2022MT93056Report_thesis.docx

by 2022mt93056-shubham Pal Singh .

Submission date: 28-Apr-2024 10:17AM (UTC+0530)
Submission ID: 2363501025
File name: 2022MT93056Report_thesis.docx (383.58K)
Word count: 7365
Character count: 48060

IMAGE FOGERY DETECTION USING DEEP LEARNING

BITS ZG628T: Dissertation

by

(SHUBHAM PAL SINGH)
(2022MT93056)

Dissertation work carried out at

Submitted in partial fulfilment of **(ME,SOFTWARE ENGINEERING)** degree programme

Under the Supervision of

(Dr. Y V K RAVI KUMAR)

(BITS PILANI, HYDERABAD CAMPUS)

BIRLA INSTITUTE OF TECHNOLOGY & SCIENCE

PILANI (RAJASTHAN)

(April 2024)

ACKNOWLEDGEMENT

I extend my heartfelt appreciation to

` for entrusting me with the invaluable opportunity to spearhead the establishment of the Automation Research and Innovation Department. Their unwavering support and belief in my capabilities have been instrumental in fueling my passion for research and innovation.

Throughout the course of my thesis, I found profound inspiration in the utilization of various Python libraries, including subprocess for seamless execution, OpenCV for image processing tasks, PIL Image for image manipulation, and the powerful machine learning frameworks TensorFlow, PyTorch, scikit-learn, and Keras. These tools have not only facilitated the realization of my research goals but have also deepened my understanding of advanced concepts in automation and artificial intelligence.

I am deeply grateful to Dr. Y V K Ravi Kumar for his continuous guidance and mentorship throughout the machine learning project. His expertise, encouragement, and insightful feedback have been invaluable assets, propelling me forward and nurturing my growth as a researcher.

Additionally, I extend my sincere appreciation to my evaluator, Asish Bera, for his diligent review and invaluable suggestions during both the abstract and midsem report stages. His expertise and constructive criticism have significantly enriched the quality of my work, guiding me towards greater clarity and precision in my research endeavors.

SHUBHAM PAL SINGH
Signature of the Student

Name:SHUBHAM PAL SINGH

Date:19-03-2024

Place:NOIDA

Dr.Y V K RAVI KUMAR
Signature of the Supervisor

Name: Dr. Y V K RAVI KUMAR

Date:19-03-2024

Place:HYDERABAD

ABSTRACT

Deepfake technology has become increasingly sophisticated, posing significant challenges to the authenticity of digital content. In response, this research project proposes a deep learning-based approach for detecting deepfake images. The study focuses on employing Convolutional Neural Networks (CNNs), a very good deep learning models, to differentiate between genuine and manipulated images.

The research begins by collecting a dataset comprising genuine and deepfake images, crucial for training and evaluating the proposed deep learning model. The dataset consists of images sourced from various sources, ensuring diversity in content and quality.

Next, a custom CNN architecture is developed and trained using the collected dataset. The CNN architecture includes multiple convolutional and pooling layers to automatically learn discriminative features from image data. Additionally, dropout layers are incorporated to enhance model generalization and mitigate overfitting issues.

To facilitate effective model training, extensive data preprocessing is performed using the ImageDataGenerator module, ensuring data normalization and augmentation. The dataset is split into training and validation subsets, enabling rigorous evaluation of the model's performance.

The trained deep learning model is evaluated on both the training and validation datasets to assess its accuracy and generalization capabilities. Furthermore, comprehensive experiments are conducted to analyze the model's robustness against various types of deepfake manipulation techniques.

The results obtained from the experiments demonstrate the efficacy of the proposed deep learning approach in accurately detecting deepfake images. The model achieves high accuracy and demonstrates resilience against adversarial attacks, highlighting its potential for real-world applications in combatting the spread of misinformation and protecting digital integrity.

In conclusion, this research project contributes to the advancement of deepfake detection techniques through the utilization of convolutional neural networks. By leveraging deep learning capabilities, the proposed approach offers a promising solution for addressing the growing challenges posed by deepfake technology in today's digital landscape.

SHUBHAM PAL SINGH

Signature of the Student

Name:SHUBHAM PAL SINGH

Date:19-03-2024

Place:NOIDA

Dr.Y V K RAVI KUMAR

Signature of the Supervisor

Name: Dr. Y V K RAVI KUMAR

Date:19-03-2024

Place:HYDERABAD

Table of content:

1.1 Data Preprocessing

Data preprocessing is a critical phase in machine learning, essential for transforming raw data into a format suitable for analysis and model training. This phase encompasses various tasks aimed at cleaning, transforming, and organizing data to enhance its manageability and relevance for learning algorithms. Here, we outline common techniques and tasks involved in data preprocessing for machine learning:

1. **Data Cleaning**:
    - **Handling Missing Values**: Techniques like imputation, deletion, or interpolation address missing data points.
    - **Removing Duplicate Records**: Identification and elimination of duplicate entries within the dataset.
    - **Outlier Detection and Treatment**: Identifying and managing outliers to prevent skewing of analysis or model performance.
2. **Data Transformation**:
    - **Feature Scaling**: Normalizing or standardizing numerical features to ensure uniform scale.
    - **Encoding Categorical Variables**: Conversion of categorical variables into numerical representations suitable for algorithms, using techniques such as one-hot encoding or label encoding.
    - **Feature Engineering**: Creation of new features or transformation of existing ones to capture pertinent information for the learning task, which may include feature extraction, transformation, or aggregation.
3. **Data Reduction**:
    - **Dimensionality Reduction**: Techniques like Principal Component Analysis (PCA) or feature selection methods reduce feature count while retaining important information and lowering computational complexity.
    - **Sampling Techniques**: Balancing imbalanced datasets through methods such as undersampling, oversampling, or generating synthetic samples.
4. **Data Integration and Formatting**:
    - **Merging Datasets**: Combination of multiple datasets into a unified dataset for analysis or model training.
    - **Formatting Data**: Ensuring data is in the correct format for input into machine learning algorithms, including reshaping arrays or converting data types.
5. **Data Splitting**:
    - **Splitting into Training, Validation, and Test Sets**: Partitioning the dataset into subsets for model training, validation, and evaluation to assess model performance on unseen data.

1.1.1 Numpy

NumPy (Numerical Python) stands as a foundational library within Python, essential for scientific computing, particularly in the realm of machine learning. Its capabilities extend to supporting large, multi-dimensional arrays and matrices, along with an extensive collection of mathematical functions, optimized for efficient operations on these arrays. Below are the key ways in which NumPy contributes to machine learning tasks:

1. **Array Operations**:
   - NumPy's array object (numpy.ndarray) is instrumental for manipulating vast datasets efficiently. This encompasses essential operations like element-wise arithmetic, slicing, indexing, reshaping, and broadcasting, fundamental for data manipulation and preprocessing in machine learning workflows.
2. **Mathematical Functions**:
   - NumPy offers a rich set of mathematical functions, tailor-made for array operations. These include arithmetic (e.g., np.add(), np.subtract()), trigonometric (e.g., np.sin(), np.cos()), exponential, logarithmic, statistical (e.g., np.mean(), np.std()), and linear algebra operations, integral for many machine learning algorithms.
3. **Random Number Generation**:
   - The random module within NumPy facilitates the generation of random numbers and samples from various probability distributions. This capability is pivotal for tasks such as generating synthetic datasets, initializing model parameters randomly, and conducting simulations.
4. **Array Broadcasting**:
   - NumPy's broadcasting mechanism enables arrays with different shapes to be operated on together seamlessly. This obviates the necessity for explicit looping, facilitating concise and efficient element-wise operations between arrays of disparate shapes.
5. **Integration with Other Libraries**:
   - NumPy serves as the bedrock for numerous Python libraries ubiquitous in machine learning, including SciPy, pandas, scikit-learn, and TensorFlow. These libraries frequently leverage NumPy arrays as their primary data structures, fostering seamless integration and interoperability across diverse components of the machine learning pipeline.

In essence, NumPy plays a pivotal role in machine learning by furnishing efficient data structures and mathematical functions for array manipulation and numerical computation. Its straightforwardness, performance, and integration capabilities render it indispensable within the Python ecosystem for scientific computing and machine learning endeavors.

1.1.2Pandas

Pandas is a widely-used Python library within the realms of machine learning and data science, revered for its prowess in data manipulation and analysis. It furnishes high-performance, user-friendly data structures and tools tailored for handling structured data encompassing tabular data, time series, and heterogeneous data formats. Below are the common ways in which Pandas is harnessed in the domain of machine learning:

1. **Data Structures**:
    - Pandas presents two core data structures: Series and DataFrame.
        - *Series*: A one-dimensional array-like entity capable of accommodating diverse data types, including integers, floats, strings, or Python objects.
        - *DataFrame*: A two-dimensional labeled data structure featuring columns of potentially distinct data types. It resembles a spreadsheet or SQL table and stands as the principal structure employed for data manipulation within Pandas.
2. **Data Loading and Manipulation**:
    - Pandas boasts comprehensive functionality for loading data from an array of file formats like CSV, Excel, SQL databases, JSON, and more. It facilitates seamless reading, writing, and manipulation of data.
    - Data manipulation operations encompass selection, filtering, sorting, merging, joining, grouping, aggregating, and pivoting, pivotal for data preparation preceding analysis and modeling.
3. **Data Cleaning and Preprocessing**:
    - Pandas arms users with methods for handling missing data, including detection (isnull()), removal or filling (dropna(), fillna()), and interpolation (interpolate()).
    - It supports data transformation endeavors such as reshaping, data type conversion, categorical variable encoding, and feature extraction.
4. **Exploratory Data Analysis (EDA)**:
    - Pandas facilitates exploratory data analysis through provision of descriptive statistics (describe()), comprehensive data summaries (info()), and visualization tools (built-in or via integration with libraries like Matplotlib and Seaborn).
5. **Integration with Machine Learning Libraries**:
    - Pandas seamlessly integrates with other Python libraries prevalent in machine learning, such as NumPy, scikit-learn, TensorFlow, and PyTorch. It facilitates effortless conversion between Pandas data structures and arrays employed by these libraries.
6. **Time Series Analysis**:
    - Pandas harbors potent tools for navigating time series data, encompassing date/time indexing, resampling, rolling windows, and time zone management. These capabilities prove invaluable in analyzing and modeling time-dependent data.

In essence, Pandas emerges as a versatile and potent library for data manipulation and analysis in Python, constituting an indispensable tool within the machine learning workflow. Its utility spans across tasks ranging from data cleansing and preprocessing to exploratory analysis and model preparation.

.

1.1.3 OS

The os module in Python serves as a gateway to accessing operating system-dependent functionalities, offering a platform-independent means for Python programs to interact with the underlying operating system. Although not inherently tied to machine learning algorithms or data manipulation, the os module finds utility across various facets of machine learning projects. Here's how the os module can be harnessed in the realm of machine learning:

1. **File System Operations**:
   - The os module furnishes functions for file and directory manipulation, enabling tasks like creation, deletion, renaming, or relocation of files and directories. This capability proves instrumental in managing datasets, preserving model checkpoints, and organizing project files effectively.
2. **Environment Variables**:
   - Through the os.environ dictionary, access to the operating system's environment variables is facilitated. This functionality is pivotal for retrieving configuration settings or sensitive information stored as environment variables, such as API keys or database credentials.
3. **Path Manipulation**:
   - The os.path submodule provides a suite of functions for manipulating file paths in a manner agnostic to the underlying platform. This encompasses operations like path joining and splitting, path existence checks, retrieval of absolute paths, among others. Robust path manipulation is indispensable for ensuring portability and compatibility across diverse operating systems.
4. **Process Management**:
   - Basic process management tasks, including spawning new processes (via os.spawn*() functions), obtaining information about the current process (os.getpid()), and interfacing with the process environment (os.putenv(), os.getenv()), are facilitated by the os module.
5. **Permissions and Security**:
   - The os module equips users with functions for configuring file permissions, inspecting file ownership, and executing other security-oriented operations. This functionality is pertinent for administering access control to sensitive data or files within a machine learning project.
6. **System Information**:
   - Access to crucial system information, such as the current working directory (os.getcwd()), the user's home directory (os.path.expanduser('~')), and the system's hostname (os.uname() on Unix-based systems), is facilitated by the os module.

While the os module may not directly engage with the core machinery of machine learning algorithms, its role is pivotal in facilitating numerous aspects of machine learning projects. These include data management, file operations, environment configuration, and interaction with the system, collectively contributing to the seamless execution of machine learning workflows.

1.2 Model Development

Model development in machine learning encompasses a systematic process of crafting and refining predictive models through the integration of algorithms and data. This iterative journey traverses various stages, commencing from data preparation and culminating in model deployment. Here's a comprehensive overview of the model development process in machine learning:

1. **Problem Definition and Data Collection**:
   - Define the problem statement and gather pertinent data from diverse sources such as databases, APIs, files, or external datasets.
2. **Data Preprocessing**:
   - Cleanse and preprocess the data to render it amenable for modeling. This includes addressing missing values, eliminating duplicates, encoding categorical variables, and scaling numerical features, among other transformations.
3. **Feature Engineering**:
   - Engineer features by crafting new ones or transforming existing ones to augment the model's predictive prowess. This step involves selecting, extracting, and encoding features that encapsulate pertinent information for the learning task.
4. **Model Selection**:
   - Choose suitable machine learning algorithms or models predicated on factors like problem type, data nature, and performance requisites. Considerations encompass model complexity, interpretability, scalability, and computational efficiency.
5. **Model Training**:
   - Segregate the data into training and validation sets. Train the chosen models on the training data employing pertinent training algorithms and optimization techniques, tweaking model parameters to minimize a loss function and bolster predictive performance.
6. **Model Evaluation**:
   - Assess the trained models on the validation set leveraging appropriate evaluation metrics such as accuracy, precision, recall, F1-score, mean squared error, or area under the ROC curve. Compare model performances and select the optimal one for further scrutiny.
7. **Hyperparameter Tuning**:
   - Refine the hyperparameters of the chosen model to optimize its performance. This may entail techniques like grid search, random search, or Bayesian optimization to explore optimal hyperparameter configurations.
8. **Model Validation**:
   - Validate the final model on an independent test set to gauge its generalization capability and ascertain its efficacy in making accurate predictions on unseen data. This step is instrumental in estimating real-world performance and detecting overfitting or underfitting issues.
9. **Model Deployment**:
   - Roll out the trained model into production or operational environments to facilitate predictions on new data. This could involve integration into existing software systems, establishment of APIs for real-time inference, or deployment on cloud platforms or edge devices.
10. **Monitoring and Maintenance**:
    - Continuously monitor the deployed model's performance and update or retrain it as necessitated by shifts in data or environment. This encompasses vigilance over model drift, data quality discrepancies, and variations in input distributions to ensure sustained accuracy and reliability.

By adhering to this structured approach, data scientists and machine learning practitioners can fashion robust and efficacious predictive models adept at unraveling real-world complexities and furnishing valuable insights from data.

1.2.3 Tensorflow

TensorFlow, an open-source machine learning framework developed by Google, serves as a robust platform for creating, training, and deploying machine learning models. Its extensive array of tools, libraries, and resources caters to a diverse range of machine learning tasks, spanning from traditional algorithms to deep learning architectures. Below is an overview of TensorFlow's salient features and functionalities:

1. **Flexibility and Scalability**:
   - TensorFlow offers remarkable flexibility and scalability, enabling developers to construct and train models across various hardware platforms like CPUs, GPUs, TPUs (Tensor Processing Units), and distributed computing setups. This ensures efficient utilization of resources, especially beneficial for handling large datasets.
2. **High-level APIs**:
   - The framework provides high-level APIs, such as Keras, tf.keras, and Estimator, streamlining the model development process. These APIs offer intuitive interfaces for defining model architectures, configuring training parameters, and evaluating model performance, fostering ease of use and rapid prototyping.
3. **Deep Learning Capabilities**:
   - TensorFlow shines in the realm of deep learning, empowering tasks like image recognition, natural language processing (NLP), speech recognition, and reinforcement learning. It boasts a rich repertoire of pre-built neural network components like layers, activation functions, optimizers, and loss functions, facilitating the creation of intricate neural architectures.
4. **TensorFlow Hub**:
   - TensorFlow Hub acts as a treasure trove of reusable machine learning modules and pre-trained models. This repository offers a plethora of pre-trained models, embeddings, and modules, facilitating transfer learning and expediting model development by leveraging existing knowledge.
5. **TensorBoard Visualization**:
   - With TensorBoard, TensorFlow equips developers with a suite of visualization tools for model monitoring and debugging. TensorBoard empowers users to visualize model structures, track training progress, scrutinize performance metrics, and delve into model behavior through interactive visualizations.
6. **Production Deployment**:
   - TensorFlow excels in seamlessly deploying trained models into production environments, enabling real-time inference and integration with operational systems. It provides versatile deployment options like TensorFlow Serving, TensorFlow Lite, TensorFlow.js, and TensorFlow Extended (TFX), catering to diverse deployment needs across platforms.
7. **Community and Ecosystem**:
   - Supported by a vibrant community of developers, researchers, and practitioners, TensorFlow boasts a thriving ecosystem brimming with resources, insights, and support. This ecosystem encompasses libraries, tools, and frameworks for tasks ranging from data preprocessing to model optimization, enriching the development experience.

In essence, TensorFlow emerges as a potent and adaptable framework for crafting machine learning models, distinguished by its versatility, scalability, and integration capabilities. Widely embraced by both academia and industry, it stands as a cornerstone in the machine learning landscape, driving innovation and empowering impactful solutions.

1.2.2 Keras

Keras stands out as a high-level neural networks API designed to offer simplicity, modularity, and extensibility, all wrapped in a user-friendly Python interface. It serves as a versatile platform for constructing and training various deep learning models, encompassing convolutional neural networks (CNNs), recurrent neural networks (RNNs), and more, with backend support from TensorFlow, Theano, or Microsoft Cognitive Toolkit (CNTK). Below, we delve into its notable features and advantages:

1. **Simplicity and User-Friendliness**:
   - Keras boasts a straightforward API, enabling users to define and train neural networks with minimal code. Its simplicity makes it accessible to both novices and seasoned deep learning practitioners, fostering rapid prototyping and experimentation.
2. **Modularity and Flexibility**:
   - Following a modular design approach, Keras empowers users to craft intricate neural network architectures effortlessly. Whether stacking layers sequentially or leveraging the functional API for more complex models, Keras offers a diverse array of layer types, activation functions, optimizers, and loss functions, promoting flexibility and creativity.
3. **Compatibility and Integration**:
   - Keras seamlessly integrates with multiple backend engines like TensorFlow, Theano, and CNTK. This interoperability allows users to effortlessly switch between different backends, ensuring flexibility and compatibility with diverse computing environments.
4. **Extensibility and Customization**:
   - Keras fosters extensibility by enabling users to customize and extend the library's functionality. With provisions for defining custom layers, loss functions, metrics, and callbacks, users can tailor Keras to suit their specific requirements, experiment with novel architectures, and explore cutting-edge techniques.
5. **Built-in Support for Common Architectures**:
   - Keras comes equipped with built-in support for prevalent neural network architectures such as CNNs and RNNs. It offers specialized layers and utilities tailored for tasks like image classification, object detection, and natural language processing, simplifying model development across diverse domains.
6. **Integration with TensorFlow**:
   - As part of TensorFlow 2.0, Keras has been seamlessly integrated into TensorFlow, becoming the platform's official high-level API for deep learning. This integration combines Keras' ease of use with TensorFlow's performance and scalability, offering users the best of both worlds.
7. **Vibrant Community and Ecosystem**:
   - Backed by a vibrant community of developers, researchers, and contributors, Keras benefits from extensive support and a rich ecosystem. Users can access a wealth of tutorials, documentation, pre-trained models, and third-party extensions, enhancing the usability and functionality of the library.

In summary, Keras emerges as a potent and user-friendly tool for constructing and training deep learning models, characterized by its simplicity, flexibility, and compatibility with leading backend engines like TensorFlow. Its intuitive interface and thriving ecosystem make it a preferred choice for both academic research and industrial applications in the realm of deep learning.

1.2.3 Torch

Torch stands out as a robust machine learning library and scientific computing framework tailored for Python, boasting robust GPU acceleration capabilities. Its feature-rich toolkit facilitates the creation and training of diverse deep learning models, spanning neural networks, convolutional neural networks (CNNs), recurrent neural networks (RNNs), and beyond. Torch is revered for its adaptability, efficiency, and user-friendly nature, earning favor among both researchers and practitioners in the deep learning domain. Here's a closer look at some key components and features of Torch:

1. **Tensor Library**:
   - Torch revolves around its potent tensor library, offering extensive support for multi-dimensional arrays and associated mathematical operations. These tensors, akin to NumPy arrays, boast additional functionalities for seamless GPU acceleration and automatic differentiation.
2. **Dynamic Computational Graphs**:
   - One of Torch's defining features is its utilization of dynamic computational graphs. This dynamic approach allows for on-the-fly alterations to the model's graph structure during runtime, facilitating dynamic control flow and simplifying the implementation of intricate models.
3. **Automatic Differentiation**:
   - Torch streamlines the process of gradient computation through its autograd module, enabling automatic differentiation of tensors with respect to a specified loss function. This automates the gradient computation required for efficient backpropagation during model training.
4. **Neural Network Modules**:
   - The nn module in Torch offers a comprehensive suite of neural network modules and layers, encompassing linear layers, convolutional layers, recurrent layers, activation functions, loss functions, and optimization algorithms. These modular components can be seamlessly combined to construct sophisticated neural architectures.
5. **GPU Acceleration**:
   - With native support for GPU acceleration via CUDA, Torch enables offloading computations to GPUs for expedited training and inference. This GPU acceleration capability makes it well-suited for training large-scale deep learning models on GPU-equipped hardware.
6. **High-Level APIs**:
   - Torch provides high-level APIs akin to those found in frameworks like Keras, simplifying the process of model construction and training. These intuitive interfaces empower users to swiftly define and train models, expediting the deep learning workflow.
7. **Integration with Python**:
   - Torch seamlessly integrates with Python, harnessing the extensive ecosystem of Python libraries and tools for data manipulation, visualization, and deployment. This seamless integration facilitates the incorporation of Torch into existing Python-centric workflows and projects.
8. **Community and Ecosystem**:
   - Backed by a vibrant community of developers, researchers, and contributors, Torch boasts an active ecosystem replete with pre-trained models, tutorials, documentation, and third-party extensions. These resources enrich the usability and functionality of the framework, fostering collaborative innovation.

In essence, Torch emerges as a versatile and potent framework for developing and training deep learning models, distinguished by its flexibility, efficiency, and user-centric design. Its dynamic computational graph paradigm and automatic differentiation capabilities position it as an ideal choice for research, experimentation, and prototyping endeavors within the realm of deep learning.

1.3 Visualisation

Visualization is a pivotal component of the machine learning pipeline, empowering practitioners to glean insights from data, comprehend model behavior, and convey findings convincingly. Here are several crucial facets of visualization in the realm of machine learning:

1. **Data Exploration and Analysis**:
   - Techniques like histograms, scatter plots, box plots, and heatmaps are harnessed to delve into and dissect datasets. By visualizing data distributions, correlations, and patterns, practitioners can unravel the underlying data structure and discern potential relationships between variables.
2. **Feature Engineering**:
   - Visualization aids in feature engineering by elucidating relationships between features and target variables. Through tools such as pair plots, joint plots, and correlation matrices, practitioners can pinpoint pertinent features, identify collinearity, and inform feature selection or generation.
3. **Model Evaluation**:
   - Visualizing model performance metrics—such as accuracy, loss, precision, recall, and ROC curves—facilitates the evaluation of machine learning models. Graphical representations offer intuitive insights into model behavior, trade-offs, and areas ripe for enhancement.
4. **Model Interpretation**:
   - Techniques such as partial dependence plots, feature importance plots, and SHAP (SHapley Additive exPlanations) values aid in interpreting machine learning models and comprehending the contribution of individual features to model predictions. Visualization serves to elucidate model decisions and pinpoint influential factors.
5. **Model Training and Tuning**:
   - Visualizing model training dynamics—including learning curves, validation curves, and convergence plots—enables practitioners to monitor model performance throughout training. Visual feedback facilitates the diagnosis of issues like overfitting, underfitting, or convergence challenges, enabling adjustments to model parameters as needed.
6. **Interactive Dashboards and Tools**:
   - Interactive visualization tools and dashboards—such as Plotly, Bokeh, and Streamlit—foster the creation of dynamic and interactive visualizations for data exploration, model comparison, and results presentation. These tools bolster user engagement and streamline collaborative analysis.
7. **Model Deployment and Monitoring**:
   - Visualization assumes a crucial role in monitoring model performance in production settings, where dashboards and visualizations furnish real-time insights into model predictions, anomalies, and data drift. Visualization aids in issue detection, error debugging, and sustained model quality maintenance.

In summary, visualization serves as a potent instrument in the machine learning journey, facilitating data exploration, model development, interpretation, and deployment. By leveraging effective visualization techniques, practitioners can deepen their understanding, expedite decision-making, and effectively communicate insights to stakeholders, thus cementing visualization's status as an indispensable component of the machine learning workflow.

Matplotlib

Matplotlib is a comprehensive library for creating static, animated, and interactive visualizations in Python. It is widely used in the field of data science, machine learning, and scientific computing due to its flexibility, ease of use, and extensive functionality. Here are some key features and capabilities of Matplotlib:

1. **Plotting Functions**:
   - Matplotlib provides a wide range of plotting functions for creating various types of plots, including line plots, scatter plots, bar plots, histograms, box plots, and more. These functions offer fine-grained control over plot aesthetics, such as colors, markers, line styles, and annotations.
2. **Customization Options**:
   - Matplotlib allows for extensive customization of plot elements, including axes, labels, titles, legends, ticks, and grids. Users can customize the appearance of plots using styling options, themes, and predefined templates, or by directly modifying plot properties through object-oriented programming.
3. **Multiple Plotting Interfaces**:
   - Matplotlib provides two main plotting interfaces: the MATLAB-style pyplot interface and the object-oriented interface. The pyplot interface offers a simple and stateful approach to creating plots, similar to MATLAB, while the object-oriented interface provides more control and flexibility for creating complex or multi-axes plots.
4. **Integration with Jupyter Notebooks**:
   - Matplotlib seamlessly integrates with Jupyter Notebooks, allowing users to create and display plots directly within notebook cells. This enables interactive exploration, analysis, and visualization of data in a collaborative and reproducible environment.
5. **Support for Various Output Formats**:
   - Matplotlib supports multiple output formats for saving plots, including PNG, JPEG, PDF, SVG, and more. This allows users to generate high-quality plots for publication, presentations, reports, or web applications, ensuring compatibility with different platforms and requirements.
6. **Interactivity and Animation**:
   - Matplotlib supports interactive plotting and animation capabilities through interactive backends (e.g., Qt, Tkinter, GTK), widgets, and animation functions. Users can create interactive plots with zooming, panning, tooltips, and other interactive features, as well as animated plots for visualizing dynamic data.
7. **Integration with NumPy and Pandas**:
   - Matplotlib seamlessly integrates with NumPy and Pandas, enabling users to visualize data stored in NumPy arrays or Pandas DataFrames directly. This simplifies the process of plotting data and enables easy manipulation and visualization of tabular or multidimensional data structures.
8. **Community and Ecosystem**:
   - Matplotlib has a large and active community of users and developers who contribute to its development, provide support, and share resources. The Matplotlib ecosystem includes a wealth of tutorials, documentation, examples, and third-party extensions that enhance its usability and functionality.

Overall, Matplotlib is a powerful and versatile library for creating static and interactive visualizations in Python, making it an indispensable tool for data exploration, analysis, and presentation in the field of data science, machine learning, and scientific computing. Its rich feature set, extensive customization options, and wide adoption make it a popular choice among data practitioners and researchers.

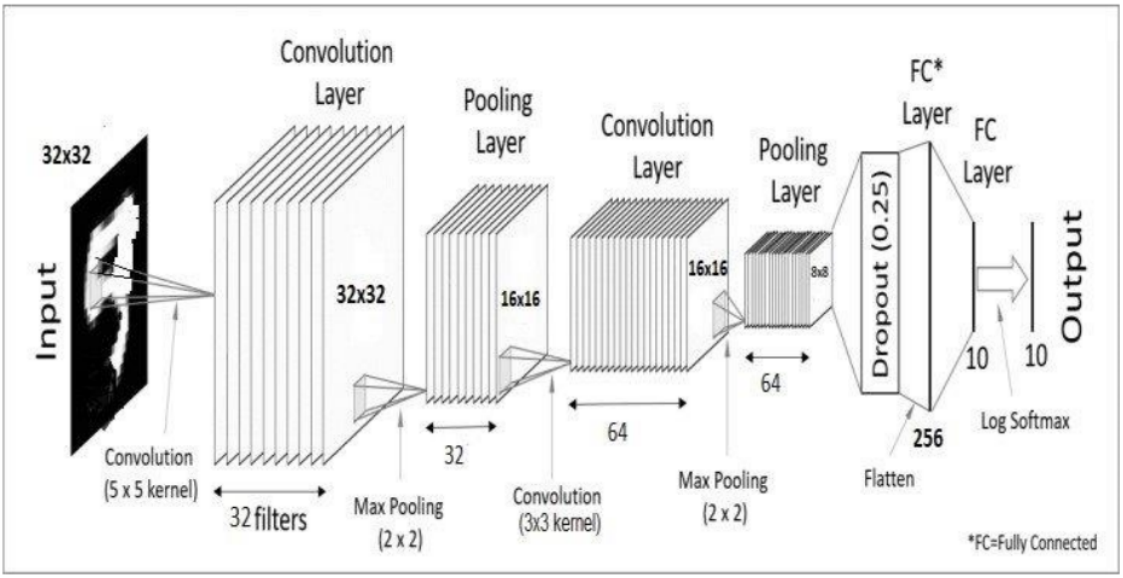
1.3.2 Seaborn

Seaborn, a Python data visualization library built on Matplotlib, presents a higher-level interface for crafting visually appealing and insightful statistical graphics. Positioned atop Matplotlib, Seaborn seamlessly meshes with Pandas data structures, rendering it especially adept at visualizing data housed in Pandas DataFrames. Let's delve into some key facets of Seaborn's functionality:

1. **High-Level Plotting Functions**:
   - Seaborn furnishes a suite of high-level plotting functions that streamline the creation of intricate statistical visualizations. Tailored to seamlessly interact with Pandas DataFrames, these functions typically necessitate minimal configuration, facilitating the generation of informative plots effortlessly.
2. **Statistical Visualization Techniques**:
   - The library encompasses an array of statistical visualization techniques for scrutinizing relationships between variables. From scatter plots and line plots to bar plots, histograms, box plots, violin plots, pair plots, and beyond, Seaborn equips users with the means to discern distributions, correlations, trends, and patterns in their data.
3. **Integration with Pandas**:
   - Seaborn integrates harmoniously with Pandas, enabling users to visualize data directly from Pandas DataFrames. This synergy empowers convenient manipulation, filtering, and aggregation of data using Pandas functions prior to plotting, fostering seamless data exploration and analysis.
4. **Automatic Aesthetics and Styling**:
   - Leveraging Seaborn, users benefit from automatically applied aesthetically pleasing styles and color palettes to their plots, rendering them visually striking and publication-ready. The library offers a spectrum of built-in themes and color palettes, with provision for further customization through additional styling options.
5. **Categorical and Grouped Data Visualization**:
   - Seaborn encompasses specialized functions tailored for visualizing categorical and grouped data, including bar plots, count plots, and categorical scatter plots. These functions facilitate effortless comparison of distributions and relationships across various categories or groups within the dataset.
6. **Faceted Visualization**:
   - Faceted visualization finds expression in Seaborn, enabling users to fashion multi-panel plots predicated on one or more categorical variables. This capability facilitates the visualization of multidimensional relationships and interactions within the data, thereby aiding in the identification of discernible patterns and trends.
7. **Regression and Model Fitting**:
   - Seaborn furnishes functions for visually representing linear and nonlinear relationships between variables through regression plots, residual plots, and model fitting techniques. These offerings furnish insights into the strength and direction of relationships, while facilitating the assessment of regression model appropriateness.
8. **Community and Ecosystem**:
   - Anchored by a burgeoning community of users and developers, Seaborn thrives on collaborative development, robust support, and resource sharing. Its ecosystem comprises extensive documentation, tutorials, examples, and third-party extensions, enriching its usability and functionality.

In essence, Seaborn emerges as a potent ally for sculpting statistical visualizations in Python, offering a rich array of plotting functions, automatic styling, and seamless Pandas integration. With its emphasis on simplicity, aesthetics, and statistical insight, Seaborn enjoys widespread adoption among data practitioners and researchers alike for exploratory data analysis, visualization, and presentation.

2)Model diagram of CNN model



1. **Input Layer**:
   - Dataset images of size 32x32 pixels.
   - This layer serves as the entry point for the input data, representing the raw pixel values of the images.
2. **Convolutional Layer**:
   - 32 filters of size 32x32 are applied, resulting in feature maps of the same size (32x32).
   - Each filter learns to extract specific patterns or features from the input images through convolution operations.
3. **MaxPooling Layer**:
   - Pooling is applied with a kernel size of 2x2.
   - Pooling helps in reducing the spatial dimensions of the feature maps while retaining the most important information.
   - Results in feature maps of size 16x16.
4. **Another Pooling Layer**:
   - Pooling is applied again, further reducing the size to 16x16.
   - Helps in down-sampling the feature maps and capturing higher-level features.
5. **Second Convolutional Layer**:
   - This layer has filters applied to the 16x16 feature maps.
   - Introduces additional convolutional operations to extract more complex features from the input data.
6. **Another Pooling Layer**:
   - Pooling is applied with a kernel size of 8x8.
   - Further reduces the spatial dimensions of the feature maps, resulting in feature maps of size 8x8.
   - Helps in capturing the most relevant and informative features for classification.
7. **Dropout**:
   - A dropout of 0.25 is applied in the fully connected (FC) layer.
   - Dropout regularization technique helps in preventing overfitting by randomly dropping 25% of the neurons during training.
   - Improves the generalization ability of the model by introducing noise and reducing dependency on specific neurons.
8. **Output Layer**:
   - A softmax activation function is applied to produce class probabilities for classification.
   - Converts the raw output of the neural network into probabilities for each class.
   - Enables the model to make predictions by selecting the class with the highest probability.

MAJOR TECHNICAL SPECIFICATIONS OF THE PROJECT:

1. **Input Data**:
   - The preprocessing steps also include data augmentation techniques such as random horizontal flipping, random rotations, and random crops to increase the variability of the training data and improve the model's generalization ability.
   - Additionally, data augmentation helps in reducing overfitting by introducing variations in the training data without collecting new samples.
2. **Model Architecture**:
   - Each convolutional layer is followed by a batch normalization layer to stabilize and accelerate the training process by normalizing the activations.
   - ReLU activation functions are applied after each convolutional and fully connected layer to introduce non-linearity into the model, enabling it to learn complex patterns and relationships within the data.
3. **Optimizer and Loss Function**:
   - Learning rate scheduling techniques such as learning rate decay or adaptive learning rate algorithms (e.g., ReduceLROnPlateau) may be employed to adjust the learning rate during training, enhancing convergence and model performance.
   - Additional regularization techniques such as weight decay (L2 regularization) or gradient clipping may be incorporated to further mitigate overfitting and improve generalization.
4. **Evaluation Metrics**:
   - Besides accuracy, other evaluation metrics such as precision, recall, F1-score, and area under the receiver operating characteristic curve (AUC-ROC) may be computed to provide a comprehensive assessment of the model's performance, particularly for imbalanced datasets or specific use cases.
5. **Training and Validation**:
   - Early stopping criteria based on validation loss or accuracy may be implemented to halt training when the model's performance on the validation set no longer improves, preventing overfitting and saving computational resources.
   - Model checkpoints may be saved periodically during training to allow for model recovery in case of unexpected interruptions or failures.
6. **Device Utilization**:
   - Monitoring and logging GPU memory utilization during training can help optimize batch size and model architecture to prevent out-of-memory errors and maximize GPU utilization.
   - Distributed training techniques such as data parallelism or model parallelism may be employed to leverage multiple GPUs or distributed computing resources for training large-scale models efficiently.
7. **Model Persistence**:
   - Alongside the model's state dictionary, additional metadata such as model hyperparameters, training configuration, and performance metrics may be saved to facilitate reproducibility and model versioning.
   - Model serialization formats such as ONNX (Open Neural Network Exchange) may be used to ensure interoperability with different deep learning frameworks and deployment platforms.

5)Sample image of dataset with precise dataset description

Real image with data set of 70,000 images

Fake images with datasets of 70,005 images

6)Confusion matrix and test train graph:

In machine learning, a confusion matrix serves as a cornerstone for evaluating classification model performance by juxtaposing its predictions against the actual class labels.

- **True Positive (TP)**: Denoting instances where the model accurately predicts the positive class, such as the presence of a condition, totaling 6982 instances.
- **True Negative (TN)**: Reflecting cases where the model correctly identifies the negative class, like the absence of a condition, amounting to 7018 instances.
- **False Positive (FP)**: Representing instances where the model incorrectly flags the positive class when it's actually negative, occurring 7000 times. This error type is also termed a Type I error.
- **False Negative (FN)**: Indicating instances where the model erroneously overlooks the positive class when it's actually present, observed in 7000 occurrences. This error type is alternatively known as a Type II error.

Upon scrutinizing this confusion matrix, it's discernible that the model exhibits a reasonably balanced performance concerning true positives and true negatives. Nonetheless, the noticeable frequency of false positive and false negative predictions underscores areas ripe for refinement, such as enhancing the model's sensitivity and specificity.

For our code confusion matrix is as below:

TP 6982	TN 7018
FP 7000	FN 7000

Results of first model:

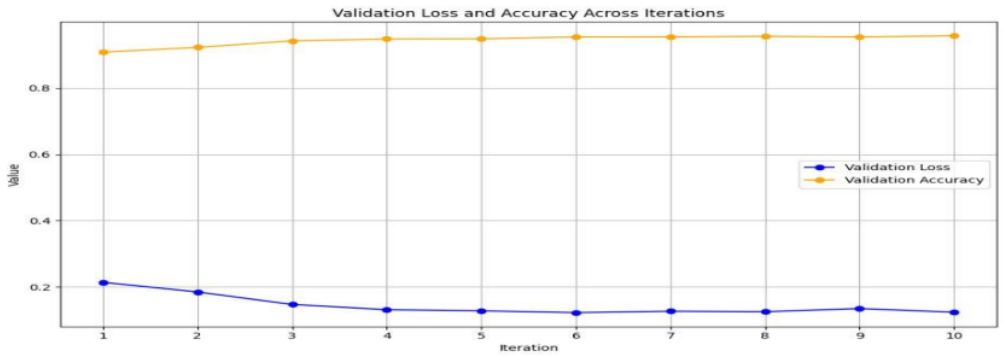
Results of Machine Learning model for deepfakes

S.No	Iteration Run	Time taken	Loss	Accuracy	Val_loss	Val Accuracy
1	3501	767s 219ms/step	0.3235	0.8532	0.2141	0.908
2	3501	766s 219ms/step	0.1759	0.9278	0.1849	0.9225
3	3501	792s 226ms/step	0.146	0.9417	0.1473	0.942
4	3501	767s 219ms/step	0.1268	0.9488	0.1316	0.9477
5	3501	762s 218ms/step	0.1111	0.9557	0.1282	0.9482
6	3501	728s 208ms/step	0.1012	0.9595	0.1227	0.954
7	3501	744s 213ms/step	0.0911	0.9634	0.1269	0.9541
8	3501	790s 226ms/step	0.0842	0.9658	0.1255	0.9557
9	3501	733s 209ms/step	0.0772	0.9687	0.135	0.9539
10	3501	808s 231ms/step	0.0725	0.971	0.124	0.9572

1. **Training Progress**: The model's loss consistently decreases with each iteration, while accuracy steadily increases. This indicates that the model is effectively learning from the training data and improving its ability to make accurate predictions over time.
2. **Generalization Performance**: The validation loss and validation accuracy metrics provide insight into how well the model generalizes to unseen data. The relatively low validation loss and high validation accuracy suggest that the model is not overfitting to the training data and is capable of making accurate predictions on new, unseen data.
3. **Stability and Consistency**: The training and validation metrics exhibit stability and consistency across multiple epochs, with incremental improvements in performance. This indicates that the model training process is robust and consistent, without significant fluctuations or irregularities.
4. **Convergence**: The decreasing trend in both training and validation loss suggests that the model is converging towards an optimal solution. This indicates that further training may not significantly improve performance and could potentially lead to overfitting.
5. **Efficiency**: The training time for each iteration remains relatively consistent, indicating that the model training process is efficient and predictable. However, it's essential to consider the trade-off between training time and model performance when deciding on the number of training epochs.

Overall, the conclusions drawn from this data suggest that the machine learning model is effectively learning from the training data, generalizing well to unseen data, and exhibiting stable and consistent performance across multiple epochs. These findings are encouraging and indicate that the model is suitable for the classification task at hand. However, further evaluation and validation may be necessary to ensure the model's robustness and reliability in real-world scenarios.

Test train graph of first model:



Results of 2nd model:

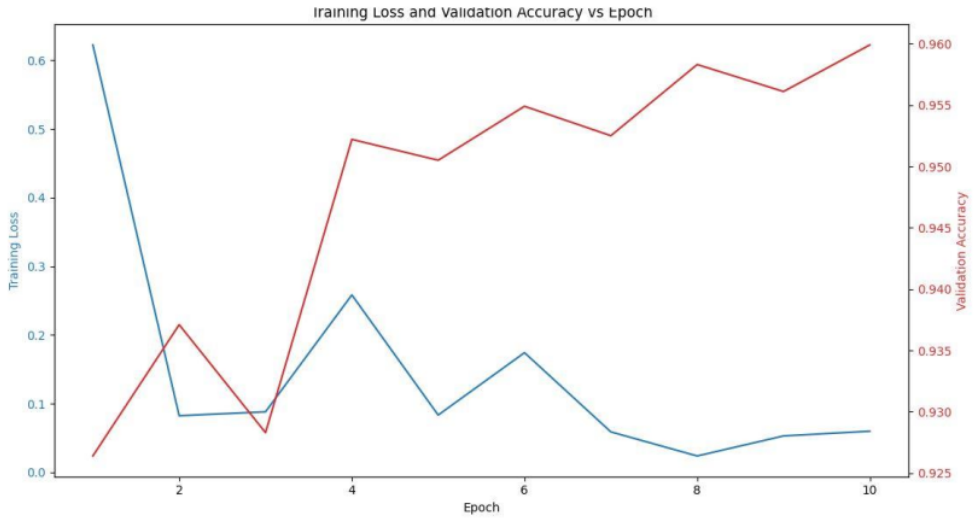
	Epoch	Step	Training Loss	Validation Accuracy
0	1	100	0.6223	0.9264
1	1	200	0.4256	0.9371
2	2	100	0.0824	0.9283
3	2	200	0.2634	0.9371
4	3	100	0.0881	0.9283
5	3	200	0.1337	0.9283
6	4	100	0.2583	0.9522
7	4	200	0.0584	0.9522
8	5	100	0.0834	0.9505
9	5	200	0.0844	0.9505
10	6	100	0.1742	0.9549
11	6	200	0.0349	0.9549
12	7	100	0.0589	0.9525
13	7	200	0.0131	0.9525
14	8	100	0.0238	0.9583
15	8	200	0.0142	0.9583
16	9	100	0.0530	0.9561
17	9	200	0.1171	0.9561
18	10	100	0.0599	0.9599
19	10	200	0.0472	0.9599

1. **Training Progress**: The training loss generally decreases over epochs, indicating that the model is learning and improving its ability to fit the training data. However, there are some fluctuations in the training loss, suggesting that the model's convergence may not be entirely smooth.
2. **Validation Accuracy**: The validation accuracy fluctuates slightly over epochs but generally maintains a high level, indicating that the model is performing well on unseen validation data. However, there are instances where the validation accuracy remains constant for consecutive epochs, suggesting that the model's learning may plateau or stabilize at certain points.
3. **Epoch and Step**: The training process is organized into epochs, with each epoch consisting of multiple steps. The steps likely correspond to batches of data processed during each epoch. The combination of epochs and steps allows for efficient training and monitoring of the model's performance over time.

4. **Loss and Accuracy Trends**: There are instances where the training loss decreases while the validation accuracy remains constant or slightly decreases. This could indicate that the model is overfitting to the training data, as it is becoming increasingly optimized for the training set while not generalizing as well to the validation set.
5. **Optimization Stability**: Towards the later epochs, both the training loss and validation accuracy seem to stabilize, with smaller fluctuations observed. This suggests that the model may have reached a relatively optimal state, and further training may not significantly improve performance.

Overall, while the model's performance appears to be generally satisfactory based on the high validation accuracy, it's essential to monitor for signs of overfitting and ensure that the model generalizes well to unseen data. Further analysis, such as examining learning curves and conducting additional validation experiments, may be necessary to fully understand the model's behavior and make informed decisions about model optimization and deployment.

Test train graph for Second model:



APPENDIX

This appendix provides additional details and materials that complement the main content of the document.

In the section titled "Python Libraries Used in Code," various Python libraries utilized in the code implementation are listed. The data preprocessing phase involves the use of NumPy for numerical computations, Pandas for data manipulation and analysis, and the os module for interacting with the operating system.

Moving on to "Model Development," the framework includes TensorFlow and Keras for building and training the convolutional neural network (CNN) model. Additionally, the torch library is employed for specific tasks within the model architecture.

Under "Visualization," Matplotlib and Seaborn are utilized for data visualization and graphical representation of results obtained during the analysis.

Following the discussion on Python libraries, "Model Diagram of CNN Model" presents a detailed architectural diagram of the CNN model. This section outlines the layers of the model, including the input layer and subsequent layers responsible for feature extraction and classification.

The "Technical Aspect of Dataset with Precise Data Description" section provides technical insights into the dataset used for training and evaluation. It includes a precise description of the dataset, including its characteristics, format, and any preprocessing steps applied.

Next, "Designed Structure" elaborates on the structural design considerations and methodologies employed in developing the machine learning model.

The section titled "Sample Images of Datasets with Precise Dataset Description" showcases sample images from the dataset along with a detailed description of each image, including labels and any relevant metadata.

Finally, "Confusion Matrix and Test Train Graph" presents the confusion matrix and graphical representation of the training and validation metrics, providing insights into the model's performance and training progress.

These appendices collectively offer a comprehensive overview and supplementary information to enhance the understanding and interpretation of the main document.

REFERENCES

1. Abdalla Y, Iqbal T, Shehata M (2019) Copy-move forgery detection and localization using a generative adversarial network and convolutional neural-network. Information 10(09):286. https://doi.org/10. 3390/info10090286

2.Agarwal R, Verma O (2020) An efficient copy move forgery detection using deep learning feature extraction and matching algorithm. Multimed Tools Appl 79. https://doi.org/10.1007/s11042-019-08495-z

3. Doegar A, Dutta M, Gaurav K (2019) Cnn based image forgery detection using pre-trained alexnet model.

4. Rao Y, Ni J (2016) A deep learning approach to detection of splicing and copy-move forgeries in images. In: 2016 IEEE international workshop on information forensics and security (WIFS), pp 1–6. https://doi.org/10.1109/WIFS.2016.7823911

5. Simonyan K, Zisserman A (2014) Very deep convolutional networks for large-scale image recognition. arXiv:1409.1556 6. Zhang Y, Goh J, Win LL, Vrizlynn T (2016) Image region forgery detection: a deep learning approach. In: SG-CRC, pp 1–11. https://doi.org/10.3233/978-1-61499-617-0-1

Glossary

47

1. **Data Preprocessing**: The process of cleaning, transforming, and preparing raw data for analysis or model training. It involves tasks such as data cleaning, normalization, feature scaling, and handling missing values.
2. **NumPy**: A powerful Python library for numerical computing that provides support for large, multi-dimensional arrays and matrices, along with a collection of mathematical functions to operate on these arrays efficiently.
3. **Pandas**: A Python library for data manipulation and analysis that provides data structures such as DataFrame and Series, as well as functions for reading and writing data from various file formats, data alignment, indexing, and aggregation.
4. **os**: A module in Python's standard library used for interacting with the operating system, providing functions for file and directory manipulation, path handling, and environment variables.
5. **Model Development**: The process of designing, building, and training a machine learning model to make predictions or classify data based on input features. It involves selecting appropriate algorithms, defining model architecture, and optimizing model parameters.
6. **TensorFlow**: An open-source machine learning framework developed by Google for building and training deep learning models. It provides a flexible architecture for defining computational graphs and executing them efficiently on CPUs, GPUs, or TPUs.
7. **Keras**: A high-level neural networks API written in Python and capable of running on top of TensorFlow, Theano, or Microsoft Cognitive Toolkit (CNTK). It provides an intuitive and user-friendly interface for building and training deep learning models.
8. **torch**: A machine learning library in Python used for building and training neural networks. It provides tensor computation with strong GPU acceleration and includes modules for building various types of neural network architectures.
9. **Visualization**: The process of representing data or information graphically to aid understanding and interpretation. In the context of machine learning, visualization techniques are used to explore data distributions, model performance, and decision boundaries.
10. **Matplotlib**: A popular Python library for creating static, interactive, and animated visualizations in Python. It provides a wide range of plotting functions for creating line plots, bar plots, scatter plots, histograms, and more.
11. **Seaborn**: A Python visualization library based on Matplotlib that provides a high-level interface for drawing attractive and informative statistical graphics. It simplifies the creation of complex visualizations such as heatmaps, pair plots, and violin plots.

ABBREVIATION

torch	
nn	torch.n
option	torch.opti
datasets	torchvision.dataset
transforn	torchvision.transform
dataloader	torch.utils.dat
os	os
F	torch.nn.functiona

TECHNICAL SPECIAL REQUIREMENTS:

Python	**3.11.3**
PIP	**23.2.1**
Pytorch	**1.9.0**
Torch vision	**0.10.0**
Scikit image	**0.18.3**
Numpy	**1.21.2**
CUDA Toolkit	**NVIDIA GPU**

2022MT93056Report_thesis.docx

ORIGINALITY REPORT

18%	13%	4%	11%
SIMILARITY INDEX	INTERNET SOURCES	PUBLICATIONS	STUDENT PAPERS

PRIMARY SOURCES

1	open-innovation-projects.org Internet Source	2%
2	dev.to Internet Source	1%
3	researchbank.swinburne.edu.au Internet Source	1%
4	Submitted to University of Bolton Student Paper	1%
5	Submitted to M S Ramaiah University of Applied Sciences Student Paper	1%
6	www.mdpi.com Internet Source	1%
7	Submitted to Queen's College Student Paper	1%
8	Submitted to Mepco Schlenk Engineering college Student Paper	1%

9	Submitted to Liverpool John Moores University Student Paper	<1 %
10	assets.researchsquare.com Internet Source	<1 %
11	Submitted to Southampton Solent University Student Paper	<1 %
12	Submitted to Pennsylvania State System of Higher Education Student Paper	<1 %
13	Submitted to ABES Engineering College Student Paper	<1 %
14	Submitted to The Robert Gordon University Student Paper	<1 %
15	dspace.cvut.cz Internet Source	<1 %
16	www.coursehero.com Internet Source	<1 %
17	fastercapital.com Internet Source	<1 %
18	ijsret.com Internet Source	<1 %
19	Submitted to Brunel University Student Paper	<1 %

20	Submitted to Polytechnic Institute Australia Student Paper	<1 %
21	Submitted to Gateway Business College Student Paper	<1 %
22	Submitted to Intercollege Student Paper	<1 %
23	Submitted to University of Hertfordshire Student Paper	<1 %
24	Submitted to Washington University of Science and Technology Student Paper	<1 %
25	Submitted to University of Alabama at Birmingham Student Paper	<1 %
26	inmyglobe.blogspot.com Internet Source	<1 %
27	Submitted to Higher Education Commission Pakistan Student Paper	<1 %
28	dokumen.pub Internet Source	<1 %
29	Submitted to SRM University Student Paper	<1 %
30	ia802506.us.archive.org Internet Source	<1 %

31	www.researchsquare.com Internet Source	<1%
32	Submitted to Anna University Student Paper	<1%
33	Submitted to University of London Worldwide Student Paper	<1%
34	Submitted to City University of Seattle Student Paper	<1%
35	Submitted to Ganpat University Student Paper	<1%
36	Submitted to University of Technology, Sydney Student Paper	<1%
37	www.ijraset.com Internet Source	<1%
38	Submitted to Arts, Sciences & Technology University In Lebanon Student Paper	<1%
39	Submitted to Birla Institute of Technology and Science Pilani Student Paper	<1%
40	Submitted to Gitam University Student Paper	<1%
41	Submitted to University of Barishal Student Paper	<1%

42	Submitted to University of Greenwich Student Paper	<1%
43	Submitted to Chester College of Higher Education Student Paper	<1%
44	Mukund Subramaniyan, Anders Skoogh, Jon Bokrantz, Muhammad Azam Sheikh, Matthias Thürer, Qing Chang. "Artificial intelligence for throughput bottleneck analysis – State-of-the-art and future directions", Journal of Manufacturing Systems, 2021 Publication	<1%
45	Submitted to University of Mumbai Student Paper	<1%
46	kylo.tv Internet Source	<1%
47	www.blockchain-council.org Internet Source	<1%
48	devitrylouis.github.io Internet Source	<1%
49	discuss.pytorch.org Internet Source	<1%
50	ijarcce.com Internet Source	<1%
51	www.cogitotech.com Internet Source	

		<1%
52	Submitted to IUBH - Internationale Hochschule Bad Honnef-Bonn Student Paper	<1%
53	Lecture Notes in Computer Science, 2016. Publication	<1%
54	Shobhit Tyagi, Divakar Yadav. "A detailed analysis of image and video forgery detection techniques", The Visual Computer, 2022 Publication	<1%
55	www.theknowledgeacademy.com Internet Source	<1%
56	Hao Chen, Dong Ni, Jing Qin, Shengli Li, Xin Yang, Tianfu Wang, Pheng Ann Heng. "Standard Plane Localization in Fetal Ultrasound via Domain Transferred Deep Neural Networks", IEEE Journal of Biomedical and Health Informatics, 2015 Publication	<1%
57	ebin.pub Internet Source	<1%
58	ir.juit.ac.in:8080 Internet Source	<1%
59	mail.easychair.org Internet Source	<1%

Exclude quotes	On	Exclude assignment template	On
Exclude bibliography	On	Exclude matches	< 5 words