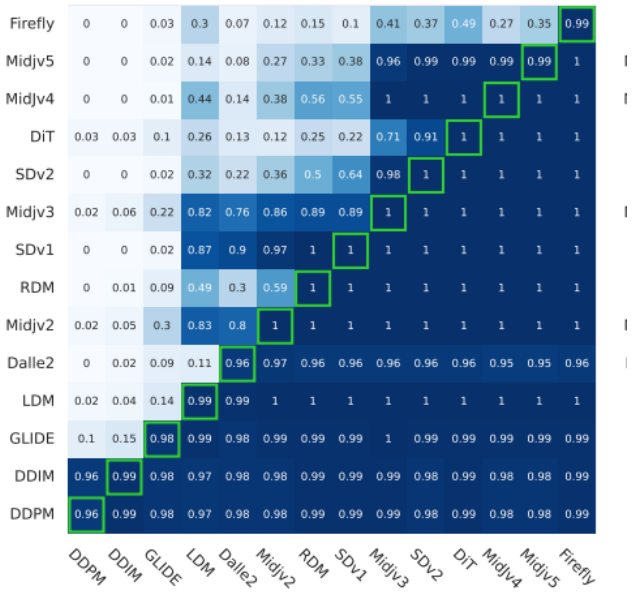


Figure 6. Along the X-axis shows the models included within the detection model, along the Y-axis show the accuracy detection at each step of the detection models training.

Limitations

- **Generalization**: Anomaly detectors trained on specific types of AI-generated content may not generalize well to others that use different generative methods. This is why as previously discussed; inclusion of multiple models is important.
- **Scalability**: Processing large volumes of content in real-time poses computational challenges, potentially limiting the scalability of this approach.
- **Evolution of AI Techniques**: From the previously discussed literature it can be inferred that as AI content generation techniques improve, they will likely produce fewer anomalies, making detection more difficult and require continual updates to detection algorithms.

Key Findings

- Watermarking while effective for detection has been shown to be circumventable in many scenarios, and directly stated as not currently up to a viable standard for a sole detection method by various studies and reports.
- Anomaly-based detection could be viewed as a more reliable detection method as it is not vulnerable to circumvention in the same way that watermarking content is. However, the findings of the reviewed literature conclude that a very targeted or diverse approach is required when training a detection model to ensure that the detector is accurate on the multitude of generative models publicly available. This detection method will also require constant evaluation and maintenance to include and account for new generative models.

Methodology

Objectives

The objective of the study is to assess the accuracy of various models trained on a wide range of datasets. The datasets will cover various objects and be classified as AI-generated content or real in the training stage. This will be done using a variety of metrics. Firstly, the training and validation accuracy scores will be recorded and comparatively analysed against the scores from the other models that have been trained for the study.

From an extensive literature review of existing studies and report the author would also like to note that not many papers have made their model parameters and the output model from the training process publicly available for peer testing. This research will seek to be transparent with the model parameters, providing code used and conducting separate evaluation testing using a custom script. This is another process the author notes it not commonly included in the existing literature. This is important due to the fact that the accuracy metrics of a machine learning model can be significantly altered by the testing data used.

This testing methodology will aim to answer the following questions,

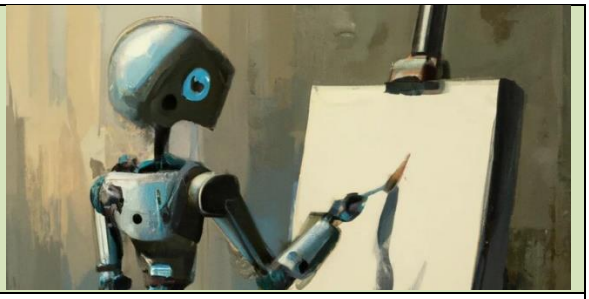
- Do existing machine learning algorithms produce acceptable training and validation accuracy scores when trained on an AI-generated and real image dataset.
- What effect does various different approaches to model generation scripts & parameters have on these training and validation scores.
- Do the training and validation accuracy values align with a varied testing dataset that have been run through the model evaluation script.
- If the values do not align, are the results still acceptable.

By answering these questions this research paper aims to achieve a greater understanding of the current ability of machine learning to **detect AI-generated content** and to **classify real and AI-generated content.**

Data Collection

Sources

Multiple datasets where sourced for training the various detection models created. The aim was to find varied datasets that were relatively recent in release, this distinction was made with the intention of gathering relevant data for recent generative models, as previously discussed, while diffusion models do generalise, previously conducted studies have found that models have to be continuously retrained with the inclusion of new models as anomalies and hallmarks of AI-generated content changes (Zhang, Wang, Jain, & Epstein, 2023). The 5 following datasets were chosen.

AI Generated Images vs Real Images (BOWMAN, 2024)				
Updated:	February 2024.			
Number of files:	975 total - 763 .jpg - 150 .png - 57 .jpeg - 5 Other			
Number of files per category:	AI-Generated Content	**539**	Real Content	**436**
About:	This is a dataset consists of web-scraped & AI-generated images. From the publishers about section: “The dataset is a captivating ensemble of images sourced from two distinct channels: web scraping and AI-generated content. The content covers many subjects; however, special emphasis was placed on these topics: people, animals, portraits, scenery, and psychedelics.” (BOWMAN, 2024) The generative models utilised for the images is varied as a result of the dataset being web-scraped. Reverse image searching returns Midjourney, Stable-Diffusion, Dall-E. Though not every image was reverse image search nor is the generative model possible to know from the indication inferred from reverse image search. **Key Features:** Web-Scraped Images: These images are harvested from various online sources across the web. Ranging from landscapes, paintings, psychedelic trips, and portraits, the web-scraped images offer a glimpse into the vast spectrum of digital imagery available online.			

ArtiFact: Real and Fake Image Dataset (Awsaf, Paul, Sarker, & Hakim, 2023)				
Updated:	February 2023.			
Number of files:	2,496,738 total, all .jpg.			
Number of files per category:	AI-Generated Content	**1,531,749**	Real Content	**964,989**
About:	This dataset is a large image dataset. The dataset includes a diverse collection of objects categories, such as humans/human faces, animals/animal faces, places, vehicles, art, etc. The creators of the dataset selected a variety of sources for both real and AI-generated images. Comprising of 8 real image sources, and 25 fake image sources.			

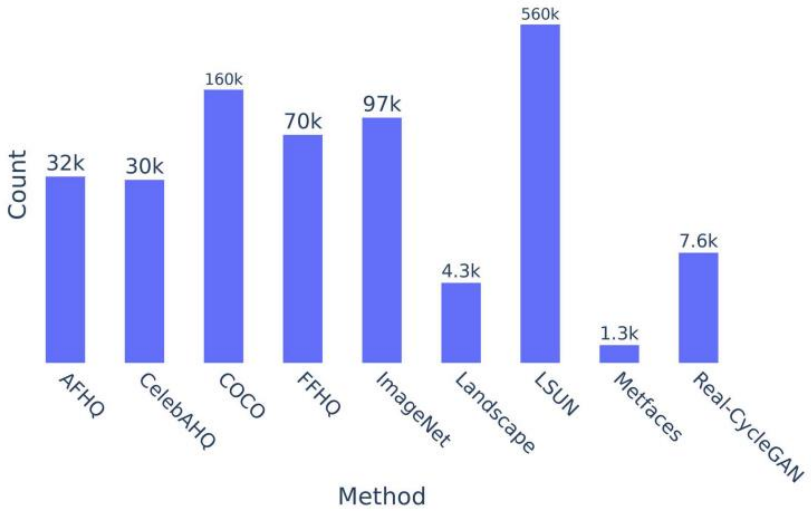


Figure 7. Distribution of different methods for real images in the ArtiFact dataset

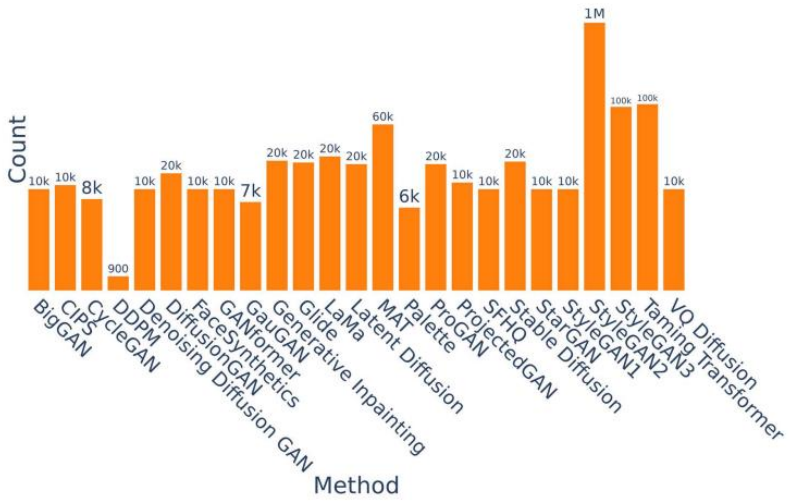
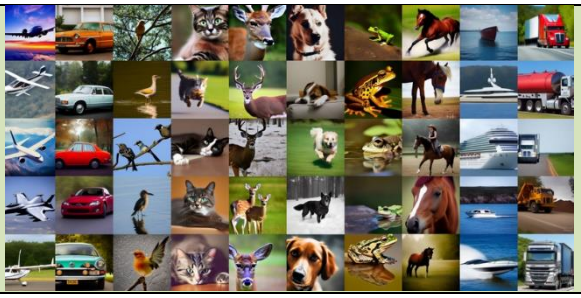
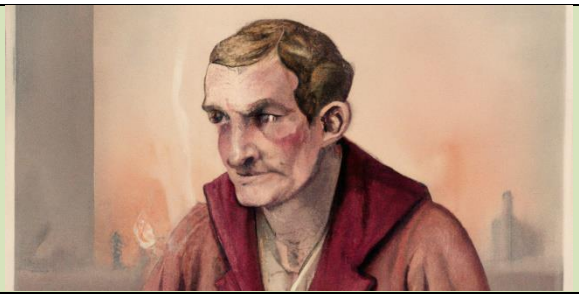


Figure 8. Distribution of different methods for fake images in the ArtiFact dataset

CIFAKE: Real and AI-Generated Synthetic Images (Lotfi & Bird, 2023) (Krizhevsky & Hinton, 2009)				
Updated:	February 2024.			
Number of files:	120,000 total, 20,000 of which are separated into a testing folder.			
Number of files per category:	AI-Generated Content	**60,000**	Real Content	**60,000**
About:	This is a popular dataset, which builds upon the original CIFAR-10 dataset from Krizhevsky & Hinton's paper "Learning multiple layers of features from tiny images". The authors, Dr Jordan J. Bird & Professor Ahmad Lotfi generated the equivalent of CIFAR-10 using Stable-Diffusion 1.4 The images have all been compressed to 32px, which makes the dataset extremely efficient in terms of storage size and also processing for training a model. This could compromise the final trained models accuracy though. Multiple comments on the Kaggle post for the dataset request the original 512px dataset referenced in Bird, J.J. and Lotfi, A., 2024. CIFAKE: Image Classification and Explainable Identification of AI-Generated Synthetic Images. (Lotfi & Bird, 2023)			

140k Real and Fake Faces (Xhlulu, 2020)				
Updated:	2020 – Month not available.			
Number of files:	140,000 total.			
Number of files per category:	AI-Generated Content	**70,000**	Real Content	**70,000**
About:	This is a dataset consists of 70,000 real faces from the Flickr dataset - https://github.com/NVlabs/ffhq-dataset compiled by Nvidia labs, as well as 70,000 images sampled from the 1 Million Fake Faces dataset - https://www.kaggle.com/datasets/tunguz/1-million-fake-faces , the fake images are GAN generated and although this paper focuses on Diffusion images, the decision to include this dataset was made due to the availability of diffusion face datasets and relevance to the field of digital forensics. This dataset would allow for filtered object classification assisted detection. This is a process involving a decision tree when classifier/detection model is being selected depending on what object has been detected in the image. This could increase accuracy.			

AI recognition dataset (Koliha, 2024)			
Updated:	February 2024		
Number of files:	21,638 images total. - 12.9k .png - 8759 .jpg		
Number of files per category:	AI-Generated Content	**17,857**	Real Content **3,781**
About:	This is a dataset comprises of real and AI-Generated images. The dataset contains images generated from various diffusion models, DALL-E and Midjourney are specifically mentioned but more are included. *Sourced from the authors Kaggle post:* **"Benefits over other datasets:** The benefit of this dataset compared to other artificially generated image datasets (such as CIFAKE) is that all images are in there original size and aspect ratio." It is worth noting the mention of CIFAKES low resolution by the author of this dataset, as this was previously recognised as a potential problem above.		

Data Labelling

For the purposes of our research, the varying means of data labelling present in each of the selected datasets was noted. A streamlined method to prepare our image dataset for analysis was required. Our primary focus was on classifying images as either real or fake, and moving them to the relevant directory for the model to take as an input during training This step is crucial for training our model to differentiate accurately between real and AI-Generated images.

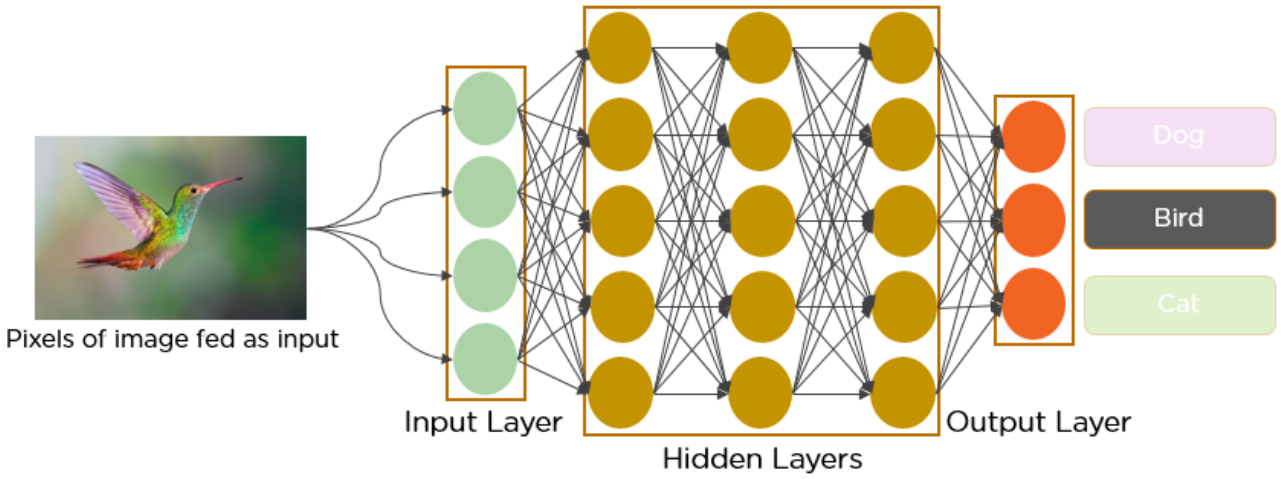
- **Automated Script for Annotation:** Given the extensive nature of our dataset, manual annotation was impractical. To efficiently handle this task, an automated script designed to assess each image was implemented, this will be discussed in further detail in a later section. The script automatically categorizes each image as 'real' or 'fake' based on the base datasets directory structure. Once an image is categorized, the script renames it accordingly and sorts it into the appropriate directory, either 'fake' or 'real', ready for input to the model we will use for research. This automation significantly accelerates the process, allowing us to quickly build a well-organized dataset ready for model training.

- **Directory Structure:** To maintain an orderly dataset, the classified images are stored in two main directories, 'real', 'ai_generated'. This organization not only aids in dataset management but also simplifies the process of retrieving specific images for model training and validation phases.

This approach not only streamlines the preparation of our dataset but also ensures that the data used for training our model is as reliable as possible. By automating the annotation process, we can focus on developing a robust model capable of accurately distinguishing between real and fake images.

Model Selection

In order to accurately classify images as AI-generated or real, we've decided to employ convolutional neural networks (CNNs), which are the go-to for image processing tasks due to their proficiency in handling pixel data, in use by Zhang et, al for their study (Zhang, Wang, Jain, & Epstein, 2023), Sarkar et, al for their projective geometry study (Sarkar, et al., 2023), Arsh Banerjee for their study (Banerjee, 2023), all to generally good results.



We're setting up an experiment to compare two main approaches: building our own CNN from scratch and using a transfer learning model based on ResNet50.

1. Building a Custom CNN: The first part of our experiment involves constructing a CNN from the ground up. The idea here is to start simple and gradually introduce complexities. We'll begin with a basic CNN structure—just a few convolutional layers followed by max pooling layers and a fully connected layer to make the final prediction. This "vanilla" model will set our baseline.
To refine our model, we'll integrate several best practices:

- **Dropout:** To prevent overfitting, dropout layers will randomly ignore certain neurons during training, forcing the data to find new paths, which improves generalization. (Brownlee, A Gentle Introduction to Dropout for Regularizing Deep Neural Networks, 2019)
- **Early Stopping:** To further combat overfitting and save training time, early stopping will halt training as soon as the validation accuracy stops improving.
- **Flattening and Shuffling:** Flattening transforms the pooled feature maps to a single column that feeds into the output layer, while shuffling ensures that the model does not learn anything from the order of the data. (Göllner, 2022)

Each of these additions aims to enhance the model's ability to learn from diverse data without memorizing it, striking a balance between accuracy and efficiency.

2. Utilizing Transfer Learning with ResNet50: For the transfer learning part, we're leaning towards ResNet50, a model known for its deep architecture and residual connections that help it learn faster and more effectively (Mascarenhas & Agarwal, 2021) without the vanishing gradient problem common in deep networks (Sarin, 2019). ResNet50 has been widely adopted for image classification tasks.
The use of a pre-trained model like ResNet50 allows us to take advantage of learned features from massive datasets like ImageNet, a technique also employed by Zhang et. al (Zhang, Wang, Jain, & Epstein, 2023), which can be particularly useful when training data is limited or when we want to speed up training. In our project, this approach could quickly provide a high-performance benchmark against which we can measure our custom-built CNN.

3. Customizations and Comparative Analysis: While ResNet50 will primarily be used in its standard form, we might need to tweak the final layers to better suit our specific task of distinguishing between AI-generated and real images. This could involve adjusting the output layer to reflect our two classes and possibly fine-tuning some of the deeper layers to adapt the pre-trained features to our specific context.
In our comparative analysis, we'll assess both models based on their accuracy and efficiency. This will help us understand the trade-offs between developing a model from scratch and using a sophisticated pre-trained system. We'll track metrics such as training time, accuracy on validation and test sets, and computational resources used, giving us a comprehensive view of which model works best for our specific needs.
This strategy not only aligns with our objective to create an effective image classifier but also provides a solid foundation for robust comparative research, giving us insights into both the potentials and limitations of custom versus pre-trained deep learning models in real-world applications.

Implementation

Experimental Setup

Hardware & Software Setup

- **CPU and GPU:**
    - AMD Ryzen 3900XT CPU paired with an NVIDIA RTX 3080 GPU. This combination offers a solid balance of processing cores and high-end GPU capabilities, which will aid in quickly training new models as parameters are tweaked.
- **24GB RAM**
- **Storage:** A 1 TB SSD provides ample space and fast data access speeds, reducing load times and speeding up the process of training and testing models.

Software and Tools:

- **Operating Systems:** Windows 10 for tasks that require CPU-based TensorFlow operations. However, for GPU-accelerated TensorFlow training, Ubuntu on Windows Subsystem for Linux 2 (WSL2), which supports CUDA and cuDNN was utilised.
- **TensorFlow Version:** TensorFlow 2.15, the latest stable release during the project. It provides all the necessary functionalities and optimizations for building and training neural network models. Default installed version 2.17 encountered bug issues with the model running out of data for no reason with an OUT_OF_RANGE error. (GitHub, 2024)
- **Development Environment:** All coding and development are done in Visual Studio Code.

Library Selection

1. **NumPy**
   NumPy is a widely used python module that contains multidimensional array and matrix data structures. (numpy, 2024)

2. **Tensorflow** (GeeksForGeeks, 2024) **& Keras** (simplilearn, 2024)
   TensorFlow is an open-source machine learning library developed by Google. TensorFlow is used to build and train deep learning models as it facilitates the creation of computational graphs and efficient execution on various hardware platforms.

   Tensorflow utilizes data glow graphs, in which the following can be considered.
   - **nodes** in the graph represent mathematical operations.
   - **edges** in the graph represent the multidimensional data arrays (called **tensors**) communicated between them. (Please note that **tensor** is the central unit of data in TensorFlow).

   Keras is a high-level, deep learning API developed by Google for implementing neural networks. It is written in Python and is used to make the implementation of neural networks easy. It also supports multiple backend neural network computation.

   Keras is relatively easy to learn and work with because it provides a python frontend with a high level of abstraction while having the option of multiple back-ends for computation purposes. This makes Keras slower than other deep learning frameworks, but relatively beginner-friendly.

   Keras allows you to switch between different back ends. The frameworks supported by Keras are:

   - Tensorflow
   - Theano
   - PlaidML
   - MXNet
   - CNTK (Microsoft Cognitive Toolkit)

   Out of these five frameworks, TensorFlow has adopted Keras as its official high-level API and the integration is a major deciding factor in its use for this research.

3. **JSON**
   Allows JSON encoding and decoding according to the RFC7159 standard within python. (python, 2024)

4. **OS**
   Allows for basic operating system interactions, open files, manipulate paths, read command line etc. (python, 2024)

5. **IPython**

6. **Shutil**

7. **Glob**

8. **Scikit-learn**

9. **PIL – pillow**

10. **Pandas**

11. **Matplotlib**
    2D plotting library which produces publication-quality figures in a variety of formats and interactive environments.

Model Utils

data-equalizer.py (geeksforgeeks, 2019), (python, 2024)

This script randomly selects a set amount of images from the training dataset and moves any excess to an overflow directory. The purpose of this script was to equalise the class sizes for training.

```
import os
import numpy as np
import shutil

def random_select_and_move(source_dir, archive_dir, select_count=3750):
    # Ensure the archive directory exists
    os.makedirs(archive_dir, exist_ok=True)

    # List all files in the source directory
    all_files = [f for f in os.listdir(source_dir) if
os.path.isfile(os.path.join(source_dir, f))]
    # Filter for images, assuming they end with common image file
extensions
    image_files = [f for f in all_files if f.lower().endswith(('.png',
'.jpg', '.jpeg', '.gif', '.bmp'))]

    # Randomly select the specified number of images
    if len(image_files) > select_count:
        selected_images = np.random.choice(image_files, size=select_count,
replace=False)
    else:
        selected_images = image_files

    # Move non-selected images to the archive directory
    for file in image_files:
        if file not in selected_images:
            shutil.move(os.path.join(source_dir, file),
os.path.join(archive_dir, file))

    print(f"Moved {len(image_files) - len(selected_images)} files to
{archive_dir}")
    print(f"{len(selected_images)} files remain in {source_dir}")

# Usage
source_directory = 'train-val-dataset/ai_generated'
archive_directory = 'overflow-dataset/ai_generated'
random_select_and_move(source_directory, archive_directory)
```

data-labeller.py (PYNative, 2024) (geeksforgeeks, 2023)

The purpose of this script is rename the files from the varying dataset sources to a common name. The decided name for this research was ‘ai_generated_<Sequential Number>_<timestamp>’ for AI generated content, and ‘real_<Sequential Number>_<timestamp>’ for real content. The names are not hugely important to the actual training of the model. It does allow for easier implementation of features in the future though, such as a file-by-file analysis.

```
import os
import time

# Define the directories containing the images
ai_generated_dir = 'raw-dataset/fakeV2'
real_dir = 'raw-dataset/real'

def rename_files(directory, prefix):
    # Initialize a counter for file numbering
    counter = 1

    # Get the current time in seconds since epoch, then format it
    timestamp = int(time.time())

    # Walk through the directory
    for dirpath, dirnames, filenames in os.walk(directory):
        for filename in filenames:
            # Extract the file extension from the original file name
            _, file_extension = os.path.splitext(filename)

            # Ensure only image files are processed
            if file_extension.lower() in ['.jpg', '.jpeg', '.png']:
                # Define the new file name with the prefix, counter, and
timestamp
                new_name =
f"{prefix}_{counter}_{timestamp}{file_extension}"

                # Get the full path of the current file and the new file
                old_file_path = os.path.join(dirpath, filename)
                new_file_path = os.path.join(dirpath, new_name)

                # Rename the file
                os.rename(old_file_path, new_file_path)

                # Increment the counter for the next file
                counter += 1

# Call the function for each directory
rename_files(ai_generated_dir, 'ai_generated')
rename_files(real_dir, 'real')
```

```
print("Files have been renamed based on their directories.")
```

data-splitter.py (geeksforgeeks, 2019) (python, 2024) (geeksforgeeks, 2020) (Rooy, 2013)

The purpose of this script is randomly select 30 images of each class from the training, remove them before training and move them to an evaluation dataset directory. This will allow for evaluation figures to be obtained for unseen but same dataset images. Results for each dataset detailed below.

```
import os
import shutil
import random

# Define the source directory and subdirectory names
source_dir = 'raw-dataset/'
ai_generated_subdir = 'fakev2'  # This may contain nested subdirectories
real_subdir = 'real'            # This may contain nested subdirectories

# Define the target directory names for AI-generated and real images
ai_generated_target_subdir = 'ai_generated'
real_target_subdir = 'real'

# Destination directories
eval_dir = 'eval-dataset'
train_val_dir = 'train-val-dataset'

# Number of images to select for evaluation
num_eval_images = 30

def setup_directories():
    # Create destination directories if they do not exist
    os.makedirs(os.path.join(eval_dir, ai_generated_target_subdir),
exist_ok=True)
    os.makedirs(os.path.join(eval_dir, real_target_subdir), exist_ok=True)
    os.makedirs(os.path.join(train_val_dir, ai_generated_target_subdir),
exist_ok=True)
    os.makedirs(os.path.join(train_val_dir, real_target_subdir),
exist_ok=True)

def move_files(destination, files):
    # Move specified files from their current location to the destination
    for file in files:
        shutil.move(file, os.path.join(destination,
os.path.basename(file)))

def split_data(source_subdir, target_subdir):
    # Define the source and initial destinations
    source_path = os.path.join(source_dir, source_subdir)
    eval_dest_path = os.path.join(eval_dir, target_subdir)
```

```
    train_val_dest_path = os.path.join(train_val_dir, target_subdir)

    # Recursively collect all files in the source directory
    all_files = []
    for dirpath, dirnames, filenames in os.walk(source_path):
        for filename in filenames:
            all_files.append(os.path.join(dirpath, filename))

    # Randomly select a subset of files for evaluation
    eval_files = random.sample(all_files, num_eval_images)

    # Identify the rest of the files for training and validation
    train_val_files = [f for f in all_files if f not in eval_files]

    # Move files to the respective directories
    move_files(eval_dest_path, eval_files)
    move_files(train_val_dest_path, train_val_files)

# Setup directories
setup_directories()

# Split data for AI-generated images and real images
split_data(ai_generated_subdir, ai_generated_target_subdir)
split_data(real_subdir, real_target_subdir)

print("Data has been split into evaluation and training/validation
datasets.")
```

random-selector.py (python, 2024) (python, 2024) (Niggo, 2020) (python, 2024)

```
import os
import shutil
import random

def copy_random_images(source_dir, target_dir, num_files=3000):
    # Ensure the target directory exists
    if not os.path.exists(target_dir):
        os.makedirs(target_dir)

    # Get a list of all image files in the source directory
    all_files = [f for f in os.listdir(source_dir) if
os.path.isfile(os.path.join(source_dir, f))]
    # Filter the list for files ending with common image file extensions
    image_files = [f for f in all_files if f.lower().endswith(('.png',
'.jpg', '.jpeg'))]

    # Check if the number of images is sufficient
    if len(image_files) < num_files:
```

```
        raise ValueError(f"Only {len(image_files)} images found in
{source_dir}, but {num_files} images are requested to copy.")

    # Randomly select 'num_files' images
    selected_files = random.sample(image_files, num_files)

    # Copy the selected images to the target directory
    for file in selected_files:
        shutil.copy(os.path.join(source_dir, file),
os.path.join(target_dir, file))

    print(f"Copied {num_files} images from {source_dir} to {target_dir}.")

# Define source and target directories
source_directory = 'raw-dataset/fake'
target_directory = 'eval-dataset/ai_generated'

# Run the function
copy_random_images(source_directory, target_directory)
```

artifface-random-selector (geeksforgeeks, 2024) (python, 2024) (python, 2024) (Niggo, 2020)

```
import os
import shutil
import random

def select_and_copy_images(root_dir, target_dir_real, target_dir_ai,
num_images=3000):
    real_dirs = {'afhq', 'celebahq', 'coco', 'ffhq', 'imagenet',
'landscape', 'lsun', 'metfaces', 'cycle_gan'}
    real_images = []
    ai_images = []

    # Walk through the root directory
    for dirpath, dirnames, filenames in os.walk(root_dir):
        # Filter image files and ignore metadata.csv
        images = [os.path.join(dirpath, name) for name in filenames if
name.lower().endswith(('.png', '.jpg', '.jpeg', '.gif', '.bmp')) and
'metadata.csv' not in name]

        # Determine if the images should be categorized as real or AI-
generated
        if any(real_dir in dirpath for real_dir in real_dirs):
            real_images.extend(images)
        else:
            ai_images.extend(images)
```

```
    # Randomly select images from each list up to the specified limit
    selected_real_images = random.sample(real_images, min(num_images,
len(real_images)))
    selected_ai_images = random.sample(ai_images, min(num_images,
len(ai_images)))

    # Ensure the target directories exist
    os.makedirs(target_dir_real, exist_ok=True)
    os.makedirs(target_dir_ai, exist_ok=True)

    # Copy the selected images to their respective target directories
    for image in selected_real_images:
        shutil.copy(image, os.path.join(target_dir_real,
os.path.basename(image)))
    for image in selected_ai_images:
        shutil.copy(image, os.path.join(target_dir_ai,
os.path.basename(image)))

    print(f"Copied {len(selected_real_images)} real images to
{target_dir_real}.")
    print(f"Copied {len(selected_ai_images)} AI-generated images to
{target_dir_ai}.")

# Main variables
root_dir = 'raw-dataset/'
target_dir_real = 'training-val-dataset/real'
target_dir_ai = 'training-val-dataset/ai_generated'

# Run the function
select_and_copy_images(root_dir, target_dir_real, target_dir_ai)
```

Models

Refined Model

(tensorflow, n.d.) (tensorflow, n.d.) (tensorflow, n.d.) (tensorflow, n.d.) (Singh, 2020) (w3schools, n.d.) (Brownlee, TensorFlow 2 Tutorial: Get Started in Deep Learning with tf.keras, 2022) (sentdex, 2018)

```
import os
import tensorflow as tf
from tensorflow.keras.preprocessing.image import ImageDataGenerator
from tensorflow.keras.models import Sequential
from tensorflow.keras.layers import Conv2D, MaxPooling2D, Flatten, Dense,
Dropout
from sklearn.model_selection import train_test_split
from matplotlib import pyplot as plt
from PIL import Image
Image.MAX_IMAGE_PIXELS = None  # This increases the limit on the maximum
number of pixels for an image loaded using PIL

# Define paths
base_dir = './train-val-dataset'  # Directory containing the training and
validation dataset

# Load images into train and validation sets
def load_data(test_size=0.2):
    # Rescale pixel values and split data into training and validation sets
    datagen = ImageDataGenerator(rescale=1./255,
validation_split=test_size)

    # Training data generator
    train_generator = datagen.flow_from_directory(
        base_dir,
        target_size=(150, 150),  # Resize images to 150x150
        batch_size=32,  # Number of images to process in a batch
        class_mode='binary',  # Type of classification (binary in this
case)
        subset='training'  # Specify subset as training
    )

    # Validation data generator
    validation_generator = datagen.flow_from_directory(
        base_dir,
        target_size=(150, 150),
        batch_size=32,
        class_mode='binary',
        subset='validation'  # Specify subset as validation
    )

    return train_generator, validation_generator

# Build the model
```

```
def build_model():
    model = Sequential([
        Conv2D(32, (3, 3), activation='relu', input_shape=(150, 150,
3)),  # Conv layer with 32 filters of size 3x3, ReLU activation
        MaxPooling2D(2, 2),  # Max pooling layer with pool size 2x2
(reduces spatial dimensions)
        Conv2D(64, (3, 3), activation='relu'),  # Another conv layer with
64 filters, increasing feature extraction
        MaxPooling2D(2, 2),
        Conv2D(128, (3, 3), activation='relu'),  # Further increase in
filters to 128, increasing complexity
        MaxPooling2D(2, 2),
        Flatten(),  # Flattens the input for the dense layers
        Dense(512, activation='relu'),  # Fully connected layer with 512
units
        Dropout(0.5),  # Dropout layer to reduce overfitting by randomly
setting input units to 0 at each update during training
        Dense(1, activation='sigmoid')  # Output layer with sigmoid
activation for binary classification
    ])

    # Compile the model with Adam optimizer and binary crossentropy as the
loss function
    model.compile(optimizer='adam', loss='binary_crossentropy',
metrics=['accuracy'])
    return model

# Train the model
def train_model(model, train_generator, validation_generator, epochs=10):
    steps_per_epoch = train_generator.samples //
train_generator.batch_size  # Calculate number of batches per epoch
    validation_steps = validation_generator.samples //
validation_generator.batch_size

    history = model.fit(
        train_generator,
        steps_per_epoch=steps_per_epoch,
        epochs=epochs,
        validation_data=validation_generator,
        validation_steps=validation_steps
    )
    return history

# Plot training results (tensorflow, n.d.) (kmario23, 2018)
def plot_training(history):
    acc = history.history['accuracy']
    val_acc = history.history['val_accuracy']
    loss = history.history['loss']
```

```
    val_loss = history.history['val_loss']

    epochs = range(len(acc))

    plt.plot(epochs, acc, 'r', label='Training accuracy')
    plt.plot(epochs, val_acc, 'b', label='Validation accuracy')
    plt.title('Training and validation accuracy')
    plt.legend()

    plt.figure()

    plt.plot(epochs, loss, 'r', label='Training loss')
    plt.plot(epochs, val_loss, 'b', label='Validation loss')
    plt.title('Training and validation loss')
    plt.legend()

    plt.show()

# Main function
def main():
    train_generator, validation_generator = load_data()
    model = build_model()
    history = train_model(model, train_generator, validation_generator)
    plot_training(history)
    model.summary()  # Print a summary of the model architecture
    model.save('models/ai-recog_refined-model.h5')  # Save the trained
model

main()
```

Refined Model with early stopping

(tensorflow, n.d.) (tensorflow, n.d.) (tensorflow, n.d.) (tensorflow, n.d.) (Singh, 2020) (w3schools, n.d.) (Brownlee, TensorFlow 2 Tutorial: Get Started in Deep Learning with tf.keras, 2022) (sentdex, 2018)

```
import tensorflow as tf
import matplotlib.pyplot as plt
import os
from tensorflow.keras.preprocessing.image import ImageDataGenerator
from tensorflow.keras.callbacks import EarlyStopping, ModelCheckpoint
from PIL import Image
Image.MAX_IMAGE_PIXELS = None

# Paths to image data
data_dir = '../train-val-dataset'

# Initialize ImageDataGenerator with specific preprocessing for AI image
detection
datagen = ImageDataGenerator(
    rescale=1./255,
    validation_split=0.2,
    brightness_range=[0.8, 1.2],  # Adjust brightness to simulate different
exposure conditions
    channel_shift_range=0.1,  # Slight shifts to emphasize color anomalies
    horizontal_flip=False,  # Avoid horizontal flips which might obscure
directional artifacts
    fill_mode='nearest'  # Fill mode for any geometric transformations
applied
)


# Setup training and validation data generators
train_generator = datagen.flow_from_directory(
    data_dir,
    target_size=(128, 128),  # Increased image size for more detailed
feature extraction
    batch_size=32,
    class_mode='binary',
    subset='training'
)
validation_generator = datagen.flow_from_directory(
    data_dir,
    target_size=(128, 128),
    batch_size=32,
    class_mode='binary',
    subset='validation'
)

# Model architecture
model = tf.keras.models.Sequential([
```

```
    # First convolutional block with batch normalization and dropout
    tf.keras.layers.Conv2D(32, (3, 3), padding='same', activation='relu',
input_shape=(128, 128, 3)),
    tf.keras.layers.BatchNormalization(),  # Normalize the activations of
the previous layer at each batch
    tf.keras.layers.MaxPooling2D(2, 2),
    tf.keras.layers.Dropout(0.2),  # Randomly sets input units to 0 to
prevent overfitting

    # Second convolutional block
    tf.keras.layers.Conv2D(64, (3, 3), padding='same', activation='relu'),
    tf.keras.layers.BatchNormalization(),
    tf.keras.layers.MaxPooling2D(2, 2),
    tf.keras.layers.Dropout(0.3),

    # Third convolutional block
    tf.keras.layers.Conv2D(128, (3, 3), padding='same', activation='relu'),
    tf.keras.layers.BatchNormalization(),
    tf.keras.layers.MaxPooling2D(2, 2),
    tf.keras.layers.Dropout(0.4),

    # Classifier
    tf.keras.layers.Flatten(),
    tf.keras.layers.Dense(256, activation='relu'),
    tf.keras.layers.BatchNormalization(),
    tf.keras.layers.Dropout(0.5),
    tf.keras.layers.Dense(1, activation='sigmoid')
])

# Compile the model
model.compile(optimizer='adam', loss='binary_crossentropy',
metrics=['accuracy'])

# Early stopping and model checkpoint to save the best model (tensorflow,
n.d.) (tensorflow, n.d.) (Brownlee, Use Early Stopping to Halt the Training
of Neural Networks At the Right Time, 2020)
early_stopping = EarlyStopping(monitor='val_loss', patience=5,
restore_best_weights=True)
model_checkpoint = ModelCheckpoint('best_model.h5', monitor='val_accuracy',
save_best_only=True)

# Train the model
history = model.fit(
    train_generator,
    steps_per_epoch=train_generator.samples // train_generator.batch_size,
    epochs=30,  # Increased number of epochs for better training, will stop
when overfitting due to early stop function
    validation_data=validation_generator,
```

```
    validation_steps=validation_generator.samples //
validation_generator.batch_size,
    callbacks=[early_stopping, model_checkpoint]
)

# Save the final model
model_save_path = '../models/art_refined-model.h5'
os.makedirs(os.path.dirname(model_save_path), exist_ok=True)
model.save(model_save_path)

# Prepare directory for saving plots
graphs_dir = 'graphs/'
os.makedirs(graphs_dir, exist_ok=True)

# Extract training history for plotting
acc = history.history['accuracy']
val_acc = history.history['val_accuracy']
loss = history.history['loss']
val_loss = history.history['val_loss']
epochs = range(1, len(acc) + 1)

# Plot and save accuracy graph (tensorflow, n.d.) (kmario23, 2018)
plt.figure()
plt.plot(epochs, acc, 'bo', label='Training Accuracy')
plt.plot(epochs, val_acc, 'b', label='Validation Accuracy')
plt.title('Training and Validation Accuracy')
plt.legend()
plt.savefig(f"{graphs_dir}Training_and_Validation_Accuracy.png")
plt.close()

# Plot and save loss graph
plt.figure()
plt.plot(epochs, loss, 'bo', label='Training Loss')
plt.plot(epochs, val_loss, 'b', label='Validation Loss')
plt.title('Training and Validation Loss')
plt.legend()
plt.savefig(f"{graphs_dir}Training_and_Validation_Loss.png")
plt.close()

print("Model and graphs have been saved.")
```

Transfer Learning Model

(tensorflow, n.d.) (tensorflow, n.d.) (Singh, 2020) (w3schools, n.d.) (Brownlee, TensorFlow 2 Tutorial: Get Started in Deep Learning with tf.keras, 2022) (sentdex, 2018) (tensorflow, n.d.) (krishnaik06, 2020) (tensorflow, n.d.)

```
import os
import tensorflow as tf
from tensorflow.keras.applications import ResNet50
from tensorflow.keras.preprocessing.image import ImageDataGenerator
from tensorflow.keras.models import Model
from tensorflow.keras.layers import Dense, GlobalAveragePooling2D
from tensorflow.keras.optimizers import Adam
from tensorflow.keras.callbacks import ModelCheckpoint, EarlyStopping
import matplotlib.pyplot as plt
from PIL import Image
Image.MAX_IMAGE_PIXELS = None

# Paths to image data
data_dir = '../train-val-dataset'

# ImageDataGenerator with validation split and suitable preprocessing for
ResNet50
datagen = ImageDataGenerator(
    preprocessing_function=tf.keras.applications.resnet50.preprocess_input,
    validation_split=0.2
)

# Setup training data generator
train_generator = datagen.flow_from_directory(
    data_dir,
    target_size=(224, 224),  # Adjust size for ResNet50
    batch_size=32,
    class_mode='binary',
    subset='training'
)

# Setup validation data generator
validation_generator = datagen.flow_from_directory(
    data_dir,
    target_size=(224, 224),
    batch_size=32,
    class_mode='binary',
    subset='validation'
)

# Load ResNet50 pre-trained on ImageNet but without the top layer
base_model = ResNet50(weights='imagenet', include_top=False,
input_shape=(224, 224, 3))
```

```
# Freeze all layers in the base model
for layer in base_model.layers:
    layer.trainable = False

# Create the custom top layers for our classification task
x = base_model.output
x = GlobalAveragePooling2D()(x)
x = Dense(1024, activation='relu')(x)
predictions = Dense(1, activation='sigmoid')(x)

# This is the model we will train
model = Model(inputs=base_model.input, outputs=predictions)

# Compile model
model.compile(optimizer=Adam(lr=0.0001), loss='binary_crossentropy',
metrics=['accuracy'])

# Setup callbacks for early stopping and best model checkpoint (tensorflow,
n.d.) (tensorflow, n.d.) (Brownlee, Use Early Stopping to Halt the Training
of Neural Networks At the Right Time, 2020)
callbacks = [
    EarlyStopping(monitor='val_loss', patience=5, verbose=1,
restore_best_weights=True),
    ModelCheckpoint('best_model_resnet50.h5', monitor='val_accuracy',
save_best_only=True, verbose=1)
]

# Train the model
history = model.fit(
    train_generator,
    steps_per_epoch=train_generator.samples // train_generator.batch_size,
    epochs=10,
    validation_data=validation_generator,
    validation_steps=validation_generator.samples //
validation_generator.batch_size,
    callbacks=callbacks
)

# Save the final model
model_save_path = '../models/art_resnet50-model.h5'
os.makedirs(os.path.dirname(model_save_path), exist_ok=True)
model.save(model_save_path)

# Prepare directory for saving plots
graphs_dir = 'graphs/'
os.makedirs(graphs_dir, exist_ok=True)

# Plot training and validation accuracy and loss
```

```
acc = history.history['accuracy']
val_acc = history.history['val_accuracy']
loss = history.history['loss']
val_loss = history.history['val_loss']
epochs = range(1, len(acc) + 1)

plt.figure()
plt.plot(epochs, acc, 'bo', label='Training Accuracy')
plt.plot(epochs, val_acc, 'b', label='Validation Accuracy')
plt.title('Training and Validation Accuracy')
plt.legend()
plt.savefig(f"{graphs_dir}Training_and_Validation_Accuracy_ResNet50.png")
plt.close()

plt.figure()
plt.plot(epochs, loss, 'bo', label='Training Loss')
plt.plot(epochs, val_loss, 'b', label='Validation Loss')
plt.title('Training and Validation Loss')
plt.legend()
plt.savefig(f"{graphs_dir}Training_and_Validation_Loss_ResNet50.png")
plt.close()

print("ResNet50 model and graphs have been saved.")
```

Evaluation Script

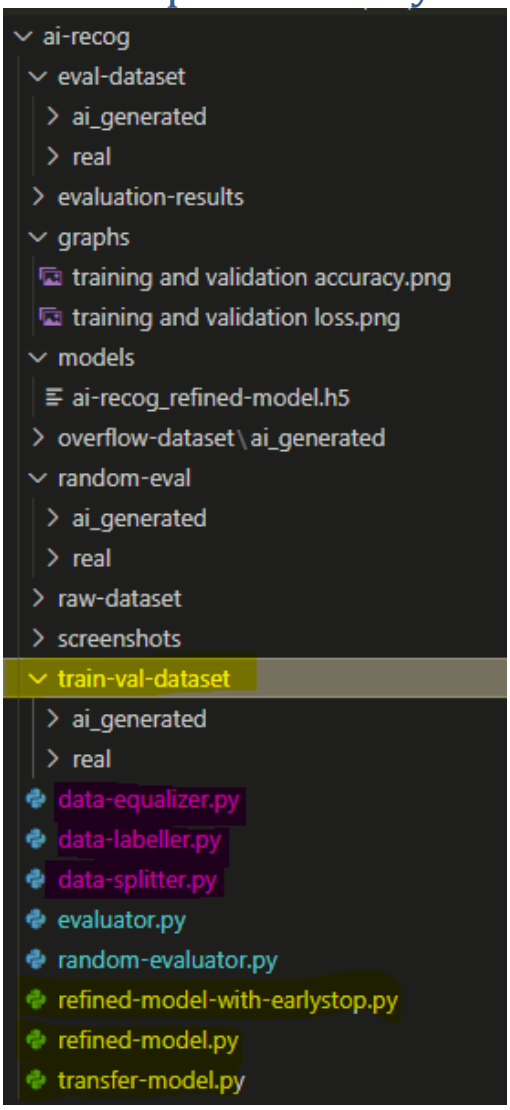
(w3 schools, n.d.) (sentdex, 2018) (Singh, 2020) (python, 2024) (python, 2024) (python, 2024) (NumPY, n.d.) (tensorflow, n.d.) (tensorflow, n.d.) (Nikneshan, 2015)

```
import os
import json
import tensorflow as tf
from tensorflow.keras.preprocessing import image
from tensorflow.keras.models import load_model
import numpy as np

# List of model paths and their respective target sizes
model_info = [
    {'path': 'models/ai-recog_refined-model.h5', 'target_size': (150,
150)},
    {'path': 'models/ai-recog_refined-early-model.h5', 'target_size': (150,
150)},
    {'path': 'models/ai-recog_resnet50-model.h5', 'target_size': (224,
224)}
]

# Directory containing images to evaluate
input_dir = 'eval-dataset'
```

```
# Directory containing random images to evaluate, these are split into
separate scripts.
# Directory containing images to evaluate
input_dir = 'random-eval-dataset'


# Function to evaluate a single model
def evaluate_model(model_path, target_size):
    # Load the trained model
    model = load_model(model_path)
    model_name = os.path.basename(model_path)

    # Initialize results structure
    results = {
        'model_name': model_name,
        'ai_generated': {'total': 0, 'total_correct': 0,
'confidence_scores': []},
        'real': {'total': 0, 'total_correct': 0, 'confidence_scores': []},
        'results': {}
    }

    # Process images and classify
    for label in ['ai_generated', 'real']:
        dir_path = os.path.join(input_dir, label)
        for img_name in os.listdir(dir_path):
            img_path = os.path.join(dir_path, img_name)
            img = image.load_img(img_path, target_size=target_size)
            img_array = image.img_to_array(img)
            img_array = np.expand_dims(img_array, axis=0)  # Make it a
batch of one
            img_array /= 255.0  # Rescale similarly as done during training

            prediction = model.predict(img_array)[0][0]
            predicted_label = 'ai_generated' if prediction < 0.2 else
'real'

            # Increment the total count
            results[label]['total'] += 1
            # Increment the correct count
            if predicted_label == label:
                results[label]['total_correct'] += 1
            # Append the confidence score
            results[label]['confidence_scores'].append(float(prediction))

    # Calculate average confidences and update the decision threshold
    results['ai_generated']['average_confidence'] =
sum(results['ai_generated']['confidence_scores']) /
len(results['ai_generated']['confidence_scores'])
```

```
    results['real']['average_confidence'] =
sum(results['real']['confidence_scores']) /
len(results['real']['confidence_scores'])

    # Calculate percentages and overall accuracy based on the new average
confidence
    results['results']['percentage_ai_accuracy'] =
(results['ai_generated']['total_correct'] /
results['ai_generated']['total']) * 100
    results['results']['percentage_real_accuracy'] =
(results['real']['total_correct'] / results['real']['total']) * 100
    overall_correct = results['ai_generated']['total_correct'] +
results['real']['total_correct']
    overall_total = results['ai_generated']['total'] +
results['real']['total']
    results['results']['overall_percentage_accuracy'] = (overall_correct /
overall_total) * 100

    return results

# Initialize a list to hold results from all models
all_results = []

# Evaluate each model and add results
for model in model_info:
    results = evaluate_model(model['path'], model['target_size'])
    all_results.append(results)

# Save results to a JSON file
results_path = 'evaluation-results/results.json'

# Different results save location for random-evaluation.py
# Save results to a JSON file
results_path = 'evaluation-results/random-input-results.json'

os.makedirs(os.path.dirname(results_path), exist_ok=True)  # Ensure the
directory exists
with open(results_path, 'w') as f:
    json.dump(all_results, f, indent=4)

print("Evaluation completed and saved to", results_path)
```

Sample Directory Structure



From top to bottom

- ai-recog: Parent folder, this folder represent the AI Recognition dataset.
    - eval-dataset: 30 randomly selected images of each category from the dataset the model has been trained on, but still unseen to the model.
    - evaluation-results: Output Directory for the above evaluation script.
    - graphs: Model plots, shows training & validation accuracy + training & validation loss.
    - models: When all models have been trained will contain all 3 models.
        - refined-model.h5
        - refined-early-model.h5
        - transfer-model.h5
    - overflow-dataset: Excess images from equalizing the class entries for AI-Generated and real images.
    - raw-dataset: Original dataset with no augmentation.
    - screenshots: Research record keeping.
    - train-val-dataset: Training & Validation dataset, files that the models are trained on.
    - Scripts
        - data-equalizer.py: Randomly selects the specified amount of images from the source directory, moves unselected excess images to the overflow directory, they wont be used for training.

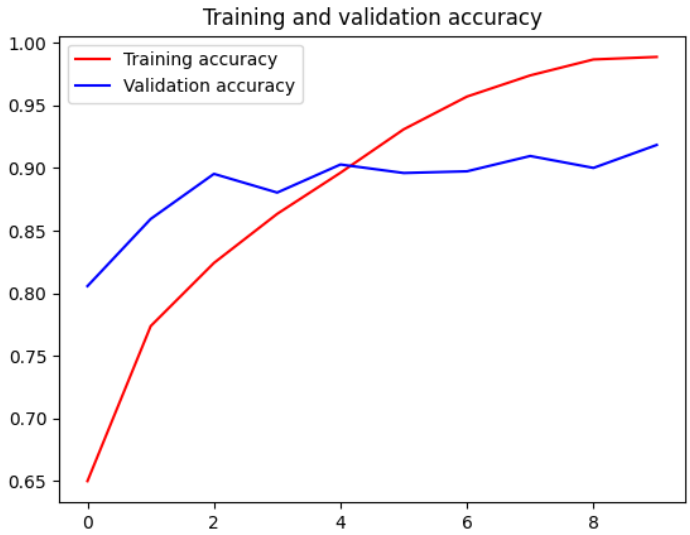
Analysis

Model Graphs

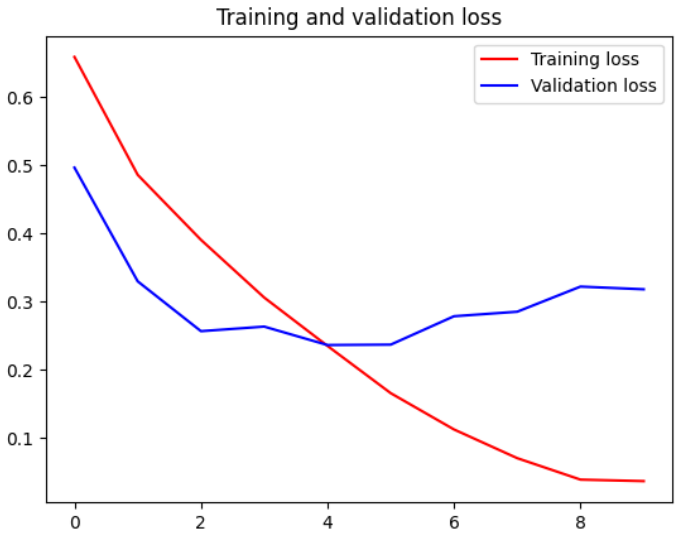
Refined Model

AI-Recog Dataset

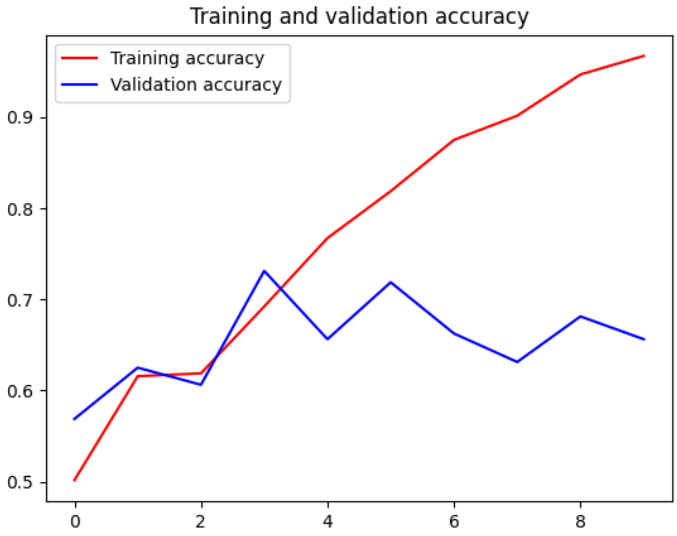
- Training & Validation Accuracy



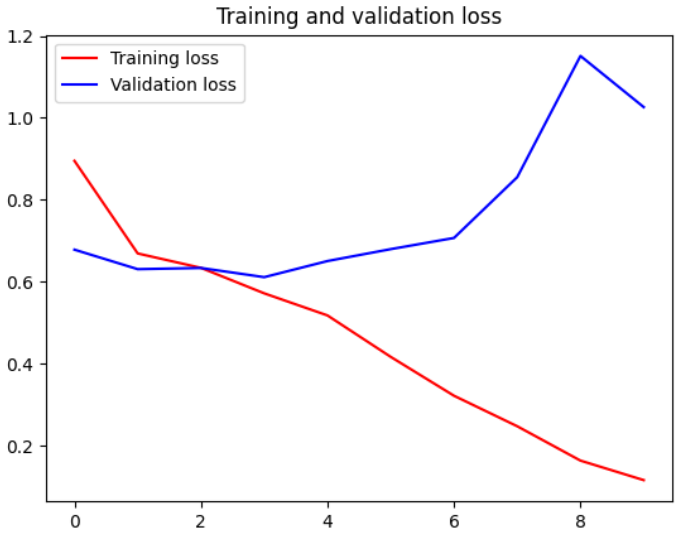
- Training & Validation Loss



- Training & Validation Accuracy

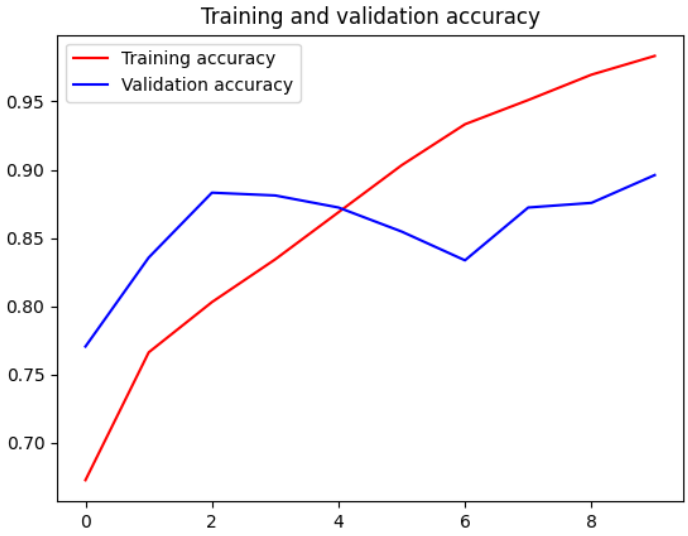


- Training & Validation Loss

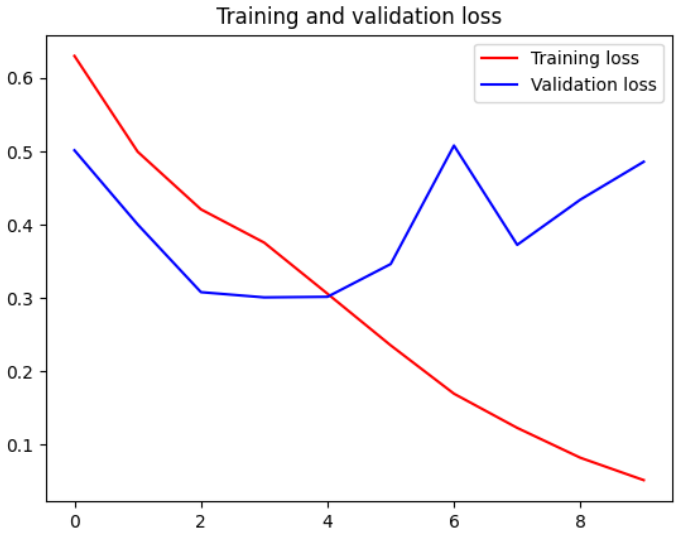


Artifact Dataset

- Training & Validation Accuracy

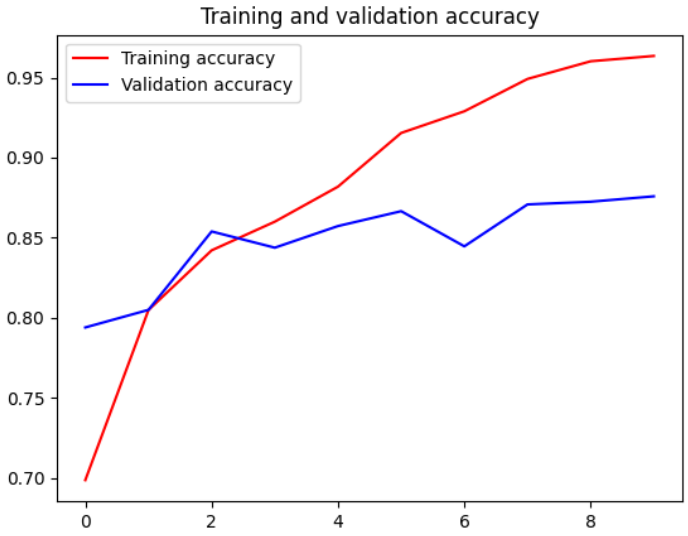


- Training & Validation Loss

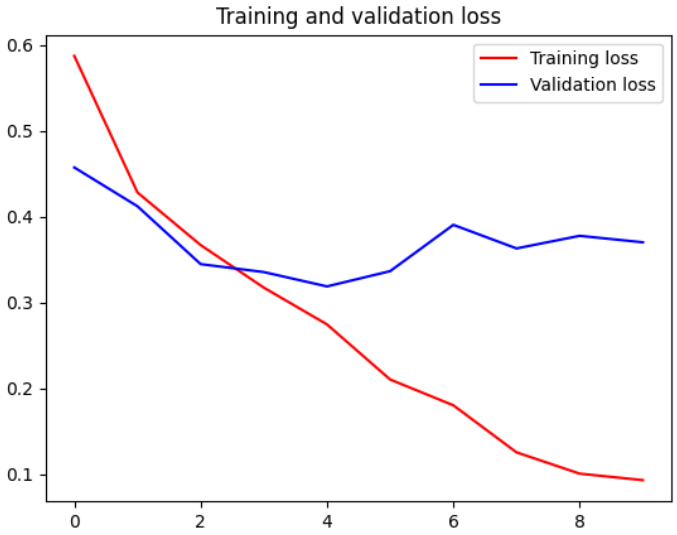


CIFAKE Dataset

- Training & Validation Accuracy

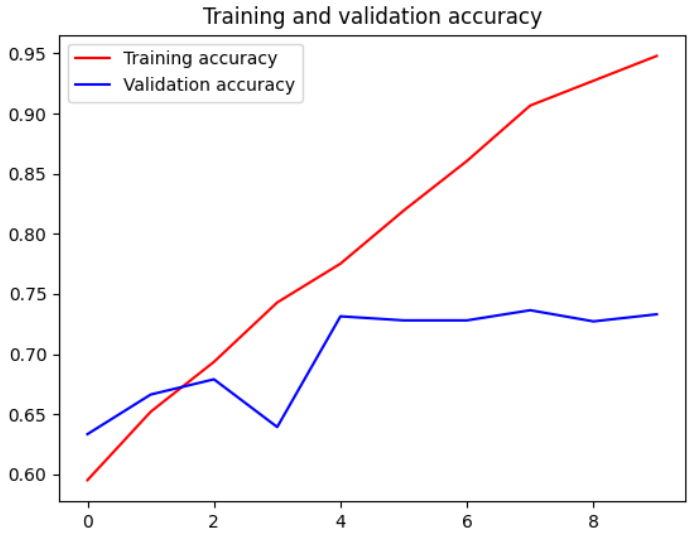


- Training & Validation Loss

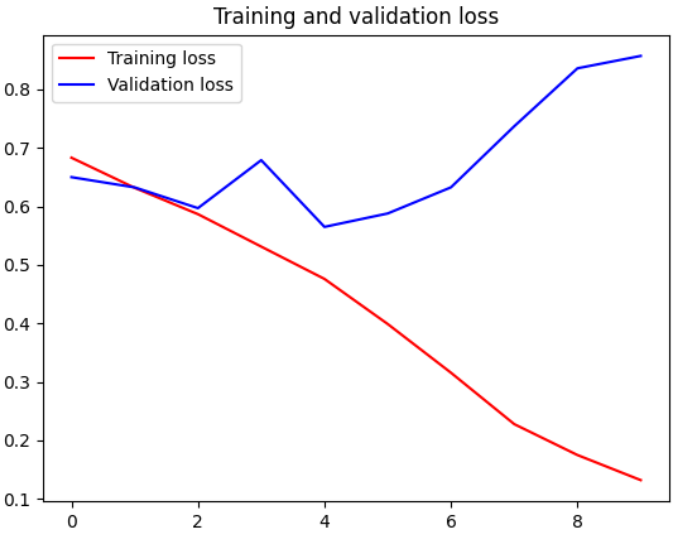


Face Dataset

- Training & Validation Accuracy



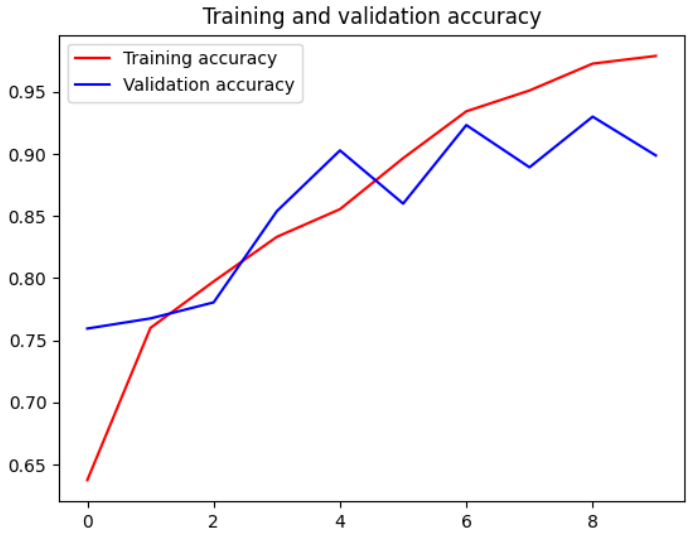
- Training & Validation Loss



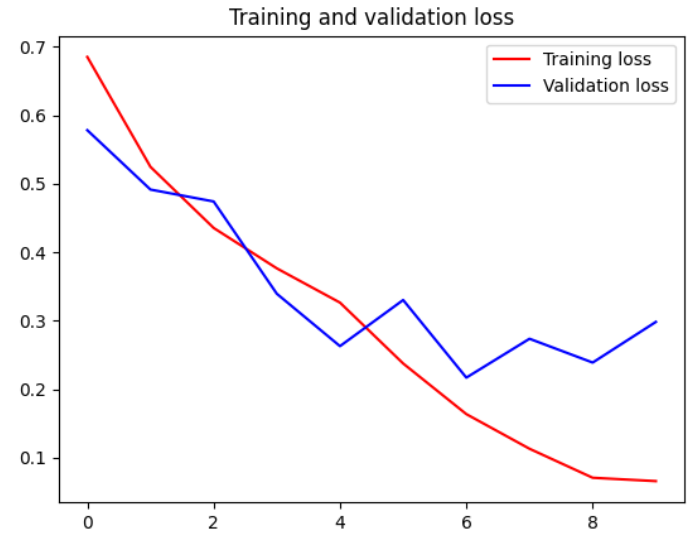
Refinded Model with Early Stopping

AI-Recog Dataset

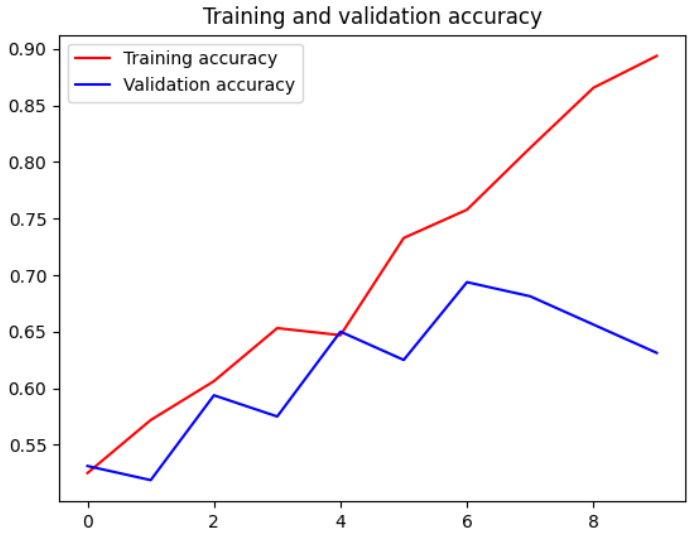
- Training & Validation Accuracy



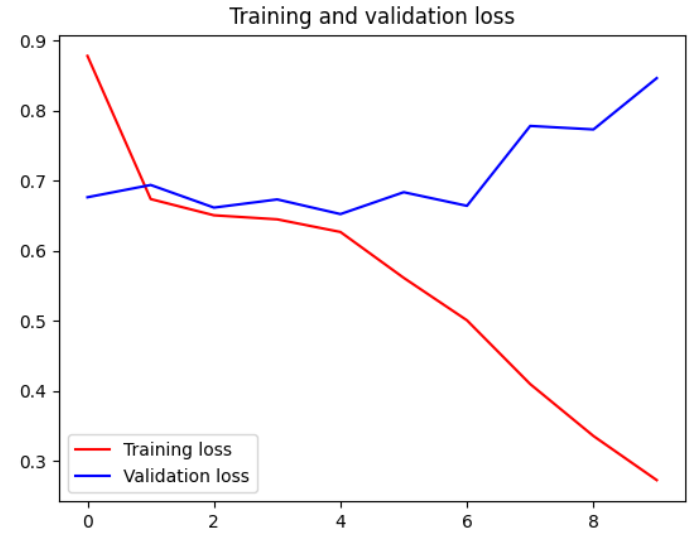
- Training & Validation Loss



- Training & Validation Accuracy

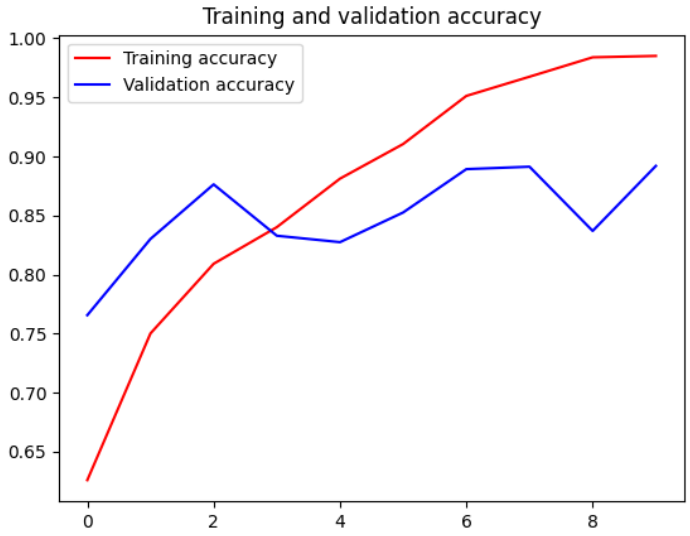


- Training & Validation Loss

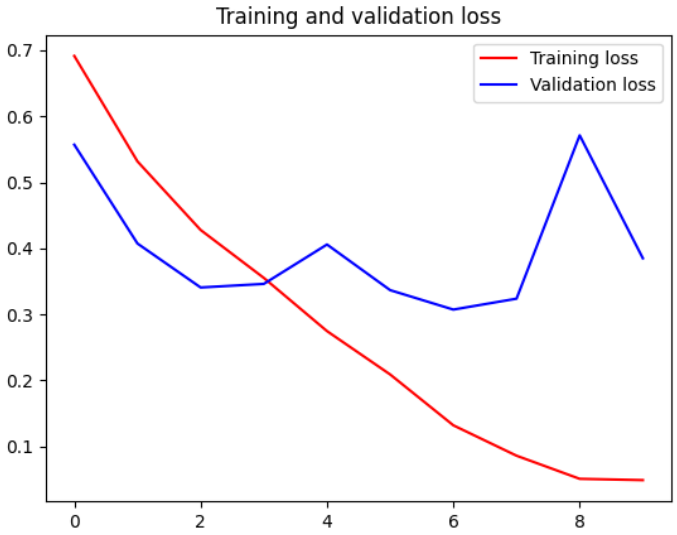


Artifact Dataset

- Training & Validation Accuracy

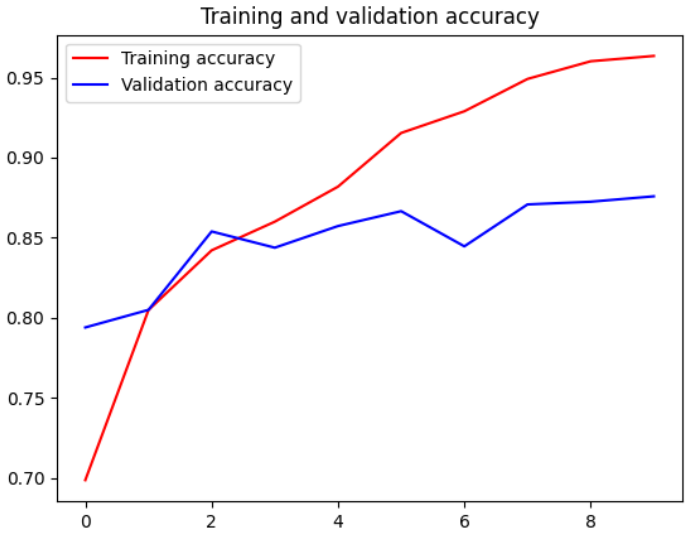


- Training & Validation Loss

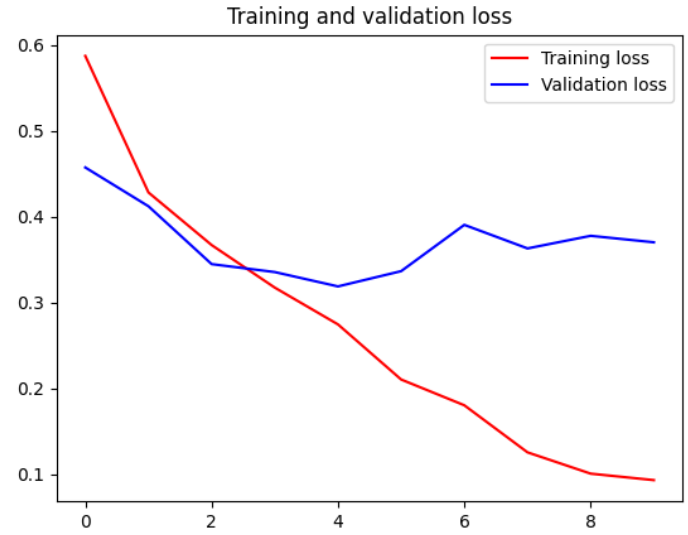


CIFAKE Dataset

- Training & Validation Accuracy

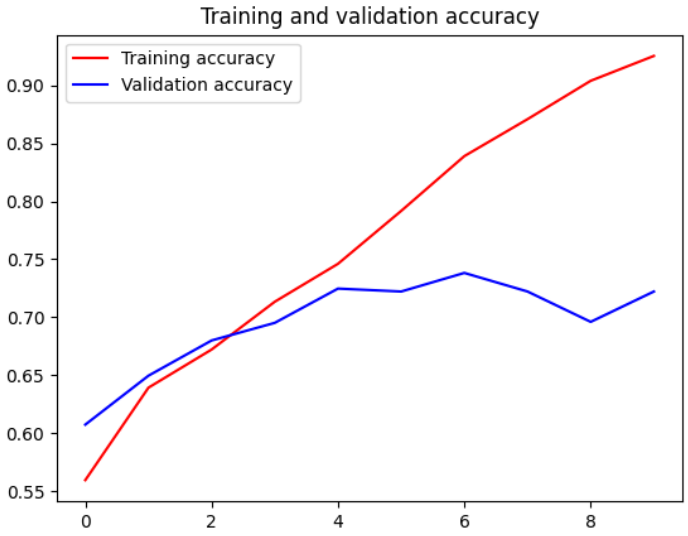


- Training & Validation Loss

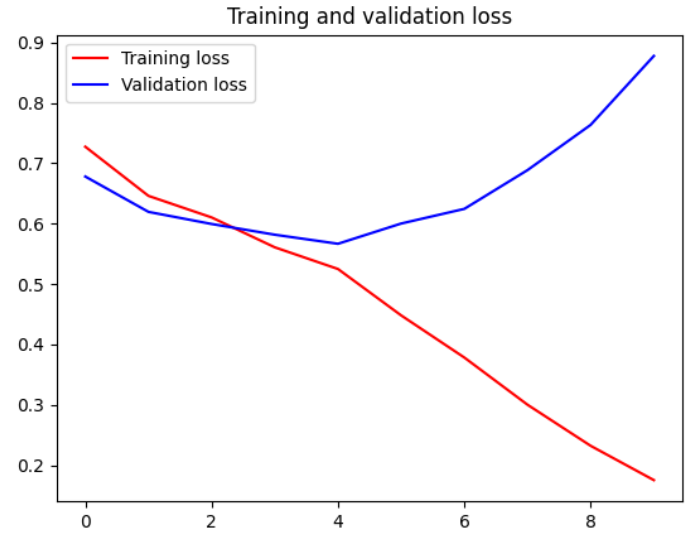


Face Dataset

- Training & Validation Accuracy



- Training & Validation Loss



Findings

Training / Validation Scores & Losses

Refined Model

AI-Recog Dataset

- Training Loss: 0.0368
- Training Accuracy: 0.9888
- Validation Loss: 0.3180
- Validation Accuracy: 0.9185

AI Art Dataset

- Training Loss: 0.1157
- Training Accuracy: 0.9672
- Validation Loss: 1.0263
- Validation Accuracy: 0.6562

Artiface Dataset

- Training Loss: 0.0519
- Training Accuracy: 0.9834
- Validation Loss: 0.4860
- Validation Accuracy: 0.8961

CIFAKE Dataset

- Training Loss: 0.0931
- Training Accuracy: 0.9635
- Validation Loss: 0.3702
- Validation Accuracy: 0.8758

Face Dataset

- Training Loss: 0.1322
- Training Accuracy: 0.9479
- Validation Loss: 0.8570
- Validation Accuracy: 0.7331

Refined Model with Early Stopping

AI-Recog Dataset

- Training Loss: 0.0660
- Training Accuracy: 0.9789
- Validation Loss: 0.2984
- Validation Accuracy: 0.8988

AI Art Dataset

- Training Loss: 0.2725
- Training Accuracy: 0.893
- Validation Loss: 0.8466
- Validation Accuracy: 0.6313

Artiface Dataset

- Training Loss: 0.0519
- Training Accuracy: 0.9834
- Validation Loss: 0.4860
- Validation Accuracy: 0.8961

CIFAKE Dataset

- Training Loss: 0.0582
- Training Accuracy: 0.9798
- Validation Loss: 0.8768
- Validation Accuracy: 0.8193

Face Dataset

- Training Loss: 0.1752
- Training Accuracy: 0.9254
- Validation Loss: 0.8777
- Validation Accuracy: 0.7221

Transfer Model

AI-Recog Dataset

- Training Loss: 0.5980
- Training Accuracy: 0.6820
- Validation Loss: 0.5294
- Validation Accuracy: 0.7425

AI Art Dataset

- Training Loss: 0.6840
- Training Accuracy: 0.5406
- Validation Loss: 0.6835
- Validation Accuracy: 0.5562

Artiface Dataset

- Training Loss: 0.5943
- Training Accuracy: 0.6825
- Validation Loss: 0.5631
- Validation Accuracy: 0.7179

CIFAKE Dataset

- Training Loss: 0.5676
- Training Accuracy: 0.7131
- Validation Loss: 0.5550
- Validation Accuracy: 0.7078

Face Dataset

- Training Loss: 0.6887
- Training Accuracy: 0.5693
- Validation Loss: 0.6886947154998779,
- Validation Accuracy: 0.5692567825317383

Real-World Evaluation Script Accuracy

Refined Model

AI-Recog Dataset

- Unseen data from training Dataset
  - AI detection accuracy – **86.6%**
  - Real Image detection accuracy – **76.6%**
  - Overall percentage accuracy – **81.6%**


```
"results": {
    "percentage_ai_accuracy": 86.66666666666667,
    "percentage_real_accuracy": 76.66666666666667,
    "overall_percentage_accuracy": 81.66666666666667
}
```


- Unseen data from various unseen Datasets
  - AI detection accuracy – **83.6%**
  - Real Image detection accuracy – **59.5%**
  - Overall percentage accuracy – **71.6%**


```
"results": {
    "percentage_ai_accuracy": 83.66336633663366,
    "percentage_real_accuracy": 59.5,
    "overall_percentage_accuracy": 71.64179104477611
}
```


AI Art Dataset

-

Artiface Dataset

- Unseen data from training Dataset
  - AI detection accuracy – **16%**
  - Real Image detection accuracy – **84%**
  - Overall percentage accuracy – **50%**


```
"results": {
    "percentage_ai_accuracy": 16.0,
    "percentage_real_accuracy": 84.0,
    "overall_percentage_accuracy": 50.0
}
```


- Unseen data from various unseen Datasets
  - AI detection accuracy – **85.6%**
  - Real Image detection accuracy – **61.5%**
  - Overall percentage accuracy – **73.6%**

```
"results": {
    "percentage_ai_accuracy": 85.64356435643565,
    "percentage_real_accuracy": 61.5,
    "overall_percentage_accuracy": 73.6318407960199
}
```


CIFAKE Dataset

-

Face Dataset

-

Refined Model with Early Stopping

AI-Recog Dataset

- Unseen data from training Dataset
  - AI detection accuracy – **70%**
  - Real Image detection accuracy – **76.6%**
  - Overall percentage accuracy – **73.3%**


```
"results": {
    "percentage_ai_accuracy": 70.0,
    "percentage_real_accuracy": 76.66666666666667,
    "overall_percentage_accuracy": 73.33333333333333
}
```


- Unseen data from various unseen Datasets
  - AI detection accuracy – **77.7%**
  - Real Image detection accuracy – **68.5%**
  - Overall percentage accuracy – **73.1%**


```
"results": {
    "percentage_ai_accuracy": 77.72277227722772,
    "percentage_real_accuracy": 68.5,
    "overall_percentage_accuracy": 73.13432835820896
}
```


AI Art Dataset

-

Artifice Dataset

- Unseen data from training Dataset
  - AI detection accuracy – **27%**
  - Real Image detection accuracy – **81%**
  - Overall percentage accuracy – **54%**


```
"results": {
    "percentage_ai_accuracy": 27.0,
    "percentage_real_accuracy": 81.0,
    "overall_percentage_accuracy": 54.0
}
```


- Unseen data from various unseen Datasets

- AI detection accuracy – **81.1%**
- Real Image detection accuracy – **65.5%**
- Overall percentage accuracy – **73.3%**


```
"results": {
    "percentage_ai_accuracy": 81.1881188118812,
    "percentage_real_accuracy": 65.5,
    "overall_percentage_accuracy": 73.38308457711443
}
```


CIFAKE Dataset

-

Face Dataset

Transfer Model

AI-Recog Dataset

- Unseen data from training Dataset
  - AI detection accuracy – **0%**
  - Real Image detection accuracy – **100%**
  - Overall percentage accuracy – **50%**


```
"results": {
    "percentage_ai_accuracy": 0.0,
    "percentage_real_accuracy": 100.0,
    "overall_percentage_accuracy": 50.0
}
```


- Unseen data from various unseen Datasets
  - AI detection accuracy – **1%**
  - Real Image detection accuracy – **99.5%**
  - Overall percentage accuracy – **50%**


```
"results": {
    "percentage_ai_accuracy": 0.9900990099009901,
    "percentage_real_accuracy": 99.5,
    "overall_percentage_accuracy": 50.0
}
```


AI Art Dataset

-

Artiface Dataset

- Unseen data from training Dataset
  - AI detection accuracy – **0%**
  - Real Image detection accuracy – **100%**
  - Overall percentage accuracy – **50%**


```
"results": {
    "percentage_ai_accuracy": 0.0,
    "percentage_real_accuracy": 100.0,
    "overall_percentage_accuracy": 50.0
}
```

- Unseen data from various unseen Datasets
    - AI detection accuracy – **3.9%**
    - Real Image detection accuracy – **98.5%**
    - Overall percentage accuracy – **50%**


```
],
"results": {
    "percentage_ai_accuracy": 3.9603960396039604,
    "percentage_real_accuracy": 98.5,
    "overall_percentage_accuracy": 50.99502487562189
}
```


CIFAKE Dataset

-

Face Dataset

Results

As can be seen from the results the models show some skill in detecting AI and real images. The accuracy are however not very high and do show some uncertainty. The transfer model that was used shows no skill in any of the tests, classifying mostly everything as real. This may be due to the model being trained on ImageNet prior and there being a flaw in the weighting, which is leading to a 'real' image verdict bias.

The refined models show promising results, it is worth noting that the early stop model never needed to call its checkpoint or early stop. This indicates that more could be achieved by allowing the model to run for more epochs or with larger dataset. Unfortunately, this was not feasible as the models take a considerable amount of time to train as is. Interestingly some models performed worse on unseen data from their original dataset using these models. The random eval dataset is made up of various diffusion generated images and a collection of real images ranging from the flickr dataset to stock images.

Future Work

As alluded to from the results, the early stopping capable model could've done with more epochs or a larger dataset. This may have led to even higher accuracy in classifying the images.

With the release of OpenAI's SORA an adaption of this tool could be made to take multiple frames from videos generated and run them through the model. This would enable detection of AI-Generated video.

Conclusion

Acknowledgements

I would like to express my deepest gratitude to my thesis advisor, Peter Alexander for his unwavering support and guidance throughout the research and writing of this thesis, he always displayed a willingness to accommodate any request for guidance for which I am truly grateful. His insights, expertise and pointers have been invaluable, and his encouragement made this work possible. I also would like to thank all my past professors and lecturers, whose teachings have profoundly shaped my academic journey and equipped me with the knowledge and skills that were essential for my research.

My sincere appreciation extends to my family for their love and unwavering support throughout my academic

journey. Thank you to my parents, Alan & Karen, for their sacrifices and for instilling in me the value of hard work. I would also like to acknowledge my classmates, for their support and for the advice and ideas they've shared with me throughout the completion of this research.

This thesis stands as a milestone in my academic journey, and I am grateful for everyone who made it possible.

References

Adobe. (2023, November 21). *Use the Object Selection tool, Select Subject, Quick Selection, or Magic Wand tools to make selections in Photoshop.* Retrieved from helpx.adobe: https://helpx.adobe.com/ie/photoshop/using/making-quick-selections.html

Agarwal, S., El-Gaaly, T., Farid, H., & Lim, S.-N. (2020). *Detecting Deep-Fake Videos from.* Univeristy of California, Berkeley, Facebook Research.

Awsaf, Paul, B., Sarker, N. H., & Hakim, Z. I. (2023, February 24). *ArtiFact: Real and Fake Image Dataset*. Retrieved from Kaggle: https://www.kaggle.com/datasets/awsaf49/artifact-dataset

Banerjee, A. (2023). *Detecting AI-Generated Images Created by Diffusion.* Dr. Xiaoyan Li.

Bianco, T., Castellano, G., Scaringi, R., & Vessio, G. (2023). *Identifying AI-Generated Art with Deep Learning*. Bari: Department of Computer Science, University of Bari Aldo Moro.

BOWMAN, C. (2024, February). *AI Generated Images vs Real Images*. Retrieved from Kaggle: https://www.kaggle.com/datasets/cashbowman/ai-generated-images-vs-real-images/data

Brownlee, J. (2019, August 6). *A Gentle Introduction to Dropout for Regularizing Deep Neural Networks*. Retrieved from machinelearningmastery: https://machinelearningmastery.com/dropout-for-regularizing-deep-neural-networks/

Brownlee, J. (2020, Augst 25). *Use Early Stopping to Halt the Training of Neural Networks At the Right Time*. Retrieved from machinelearningmastery: https://machinelearningmastery.com/how-to-stop-training-deep-neural-networks-at-the-right-time-using-early-stopping/

Brownlee, J. (2022, August 2). *TensorFlow 2 Tutorial: Get Started in Deep Learning with tf.keras*. Retrieved from machinelearningmastery: https://machinelearningmastery.com/tensorflow-tutorial-deep-learning-with-tf-keras/

Chang, Z., Koulieris, G., & Shum, H. P. (2023). *On the Design Fundamentals of Diffusion Models: A Survey.* IEEE.

Gatys, L. A., Ecker, A. S., & Bethge, M. (2015). *A Neural Algorithm of Artistic Style.* Werner Reichardt Centre for Integrative Neuroscience and Institute of Theoretical Physics, University of Tubingen, Germany ¨, Bernstein Center for Computational Neuroscience, Tubingen, Germany ¨, Graduate School for Neural Information Processing, Tubingen.

Gatys, L. A., Ecker, A. S., & Bethge, M. (2016). *Image Style Transfer Using Convolutional Neural Networks.*

geeksforgeeks. (2019, June 20). *Python | shutil.move() method*. Retrieved from geeksforgeeks: https://www.geeksforgeeks.org/python-shutil-move-method/

geeksforgeeks. (2020, December 29). *Create a directory in Python.* Retrieved from geeksforgeeks: https://www.geeksforgeeks.org/create-a-directory-in-python/

geeksforgeeks. (2023, Jan 13). *Get current timestamp using Python*. Retrieved from geeksforgeeks: https://www.geeksforgeeks.org/get-current-timestamp-using-python/

GeeksForGeeks. (2024, Jan 03). *Introduction to TensorFlow*. Retrieved from geeksforgeeks: https://www.geeksforgeeks.org/introduction-to-tensorflow/

geeksforgeeks. (2024, March 14). *os.walk() in Python*. Retrieved from geeksforgeeks: https://www.geeksforgeeks.org/os-walk-python/

GitHub. (2024, February 14). *Local rendezvous is aborting with status: OUT_OF_RANGE: End of sequence warning when iterating over a dataset #62963*. Retrieved from github: https://github.com/tensorflow/tensorflow/issues/62963

Göllner, S. (2022, July 14). *How to reduce training parameters in CNNs while keeping accuracy >99%*. Retrieved from towardsdatascience: https://towardsdatascience.com/how-to-reduce-training-parameters-in-cnns-while-keeping-accuracy-99-a213034a9777

Goodfellow, I. J., Pouget -Abadie, J., Mirza, M., Xu, B., Warde-Farley, D., Ozair, S., . . . Bengio, Y. (2014). *Generative Adversarial Nets.* Montreal: University of Montreal.
Google Research. (2022, May). *Imagen - unprecedented photorealism x deep level of language understanding*. Retrieved from https://imagen.research.google/: https://imagen.research.google/
Google Research. (2022, June). *PARTI - Pathway Autoregressive Text-to-Image Model* . Retrieved from research.google: https://parti.research.google/
Grudin, J. (2017). *A Moving Target: The Evolution of HCI.* Morgan & Claypool Publishers.
Hochreiter, S., & Schmidhuber, J. (1997). *LONG SHORT-TERM MEMORY.* Viganello, Munich: IDSIA - Dalle Molle Institute for Artificial Intelligence & Technical University of Munich.
Jiang, Z., Zhang, J., & Gong, N. Z. (2023). *Evading Watermark based Detection of AI-Generated Content.* Duke University.
Karras, T., Laine, S., & Aila, T. (2019). *A Style-Based Generator Architecture for Generative Adversarial Networks.* NVIDIA.
kmario23. (2018, October 18). *Introduction-to-NumPy-Matplotlib-TensorFlow - understanding-matplotlib.ipynb*. Retrieved from github: https://github.com/kmario23/Introduction-to-NumPy-Matplotlib-TensorFlow/blob/master/understanding-matplotlib.ipynb
Koliha, N. (2024, February). *AI recognition dataset*. Retrieved from Kaggle: https://www.kaggle.com/datasets/superpotato9/dalle-recognition-dataset?resource=download
krishnaik06. (2020, July 1). *Transfer Learning Resnet 50.ipynb*. Retrieved from github: https://github.com/krishnaik06/Deep-Learning-Car-Brand/blob/master/Transfer%20Learning%20Resnet%2050.ipynb
Krizhevsky, A., & Hinton, G. (2009). *Learning Multiple Layers of Features from Tiny Images.* University of Toronto.
Lenz, W. (1920). *Beiträge zum Verständnis der magnetischen Eigenschaften in festen Körpern.* Physikalische Zeitschrift.
Lotfi, A., & Bird, J. J. (2023, March 24). *CIFAKE: Real and AI-Generated Synthetic Images - Can Computer Vision detect when images have been generated by AI?* Retrieved from Kaggle: https://www.kaggle.com/datasets/birdy654/cifake-real-and-ai-generated-synthetic-images
Luccioni, S., Jernite, Y., Thomas, D., Witko, E., Ozoani, E., Fukano, J., . . . Mitchel, M. (2024, February 26). *AI Watermarking 101: Tools and Techniques*. Retrieved from huggingface: https://huggingface.co/blog/watermarking#:~:text=In%20AI%20specifically%2C%20watermarking%20involves,either%20by%20humans%20or%20algorithmically.
Madiega, T. (2023). *Generative AI and watermarking.* European Parliamentary ResearchService.
Marcel, P. K. (2018). *DeepFakes: a New Threat to Face Recognition?* arxiv.
Mascarenhas, S., & Agarwal, M. (2021). *A comparison between VGG16, VGG19 and ResNet50 architecture frameworks for Image Classification.* IEEE.
Niggo. (2020, July 10). *Copy random files by 5 each to different folder*. Retrieved from Stackoverflow: https://stackoverflow.com/questions/62831060/copy-random-files-by-5-each-to-different-folder
Nikneshan, A. (2015, December 28). *Create JSON object with variables from an array*. Retrieved from stackoverflow: https://stackoverflow.com/questions/34489706/create-json-object-with-variables-from-an-array
numpy. (2024). *NumPy Documentation*. Retrieved from numpy: https://numpy.org/doc/stable/index.html
NumPY. (n.d.). *numpy.expand_dims*. Retrieved from numpy: https://numpy.org/doc/stable/reference/generated/numpy.expand_dims.html
OpenAI. (2021, January 5). *DALL·E: Creating images from text*. Retrieved from openai: https://openai.com/index/dall-e/
PYNative. (2024, January 25). *Rename Files in Python*. Retrieved from pynative: https://pynative.com/python-rename-file/
python. (2024). *json — JSON encoder and decoder*. Retrieved from docs.python: https://docs.python.org/3/library/json.html#
python. (2024). *os — Miscellaneous operating system interfaces*¶. Retrieved from docs.python: os — Miscellaneous operating system interfaces¶
python. (2024). *os.path — Common pathname manipulations*. Retrieved from python docs: https://docs.python.org/3/library/os.path.html

python. (2024). *random — Generate pseudo-random numbers*. Retrieved from docs.python: https://docs.python.org/3/library/random.html
python. (2024). *shutil — High-level file operations*. Retrieved from docs.python: https://docs.python.org/3/library/shutil.html
Ramesh, A., Dhariwal, P., Nichol, A., Chu, C., & Chen, M. (2022). *Hierarchical Text-Conditional.* OpenAI.
Rooy, J. L. (2013, August 23). *How to do a recursive sub-folder search and return files in a list?* Retrieved from stackoverflow: https://stackoverflow.com/questions/18394147/how-to-do-a-recursive-sub-folder-search-and-return-files-in-a-list
Salpekar, O. (2020). *DeepFake Image Detection.* Stanford: Stanford University.
Sarin, S. (2019, December 29). *VGGNet vs ResNet*. Retrieved from towardsdatascience: https://towardsdatascience.com/vggnet-vs-resnet-924e9573ca5c
Sarkar, A., Mai, H., Mahapatra, A., Lazebnik, S., Forsyth, D., & Bhattad, A. (2023). *Shadows Don't Lie and Lines Can't Bend! Generative Models don't know Projective Geometry...for now.* University of Illinois, Toyota Technological Institute at Chicago.
sentdex. (2018, August 31). *How to use your trained model - Deep Learning basics with Python, TensorFlow and Keras p.6*. Retrieved from youtube: https://www.youtube.com/watch?v=A4K6D_gx2Iw
simplilearn. (2024, February 15). *What Is Keras: The Best Introductory Guide To Keras*. Retrieved from simplilearn: https://www.simplilearn.com/tutorials/deep-learning-tutorial/what-is-keras#:~:text=Keras%20is%20a%20high%2Dlevel,multiple%20backend%20neural%20network%20computation.
Singh, R. R. (2020, April 13). *Basics of TensorFlow 2.0 and Training a Model*. Retrieved from medium: https://medium.com/analytics-vidhya/basics-of-tensorflow-2-0-and-training-a-model-bf33cf4dff5a
Srinivasan, S. (2024). *Detecting AI fingerprints: A guide to watermarking and beyond.* Washington: The Brookings Institution.
Tancik, M., Mildenhall, B., & Ng, R. (2020). *StegaStamp: Invisible Hyperlinks in Physical Photographs.* Berkeley: University of California, Berkeley.
tensorflow. (n.d.). *Displaying image data in TensorBoard* . Retrieved from tensorflow: https://www.tensorflow.org/tensorboard/image_summaries
tensorflow. (n.d.). *Image classification* . Retrieved from tensorflow: https://www.tensorflow.org/tutorials/images/classification
tensorflow. (n.d.). *tf.keras.callbacks.EarlyStopping* . Retrieved from tensorflow: https://www.tensorflow.org/api_docs/python/tf/keras/callbacks/EarlyStopping
tensorflow. (n.d.). *tf.keras.callbacks.ModelCheckpoint* . Retrieved from https://www.tensorflow.org/api_docs/python/tf/keras/callbacks/ModelCheckpoint
tensorflow. (n.d.). *tf.keras.layers.Conv2D* . Retrieved from tensorflow: https://www.tensorflow.org/api_docs/python/tf/keras/layers/Conv2D
tensorflow. (n.d.). *tf.keras.layers.GlobalAveragePooling2D* . Retrieved from tensorflow: https://www.tensorflow.org/api_docs/python/tf/keras/layers/GlobalAveragePooling2D
tensorflow. (n.d.). *tf.keras.layers.MaxPool2D* . Retrieved from tensorflow: https://www.tensorflow.org/api_docs/python/tf/keras/layers/MaxPool2D
tensorflow. (n.d.). *tf.keras.preprocessing.image.ImageDataGenerator* . Retrieved from tensorflow: https://www.tensorflow.org/api_docs/python/tf/keras/preprocessing/image/ImageDataGenerator
tensorflow. (n.d.). *tf.keras.Sequential* . Retrieved from tensorflow: https://www.tensorflow.org/api_docs/python/tf/keras/Sequential
tensorflow. (n.d.). *Training & evaluation with the built-in methods* . Retrieved from tensorflow: https://www.tensorflow.org/guide/keras/training_with_built_in_methods
tensorflow. (n.d.). *Transfer learning and fine-tuning* . Retrieved from tensorflow: https://www.tensorflow.org/tutorials/images/transfer_learning
Turing, A. (1950). *Computing Machinery and Intelligence.* Mind.
Vasse'i, R. M., & Udoh, G. (2024). *In Transparency We Trust? Evaluating the Effectiveness of Watermarking and Labeling AI-Generated Content.* Mozilla Research.
w3 schools. (n.d.). *TensorFlow Models - Using the Model*. Retrieved from w3schools: https://www.w3schools.com/ai/ai_tensorflow_model.asp
w3schools. (n.d.). *TensorFlow Models - Creating a Model, Adding Layers to the Model, Compiling the Model,*

Training the Model. Retrieved from w3schools: https://www.w3schools.com/ai/ai_tensorflow_model.asp

Winograd, T. (1971). *Procedures as a represntation for data in a computer program for understanding natural language.* Boston: Massachusetts Insitute of Technology.

Xhlulu. (2020). *140k Real and Fake Faces*. Retrieved from Kaggle: https://www.kaggle.com/datasets/xhlulu/140k-real-and-fake-faces/data

Yaeger, L. (2002). *A Brief, Early History of Computer Graphics in Film.*

Zhang, R., Wang, O., Jain, I., & Epstein, D. C. (2023). *Online Detection of AI-Generated Images.* Adobe Research.

Zhang, X., Karaman, S., & Chang, S.-F. (2019). *Detecting and Simulating Artifacts in GAN Fake.* Columbia: Columbia University.