Image Classification of Fake vs. Real Images

Team Name: Sahab

Team Members: Agam Harpreet Singh & Aarav Dawer Bhojwani

February 10, 2025

1 Introduction

In this project, our goal was to build a robust image classification system capable of distinguishing between *fake* and *real* images. The development process evolved through several stages—from an initial simple Convolutional Neural Network (CNN) to a complex model employing transfer learning with EfficientNetB0, augmented by extensive data preprocessing and cleaning. This report outlines our approach, detailing the progression of model complexity, the data cleaning and preprocessing steps, and the final implementation used for inference.

2 Project Repository

The complete project details, including source code, experiment results, and further documentation, are available on our GitHub repository. You can access the repository at:

`https://github.com/Agam77055/Deepfake_detection`

3 Data Preprocessing and Cleaning

3.1 Data Collection and Initial Challenges

- **Dataset Structure:** The raw dataset was organized into two main subdirectories:
  - `fake/` — Supposed to contain fake images.
  - `real/` — Supposed to contain real images.
- **Data Quality Issues:** During our initial analysis, we observed that the dataset contained misclassified images (e.g., real images mistakenly placed in the fake folder, and vice versa). Such inconsistencies could negatively impact the model's ability to learn meaningful features.

3.2 Data Cleaning Process

– **Automated Separation:** We developed a Python script that scanned through the dataset directories, examined the images, and separated them based on content analysis and available metadata. The script flagged any mismatches (where real images appeared in the fake dataset or vice versa) for manual verification. The cleaned images were then re-assigned to the correct folder.

– **Label Assignment:** After cleaning, labels were assigned programmatically:

  – **0** for images in the `fake` folder.

  – **1** for images in the `real` folder.

– **Image Preprocessing:** Each image was:

  – Read from disk.

  – Decoded and resized to the standard input size of **224** × **224** pixels (as required by EfficientNetB0).

  – Converted to a floating point format with values in the range [0, 255] and then preprocessed using the EfficientNet-specific preprocessing function (which typically scales values to a range of approximately [-1, 1]).

4 Model Development

4.1 Early Models: Simple CNN Architecture

– **Initial Attempts:** Our first approach employed a simple CNN architecture with a few convolutional and pooling layers.

– **Limitations:** Despite our efforts, the basic model underfit the data (with training accuracy around 65%), indicating insufficient capacity to learn the complex patterns present in the dataset.

4.2 Increasing Model Complexity

– We experimented by adding more convolutional layers, dropout layers for regularization, and fully connected (dense) layers to capture higher-level features. Although these modifications improved performance, the model still did not achieve the desired accuracy.

4.3 Advanced Approach: Transfer Learning with EfficientNetB0

– **Rationale:** To leverage pre-trained features and enhance performance, we adopted a transfer learning approach using EfficientNetB0—a state-of-the-art architecture pre-trained on the ImageNet dataset.

– **Model Architecture:**

  - **Base Model:** We loaded EfficientNetB0 without its top classification layers, enabling us to use its powerful feature extraction capabilities.

  - **Data Augmentation:** To improve generalization and reduce overfitting, data augmentation layers (such as RandomFlip and RandomRotation) were incorporated into the model pipeline.

  - **Classification Head:** After passing the preprocessed images through the base model, global average pooling was applied, followed by a dense layer (with ReLU activation) and dropout (20%) before the final softmax output layer that predicts one of the two classes.

- **Training Details:** The model was compiled with the Adam optimizer (using a learning rate of 1e-4) and trained for 25 epochs. Evaluation on training, validation, and test sets yielded approximately 91% test accuracy and a comparable F1 score.

- **Model Saving:** After training, the model was saved (e.g., as `my_trained_model.h5`) so that it could later be loaded for inference on new test data.

5 Inference and Prediction

In a subsequent phase, we utilized the saved model to generate predictions on a set of test images. The test images were named sequentially (e.g., `1.png`, `2.png`, . . . , `500.png`) and stored in a separate folder. A dedicated prediction script was written to:

- Load each test image and apply the same preprocessing steps used during training.

- Use the loaded model to predict the image's class.

- Map the prediction (with `0` corresponding to *fake* and `1` corresponding to *real*) and output the results in a JSON file format.

An example of the JSON output is as follows:

```
{
    "index": 4,
    "prediction": "real"
},
{
    "index": 5,
    "prediction": "real"
}
```

This JSON file can be used for further evaluation or as the required submission format.

6 Conclusion

Through an iterative process of development and model refinement, we evolved from a basic CNN to a sophisticated transfer-learning model using EfficientNetB0. Key improvements included:

- **Rigorous Data Cleaning:** Correcting mislabeled images ensured the quality of the training data.
- **Advanced Preprocessing:** Standardizing images to 224 × 224 pixels and applying EfficientNet-specific preprocessing.
- **Enhanced Model Architecture:** Leveraging a pre-trained EfficientNetB0 combined with data augmentation and dropout layers resulted in robust performance.
- **Effective Inference Pipeline:** Automating predictions on a new set of test images and outputting results in a structured JSON format.

The final model, achieving approximately 90% test accuracy, demonstrates significant progress over our initial approaches and serves as a strong foundation for further improvements or deployment.