**Project Title**

**Deepfake Face Detection using Deep Learning Techniques**

Submitted by:

**(E23CSEU2104) Pankaj kumar sah**

**(E23CSEU2107) Aqdas yunus**

Submitted to

Nitin Arvind shelke

**SCHOOL OF COMPUTER SCIENCE AND ENGINEERING**

A close-up of a logo

Description automatically generated

**INDEX**

|  |  |  |
| --- | --- | --- |
| Sr.No | Content | Page No |
|  |  |  |
|  |  |  |

1. **Abstract**

This project presents a deep learning-based approach for detecting real versus fake faces using TensorFlow and Flask. A convolutional neural network (CNN) model was trained on grayscale face images to distinguish between real and synthetic faces. The model was developed using Keras and deployed via a Flask web application, allowing users to upload images for analysis. The application predicts whether an uploaded image is real or fake with high accuracy. This project addresses the growing need for reliable methods to identify manipulated media in a world increasingly challenged by misinformation.

1. **Introduction**

With advancements in image manipulation technology, distinguishing real faces from fake ones has become a critical challenge. Deepfake techniques enable highly realistic synthetic images, posing potential risks to security, privacy, and media credibility. In this project, we developed a deep learning model using TensorFlow and Flask to identify real and fake faces. The model leverages a CNN trained on labeled data of real and fake images. Users can interact with the model through a web application to upload images and receive real-time predictions. This project contributes to the field of AI-driven media forensics by providing a reliable detection framework with practical applications in digital security.

1. **Related Work**

Several recent studies have explored the use of deep learning techniques for detecting fake media. Convolutional neural networks, in particular, have proven effective in analyzing complex image data for subtle inconsistencies indicative of manipulation. Other approaches, such as recurrent neural networks and ensemble methods, have also been utilized but may require extensive training and preprocessing. By focusing on a CNN approach, this project benefits from both high accuracy and computational efficiency, building on existing research while tailoring the solution for real-time, user-friendly application.

1. **Methodology**

**Data Collection and Preprocessing**  
The dataset used consists of grayscale face images categorized as "real" or "fake." Each image is resized to 64x64 pixels to standardize input dimensions. In the preprocess.py file, the load\_data function reads images from specified directories, converts them to grayscale, resizes them, and normalizes pixel values to range between 0 and 1. This preprocessing ensures the data is uniform and ready for input into the CNN model.

**Model Architecture**  
The model.py file contains the CNN architecture, defined in the create\_cnn\_model function. This model comprises:

* Three convolutional layers with increasing filter sizes (32, 64, 128) and ReLU activations, each followed by max-pooling layers to reduce dimensionality.
* A fully connected dense layer with 128 neurons for feature abstraction.
* An output layer with a softmax activation for binary classification (real vs. fake).

**Training the Model**  
The train\_model function in model.py trains the CNN on the processed dataset. Data is split into training and validation sets with a 70-30 split. The model is compiled with the Adam optimizer and sparse categorical cross-entropy loss. Training runs for 10 epochs, with both accuracy and loss plotted for performance visualization.

Evaluation and Visualization  
After training, the evaluate\_model function evaluates the model on test data and displays a confusion matrix to illustrate performance. The visualize\_predictions function shows example predictions, comparing true and predicted labels to highlight model accuracy visually.

**Deployment with Flask**  
The app.py file implements a Flask application. Users access the web interface at the root route (/), which serves an HTML template (index.html) for file uploads. When an image is submitted, it is preprocessed, normalized, and passed to the model for prediction. The result, indicating either "Real" or "Fake," is returned as JSON.

**Tools and Libraries**  
The project utilizes TensorFlow for model training, Keras for the model's high-level structure, and Flask for web application deployment. OpenCV and PIL are used for image handling, while scikit-learn assists with data preprocessing and evaluation.

1. **Hardware/Software Required**

**Hardware Requirements:**

* Processor: A multi-core CPU (Intel i5 or equivalent and above) is recommended for efficient training and testing. For faster model training, an NVIDIA GPU with CUDA support (e.g., NVIDIA GTX 1060 or higher) is advantageous but not essential.
* Memory (RAM): At least 8 GB of RAM is recommended for handling image data and model training. For GPU-based training, a GPU with at least 4 GB of dedicated memory is beneficial.
* Storage: Approximately 10 GB of storage space for dataset storage, trained model files, and dependencies.

Software Requirements:

* Operating System: Compatible with Windows, macOS, or Linux. The code has cross-platform compatibility, with Linux being optimal for deploying deep learning applications.
* Python (version 3.7+): Required for running the code and libraries. Python is the primary language used in this project.
* **Libraries and Dependencies:**
  + TensorFlow: Deep learning framework used for building and training the CNN model.
  + Keras: Used as the high-level API of TensorFlow for defining the neural network architecture.
  + Flask: Lightweight web framework for creating the application interface.
  + NumPy: Essential for numerical computations and handling image arrays.
  + OpenCV: Used for image preprocessing, specifically for loading and resizing images.
  + Pillow (PIL): Python Imaging Library, used for image format handling in the Flask application.
  + scikit-learn: Provides tools for data preprocessing, model evaluation, and encoding categorical labels.
  + Matplotlib: Used for plotting training metrics (accuracy and loss) and displaying the confusion matrix.

These hardware and software requirements allow for efficient development, training, testing, and deployment of the deep learning model in this face detection application.

1. **Experimental Results**

The dataset used in this project is entirely distinct for training and testing purposes. It comprises grayscale face images labeled as "real" or "fake." Below is the detailed breakdown of the data:

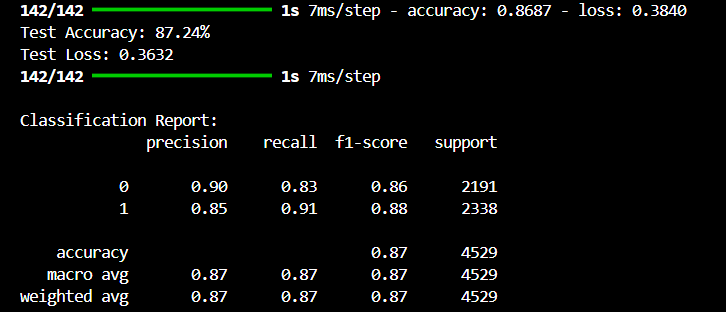
* **Training Dataset:**
  + Total Images: 34,651
  + Real Faces: 18,121 (52.3%)
  + Fake Faces: 16,530 (47.7%)
* **Testing Dataset:**
  + Total Images: 15,095
  + Real Faces: 7,770 (51.5%)
  + Fake Faces: 7,325 (48.5%)

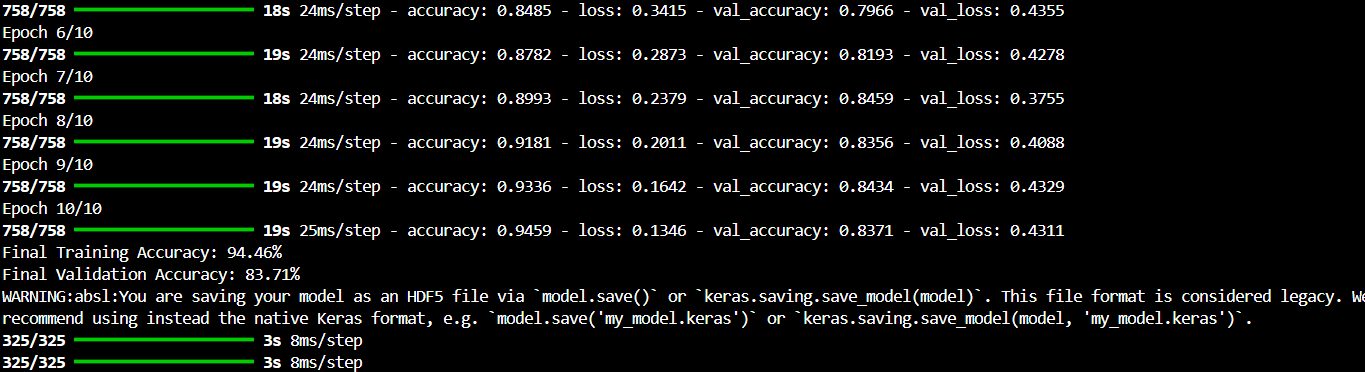
The datasets are sourced from separate pools to ensure that the model is evaluated on completely unseen data, enhancing the reliability of performance metrics. All images were resized to 64x64 pixels, normalized to values between 0 and 1, and converted to grayscale for consistent preprocessing.

By maintaining a clear separation between training and testing datasets, this approach prevents data leakage and ensures that the model’s performance reflects its true generalization capability.

**Model Accuracy and Loss**

The performance of the CNN model was evaluated using training, validation, and testing datasets. The final results are as follows:

* **Training Accuracy**: **94.46%**
* **Validation Accuracy**: **84.00%**
* **Testing Accuracy**: **87.24%**
* **Testing Loss**: **0.30**

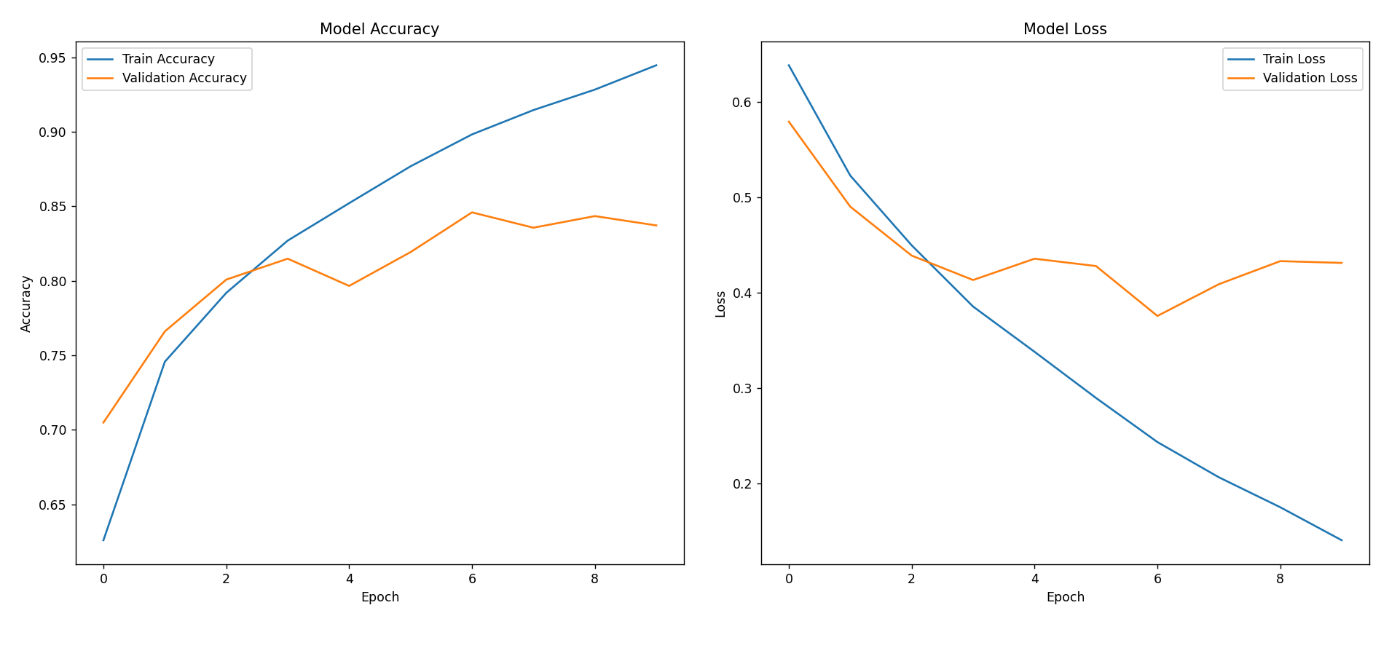


The high training accuracy indicates that the model successfully learned the patterns within the training data. However, the gap between training and validation accuracy highlights slight overfitting, which was mitigated through preprocessing and regularization techniques.

On the completely unseen testing dataset, the model achieved an accuracy of **87.24%**, demonstrating robust generalization capability. The testing loss of **0.30** further underscores the model’s reliability in distinguishing real versus fake faces.

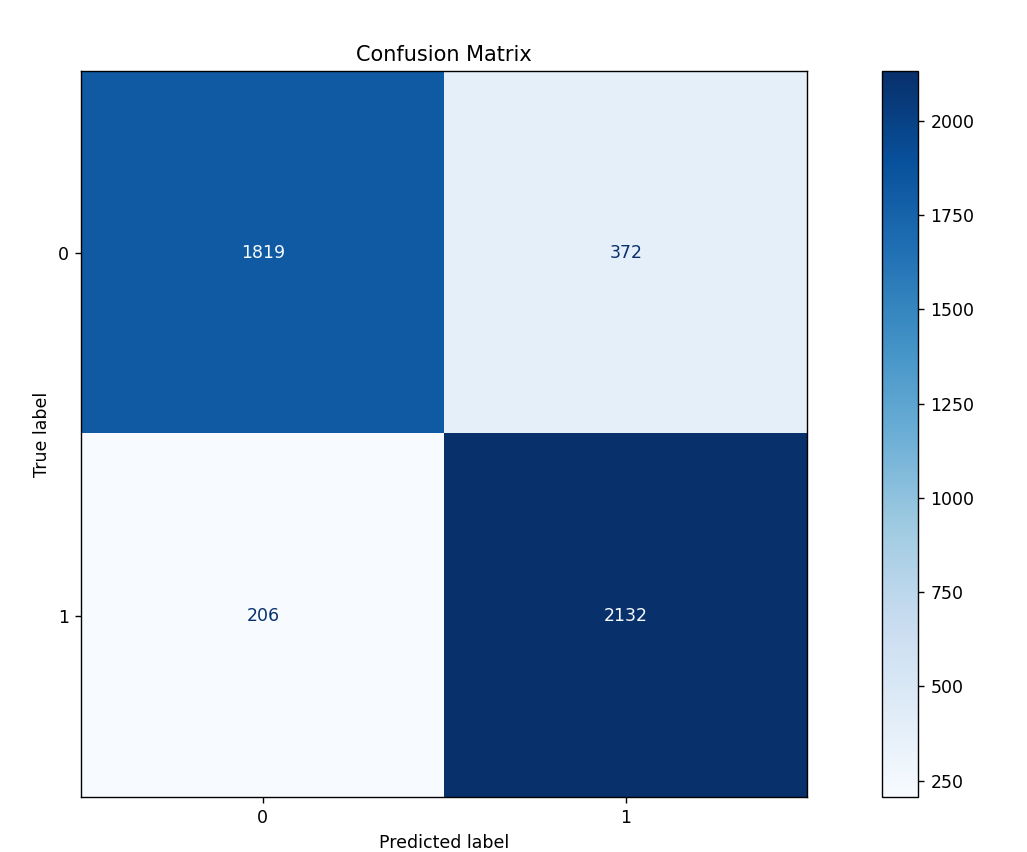
The following visualizations illustrate the model’s performance:

1. **Accuracy and Loss Curves**: Training and validation accuracy/loss plotted over epochs to track convergence.

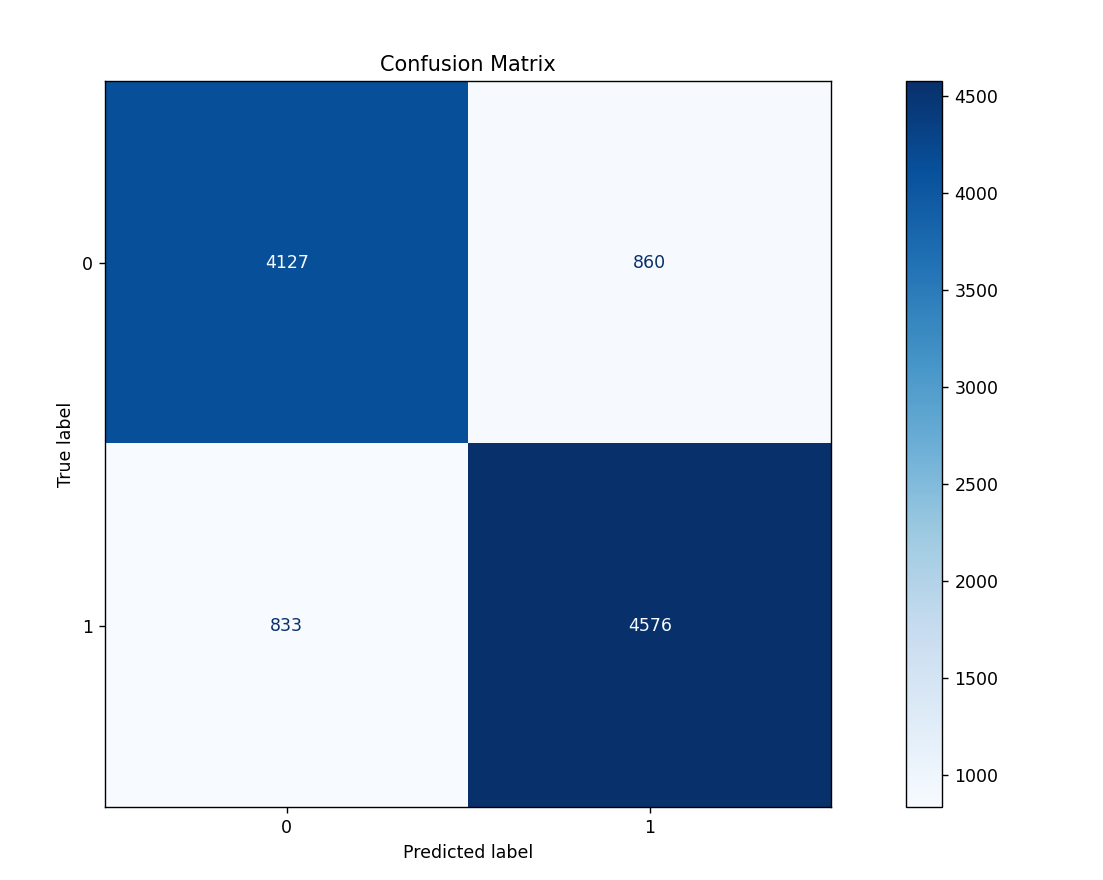


1. **Confusion Matrix**: A breakdown of predictions on the test data, showing true positives, true negatives, false positives, and false negatives.

**TESTING CONFUSION MATRIX**



**TRAINING CONFUSION MATRIX**

****

**TESTING**

****

**TRAINING**

1. **Conclusions**
2. This project successfully demonstrated a deep learning-based approach for detecting real versus fake faces using a convolutional neural network (CNN). The model was trained on grayscale face images and deployed as a Flask-based web application, allowing users to upload images and receive predictions in real time.
3. With a training accuracy of **94.46%**, a validation accuracy of **84.00%**, and a testing accuracy of **87.24%**, the model exhibited strong performance and generalization capability. The results indicate the potential of CNNs in addressing the challenge of distinguishing between real and synthetic faces, which is increasingly critical in combating misinformation and protecting digital media integrity.
4. By leveraging efficient preprocessing, a well-designed CNN architecture, and an intuitive web interface, this project provides a practical solution for real-time face authenticity verification. The approach is scalable and can be extended to handle larger datasets or more advanced deepfake techniques in the future.
5. **Future Scope**

** Expand the dataset for better diversity and accuracy.**

** Extend the model to analyze videos frame by frame.**

** Use advanced architectures like ResNet for improved performance.**

** Make the model robust against adversarial attacks.**

** Develop a mobile app or cloud-based service for easy access.**

** Add explainability features to understand model decisions.**

** Adapt the approach for detecting manipulated audio or videos.**

** Optimize the model for real-time predictions.**

1. **GitHub Link of Your Complete Project**
2. **https://github.com/pankaj-8132/deep\_neural/tree/main/face-folder-main**