**Fake Image Detection using Deep Learning.**

**Objective:**

The aim of this project is to develop a Convolutional Neural Network (CNN)-based image classification model that can distinguish between **real and fake images.**

**Problem Statement:**

A picture may be worth a thousand words, but today it takes far fewer words than that to prompt a generative AI model to create fake photographs. Photo manipulation has been a concern for years with the rise of advanced editing software, but only recently has the game changed entirely. With the rise of AI-generated content and deep fakes, detecting fake images has become crucial.

### **Dataset:**

The dataset is taken from Kaggle datasets containing both real and fake images. The structure of dataset was like  
 Dataset  
 –Real

–Fake

I have splitted the data into training, validation and testing using train\_test\_split() and divided the data as follows:

* Total Images: 1,890
  + Training Set: 1,632 images (80%)
  + Validation Set: 129 images (10%)
  + Test Set: 129 images (10%)

Later, I assigned binary values to real and fake images for better training.

* Labels: Binary
  + 0: Real image
  + 1: Fake image

### **Technologies Used:**

* Language: Python
* Libraries: TensorFlow, Keras, NumPy, Streamlit, OpenCV, PIL
* Model: Sequential CNN
* IDE: Jupyter Notebook / VS Code
* Deployment: Streamlit

### **Methodology:**

**Step 1: Load Dataset**

Loaded the dataset into the project directory.

**Step 2: Install requirements**

Imported all the required modules and python libraries.

numpy- Used for handling arrays and preprocessing image data efficiently (e.g., converting images to arrays, normalization)

seaborn- Used for visualizing model performance metrics like accuracy/loss with attractive statistical plots.

os- Helps navigate directories, access image files, and manage file paths during data loading.

cv2 *(OpenCV)*- Used for reading, resizing, and manipulating image data in real-time.

tensorflow- Core deep learning library used to build, train, evaluate, and deploy the CNN model.

**Step 3: Data Pre-Processing**

Pre-processed the data by reading it and converting it into array for normalization.

Resized the image into 128x128 height and width.

Labeled the data into Real and Fake [0,1]

**Step 4: Splitting the data**

Splitted the data into train, validation and test using train\_test\_split() function and dividing it into 80:10:10 ratio.

**Step 5: Design the model architecture.**

Designed a Sequential Model having 4 Convolutional Layers and 2 Dense Layers, I tried it with more layers but as the dataset was comparatively small, it led to less model accuracy so I decided to keep the model layers simple and short.

The first layer started with 32 filters and kernel of 3x3.

The number of filters are doubled at every next layer and kernel is kept same.

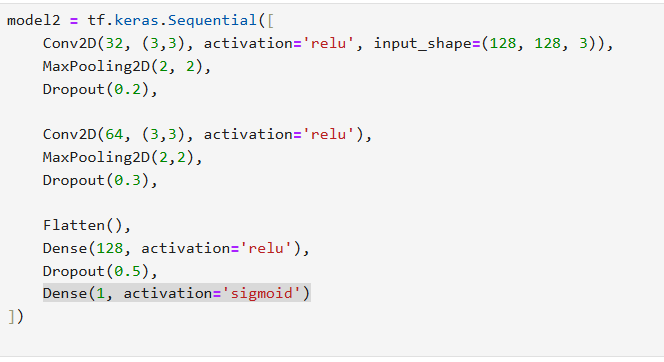
I introduced some Max Pooling Layers after Convolutional Layers to avoid over-fitting and reduce Computational Costs.

The Output from Covolutional Layer is Flattened and passed over to Dense Layers.

Some Dropout Layers were also introduced throught the model to randomly ignore some of the neurons and reduce over-fitting.

We used ReLU activation in all layers except output layer to reduce computation cost and introduce non-linearity.

Finally the Output Layer was constructed containing 1 neuron for binary classification and sigmoid activation function as the data was divided into two groups – real and fake.



**Step 6: Training and evaluation of the Model**

I compiled and trained the model with Optimizer as adam, and binary\_crossentropy for loss.

Saved the best model with most accuracy, and used early stopping to prevent overfitting of the model.

**Step 7: Deployment**

I used Streamlit for deploying the model.

Streamlit is an open-source Python library that lets you quickly build and share interactive web apps for data science and machine learning projects with just a few lines of code.

Users can upload an image and receive a **real/fake** classification

### **Results:**

Accuracy: ~0.98

**Challenges Faced and How I overcame them:**

1. Dataset imbalance and small size initially led to overfitting and so I have to change the model architecture to keep it simple.
2. Initially, I used wrong metric i.e. val\_loss to save the model, this caused the best model not to be saved, later I changed it to val\_accuracy which saved the best and most accurate model.

