Fake Face Detection Using CNN

Real and Fake face detection using CNN:

CNN stands for Convolutional Neural Network, CNNs are often used in deep learning, which involves training the network with large datasets to improve its accuracy and ability to recognize complex patterns in data. It is composed of layers of interconnected nodes, called neurons, that process and extracts features from input data. These filters learn to identify patterns and features within the data, such as edges or textures, which can be used to recognize objects or classify images.

Downloaded the dataset from Kaggle it has 70k real faces and 70k fake faces. Real and fake images were combined and resized all the images into 256px, and split the data into train, validation, and test set. CSV files also included information regarding the path of images, labels, and their image id are present. From this, we have taken 150 real and 150 fake images from the above dataset and created our own CSV files and folders for 300 images. CSV files contain the path of the images, label, and label in string form.

Convolutional Neural Networks (CNNs) can be used to detect fake images with a high degree of accuracy. CNN involves training a convolutional neural network to differentiate between real and fake faces. This can be done by training the network on a large dataset of real and fake faces, with labels indicating which are real and which are fake.

Code explanation:

- Import all the necessary libraries for reading the data and implementing CNN architecture.
- Mounting the Google Drive in the Colab environment by running this command and following the authentication process.
- Setting the path to the dataset directory where the image data is stored.
- creating three datasets using the Keras. The directory argument specifies the path.
- The label_mode argument specifies the type of labels that will be returned, which in this case is binary.
- The image_size argument specifies the dimensions to which the input images will be resized, which in this case is 256x256 pixels.
- The batch_size argument specifies the size of the batches that the data will be divided into during training.

- Define a data augmentation pipeline using the Sequential class and several image transformation layers, including RandomFlip, RandomTranslation, RandomRotation, RandomZoom, and Rescaling.
- This pipeline will be used to generate augmented images during training to help the model generalize better to new images.
- Here we have defined the function: where we compute the recall score, which is the proportion of true positive predictions out of all actual positive cases.
- Computed the precision score, which is the proportion of true positive predictions out of all positive predictions.
- Computes the F1 score, which is the harmonic mean of precision and recall. It does this by calling the *get_precision()* and *get_recall()* functions and using their values to compute the F1 score.
- Define a convolutional neural network (CNN) model using the Keras API for image classification tasks.
- The CNN architecture consists of four convolutional layers with max pooling, followed by two fully connected layers with batch normalization and ReLU activation, and a sigmoid activation output layer.
- The model takes an input shape of (256, 256) and uses the He uniform initialization for the kernel weights and zero initialization for the biases. The model summary shows the architecture and the number of parameters for each layer.
- Compile the model using the Adam optimizer, binary cross-entropy loss function, and several metrics including accuracy, get_f1, get_precision, and get_recall.
- Train the model using the fit method, passing in the training and validation data, number of epochs, and other parameters. During training, the model will generate augmented images using the data augmentation pipeline defined earlier.
- After training, evaluate the model using the test data, which was not used during training. The test results include the test loss, accuracy, recall, precision, and F1 score.
- Defines a function ResNet() that creates a new model by adding some layers on top of a pre-trained ResNet50 model. The pre-trained model is loaded from the applications module of Keras.
- Performing hyperparameter tuning for a convolutional neural network (CNN) model using the Keras library.
- The hyperparameters include the learning rate, batch size, number of epochs, optimizer, and regularizer.
- Using nested for loop to iterate through all possible combinations of optimizers and regularizers. For each combination, it builds a CNN model using the CNNmodel function with an input shape of (256, 256).

- Then compile the model with the current optimizer and binary cross-entropy loss function.
- The compiled model is then trained using the fit() function with the specified batch size, number of epochs, and validation data.
- The EarlyStopping callback is also used to stop training if the validation loss does not improve for 5 epochs.
- After training, evaluate the model using the test data and prints the loss and accuracy.

Result:

1.

- Accuracy: 50.0
- Recall: 100.0
- F1_score: 66.042

2.

- Loss: 1.2607940435409546
- Accuracy: 0.5444444417953491

Conclusion:

A Convolutional Neural Network (CNN) model was developed to detect fake images that were generated using Generative Adversarial Networks (GANs). The model was trained using a dataset and its accuracy, recall, and f1_score was printed to evaluate its performance. Hyperparameters were tuned for the model and it was observed that the accuracy was higher than that of the normal CNN model.