DeepFake Image Classification Report

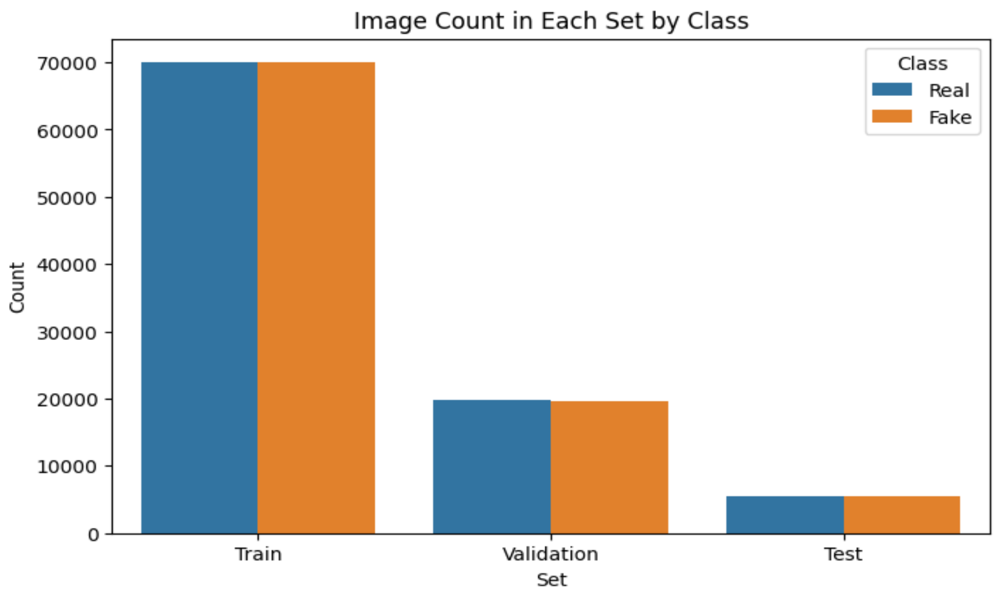
Team Members:
Hemanth Kumar Bola, Chandini Marrapu, Harika Devi Alla, Deepika Panjala

1. Introduction

Deepfakes are synthetic media generated using artificial intelligence, particularly deep learning techniques. They create highly realistic images, videos, and audio clips that can make individuals appear to say or do things they never actually did. While such technology can have creative and entertainment-related applications, it poses a growing threat to society by enabling the spread of misinformation, identity fraud, and political manipulation. To address this issue, our project aims to develop a Convolutional Neural Network (CNN) that can accurately classify facial images as either real or fake. The classifier is trained on a large dataset of labeled images and is designed to learn subtle patterns that distinguish deepfakes from genuine visuals, thereby contributing to digital media forensics and online safety.

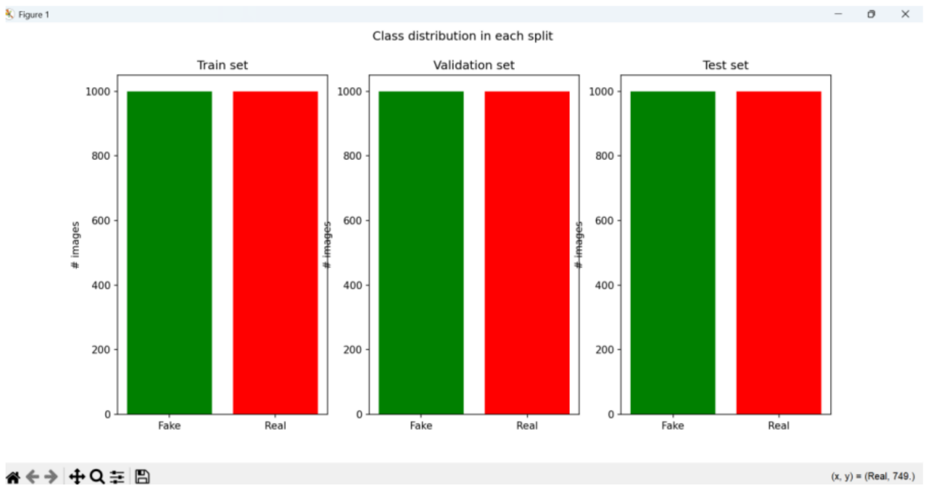
2. Dataset Description

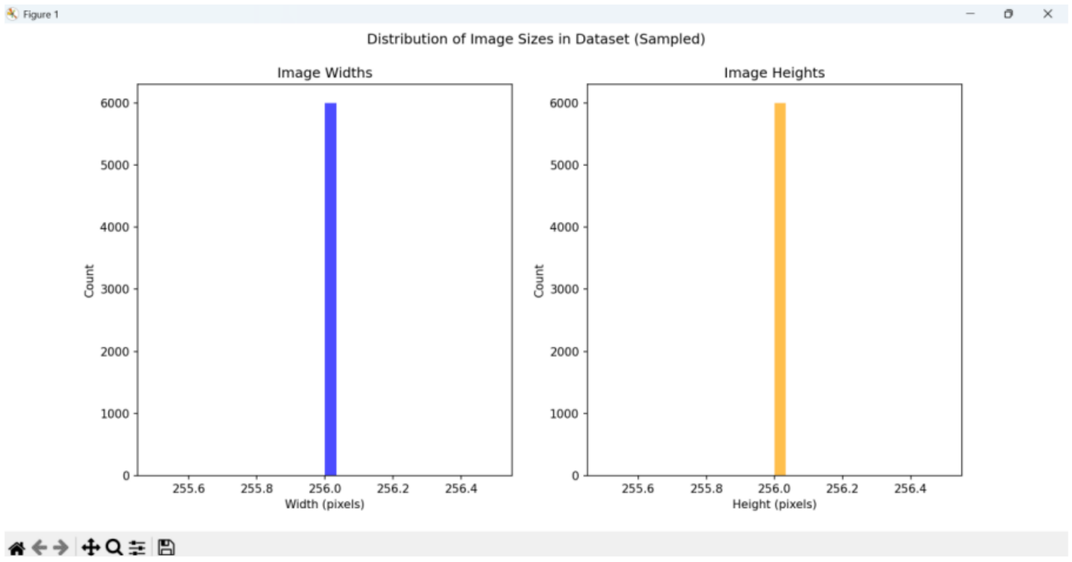
The dataset used in this project is titled 'DeepFake and Real Images' and is publicly available on Kaggle. It was created by Manjil Karki to support the development of machine learning models capable of detecting manipulated facial images. The dataset contains two classes: Real and Fake. Real images depict actual human faces, while Fake images are AI-generated deepfakes. The dataset comprises approximately 190,000 images, which are divided into training (140,002 images), validation (39,428 images), and test (10,905 images) sets. This distribution allows for robust model training and evaluation. The images are stored in JPEG format and are well-suited for use in computer vision tasks. The dataset was loaded into the project using Keras' ImageDataGenerator utility, which facilitates real-time image augmentation and directory-based data labeling.



3. Exploratory Data Analysis & Preprocessing

In the data exploration step, we first traversed the Train, Validation, and Test directories to count how many real and fake images are in each split and printed those counts. We then visualized the class balance by plotting side-by-side bar charts for each split, showing the number of real versus fake images. To get a sense of the raw data, we displayed one example image from each class in the training set. Finally, we collected up to 1,000 image dimensions per class across all splits, identified and printed the ten most common width×height combinations, and plotted histograms of image widths and heights to reveal the overall distribution of image sizes in the dataset.





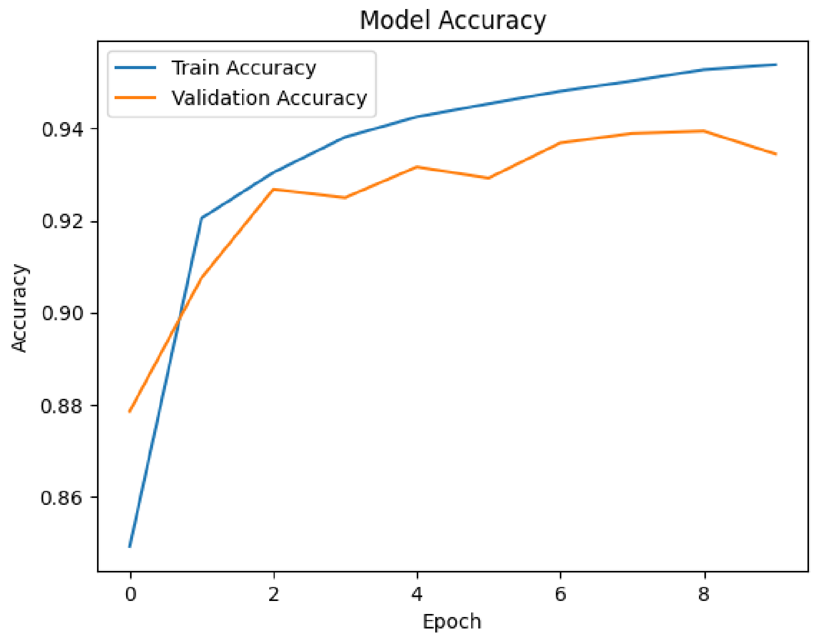
4. CNN Model Architecture

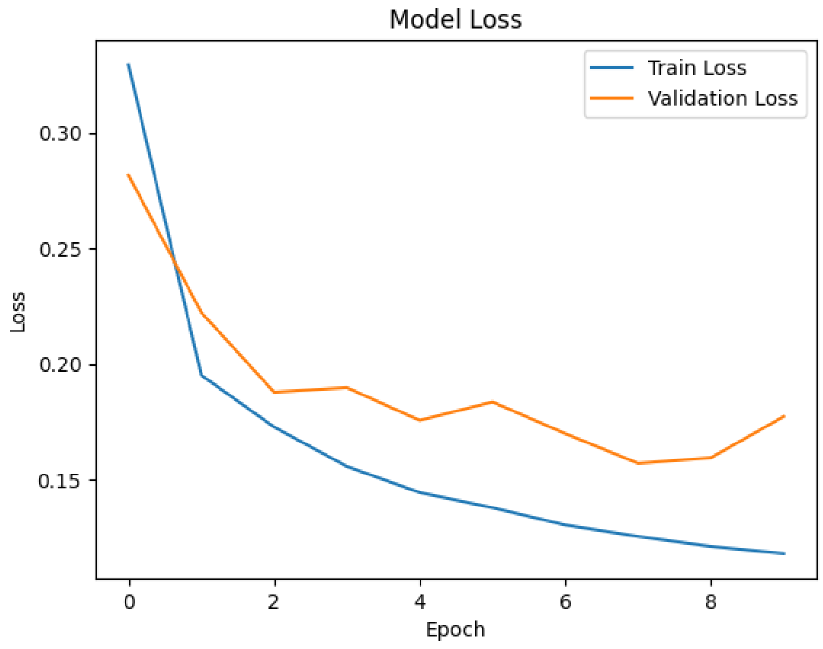
The model architecture was implemented using Keras' Sequential API, a linear stack of layers that allows data to flow in a single path from input to output. This approach is particularly useful for image classification tasks where a straightforward, layer-by-layer structure simplifies the implementation and debugging process. We employed a Convolutional Neural Network (CNN) because of its inherent ability to automatically and adaptively learn spatial hierarchies of features from input images. CNNs use filters to extract low-level features like edges and shapes in initial layers and progressively learn more abstract features in deeper layers.

In our model, the sequence starts with two convolutional layers using ReLU activation functions, followed by max pooling layers to reduce spatial dimensions. After these, a flattening layer transforms the 2D outputs into a 1D vector, which is then passed to a fully connected dense layer with ReLU. The final layer is a single-neuron output with sigmoid activation for binary classification. The simplicity of the sequential model, combined with the feature-learning capability of CNNs, makes this architecture ideal for the deepfake detection task. It offers both computational efficiency and high classification accuracy, serving as a strong foundation for potential future expansion into more complex or pretrained architectures.

4.1. Training & Evaluation

The model was trained for 10 epochs using the Adam optimizer, which adapts the learning rate for each parameter, and binary crossentropy loss, which is suitable for binary classification tasks. Training involved feeding batches of images from the training set, while validation data was used to monitor model performance after each epoch. The training accuracy improved steadily over time, reaching approximately 95%, while the validation accuracy peaked at around 94%, indicating effective learning without significant overfitting. Simultaneously, the training loss declined continuously, and the validation loss remained stable throughout the training, confirming that the model generalized well to unseen data. The accuracy and loss trends were visualized using line plots to confirm the model's convergence.



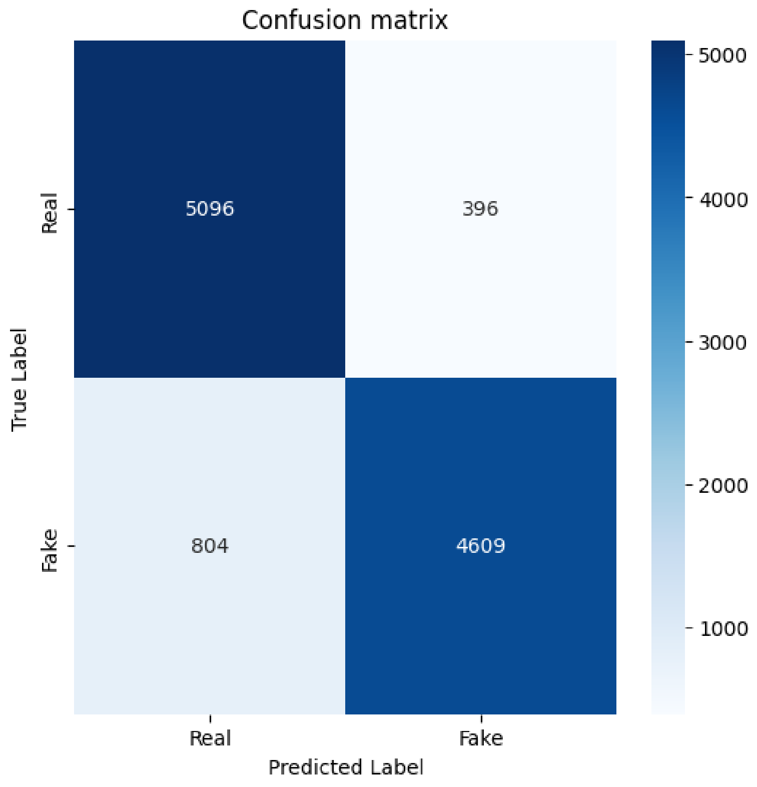


4.2. Results

After training, the model was evaluated on the test dataset containing 10,905 images. It achieved a final accuracy of 89.00% with a loss value of 0.1823. Further analysis using classification metrics showed that for real images, the model achieved a precision of 0.86, recall of 0.93, and an F1-score of 0.89. For fake images, the precision was 0.92, recall was 0.85, and F1-score was 0.88. The macro and weighted averages for all metrics stood at 0.89. These results indicate that the model performs slightly better in identifying fake images, while being more sensitive in detecting real images, offering a balanced performance ideal for practical applications.

4.3. Confusion Matrix Analysis

The confusion matrix provided further insights into model behavior. Out of the 10,905 test images, 4609 fake images were correctly identified as fake (True Positives), and 5096 real images were correctly classified as real (True Negatives). There were 396 false positives, where real images were incorrectly labeled as fake, and 804 false negatives, where fake images were mislabeled as real. This distribution suggests the model adopts a conservative classification approach, slightly favoring the real class to avoid false alarms, which is preferable in many real-world detection scenarios.

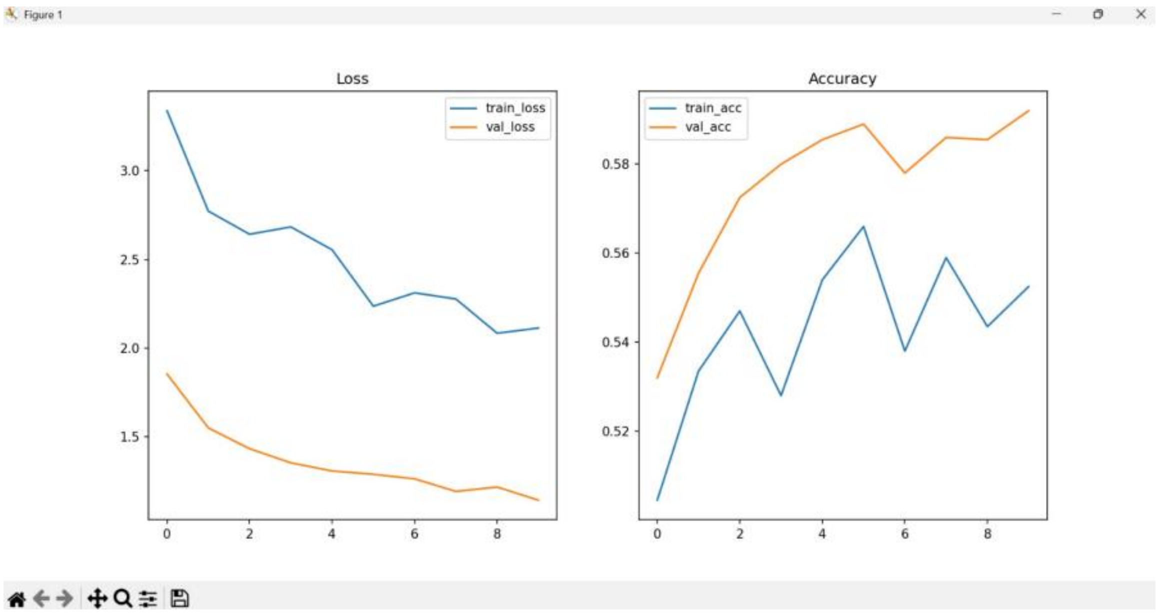


5. Xception model

The Xception-based model leverages transfer learning by importing the pre-trained Xception convolutional base (with ImageNet weights) and excluding its fully connected head. All layers of this base are frozen (base.trainable = False) to retain previously learned feature representations. An input tensor of shape (128 × 128 × 3) is passed through the Xception base in inference mode, followed by a global average pooling layer to collapse spatial dimensions into a single feature vector. A dropout layer with a 30 % rate regularizes the model by randomly deactivating neurons during training, and a final dense layer with a sigmoid activation outputs the probability of an image being fake. The model is compiled with the Adam optimizer (learning rate = 1e-4), binary cross-entropy loss, and accuracy as the primary metric.

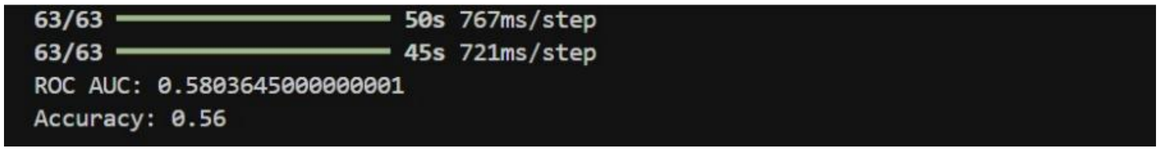
5.1. Training and the evaluation

Training begins by fitting the Xception model on the train_ds dataset while validating on val_ds for up to 10 epochs. Two callbacks ensure efficient training: an early stopping callback (patience = 3) halts training when validation loss stops improving, and a model checkpoint callback saves only the best model weights to best_model_xception.h5. Verbose logging provides epoch-by-epoch loss and accuracy. After training, the model's history can be visualized by plotting training versus validation loss and accuracy curves, highlighting convergence behavior and potential over- or under-fitting .



5.2. Results

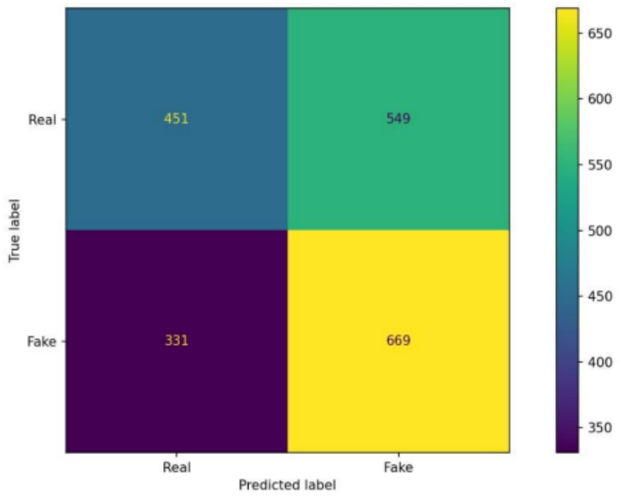
Upon completion of training, performance metrics—including final validation accuracy and loss are recorded. The model's effectiveness is further quantified on the held-out test set: predictions are generated as probabilities, and an ROC AUC score measures discrimination capability. Accuracy is computed by thresholding probabilities at 0.5, yielding the overall proportion of correctly classified images. These metrics, alongside those from other architectures (e.g., the simple CNN and ResNet50 variants), are collated into a summary table to facilitate direct comparison of model strengths and weaknesses.



```
63/63 ━━━━━━━━━━━━━━━━━━━━ 50s 767ms/step
63/63 ━━━━━━━━━━━━━━━━━━━━ 45s 721ms/step
ROC AUC: 0.5803645000000001
Accuracy: 0.56
```

5.3. Confusion Matrix Analysis

To gain deeper insight into classification errors, the test-set predictions (y_pred) and true labels (y_true) feed into a confusion matrix. This matrix delineates true positives (correctly detected fakes), true negatives (correctly detected reals), false positives (real images misclassified as fake), and false negatives (fake images missed). A ConfusionMatrixDisplay then visualizes these counts, enabling identification of asymmetric error patterns—such as if the model is biased toward marking too many reals as fakes or vice versa. Such analysis guides threshold adjustments or data augmentation strategies to balance sensitivity and specificity.

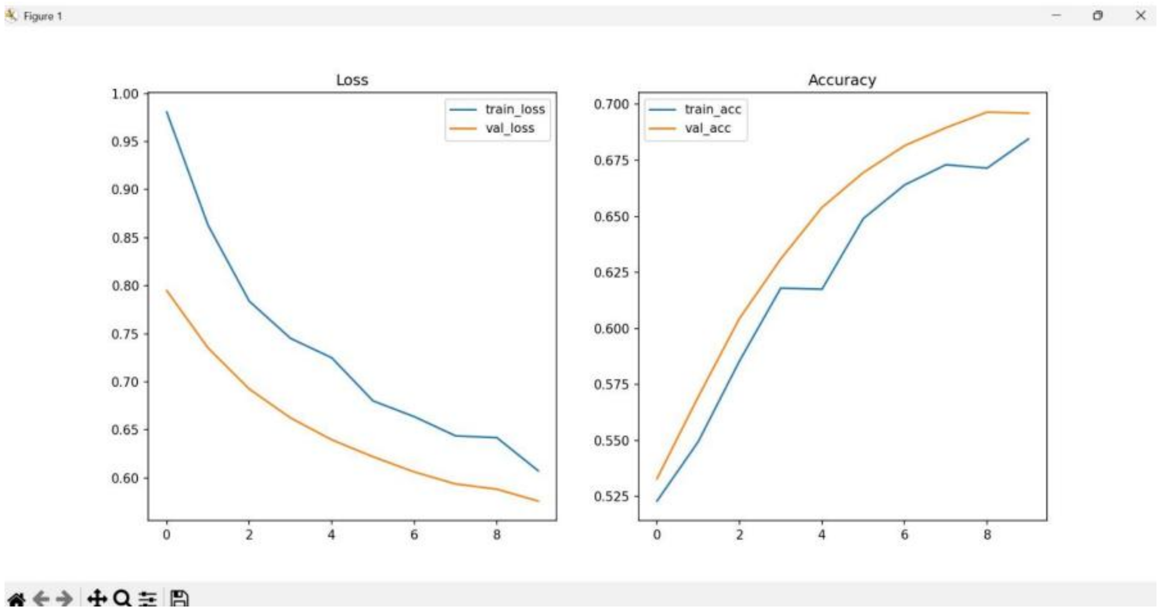


6. Resnet Model

The ResNet50-based classifier uses transfer learning by loading the ResNet50 convolutional backbone with ImageNet weights (excluding its top layers) and freezing all its parameters (base.trainable = False). Input images of size 128 × 128 × 3 pass through this backbone, after which a global average pooling layer reduces the spatial feature maps to a single vector. A dropout layer (rate = 0.3) follows to mitigate overfitting, and a final dense layer with a sigmoid activation produces the probability of an image being fake. The model is compiled with the Adam optimizer (learning rate = 1e-4), binary cross-entropy loss, and accuracy as the evaluation metric .

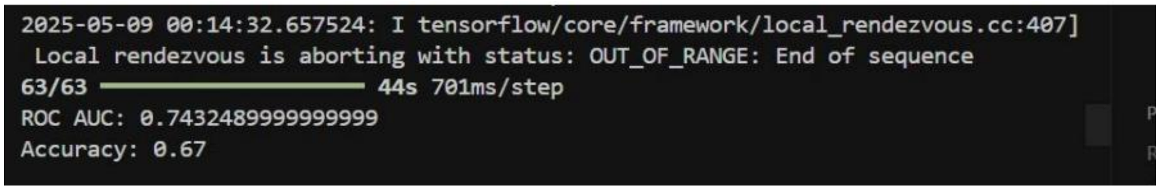
6.1. Training and the Evaluation

Training is conducted over up to 10 epochs on the train_ds dataset, with validation performed on val_ds. Two callbacks facilitate efficient learning: an EarlyStopping callback halts training if the validation loss fails to improve for three consecutive epochs, and a ModelCheckpoint callback saves the best-performing weights to best_model_resnet.h5. Training progress (loss and accuracy for both train and validation sets) is logged epoch by epoch, and can be visualized via loss/accuracy curves to assess convergence and detect under- or over-fitting .



6.2. Results

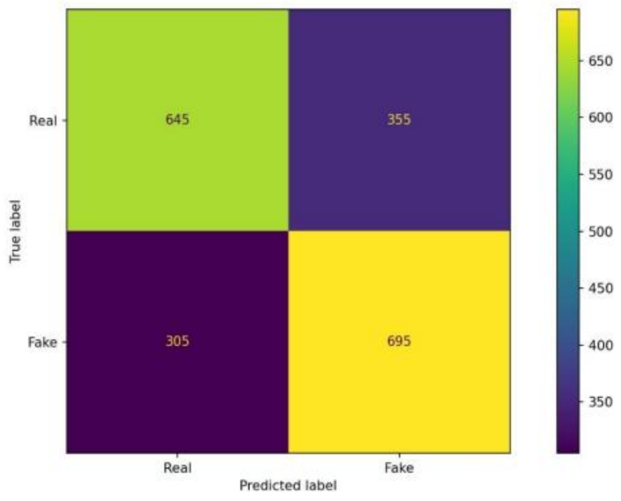
Upon completion of training, the ResNet50 model's final validation accuracy and loss are recorded. On the held-out test set, predictions are generated as probabilities and evaluated using ROC AUC to measure discriminative power, while overall accuracy is computed by thresholding probabilities at 0.5. These metrics are then compared alongside those from the simple CNN and Xception models in a summary table, which highlights the ResNet50 variant's relative strengths in classification performance.



```
2025-05-09 00:14:32.657524: I tensorflow/core/framework/local_rendezvous.cc:407]
 Local rendezvous is aborting with status: OUT_OF_RANGE: End of sequence
63/63 ━━━━━━━━━━━━━━━━━━━━ 44s 701ms/step
ROC AUC: 0.7432489999999999
Accuracy: 0.67
```

6.3. Confusion Matrix Analysis

To understand classification behavior in detail, the true labels (y_true) and predicted labels (y_pred, obtained by thresholding predicted probabilities) feed into a confusion matrix. A ConfusionMatrixDisplay visualizes counts of true negatives, true positives, false negatives, and false positives. By examining this matrix, one can identify whether the ResNet50 model tends to over-predict fakes (false positives) or miss actual fakes (false negatives), informing decisions on threshold adjustment or further data augmentation to balance sensitivity versus specificity.



6.3. Challenges and Improvements

While our transfer-learning approaches achieved strong overall accuracy, several challenges emerged that point to avenues for improvement. First, freezing the Xception and ResNet50 backbones preserved pretrained features but limited the models' ability to adapt to subtle deepfake artifacts in our specific dataset, contributing to the 396 false positives and 804 false negatives observed in the CNN's confusion matrix (real-vs-fake bias) . Second, using a uniform 128 × 128 input resolution and modest data augmentation may have constrained the models' capacity to learn fine-grained spatial cues, especially in high-quality fakes. Third, early stopping after just three stagnant epochs, while preventing overfitting, risks halting before the networks fully converge on difficult cases. Finally, evaluation solely at a 0.5 probability threshold overlooks opportunities to optimize sensitivity-specificity trade-offs for real-world deployment.

To address these issues, we recommend fine-tuning upper layers of the pretrained backbones unfreezing a subset of deeper blocks to better capture domain-specific features and experimenting with larger input sizes (e.g., 224 × 224) alongside stronger augmentation (color jitter, Gaussian blur). Extending patience in the early-stopping callback and incorporating learning-rate scheduling can foster deeper convergence, while threshold calibration (e.g., via ROC curve analysis) could rebalance false positive/negative rates. Finally, exploring ensemble methods combining CNN, Xception, and ResNet predictions and leveraging advanced architectures such as EfficientNet or vision transformers may further boost detection robustness.