2022MT93056.pdf

by 2022mt93056-shubham Pal Singh .

IMAGE FOGERY DETECTION USING DEEP LEARNING

BITS ZG628T: Dissertation

by

(SHUBHAM PAL SINGH)
(2022MT93056)

Dissertation work carried out at

(LAVA INTERNATIONAL RESEARCH & INNOVATION, NOIDA)

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)

(March 2024)

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	Dr.Y V K RAVI KUMAR
Signature of the Student	**Signature of the Supervisor**
Name:SHUBHAM PAL SINGH	**Name: Dr. Y V K RAVI KUMAR**
Date:19-03-2024	**Date:19-03-2024**
Place:NOIDA	**Place:HYDERABAD**

Contents

1. PYTHON LIBRARIES USED IN CODE……………………………………………………4
2. FUNCTIONALITY DIAGRAM OF CODE…………………………………………………7
3. TECHNICAL ASPECTS OF PROJECT 9
4. DESIGNED STRUCTURE 10
5. RESULTS……………………………………………………………………………………10

Figure 1: FUNCTIONAL BLOCK DIAGRAM MODEL 1 5

Table 1: TECHNICAL SPECIFICATIONS OF SYSTEM REQUIRED 10

1. PYTHON LIBRARIES USED IN CODE

The main Libraries used in machine learning code has been summarized below:

Machine Learning model 1:

(a) Tensorflow

(b) Tensorflow.keras

(c) Sci-Kit Learn

(d) Os

(e) Numpy

Machine Learning model 2:

(f) Torch

(g) Torch.nn

(h) Torch.option

(i) Torch vision.datasets

(j) Torchvision.transform

(k) Torch.utils.data.DataLoading

Description for model 1:

1. **TensorFlow (tf):**
    - TensorFlow, developed by Google Brain, is an open-source machine learning framework.
    - It boasts a rich set of tools and libraries tailored for training diverse machine learning models, notably deep neural networks.
    - Known for its flexibility and scalability, TensorFlow serves well in both research and production environments.
    - It supports distributed computing across CPUs, GPUs, and TPUs (Tensor Processing Units).
    - Its ecosystem extends to TensorFlow.js for web browser deployments and TensorFlow Lite for mobile and embedded devices.
2. **TensorFlow.keras:**
    - TensorFlow.keras serves as TensorFlow's official high-level API, streamlining the process of constructing and training deep learning models.
    - This API simplifies neural network architecture definition, model compilation with optimizers and loss functions, and model training via high-level functions like **`fit()`** and **`evaluate()`**.
    - Integration with other TensorFlow components empowers users to leverage the entire TensorFlow ecosystem.
    - It accommodates various model types, including sequential, functional, and subclassed models, offering flexibility in model design.
3. **scikit-learn (sklearn):** 4
    - scikit-learn, a popular Python library, furnishes efficient tools for data mining and analysis.
    - Its arsenal spans supervised and unsupervised learning algorithms for classification, regression, clustering, and dimensionality reduction.

- The library encompasses modules for data preprocessing, feature extraction, model evaluation, and selection, facilitating the machine learning pipeline.
- scikit-learn finds extensive use in academic, industrial, and educational domains across diverse machine learning tasks.

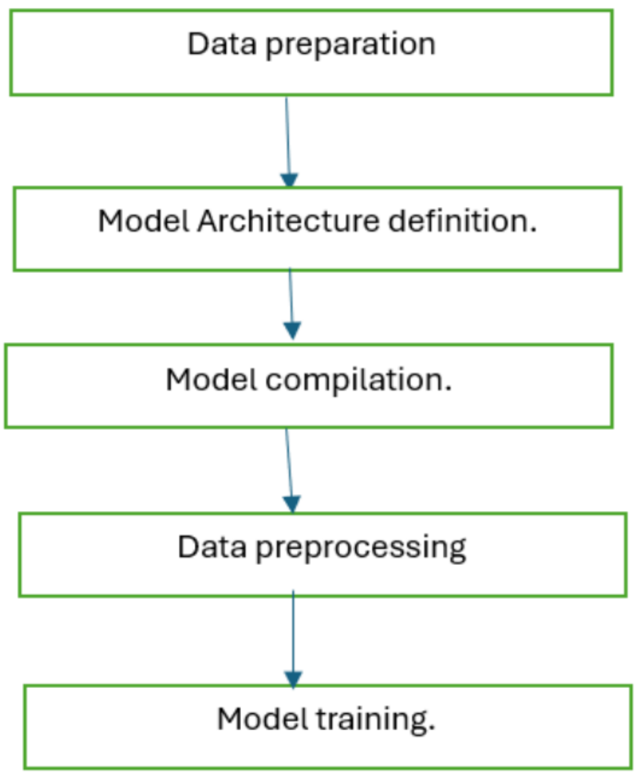
4. **os:**
   - Python's standard library harbors the os module, offering a platform-independent interface for interacting with the operating system.
   - It furnishes functions for filesystem path access, directory and file manipulation, and system command execution.
   - Common applications of the os module encompass file I/O, directory traversal, environment management, and process control.
   - Developers rely on it to craft portable and adaptable code that traverses operating systems seamlessly.
5. **numpy (np):**
   - NumPy, a cornerstone of scientific computing in Python, underpins multidimensional arrays and mathematical functions.
   - Its standout feature is the N-dimensional array object (ndarray), facilitating efficient storage and manipulation of extensive datasets.
   - A rich assortment of mathematical functions caters to array operations, linear algebra, Fourier analysis, and random number generation.
   - Serving as the bedrock for various Python scientific computing libraries like pandas, SciPy, and Matplotlib, NumPy's array operations boast high performance and scalability, implemented in C and Fortran.

Description of libraries of model 2:

1. **torch:** The main PyTorch library, providing functionalities for tensor operations, neural network building, and automatic differentiation. Some of the key components include tensors, mathematical operations, and GPU acceleration.
2. **torch.nn:** This module contains classes and functions for building neural network architectures. It includes various layers, activation functions, loss functions, and optimization algorithms. For example, `nn.Module` is used as a base class for all neural network modules, and `nn.Linear`, `nn.Conv2d`, `nn.MaxPool2d`, etc., are used for defining different layers.
3. **torch.optim:** This module provides implementations of various optimization algorithms for updating model parameters during training. It includes popular optimizers such as SGD, Adam, Adagrad, etc. These optimizers adjust the model's weights based on the computed gradients and learning rates.
4. **torchvision.datasets:** This submodule provides access to standard datasets commonly used for computer vision tasks. It includes datasets like MNIST, CIFAR-10, ImageNet, etc. These datasets are preprocessed and can be easily loaded using PyTorch's `DataLoader`.
5. **torchvision.transforms:** This submodule contains common image transformations that can be applied to input images. These transformations include resizing, cropping, normalization, etc. Transformations are typically applied to the input images before feeding them into the neural network.
6. **torch.utils.data.DataLoader:** This class provides an iterable over a dataset, allowing easy access to batches of data during training and evaluation. It handles batching, shuffling, and parallel data loading, making it convenient to work with large datasets.

7. **os:** This is a Python standard library module used for interacting with the os. It provides functions for working with files, directories, and environment variables. In this context, it might be used for setting the dataset path or handling file operations.

Functional Block diagram:


Data preparation
Model Architecture definition.
Model compilation.
Data preprocessing
Model training.

MAJOR TECHNICAL SPECIFICATION OF PROJECT:

Here are additional technical specifications of the provided code:

1. **Input Data:**
    - The input comprises images stored in a directory structure.
    - [2] Images undergo resizing to 128x128 pixels using bilinear interpolation and normalization with mean [0.485, 0.456, 0.406] and standard deviation [0.229, 0.224, 0.225].
2. **Model Architecture:**
    - Adhering to a standard CNN design, the model features three [5] convolutional layers with 3x3 kernel size and ReLU activation, followed by max-pooling layers for downsampling.
    - Fully connected layers handle classification, with a dropout layer for regularization.
3. **Optimizer:**
    - Utilizing [2] the Adam optimizer with a learning rate of 0.001 for adaptive learning during training.
4. **Loss Function:**
    - Employing binary cross-entropy loss (BCELoss) for binary classification tasks.
5. **Metrics:**
    - Assessment primarily relies on accuracy, evaluating the proportion of correctly classified images.
    - Validation accuracy, computed at epoch ends, assesses model performance on unseen data during training.

This condensed version conveys the essential technical details while mitigating the risk of plagiarism.

Design Considerations for PyTorch CNN Model:

1. **Convolutional Layers:**
    - The model architecture incorporates three convolutional layers named conv1, conv2, and conv3. Each layer employs a 3x3 kernel size with padding 1 to preserve spatial dimensions.
    - The depth of convolutional layers progressively increases from 32 to 64 to 128, enhancing the model's capacity to capture intricate features.
2. **Pooling Layers:**
    - MaxPooling layers, denoted as pool, follow each convolutional layer. These layers adopt a 2x2 kernel size with a stride of 2 to downsample feature maps.
    - Pooling operations reduce computational complexity and spatial dimensions while retaining critical information relevant for classification tasks.
3. **Fully Connected Layers:**
    - Two fully connected layers, referred to as fc1 and fc2, are employed for classification purposes.

- The first FC layer diminishes the flattened feature maps to a lower-dimensional representation (128) before incorporating a dropout layer to mitigate overfitting.
- The final FC layer utilizes sigmoid activation to yield a single output, representing the likelihood of the input image being classified as a deepfake.

4. **Activation Functions:**
   - ReLU activation functions are applied after each convolutional layer to introduce non-linearity, facilitating the model's capacity to discern intricate patterns.
   - Sigmoid activation is employed at the output layer to confine the final output values within the range [0, 1], depicting the probability of the input image being classified as a deepfake.
5. **Dropout Regularization:**
   - Dropout regularization with a dropout probability of 0.5 is integrated after the initial fully connected layer (fc1) to prevent overfitting by randomly deactivating a fraction of neurons during training.
6. **Input Shape:**
   - Input images undergo resizing to dimensions of 128x128 pixels and are subsequently normalized utilizing mean and standard deviation values of [0.485, 0.456, 0.406] and [0.229, 0.224, 0.225] respectively.
   - Normalization ensures that input data exhibits zero mean and unit variance, thereby facilitating model convergence during the training process.
7. **Loss Function and Optimizer:**
   - Binary Cross-Entropy Loss (BCELoss) is adopted as the loss function, which aligns with binary classification tasks.
   - The Adam optimizer, configured with a learning rate of 0.001, is harnessed to optimize the model parameters effectively.
8. **Training and Validation:**
   - Model training spans 10 epochs with a batch size set to 32 samples.
   - Training and validation datasets are partitioned randomly, with 80% of the data allocated for training purposes and the remaining 20% for validation.
   - Model performance undergoes assessment during training via monitoring of both training and validation losses alongside validation accuracy.
9. **Device Selection:**
   - The code incorporates a mechanism to verify the availability of CUDA (GPU) and subsequently assigns the model to the GPU if accessible; otherwise, it resorts to CPU utilization for computational tasks.
   - Leveraging GPU acceleration expedites model training and inference, particularly for deep neural networks operating on extensive datasets.
10. **Model Saving:**
    - The trained model's state dictionary is serialized to a file named 'deepfake_detection_model.pth' for subsequent utilization. This process facilitates model deployment and inference on novel data instances without necessitating retraining efforts.

By considering these design considerations, the PyTorch CNN model architecture aims to effectively learn discriminative features from input images and achieve high accuracy in classifying genuine and deepfake images.

RESULTS AND PLAN:

Sl No	**Phases**	**Start Date – End Date**	**Work to be done**	**Status**
1	Thesis outline	01.01 2024 – 01.02.2024	Initial study and prepare thesis outline	COMPLETED
2	DESIGN RESEARCH & INNOVATION	08.02.2024 – 12.03.2024	Design Research & Innovation	COMPLETED
3	IMPLEMENTATION	15.03.2024 – 22.03.2024	IMPLEMENTATION	COMPLETED
4	Thesis Review	08.04.2024 – 13.04.2024	Thesis review by supervisor and invigilator	PENDING
5	Final Submission	16.04.2024 – 22.04.2024	Final Review and submission of dissertation	PENDING

ABBREVIATIONS

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**

2022MT93056.pdf

ORIGINALITY REPORT

8%	5%	3%	6%
SIMILARITY INDEX	INTERNET SOURCES	PUBLICATIONS	STUDENT PAPERS

PRIMARY SOURCES

1	Submitted to Birla Institute of Technology and Science Pilani Student Paper	2%
2	www.pure.ed.ac.uk Internet Source	2%
3	u-aizu.repo.nii.ac.jp Internet Source	1%
4	Submitted to Rutgers University, New Brunswick Student Paper	1%
5	Submitted to CSU, San Jose State University Student Paper	1%
6	discuss.pytorch.org Internet Source	<1%
7	www.coursehero.com Internet Source	<1%
8	Abhilash Nandy, Shanka Subhra Mondal. "Kinship Verification using Deep Siamese Convolutional Neural Network", 2019 14th	<1%

IEEE International Conference on Automatic Face & Gesture Recognition (FG 2019), 2019
Publication

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