A

PROJECT REPORT

On

AI VS REAL IMAGES

PIXELTRUTH

Submitted by

(Group No.-5)

Rohit	(220657)
Radhika Bhati	(220650)
Harshita Rupani	(220333)
Deepanshu Aggarwal	(220660)

CSE-4 - Batch 2022-2026

Submitted to

Dr. Hirdesh Kumar Pharasi

(Assistant Professor, School of Engineering and Technology)

May 2024

CANDIDATE'S DECLARATION

We Deepanshu Aggarwal, Radhika Bhati, Rohit, Harshita Rupani hereby declare that the project entitled ″PIXELTRUTH " in fulfilment of completion of the 4th-semester course –PRJ 2 as part of the Bachelor of Technology (B. Tech) program at the School of Engineering and Technology, BML Munjal University is an authentic record of our work carried out under the supervision of Dr. Hirdesh Kumar Pharasi. Due acknowledgments have been made in the text of the project to all other materials used.

This project was done in full compliance with the requirements and constraints of the prescribed curriculum.

1.Rohit

2.Radhika Bhati

3.Harshita Rupani

4.Deepanshu Aggarwal

SUPERVISOR'S DECLARATION

This is to certify that the above statement made by the candidate is correct to the best of my knowledge.

Faculty Supervisor Name: Dr. Hirdesh Kumar Pharasi

Signature:

ABSTRACT

In today's digitally saturated world, the distinction between authentic visual content and AI-generated imagery has become increasingly blurred, posing significant challenges in the realms of misinformation, deception, and ethical integrity. The proliferation of AI-generated images, capable of seamlessly mimicking reality, has catalyzed the dissemination of fabricated events, false narratives, and manipulated information, eroding trust and exacerbating societal divisions. In response to this pressing issue, our project, "PixelTruth," endeavors to develop a robust technological solution aimed at restoring transparency, promoting ethical standards, and empowering users to discern between real and AI-generated images effectively.

The core objective of "PixelTruth" is to engineer a sophisticated technology framework capable of accurately differentiating between authentic photographs and AI-generated imagery. To achieve this, we adopted a multifaceted approach encompassing data preprocessing, model architecture design, extensive training, and rigorous evaluation. Leveraging a diverse dataset comprising both real and AI-generated images, we meticulously curated and preprocessed the data to ensure uniformity and optimize model performance. Through iterative experimentation and refinement, we devised a Convolutional Neural Network (CNN) model utilizing TensorFlow and Keras, incorporating convolutional layers, max-pooling layers, and dropout layers to mitigate overfitting and enhance generalization.

Keywords:

Digital deception, Proliferation of AI-generated images, Fabricated events, User empowerment , Model robustness, Hardware limitations, Innovative strategies, Image verification, Comprehensive solution.

ACKNOWLEDGEMENT

I am highly grateful to Dr. Hirdesh Kumar Pharasi, BML Munjal University, Gurugram, for providing supervision to carry out the Project from February – May 2024.

Dr. Hirdesh Kumar Pharasi has provided great help in carrying out our work and is acknowledged with reverential thanks. Without the wise counsel and able guidance, it would have been impossible to complete the training in this manner.

We would like to express thanks profusely to thank Dr. Hirdesh Kumar Pharasi, for stimulating me time to time. We would also like to thank entire team of BML Munjal University. We would also thank our groupmates who devoted their valuable time and helped me in all possible ways towards successful completion.

TABLE OF CONTENT

Chapter – 1 INTRODUCTION

In today's digital age, the ease of content creation and dissemination has revolutionized communication, empowering individuals and organizations to share information and ideas effortlessly. However, this proliferation of digital content has also introduced challenges related to authenticity and trustworthiness. The emergence of AI-generated images, capable of seamlessly replicating reality, has blurred the lines between fact and fiction, complicating the task of discerning genuine visual content from fabricated or manipulated imagery.

Against this backdrop, our project, "PixelTruth," arises as a response to the pressing need for reliable methods of authenticating digital media in an era of rampant misinformation and digital deception. The exponential growth of AI-generated images has ushered in a new era of media manipulation, wherein the boundaries between reality and simulation are increasingly difficult to discern. This phenomenon poses significant ethical, societal, and security implications, as fabricated images can be used to spread false narratives, manipulate public opinion, and even fabricate evidence.

The overarching goal of "PixelTruth" is to develop a robust technological framework capable of accurately distinguishing between authentic photographs and AI-generated imagery. By leveraging the power of machine learning and image recognition algorithms, we aim to empower users with the ability to verify the authenticity of visual content encountered online. Through a combination of data preprocessing, model training, and rigorous evaluation, our project seeks to provide a reliable solution for combating the proliferation of AI-generated images and safeguarding the integrity of digital communication channels.

At its core, "PixelTruth" represents a commitment to transparency, accountability, and ethical integrity in the digital realm. By equipping users with the tools and knowledge needed to navigate the complexities of the digital landscape, we aspire to foster a culture of critical

thinking, skepticism, and responsible media consumption. Through collaborative efforts with experts in technology, ethics, and media literacy, we aim to address the challenges posed by AI-generated imagery and promote a safer, more trustworthy online environment for all users.

Chapter – 2 LITERATURE REVIEW

PAPER TITLE	AUTHOR	KEY FINDINGS
Quantifying the Performance Gap between Real and AI-Generated Images (2023) [8]	Shivani Atul Bhinge, Piyush Nagpal	By analyzing various performance metrics like accuracy, precision, recall, and F1 score, this research paper aim to identify disparities between the two image types. Insights into the strengths and limitations of AI-generated synthetic images are provided, along with guidance for incorporating them into real-world applications. The study contributes to advancing computer vision by enhancing understanding of the suitability and reliability of AI-generated synthetic images, using the CIFAKE dataset for model training .
GenImage: A Million-Scale Benchmark for Detecting AI-Generated Images (2024) [9]	Mingjian Zhu et al.	This research paper introduces the GenImage dataset, designed for training and evaluating detectors to distinguish between AI-generated fake images and real ones. It conducts a thorough analysis of dataset characteristics and proposes evaluation tasks resembling real-world conditions, such as cross-generator image

		classification and degraded image classification. By assessing detector performance across different image synthesis techniques and on degraded images, the study aims to advance detector development tailored for AI-generated image detection.
Harnessing Machine Learning for Discerning AI-Generated Synthetic Images (2023) [10]	Yuyang Wang, Yizhi Hao, Amando Xu Cong	The research addresses the challenge of identifying AI-generated synthetic images in digital media using machine learning techniques. By enhancing model accuracy through transfer learning and employing the CIFAKE dataset, which contains both real and fake images, it aims to improve synthetic image detection precision. Comparative analysis with traditional methods, like Support Vector Machine (SVM), evaluates the effectiveness of the optimized models.
CIFAKE: Image Classification and Explainable Identification of AI-Generated Synthetic Images (2023) [11]	JORDAN J. BIRD, AHMAD LOTFI	This research paper aims to distinguish between real and AI-generated photographs by: (a) Introducing the CIFAKE dataset, comprising 120,000 images, for research purposes; (b) Developing a computer vision method to enhance recognition of AI-generated images; (c) Advocating for Explainable AI to understand

		image recognition processes and visualize significant features. It addresses societal and ethical concerns regarding image authenticity amid rapid advancements in AI-generated image generation.

Chapter – 3 OBJECTIVES

The primary objective of "PixelTruth" is to pioneer a comprehensive technological solution that effectively distinguishes between authentic photographs and AI-generated images, thereby combating the proliferation of digital deception and fostering trust and transparency in the digital landscape. In response to the escalating challenges posed by the widespread dissemination of AI-generated media, our project aims to address the following key objectives:

Firstly, we seek to develop a robust technological framework that can accurately discern between real and AI-generated images. Leveraging advanced machine learning techniques, image recognition algorithms, and model optimization strategies, we aim to create a scalable solution capable of adapting to evolving threats in digital deception. By prioritizing the development of a reliable and efficient system, we lay the foundation for effective image authentication and verification processes.

Additionally, "PixelTruth" is driven by a commitment to upholding ethical standards in digital media. Our objective is to mitigate the dissemination of misinformation, prevent the manipulation of public opinion, and combat the propagation of false narratives. Through transparency, accountability, and responsible media consumption, we aim to foster a culture of integrity and trust in the digital realm, thereby promoting a more informed and discerning society.

Furthermore, central to our mission is the empowerment of users with the tools and knowledge needed to navigate the complexities of the digital landscape. By offering a user-friendly interface and intuitive image authentication capabilities, we democratize access to digital authenticity, enabling individuals and organizations to make informed decisions about the veracity of visual content encountered online. Through education, awareness, and empowerment, we aim to cultivate a community of critical thinkers capable of discerning truth from fiction in an era of rampant digital deception.

Chapter – 4 FLOW OF DATA PRE-PROCESSING

Our data pre-processing pipeline was designed to ensure that the dataset used for training and testing the model was comprehensive, consistent, and optimized for performance. Here is a detailed explanation of each step involved:

Dataset Acquisition:

We started by acquiring the CIFAKE dataset, which includes 60,000 AI-generated images and 60,000 real images from the CIFAR-10 dataset. The CIFAR-10 dataset is a well-known collection of images used for machine learning and computer vision tasks.

Dataset Categorization:

The acquired dataset was categorized into two classes: REAL (images from CIFAR-10) and FAKE (AI-generated images). This classification was crucial for training the model to distinguish between authentic and synthetic images.

Dataset Supplementation:

To enhance the diversity of our dataset, we supplemented it with 660 additional images via web scraping. This included 330 AI-generated images and 330 real photographs. These images were collected to ensure the dataset captured a wide range of visual features and scenarios.

Dataset Reduction:

Due to hardware limitations, we had to reduce the size of the training and testing datasets. The training dataset was reduced by 70%, and the testing dataset was reduced by 90%. This resulted in a revised dataset consisting of 30,000 images for training (15,000 real and 15,000 AI-generated) and 2,000 images for testing (1,000 real and 1,000 AI-generated).

Image Naming and Formatting:

We standardized the naming convention of the images by replacing spaces with underscores. This

step was necessary to avoid issues related to file naming during data processing.

Image Preprocessing:

Resizing: All images were resized to a consistent size of 48x48 pixels. Resizing ensured uniform dimensions, which is essential for efficient processing and analysis by the Convolutional Neural Network (CNN) model.

Normalization: The pixel values of all images were normalized to the range [0,1]. Pixel normalization standardizes the pixel values, which helps in optimizing model performance by ensuring that the model processes the image data uniformly.

Initial Model Training:

We initially trained the model successfully on the reduced dataset. After achieving satisfactory results, we decided to increase the dataset size to improve model accuracy and robustness.

Attempted Re-training with Larger Dataset:

We attempted to re-train the model on a larger dataset of 70,000 images (35,000 real and 35,000 AI-generated). However, we faced hardware limitations once again, which necessitated another reduction in dataset size.

Final Model Training:

Ultimately, we successfully re-trained the model using a dataset of approximately 50,000 images (25,000 real and 25,000 AI-generated) along with the 660 web-scraped images.

Additionally, we compiled a set of real images of various Indian celebrities from Google and generated corresponding AI-generated images using Leonardo.ai. These images were also included in the dataset to further enhance the model's ability to differentiate between real and AI-generated content.

Chapter – 5 METHODOLOGY

5.1. Introduction to Languages

Python

Python is the primary language used for data analysis, machine learning, and backend development in your project. Its simplicity, readability, and extensive libraries such as Pandas, NumPy, and Scikit-learn make it well-suited for handling and analyzing large datasets, implementing machine learning algorithms, and processing data efficiently.

HTML (Hypertext Markup Language)

HTML is utilized for creating the structure and content of web pages in your project. It defines the elements and layout of the user interface, including forms, buttons, text, and other components necessary for presenting information to users.

5.2.Data Preparation

The foundation of our project's methodology lies in the careful curation and preparation of the dataset. We began by acquiring the CIFAKE dataset, consisting of 60,000 AI-generated images and an equal number of real images from CIFAR-10. To augment the dataset, we performed web scraping to collect an additional 660 images, comprising both AI-generated and real photographs. Due to hardware limitations, we prudently reduced the size of the training and testing datasets, resulting in a revised dataset of 50,000 images for training and 5,000 images for testing. To ensure uniformity and optimal model performance, we standardized the dataset by resizing all images to a fixed size of 48x48 pixels and normalizing pixel values to the range [0,1]. Additionally, we conducted data preprocessing by replacing spaces in image names with underscores, facilitating efficient processing and analysis.

5.3. Model Architecture Design

We designed a Convolutional Neural Network (CNN) model using Keras, a high-level neural networks API running on top of TensorFlow. The architecture of our model consisted of sequential layers, each serving a specific purpose in the process of image classification.

Input Layer: The input shape of our images was (48, 48, 3), indicating an image size of 48x48 pixels with three color channels (RGB).

Convolutional Layers: We incorporated four convolutional layers, each followed by a Rectified Linear Unit (ReLU) activation function. The convolutional layers were responsible for detecting patterns and features within the input images. We used filters of size (3, 3) with increasing numbers of filters (32, 64, 128, and 256) in successive layers to extract hierarchical features.

MaxPooling Layers: After each convolutional layer, we added a MaxPooling layer with a pool size of (2, 2) to downsample the feature maps, reducing computational complexity and extracting dominant features.

Dropout Layers: To prevent overfitting and improve model generalization, we incorporated dropout layers after each max-pooling layer with a dropout rate of 0.2. Dropout randomly deactivates a fraction of neurons during training, forcing the network to learn more robust features.

Flatten Layer: Following the final convolutional layer, we flattened the feature maps into a one-dimensional vector to feed into the dense layers.

Dense Layers: We added three fully connected dense layers with ReLU activation functions. The

first dense layer had 128 units, followed by dropout regularization with a dropout rate of 0.5 to mitigate overfitting. The subsequent dense layer had 64 units, again followed by dropout regularization. The final dense layer had a single neuron with a sigmoid activation function, producing a binary output indicating the likelihood of the input image being AI-generated or real.

5.4. Model Training

Having defined the model architecture, we compiled the model using the Adam optimizer and binary cross-entropy loss function. The training process involved feeding the model with the training data for 15 epochs while periodically validating its performance on the testing data. This iterative process allowed the model to learn and adapt to the characteristics of the dataset, optimizing its ability to distinguish between real and AI-generated images.

5.5. Model Evaluation

Upon completing the training process, we saved the trained model for future use and evaluation. To assess its performance, we loaded the model and computed evaluation metrics such as accuracy, precision, recall, and F1-score using the classification report from scikit-learn. This comprehensive evaluation provided insights into the model's efficacy in accurately classifying images and discriminating between real and AI-generated content.

5.6. Deployment

The culmination of our methodology was the deployment of the trained model on a user-friendly web interface. Leveraging Streamlit, we created a platform where users could upload downloaded images or input image URLs for verification. Upon submission, the model processed the images and presented the results, indicating whether the image was AI-generated or real. This deployment facilitated seamless interaction with the model, enabling users to verify image authenticity with ease.

Chapter – 6 RESULTS AND DISCUSSION

6.1. Summary Of Model

Layer (type)	Output Shape	Param #
conv2d_8 (Conv2D)	(None, 46, 46, 32)	896
max_pooling2d_8 (MaxPooling2D)	(None, 23, 23, 32)	0
dropout_12 (Dropout)	(None, 23, 23, 32)	0
conv2d_9 (Conv2D)	(None, 21, 21, 64)	18,496
max_pooling2d_9 (MaxPooling2D)	(None, 10, 10, 64)	0
dropout_13 (Dropout)	(None, 10, 10, 64)	0
conv2d_10 (Conv2D)	(None, 8, 8, 128)	73,856
max_pooling2d_10 (MaxPooling2D)	(None, 4, 4, 128)	0
dropout_14 (Dropout)	(None, 4, 4, 128)	0
conv2d_11 (Conv2D)	(None, 2, 2, 256)	295,168
max_pooling2d_11 (MaxPooling2D)	(None, 1, 1, 256)	0
dropout_15 (Dropout)	(None, 1, 1, 256)	0
flatten_2 (Flatten)	(None, 256)	0
dense_6 (Dense)	(None, 64)	16,448
dense_7 (Dense)	(None, 64)	4,160
dense_8 (Dense)	(None, 1)	65

Table 2 – Summary of the Model

The provided summary outlines the architecture of a Convolutional Neural Network (CNN) designed for image processing tasks, likely targeting binary classification. The model begins with an implicit input layer that presumably handles images of a certain dimension, inferred to be around (48, 48, 3) based on subsequent layer dimensions. The first layer is a convolutional layer (conv2d_8) with 32

filters, each of size 3x3, which produces an output feature map of size (46, 46, 32). This slight reduction in spatial dimensions is due to the convolution operation, which typically involves a stride or padding adjustment.

Following the initial convolution, a max-pooling layer (max_pooling2d_8) reduces the feature map size to (23, 23, 32), effectively halving the spatial dimensions while retaining the depth of 32. A dropout layer (dropout_12) is then applied, introducing regularization by randomly setting a fraction of the input units to zero during training to prevent overfitting.

The next convolutional layer (conv2d_9) increases the depth to 64 filters, outputting a feature map of (21, 21, 64). This is again followed by a max-pooling layer (max_pooling2d_9), reducing the size to (10, 10, 64), and another dropout layer (dropout_13) for regularization. The model continues to deepen with a third convolutional layer (conv2d_10), which outputs (8, 8, 128) feature maps, followed by max-pooling (max_pooling2d_10) down to (4, 4, 128), and dropout (dropout_14).

The final convolutional layer (conv2d_11) significantly increases the depth to 256 filters and reduces the spatial dimension to (2, 2, 256). This is followed by a max-pooling layer (max_pooling2d_11), which compresses the feature maps to (1, 1, 256), and a dropout layer (dropout_15). At this stage, the model has effectively transformed the input image into a compact and highly abstract representation.

The compact feature maps are then flattened into a 1-dimensional vector of 256 elements by the flatten layer (flatten_2). This vector is fed into a dense (fully connected) layer (dense_6) with 64 neurons, which learns to combine the extracted features in meaningful ways. Another dense layer (dense_7) with 64 neurons follows, continuing the feature combination and abstraction process. Finally, the model outputs through a dense layer (dense_8) with a single neuron, which is typically used for binary classification tasks by applying a sigmoid activation function to produce a probability score.

Overall, this CNN architecture is designed to progressively extract and refine features from input images through a series of convolutional, pooling, and dropout layers, before flattening the data and making a classification decision through fully connected layers. The use of dropout layers at various stages helps to mitigate overfitting, ensuring that the model generalizes well to new, unseen data

6.2. Evaluation Metrices.

	precision	recall	f1-score	support
0	0.91	0.94	0.93	2500
1	0.94	0.91	0.92	2500
accuracy			0.93	5000
macro avg	0.93	0.93	0.93	5000
weighted avg	0.93	0.93	0.93	5000

Fig 1 – Evaluation Metrices

The model's performance metrics, evaluated on a test set of 5000 samples, demonstrate robust classification capabilities in a binary classification context. The precision, recall, and F1-score for each class (0 and 1) are high, with class 0 achieving a precision of 0.91, recall of 0.94, and an F1-score of 0.93, while class 1 attains a precision of 0.94, recall of 0.91, and an F1-score of 0.92. These metrics indicate that the model is highly effective at distinguishing between the two classes, maintaining a good balance between precision (the ability to avoid false positives) and recall (the ability to identify all positive instances). The overall accuracy of the model is 93%, showing that 93% of the total predictions are correct. Both the macro average and weighted average of precision, recall, and F1-score are 0.93, reflecting consistent performance across classes and accounting for class distribution. This close alignment of precision and recall within each class suggests that the model does not disproportionately favor one class over the other, which is crucial in applications where both types of classification errors

carry significant consequences. In summary, the model demonstrates robust and balanced performance across key metrics, underscoring its suitability for binary classification tasks and its capability to generalize well on unseen data, making it a reliable tool for practical applications.

6.3. Loss Analysis

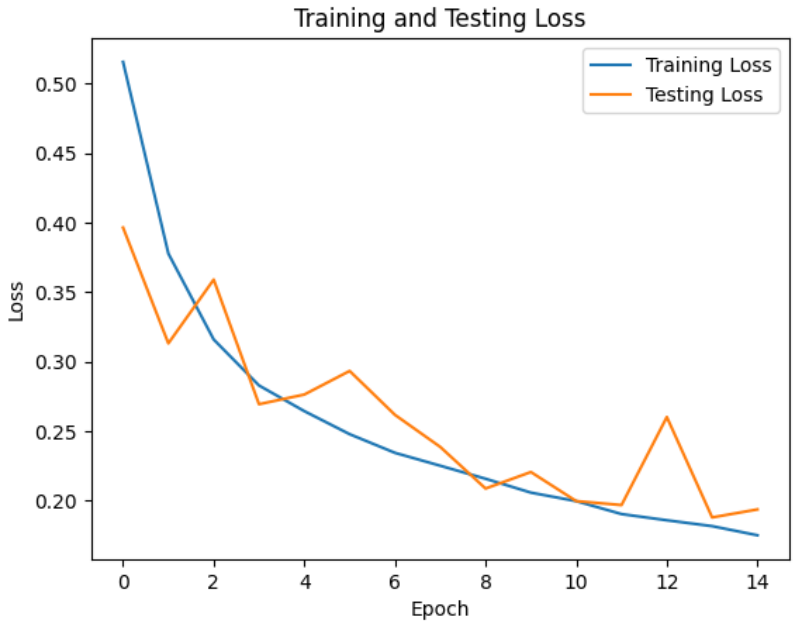

Fig 2 – Training and Testing Loss Graph

- **Epoch 1:** The initial training loss is 0.5852, and validation loss is 0.3963. The higher training loss compared to validation loss suggests that the model is initially overfitting less but is still adjusting its parameters.
- **Epoch 2:** Training loss drops significantly to 0.3979, with validation loss decreasing to 0.3131. This indicates effective learning and reduction in errors.
- **Epoch 3-4:** Training loss continues to decrease to 0.3219 and 0.2824, while validation loss fluctuates, momentarily increasing to 0.3589 before dropping to 0.2693. The initial increase in validation loss might suggest slight overfitting, which the model adjusts in subsequent epochs.
- **Epoch 5-6:** Training loss further reduces to 0.2725 and 0.2496, with validation loss slightly increasing to 0.2763 and 0.2932. The model might be overfitting slightly, but overall, the trend indicates improvement.

- **Epoch 7-8:** Training loss decreases to 0.2374 and 0.2249, with validation loss decreasing to 0.2617 and 0.2386. This indicates better learning and reduced overfitting.
- **Epoch 9-10:** Training loss continues to drop to 0.2197 and 0.2079, while validation loss decreases significantly to 0.2087 and slightly increases to 0.2206. The model shows effective learning and generalization.
- **Epoch 11-12:** Training loss reduces to 0.1982 and 0.1903, with validation loss decreasing to 0.1997 and 0.1968. These results demonstrate the model's continued improvement in minimizing errors.
- **Epoch 13-15:** Training loss reaches a low of 0.1777, and validation loss stabilizes around 0.1879 to 0.1937. The model maintains consistent performance with minimal overfitting.

6.4. Accuracy Analysis

Fig 3 – Training and Testing Accuracy Graph

- **Epoch 1:** The model starts with a training accuracy of 66.41% and a validation accuracy of 82.50%. This initial discrepancy indicates that the model begins by learning general patterns but may not yet be capturing all the intricacies of the validation data.
- **Epoch 2:** Significant improvement is observed, with training accuracy rising to 82.08% and validation accuracy to 86.70%. This suggests effective learning and a good adaptation to both training and validation data.
- **Epoch 3-4:** Training accuracy continues to improve, reaching 86.15% and 87.97%, respectively, while validation accuracy shows fluctuations, momentarily dropping to 84.74% before rising to 88.60%. This fluctuation might indicate the model's struggle to generalize some specific patterns in the validation data.
- **Epoch 5-6:** Training accuracy consistently increases to 88.44% and 89.58%. Validation accuracy stabilizes around 88.48% and 88.32%, with slight variations suggesting that the model is still fine-tuning its understanding of the data.
- **Epoch 7-8:** Training accuracy reaches 90.22% and 90.83%, while validation accuracy improves to 89.42% and 90.34%. The model shows consistent improvement and better generalization capabilities.
- **Epoch 9-10:** Training accuracy further improves to 91.07% and 91.74%, with validation accuracy rising to 92.06% and 91.20%. This steady increase demonstrates the model's enhanced learning efficiency and robustness.
- **Epoch 11-12:** Training accuracy reaches 91.86% and 92.17%, and validation accuracy climbs to 92.22% and 92.46%. These metrics suggest that the model is achieving higher generalization performance.
- **Epoch 13-15:** Training accuracy peaks at 93.01%, and validation accuracy stabilizes around 92.70% to 92.54%. The model demonstrates strong and consistent performance across both datasets.

Hence, The training and validation accuracy curves demonstrate a steady improvement,

with the model achieving over 93% training accuracy and around 92.70% validation accuracy by the end of the 15 epochs. The loss curves corroborate this trend, showing consistent reduction in both training and validation loss, indicative of effective learning and generalization. The occasional fluctuations in validation metrics highlight the model's fine-tuning process to adapt to new data patterns, eventually stabilizing to demonstrate robust performance. Overall, the results reflect the model's strong ability to learn and generalize from the training data, making it reliable for practice application.

6.5. Model Deployment Results:

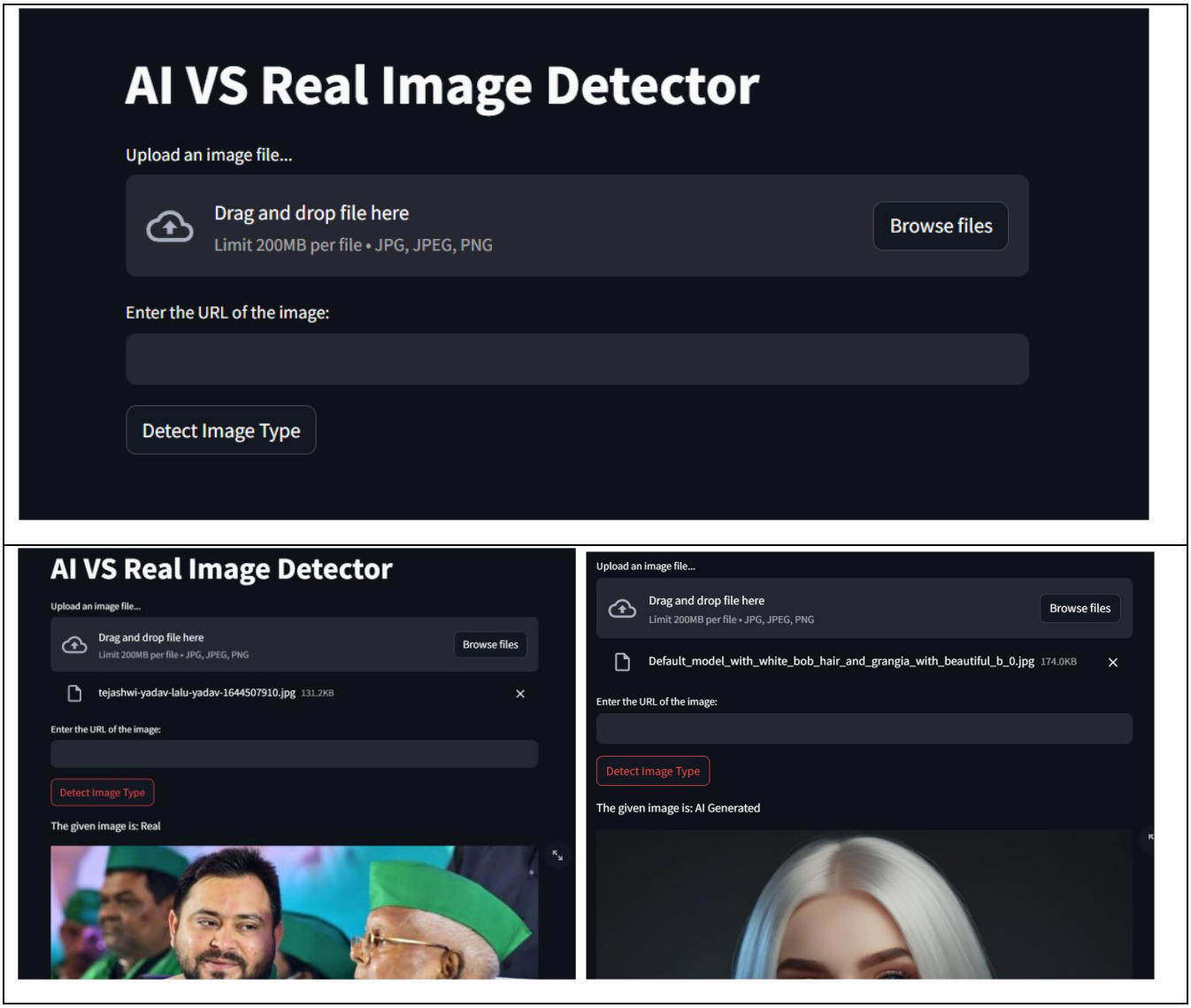

Fig 4 – Model Deployment Results

Below is a demonstration of "PixelTruth" in action, showcasing its user-friendly interface designed to authenticate visual content. Users have two options to analyze images: they can either upload a file directly from their device or enter a URL to assess an online image. This functionality is pivotal in today's digital landscape, where distinguishing between authentic and AI-generated images is increasingly challenging. "PixelTruth" utilizes a Convolutional Neural Network (CNN) model, trained rigorously to detect subtle differences between real and fabricated images. This model, now deployed, provides a seamless and reliable way for users to verify the authenticity of visual content, helping to combat misinformation and digital deception. By leveraging advanced machine learning algorithms, "PixelTruth" empowers individuals to make informed decisions about the media they encounter, promoting transparency, accountability, and ethical integrity in digital communication.

Chapter – 7 FUTURE OUTLOOK

As we look towards the future, the horizon for "PixelTruth" extends beyond its current capabilities, presenting a myriad of opportunities for expansion, enhancement, and innovation. With the rapid evolution of technology and the ever-changing landscape of digital media, our project is poised to adapt and thrive in the face of emerging challenges and opportunities.

One promising avenue for future development lies in broadening the scope of the model to encompass not only images but also other forms of media, such as videos, audio recordings, and textual content. By extending the model's capabilities to detect AI-generated content across diverse modalities, we can provide users with a comprehensive solution for identifying synthetic media, thereby promoting greater trust and reliability in digital content authentication efforts. This holistic approach would enable "PixelTruth" to serve as a versatile tool for combating digital deception across a wide range of media formats, enhancing its impact and relevance in an increasingly interconnected world.

Furthermore, advancements in machine learning and artificial intelligence present exciting opportunities for refining and optimizing the performance of the model. By leveraging state-of-the-art techniques such as transfer learning, ensemble methods, and adversarial training, we can enhance the model's accuracy, robustness, and efficiency, enabling it to adapt to new challenges and adversarial attacks with ease. Additionally, ongoing research in explainable AI and interpretability techniques can help elucidate the inner workings of the model, providing users with insights into how decisions are made and increasing transparency in the authentication process.

Collaboration and partnership are also key drivers of future growth and innovation for "PixelTruth."

By forging alliances with industry stakeholders, academic institutions, and civil society organizations, we can leverage collective expertise and resources to advance the state of the art in digital content authentication. Collaborative initiatives can facilitate data sharing, benchmarking, and knowledge exchange, accelerating progress towards our shared goal of fostering trust and authenticity in the digital ecosystem.

Chapter – 8 IMPLEMENTATION

8.1. Python Notebook Code

```
#Reducing the size of the datasets due to hardware and software resource restrictioins
#Training dataset - reduced by 58.3%
import os
import random
import shutil

aigenerated_folder = "C:\dataset\\ai-v-real-master\\ai-v-real-master\data\Processed Data_2\Training Data\AIGenerated"
real_folder = "C:\dataset\\ai-v-real-master\\ai-v-real-master\data\Processed Data_2\Training Data\Real"

# Function to get the number of images in a folder
def count_images(folder_path):
    return len([f for f in os.listdir(folder_path) if os.path.isfile(os.path.join(folder_path, f))])

# Function to delete % of the images in a folder
def reduce_folder_size(folder_path):
    num_images_before = count_images(folder_path)
    num_images_to_delete = int(num_images_before * 0.583)
    all_images = [f for f in os.listdir(folder_path) if os.path.isfile(os.path.join(folder_path, f))]
    images_to_delete = random.sample(all_images, num_images_to_delete)
    for image in images_to_delete:
        os.remove(os.path.join(folder_path, image))
    num_images_after = count_images(folder_path)
    print(f"Deleted {num_images_to_delete} images from {folder_path}")
    print(f"Number of images in {folder_path} before: {num_images_before}, after: {num_images_after}")

# Print the number of images in each folder before reducing the size
print(f"Number of images in {aigenerated_folder}: {count_images(aigenerated_folder)}")
print(f"Number of images in {real_folder}: {count_images(real_folder)}")

# Reduce the size of the AiGenerated and Real folders
reduce_folder_size(aigenerated_folder)
reduce_folder_size(real_folder)
……
```

```
#Testing dataset - reduced by 75%
import os
import random
import shutil

aigenerated_folder = "C:\dataset\\ai-v-real-master\\ai-v-real-master\data\Processed Data_2\Testing Data\AIGenerated"
real_folder = "C:\dataset\\ai-v-real-master\\ai-v-real-master\data\Processed Data_2\Testing Data\Real"

# Function to get the number of images in a folder
def count_images(folder_path):
    return len([f for f in os.listdir(folder_path) if os.path.isfile(os.path.join(folder_path, f))])

# Function to delete % of the images in a folder
def reduce_folder_size(folder_path):
    num_images_before = count_images(folder_path)
    num_images_to_delete = int(num_images_before * 0.75)
    all_images = [f for f in os.listdir(folder_path) if os.path.isfile(os.path.join(folder_path, f))]
    images_to_delete = random.sample(all_images, num_images_to_delete)
    for image in images_to_delete:
        os.remove(os.path.join(folder_path, image))
    num_images_after = count_images(folder_path)
    print(f"Deleted {num_images_to_delete} images from {folder_path}")
    print(f"Number of images in {folder_path} before: {num_images_before}, after: {num_images_after}")

# Print the number of images in each folder before reducing the size
print(f"Number of images in {aigenerated_folder}: {count_images(aigenerated_folder)}")
print(f"Number of images in {real_folder}: {count_images(real_folder)}")

# Reduce the size of the AiGenerated and Real folders
reduce_folder_size(aigenerated_folder)
reduce_folder_size(real_folder)

…….
data = "C:\dataset\\ai-v-real-master\\ai-v-real-master\data\Processed Data_2\Training Data"
categories = ['Real', 'AIGenerated']

…….

#Replacing the spaces in file names with _
import os
```

```
aigenerated_folder ="C:\dataset\\ai-v-real-master\\ai-v-real-master\data\Processed Data_2\Training Data\AIGenerated"
for filename in os.listdir(aigenerated_folder):
    if ' ' in filename:
        os.rename(os.path.join(aigenerated_folder, filename), os.path.join(aigenerated_folder, filename.replace(' ', '_')))

……
import os

real_folder = "C:\dataset\\ai-v-real-master\\ai-v-real-master\data\Processed Data_2\Training Data\Real"
for filename in os.listdir(real_folder):
    if ' ' in filename:
        os.rename(os.path.join(real_folder, filename), os.path.join(real_folder, filename.replace(' ', '_')))

…….
img_size = 48
training_data = []

……
pip install opencv-python-headless

…..
#Resizing the images and Normalizing pixel values
import os
import cv2 as cv
for category in categories:
    path = os.path.join(data, category)
    classes = categories.index(category)
    for img in os.listdir(path):
        img_path = os.path.join(path, img)
        img_array = cv.imread(img_path)
        if img_array is not None:
            img_resized = cv.resize(img_array, (img_size, img_size))
            img_resized = img_resized / 255.0  # Normalize pixel values to [0, 1]
            training_data.append([img_resized, classes])
        else:
            print(f"Failed to load image: \"{img_path}\"")

print(f"Total images processed: {len(training_data)}")
…….
len(training_data), training_data[0][0].shape
```

```
……
random.shuffle(training_data)
import numpy as np

……
X_train = []
y_train = []

for features, label in training_data:
    X_train.append(features)
    y_train.append(label)

X_train = np.array(X_train).reshape(-1, img_size, img_size, 3)
y_train = np.array(y_train)

……
X_train.shape
y_train.shape

……
X_train[0].shape

……
pickle_out = open("X_train.pickle", "wb")
pickle.dump(X_train, pickle_out, protocol=4)
pickle_out.close()

pickle_out = open("y_train.pickle", "wb")
pickle.dump(y_train, pickle_out, protocol=4)
pickle_out.close()
……
pickle_in = open("X_train.pickle", "rb")
X_train = pickle.load(pickle_in)

pickle_in = open("y_train.pickle", "rb")
y_train = pickle.load(pickle_in)

……
X_train.shape
y_train.shape

……
data = "C:\dataset\\ai-v-real-master\\ai-v-real-master\data\Processed Data_2\Testing Data"
categories = ['Real', 'AIGenerated']
```

```
img_size = 48
testing_data = []

i = 0
for category in categories:
    path = os.path.join(data,category)
    classes = categories.index(category)
    for img in os.listdir(path):
        i = i + 1
        img_array = cv.imread(os.path.join(path,img))
        new_array = cv.resize(img_array, (48,48))
        new_array = new_array/255
        testing_data.append([new_array, classes])

random.shuffle(testing_data)

X_test = []
y_test = []

for features, label in testing_data:
    X_test.append(features)
    y_test.append(label)

X_test = np.array(X_test).reshape(-1, img_size, img_size, 3)
y_test = np.array(y_test)

…….
X_test.shape
y_test.shape

…..
pip install --upgrade tensorflow
…….
import tensorflow as tf
from tensorflow import keras
from tensorflow.keras import layers, models, Sequential

…..
model = keras.Sequential([
    keras.layers.Conv2D(32, (3, 3), activation='relu', input_shape=(48, 48, 3)),
    keras.layers.MaxPooling2D((2, 2)),
    keras.layers.Dropout(0.2),
```

```
    keras.layers.Conv2D(64, (3, 3), activation='relu'),
    keras.layers.MaxPooling2D((2, 2)),
    keras.layers.Dropout(0.2),

    keras.layers.Conv2D(128, (3, 3), activation='relu'),
    keras.layers.MaxPooling2D((2, 2)),
    keras.layers.Dropout(0.2),

    keras.layers.Conv2D(256, (3, 3), activation='relu'),
    keras.layers.MaxPooling2D((2, 2)),
    keras.layers.Dropout(0.2),

    keras.layers.Flatten(),
    keras.layers.Dense(128, activation='relu'),
    keras.layers.Dropout(0.5),  # Dropout added
    keras.layers.Dense(64, activation='relu'),
    keras.layers.Dropout(0.5),  # Dropout added
    keras.layers.Dense(1, activation='sigmoid')
])

…..
model.compile(
    optimizer='adam',
    loss='binary_crossentropy',
    metrics=['accuracy']
)
…..
# #Setting the parameters for the model
# model = keras.Sequential([
#     keras.layers.Conv2D(32,(3,3), activation='relu', input_shape = (48,48,3)),
#     keras.layers.MaxPool2D((2,2)),
#     keras.layers.Dropout(0.2),

#     keras.layers.Conv2D(64,(3,3), activation='relu'),
#     keras.layers.MaxPool2D((2,2)),
#     keras.layers.Dropout(0.2),

#     keras.layers.Conv2D(128,(3,3), activation='relu'),
#     keras.layers.MaxPool2D((2,2)),
#     keras.layers.Dropout(0.2),

#     keras.layers.Conv2D(256,(3,3), activation='relu'),
#     keras.layers.MaxPool2D((2,2)),
#     keras.layers.Dropout(0.2),
```

```
#     keras.layers.Flatten(),
#     keras.layers.Dense(64, activation='relu'),
#     keras.layers.Dense(64, activation='relu'),
#     keras.layers.Dense(1, activation='sigmoid')
# ])

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

……
model.summary()
…
#Training the model
history = model.fit(X_train, y_train, epochs=15, validation_data=(X_test, y_test))

….
#Saving the model
model.save("AIGeneratedModel2.h5")

…..
#Loading the saved model
model_new = keras.models.load_model("AIGeneratedModel2.h5")

….
model_new.evaluate(X_test, y_test)
y_pred = model_new.predict(X_test)
y_pred.shape

……
y_predicted = []

for arr in y_pred:
    if arr[0] <= 0.5:
        y_predicted.append(0)
    else:
        y_predicted.append(1)

y_predicted = np.array(y_predicted)

….
```

```
y_predicted.shape

….
from gettext import install

pip install scikit-learn

….
from sklearn.metrics import classification_report

print(classification_report(y_test, y_predicted))

…..
!pip install matplotlib
import matplotlib.pyplot as plt # type: ignore

# Plot the training and testing loss
plt.plot(history.history['loss'], label='Training Loss')
plt.plot(history.history['val_loss'], label='Testing Loss')
plt.xlabel('Epoch')
plt.ylabel('Loss')
plt.title('Training and Testing Loss')
plt.legend()
plt.show()

# Plot the training and testing accuracy
plt.plot(history.history['accuracy'], label='Training Accuracy')
plt.plot(history.history['val_accuracy'], label='Testing Accuracy')
plt.xlabel('Epoch')
plt.ylabel('Accuracy')
plt.title('Training and Testing Accuracy')
plt.legend()
plt.show()

……
```

8.2. Model Deployment Code

```
def find_out(path_img):
    img_arr = cv.imread(path_img)
    plt.imshow(img_arr)
    new_arr = cv.resize(img_arr, (48,48))
    new_arr = new_arr/255
    test = []
```

```
    test.append(new_arr)
    test = np.array(test).reshape(-1, img_size, img_size, 3)
    y = model_new.predict(test)
    if y[0] <= 0.5:
        print("The given image is Real.")
    else:
        print("The given image is AI Generated.")

…..
import cv2 as cv
path_img = "C:\Downloadss\content\dataset_train\Real\cityscapes_39.jpeg"
find_out(path_img)

….
path_img = "F:\Downloads\check2.jpg"
find_out(path_img)


CODE USE FOR DEVELPOMENT
import streamlit as st
import requests
from tensorflow.keras.models import load_model
import numpy as np
import cv2 as cv
from PIL import Image
from io import BytesIO
import os

# Load the trained model
model = load_model("AIGeneratedModelUpdated.h5")

st.set_page_config(
    page_title="AI VS Real Image Detector",
    page_icon="🖼",
)

st.title("AI VS Real Image Detector")

# Option to upload an image file
uploaded_file = st.file_uploader("Upload an image file...", type=["jpg", "jpeg", "png"])
# Option to input an image URL
image_url = st.text_input("Enter the URL of the image:", "")
```

```
def detect_image_type(image):
    try:
        img_arr = np.array(image)
        # Convert the image to RGB format if it's not already in RGB
        if len(img_arr.shape) == 2:
            img_arr = cv.cvtColor(img_arr, cv.COLOR_GRAY2RGB)
        elif len(img_arr.shape) == 3 and img_arr.shape[2] == 4:
            img_arr = cv.cvtColor(img_arr, cv.COLOR_RGBA2RGB)
        elif len(img_arr.shape) != 3 or img_arr.shape[2] != 3:
            raise ValueError("Unsupported image format")

        new_arr = cv.resize(img_arr, (48, 48)) / 255.0
        test = np.array([new_arr])
        result = model.predict(test)
        return "AI Generated" if result[0][0] > 0.5 else "Real"
    except Exception as e:
        st.error(f"Error detecting image type: {e}")
        return None

def fetch_image_from_url(url):
    try:
        response = requests.get(url)
        if response.status_code == 200:
            # Check if the content type is an image
            content_type = response.headers['content-type']
            if 'image' in content_type:
                return Image.open(BytesIO(response.content))
            else:
                st.error(f"Invalid content type: {content_type}. Please provide a valid image URL.")
                return None
        else:
            st.error(f"Failed to fetch image from URL. Status code: {response.status_code}")
            return None
    except Exception as e:
        st.error(f"Error fetching image from URL: {e}")
        return None

def is_image_url(url):
    _, ext = os.path.splitext(url)
    return ext.lower() in ['.jpg', '.jpeg', '.png', '.gif', '.bmp']

if st.button("Detect Image Type"):
    if uploaded_file is not None:
        try:
```

```
            image = Image.open(uploaded_file)
            image_type = detect_image_type(image)
            st.write(f"The given image is: {image_type}")
            st.image(image, caption='Uploaded Image', use_column_width=True)
        except Exception as e:
            st.error(f"Error processing uploaded image: {e}")
    elif image_url:
        if is_image_url(image_url):
            image = fetch_image_from_url(image_url)
            if image is not None:
                try:
                    image_type = detect_image_type(image)
                    st.write(f"The given image is: {image_type}")
                    st.image(image, caption='Uploaded Image', use_column_width=True)
                except Exception as e:
                    st.error(f"Error detecting image type: {e}")
        else:
            st.error("Invalid image URL. Please provide a valid URL pointing directly to an image file.")
    else:
        st.warning("Please upload an image file or provide the URL of the image.")
```

REFERENCES

1. Flask Documentation. (n.d.). Retrieved from https://flask.palletsprojects.com/
2. Python Software Foundation. (n.d.). pickle — Python object serialization. Retrieved from https://docs.python.org/3/library/pickle.html
3. NumPy. (n.d.). Retrieved from https://numpy.org/
4. OpenCV. (n.d.). Retrieved from https://opencv.org/
5. TensorFlow. (n.d.). Retrieved from https://www.tensorflow.org/
6. Keras. (n.d.). Retrieved from https://keras.io/
7. Streamlit. (n.d.). Retrieved from https://streamlit.io/
8. "Quantifying the Performance Gap between Real and Ai-Generated ... - SSRN." 06 Oct. 2023, https://papers.ssrn.com/sol3/papers.cfm?abstract_id=4594547.
9. "GenImage: A Million-Scale Benchmark for Detecting AI-Generated Image." 14 Jun. 2023, https://arxiv.org/abs/2306.08571.
10. "Harnessing Machine Learning for Discerning AI-Generated Synthetic Images." 14 Jan. 2024, https://arxiv.org/abs/2401.07358.
11. "CIFAKE: Image Classification and Explainable Identification of AI ...." 24 Mar. 2023, https://arxiv.org/abs/2303.14126.