**Data Preparation/Feature Engineering**

**1. Overview**

The data-building and feature engineering phase is an important step in the life cycle of the machine learning project, whose goal is to convert the raw data into a format suitable for model training This phase includes several key components:

**Data Cleaning:**  
Ensuring data quality by addressing missing values, outliers, and inconsistencies in the raw data. But actually we have image data which is does not have any missing values.

**Feature Engineering:**  
to develop new features or modify existing features to increase the model’s ability to accurately predict.

**Handling Categorical Data:**Encoding categorical variables into numerical representations to make them more manageable for machine learning models.

**Data Augmentation:**  
To introduce variety to image datasets by applying random transformations such as rotation, flipping, and zoom.

**Data Splitting:**  
Dividing the dataset into training, validation, and test sets.

The importance of this step is to produce sophisticated representative data with well-designed features, which allows machine learning models to efficiently learn and apply to new unseen data.

**2. Data Collection**

The DFDC dataset was created for the Deepfake Detection Challenge organized by Facebook. It is a publicly available dataset hosted on Kaggle and is designed to facilitate research and development in the detection of deepfake videos.

First, we are setting up the Kaggle API credentials in Google Colab and then downloading the dataset for the "deepfake-detection-challenge" competition. The code begins by installing the Kaggle Python library (‘Kaggle’) and uploading the Kaggle API credentials file (Kaggle.json) to the Google Colab environment.  
The next step involves using the Kaggle API to download the dataset for the "deepfake-detection-challenge" competition.  
# !kaggle competitions download -c deepfake-detection-challenge

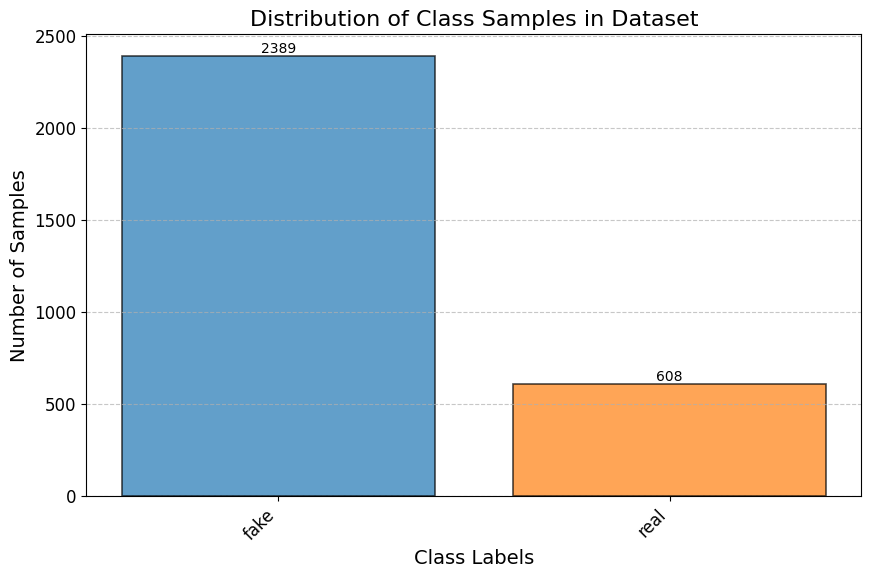
downloaded file is a zip archive, it needs to be unzipped to access the dataset files.  
# !unzip deepfake-detection-challenge.zip

**3. Data Cleaning**

In our dataset there are two directories: Real and Fake. After taking frames from the videos, we have [(0, 2389), (1, 608)] images. 0 indicates fake and 1 indicates real.

**4. Exploratory Data Analysis (EDA)**

The exploratory data analysis (EDA) on the dataset involves visualizing the distribution of class samples.  
Class Distribution Visualization: The bar plot illustrates the distribution of class samples in the dataset.  
Class 0 is represented by blue bars, and Class 1 is represented by orange bars.  
The x-axis corresponds to class labels, and the y-axis represents the number of samples.



The first three samples from both real and fake directories.





**5. Feature Engineering**

Data Augmentation: Applying data augmentation techniques to artificially increase the size of the training dataset. Common augmentations include rotation, flipping, zooming, and slight changes in brightness and contrast.

Breaking down the parameters in my code:  
rescale=1/255.0: Normalizes pixel values to be in the range [0, 1].  
rotation\_range=40: Randomly rotates the image by a value between -40 and 40 degrees.  
width\_shift\_range=0.2: Shifts the image horizontally by a fraction of its width (20% in this case).  
height\_shift\_range=0.2: Shifts the image vertically by a fraction of its height (20% in this case).  
shear\_range=0.2: Applies shearing transformation.  
zoom\_range=0.2: Randomly zooms into the image.  
horizontal\_flip=True: Randomly flips the image horizontally.  
fill\_mode='nearest': Specifies the method to fill in newly created pixels during transformations.  
validation\_split=0.2: Reserves 20% of the data for validation.

The image is resized to (224, 224) pixels before saving.

**6. Data Transformation**

cv2.imwrite(filename, cv2.resize(crop\_img, (224, 224)))

In the above line, cv2.resize(crop\_img, (224, 224)) resizes the cropped face image (crop\_img) to a size of (224, 224) pixels before it is written to the file.

rescale=1/255.0,  
Here, rescale=1/255.0 scales the pixel values to be in the range [0, 1], which is a common normalization technique for image data.

**Model Exploration**

**1. Model Selection**

I am using InceptionResnetv2 pretrained model for detecting deepfakes. InceptionResNetV2 is a powerful deep neural network architecture that exhibits excellent performance in image-related tasks. It is known for its ability to capture shapes and complexity in images, making it suitable for tasks such as facial recognition and deepfake detection. Pretrained models trained on big data (such as ImageNet) have already learned common features that can be useful for various computer vision tasks. InceptionResNetV2 combines the capabilities of the Inception and ResNet frameworks. Inception modules enable efficient feature extraction with different receptive field sizes, while ResNet connections address the vanishing gradient problem during training. This combination often results in improved performance.

**Strengths:**  
**Feature Extraction:** InceptionResNetV2 excels at extracting hierarchical features from images, making it suitable for tasks where understanding complex patterns is crucial.

**Transfer Learning:** The ability to leverage knowledge from pretrained models allows for effective transfer learning, even with limited labeled data.

**Weaknesses:**  
**Computational Intensity:** InceptionResNetV2 is a deep and complex model, which makes it computationally intensive. Training and fine-tuning such models can be resource-intensive.

**Overfitting** **Risk:** Deep models like InceptionResNetV2 can be prone to overfitting, especially when dealing with a small dataset. Proper regularization techniques and data augmentation are essential to mitigate this risk.

**2. Model Training**

The InceptionResNetV2 model is loaded with weights pre-trained on the ImageNet dataset. The model is configured to accept input images of dimensions (X, Y, 3). All layers of the InceptionResNetV2 base model are set to non-trainable to preserve the pre-trained weights.

The model architecture includes a Global Average Pooling 2D layer, a Flatten layer, a Dense layer with 256 neurons and ReLU activation, and a final Dense layer with a sigmoid activation function for binary classification (real or fake).

The model is compiled using the Adam optimizer and binary cross\_entropy loss.   
Early Stopping: Monitors validation accuracy and stops training if there is no improvement for a certain number of epochs (patience set to 5).  
ModelCheckpoint: Saves the model with the best validation accuracy.  
ReduceLROnPlateau: Reduces the learning rate if there is no improvement in validation accuracy for a certain number of epochs.

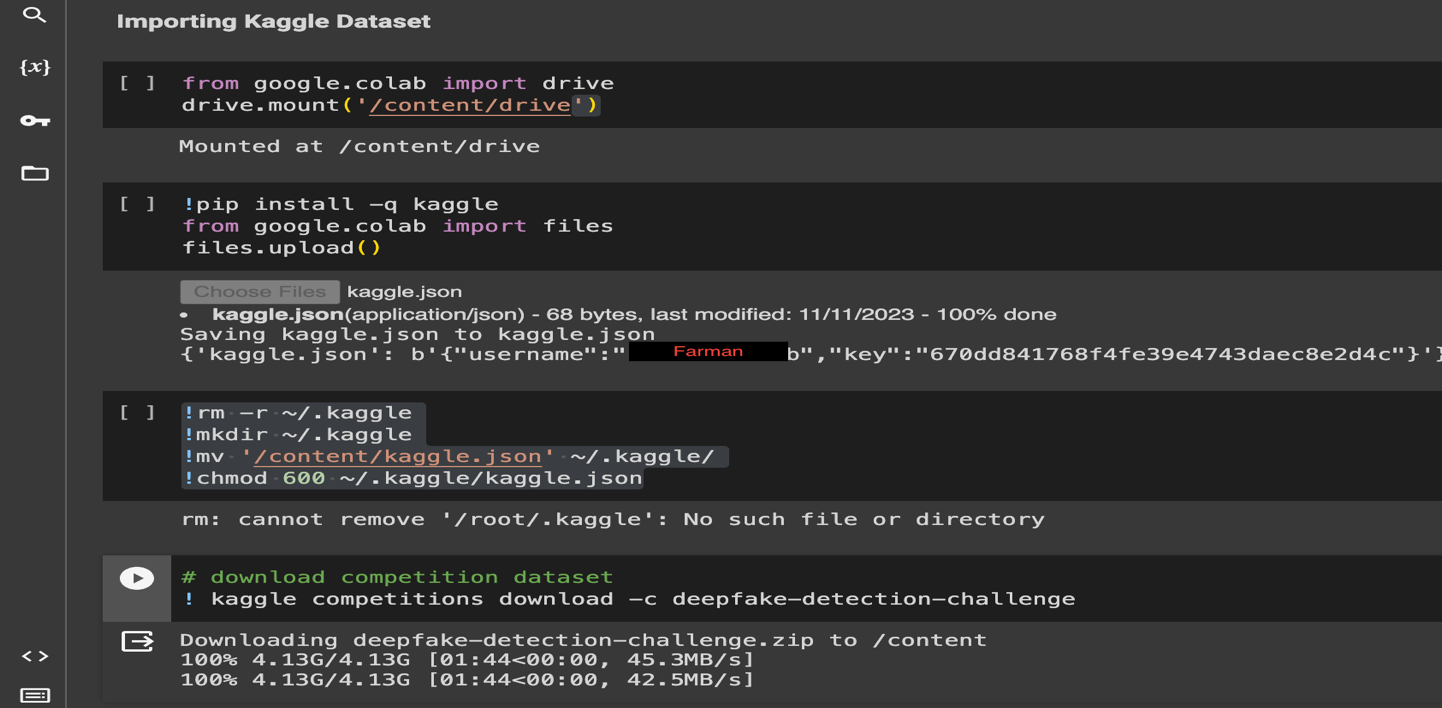
The model is trained for 50 epochs with a specified batch size (BATCH\_SIZE).  
The training process includes callbacks for early stopping, model checkpointing, and learning rate reduction.

**3. Model Evaluation**

The loaded model is evaluated on the test set, and metrics such as test loss and accuracy are extracted. Using scikit-learn metrics, precision, recall, and the confusion matrix are calculated based on the predicted and true labels.

|  |
| --- |
| 6/6 [==============================] - 18s 2s/step - loss: 0.4165 - accuracy: 0.8262  Test Loss: 0.4165 Test Accuracy: 82.62% 6/6 [==============================] - 25s 3s/step Precision: 0.2093 Recall: 0.1192 Confusion Matrix: [[529 68] [133 18]] |

**4. Code Implementation**



EDA part is already explained in **4. Exploratory Data Analysis (EDA).**