```
!pip install kaggle

from google.colab import files
files.upload()

!mkdir -p ~/.kaggle
!cp kaggle.json ~/.kaggle/
!chmod 600 ~/.kaggle/kaggle.json

!kaggle datasets download -d manjilkarki/deepfake-and-real-images

!unzip -o deepfake-and-real-images.zip
```

Show hidden output

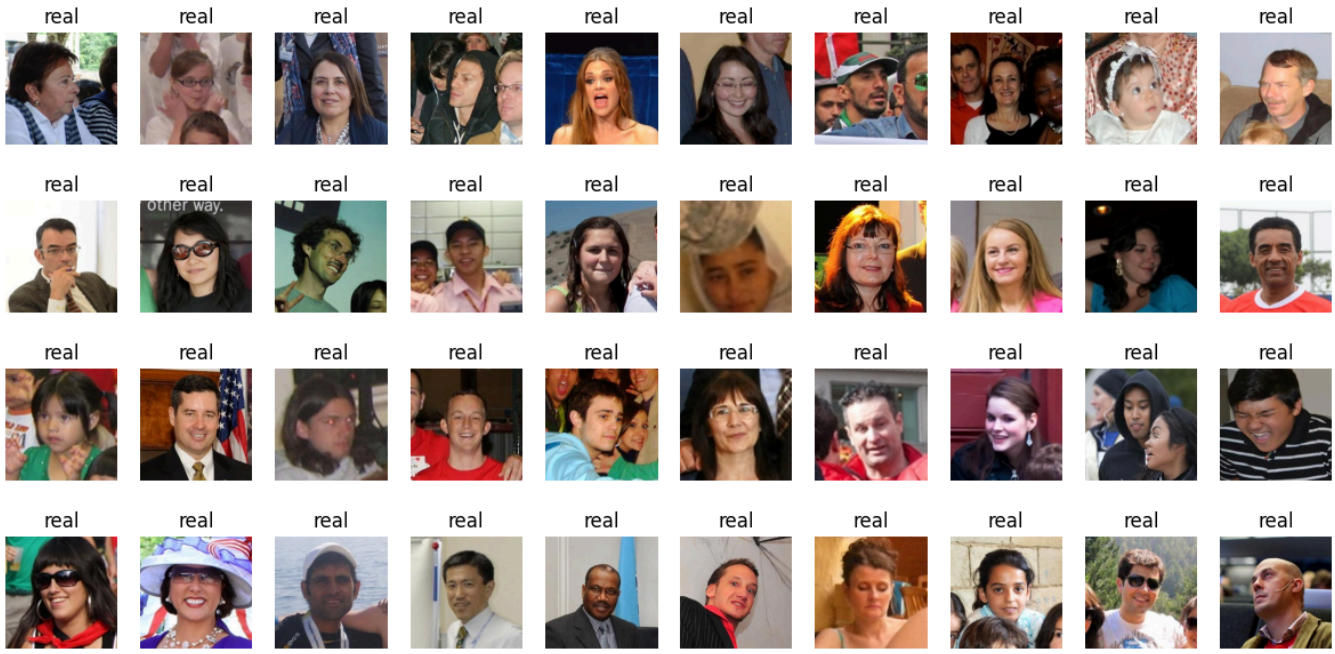
```
import os.path
import cv2
import numpy as np
import pandas as pd
import matplotlib.pyplot as plt
import tensorflow as tf
import seaborn as sns
from pathlib import Path
from tqdm import tqdm
from time import perf_counter

from sklearn.model_selection import train_test_split
from sklearn.metrics import classification_report,accuracy_score
from IPython.display import Markdown, display
```

```
from pathlib import Path
import pandas as pd
import os

def image_to_df(filepaths):
  # Extract the labels from the filepaths
  labels = [str(filepath).split(".")[-2].split("/")[-1].split("_")[0] for filepath in filepaths]

  # Create a pandas Series for the filepaths and labels
  filepath = pd.Series(filepaths, name='Data').astype(str)
  labels = pd.Series(labels, name='Label')

  # Combine the Series into a DataFrame
  df = pd.concat([filepath, labels], axis=1)

  return df

# Set the data path
data_path = "/content/Dataset"

# Get the paths to the Train, Test, and Validation directories
train_path = os.path.join(data_path, "Train")
test_path = os.path.join(data_path, "Test")
val_path = os.path.join(data_path, "Validation")

# Get the paths to the Real and Fake directories within each of the above directories
train_real_path = os.path.join(train_path, "Real")
train_fake_path = os.path.join(train_path, "Fake")
test_real_path = os.path.join(test_path, "Real")
test_fake_path = os.path.join(test_path, "Fake")
val_real_path = os.path.join(val_path, "Real")
val_fake_path = os.path.join(val_path, "Fake")

# Get the filepaths for all the images in each of the Real and Fake directories
train_real_filepaths = list(Path(train_real_path).glob(r'**/*.jpg'))
train_fake_filepaths = list(Path(train_fake_path).glob(r'**/*.jpg'))
test_real_filepaths = list(Path(test_real_path).glob(r'**/*.jpg'))
test_fake_filepaths = list(Path(test_fake_path).glob(r'**/*.jpg'))
val_real_filepaths = list(Path(val_real_path).glob(r'**/*.jpg'))
val_fake_filepaths = list(Path(val_fake_path).glob(r'**/*.jpg'))

# Combine the filepaths into a single list
filepaths = train_real_filepaths + train_fake_filepaths + test_real_filepaths + test_fake_filepaths + val_real_filepaths + val_fake_file

df = image_to_df(filepaths)
```

```
print(f'Number of training pictures : {df.shape[0]}\n')
print(f'Number of training labels : {df.shape[1]}\n')
```

```
Number of training pictures : 190335

Number of training labels : 2
```

```
import matplotlib.pyplot as plt

fig, axes = plt.subplots(nrows=4, ncols=10, figsize=(15, 7),
                         subplot_kw={'xticks': [], 'yticks': []})

for i, ax in enumerate(axes.flat):
  ax.imshow(plt.imread(df['Data'].iloc[i]))
  ax.set_title(df['Label'].iloc[i])
  ax.axis('off')

plt.subplots_adjust(hspace=0.5)
plt.show()
```




```
from sklearn.model_selection import train_test_split

train_df, test_df = train_test_split(df, test_size=0.1, random_state=0)
train_df.shape, test_df.shape
```

```
((171301, 2), (19034, 2))
```

```
def create_gen():
  train_generator = tf.keras.preprocessing.image.ImageDataGenerator(
      preprocessing_function=tf.keras.applications.inception_v3.preprocess_input,
      validation_split=0.2
  )

  test_generator = tf.keras.preprocessing.image.ImageDataGenerator(
      preprocessing_function=tf.keras.applications.inception_v3.preprocess_input)

  train_image = train_generator.flow_from_dataframe(
      dataframe=train_df, x_col="Data", y_col="Label", target_size=(224,224), class_mode='binary',
      rescale=1./255, vertical_flip=True, rotation_range=45, width_shift_range=0.2, height_shift_range=0.2,
      fill_mode="nearest", subset='training', batch_size=32, shuffle=True)

  val_image = train_generator.flow_from_dataframe(
      dataframe=train_df, x_col="Data", y_col="Label", target_size=(224,224), class_mode='binary',
      rescale=1./255, vertical_flip=True, rotation_range=45, width_shift_range=0.2, height_shift_range=0.2,
      fill_mode="nearest", subset='validation', batch_size=32, shuffle=False
  )

  test_image = test_generator.flow_from_dataframe(
      dataframe=test_df, x_col='Data', y_col='Label', target_size=(224,224), class_mode='binary',
      shuffle=False, batch_size=32
  )
```

```
  return train_generator, test_generator, train_image, val_image, test_image

train_generator,test_generator,train_images,val_images,test_images = create_gen()
```

```
Found 137041 validated image filenames belonging to 2 classes.
Found 34260 validated image filenames belonging to 2 classes.
Found 19034 validated image filenames belonging to 2 classes.
```

```
import tensorflow as tf

def get_cnn_model():
    # Define the input shape
    inputs = tf.keras.Input(shape=(224, 224, 3))

    # First Convolutional Block
    x = tf.keras.layers.Conv2D(32, (3, 3), activation='relu', padding='same')(inputs)
    x = tf.keras.layers.MaxPooling2D((2, 2))(x)

    # Second Convolutional Block
    x = tf.keras.layers.Conv2D(64, (3, 3), activation='relu', padding='same')(x)
    x = tf.keras.layers.MaxPooling2D((2, 2))(x)

    # Third Convolutional Block
    x = tf.keras.layers.Conv2D(128, (3, 3), activation='relu', padding='same')(x)
    x = tf.keras.layers.MaxPooling2D((2, 2))(x)

    # Flatten and Dense Layers
    x = tf.keras.layers.Flatten()(x)
    x = tf.keras.layers.Dense(64, activation='relu')(x)
    x = tf.keras.layers.Dense(64, activation='relu')(x)

    # Output Layer
    outputs = tf.keras.layers.Dense(1, activation='sigmoid')(x)

    # Build the Model
    model = tf.keras.Model(inputs=inputs, outputs=outputs)

    # Compile the Model
    model.compile(optimizer=tf.keras.optimizers.Adam(learning_rate=0.001),
                  loss='binary_crossentropy',
                  metrics=['accuracy'])
    return model

# Create and compile the CNN model
cnn_model = get_cnn_model()
```

```
import tensorflow as tf
from tensorflow.keras.applications import MobileNetV2
from tensorflow.keras.callbacks import EarlyStopping, ReduceLROnPlateau

# Further reduce image size and batch size for speed
IMG_SIZE = 64  # Smaller image size to improve speed
BATCH_SIZE = 128  # Larger batch size if memory allows

# Enable mixed precision if using a compatible GPU
tf.keras.mixed_precision.set_global_policy("mixed_float16")

# Load a lightweight model (MobileNetV2) with pre-trained weights and freeze layers
base_model = MobileNetV2(input_shape=(IMG_SIZE, IMG_SIZE, 3), include_top=False, weights="imagenet")
base_model.trainable = False

# Add custom layers on top of the base model
inputs = tf.keras.Input(shape=(IMG_SIZE, IMG_SIZE, 3))
x = base_model(inputs, training=False)
x = tf.keras.layers.GlobalAveragePooling2D()(x)
x = tf.keras.layers.Dense(64, activation="relu")(x)
outputs = tf.keras.layers.Dense(1, activation="sigmoid", dtype=tf.float32)(x)  # Set dtype for mixed precision compatibility
cnn_model = tf.keras.Model(inputs, outputs)

# Compile the model with a reduced learning rate
cnn_model.compile(optimizer=tf.keras.optimizers.Adam(learning_rate=0.0005),
                  loss='binary_crossentropy',
                  metrics=['accuracy'])

# Update the data generator with resizing
def create_gen():
    train_generator = tf.keras.preprocessing.image.ImageDataGenerator(
        preprocessing_function=tf.keras.applications.mobilenet_v2.preprocess_input,
        validation_split=0.2,
        horizontal_flip=True,  # Optional data augmentation
```

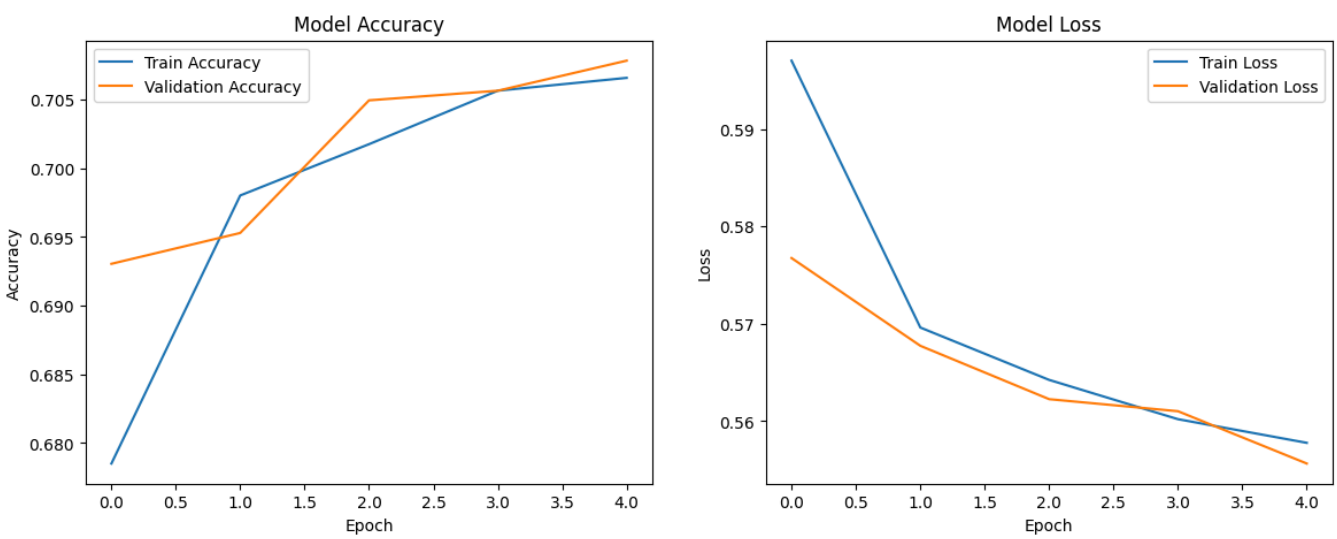
```
        rotation_range=30,  # Optional data augmentation
        zoom_range=0.2,  # Optional data augmentation
        width_shift_range=0.2,  # Optional data augmentation
        height_shift_range=0.2  # Optional data augmentation
    )

    test_generator = tf.keras.preprocessing.image.ImageDataGenerator(
        preprocessing_function=tf.keras.applications.mobilenet_v2.preprocess_input
    )

    train_image = train_generator.flow_from_dataframe(
        dataframe=train_df,
        x_col="Data",
        y_col="Label",
        target_size=(IMG_SIZE, IMG_SIZE),  # Resize images here
        class_mode='binary',
        batch_size=BATCH_SIZE,
        shuffle=True,
        subset='training'
    )

    val_image = train_generator.flow_from_dataframe(
        dataframe=train_df,
        x_col="Data",
        y_col="Label",
        target_size=(IMG_SIZE, IMG_SIZE),  # Resize images here
        class_mode='binary',
        batch_size=BATCH_SIZE,
        shuffle=False,
        subset='validation'
    )

    test_image = test_generator.flow_from_dataframe(
        dataframe=test_df,
        x_col='Data',
        y_col='Label',
        target_size=(IMG_SIZE, IMG_SIZE),  # Resize images here
        class_mode='binary',
        shuffle=False,
        batch_size=BATCH_SIZE
    )

    return train_image, val_image, test_image

# Create data generators
train_images, val_images, test_images = create_gen()

# Add callbacks for adaptive learning and early stopping
early_stopping = EarlyStopping(monitor='val_loss', patience=2, restore_best_weights=True)
reduce_lr = ReduceLROnPlateau(monitor='val_loss', factor=0.5, patience=1, min_lr=1e-6)

# Train the model with fewer epochs
epochs = 5

# Start training
history = cnn_model.fit(
    train_images,
    validation_data=val_images,
    epochs=epochs,
    callbacks=[early_stopping, reduce_lr],
    verbose=1
)
```

```
<ipython-input-21-9599d02b766f>:13: UserWarning: `input_shape` is undefined or non-square, or `rows` is not in [96, 128, 160, 192, 2
  base_model = MobileNetV2(input_shape=(IMG_SIZE, IMG_SIZE, 3), include_top=False, weights="imagenet")
Found 137041 validated image filenames belonging to 2 classes.
Found 34260 validated image filenames belonging to 2 classes.
Found 19034 validated image filenames belonging to 2 classes.
Epoch 1/5
/usr/local/lib/python3.10/dist-packages/keras/src/trainers/data_adapters/py_dataset_adapter.py:121: UserWarning: Your `PyDataset` cl
  self._warn_if_super_not_called()
1071/1071 ━━━━━━━━━━━━━━━━━━━━ 3111s 3s/step - accuracy: 0.6526 - loss: 0.6367 - val_accuracy: 0.6931 - val_loss: 0.5768 - learning_
Epoch 2/5
1071/1071 ━━━━━━━━━━━━━━━━━━━━ 3153s 3s/step - accuracy: 0.6956 - loss: 0.5719 - val_accuracy: 0.6953 - val_loss: 0.5677 - learning_
Epoch 3/5
1071/1071 ━━━━━━━━━━━━━━━━━━━━ 3150s 3s/step - accuracy: 0.7011 - loss: 0.5651 - val_accuracy: 0.7050 - val_loss: 0.5623 - learning_
Epoch 4/5
1071/1071 ━━━━━━━━━━━━━━━━━━━━ 3119s 3s/step - accuracy: 0.7045 - loss: 0.5613 - val_accuracy: 0.7057 - val_loss: 0.5610 - learning_
Epoch 5/5
1071/1071 ━━━━━━━━━━━━━━━━━━━━ 3096s 3s/step - accuracy: 0.7066 - loss: 0.5578 - val_accuracy: 0.7079 - val_loss: 0.5556 - learning_
```

```
# Evaluate on the test set
test_loss, test_accuracy = cnn_model.evaluate(test_images)
print(f'Test Loss: {test_loss:.4f}')
print(f'Test Accuracy: {test_accuracy:.4f}')
```

```
149/149 ━━━━━━━━━━━━━━━━━━━━ 330s 2s/step - accuracy: 0.6630 - loss: 0.6278
Test Loss: 0.6280
Test Accuracy: 0.6623
```

```
import matplotlib.pyplot as plt

def plot_history(history):
    # Plot training & validation accuracy values
    plt.figure(figsize=(14, 5))

    # Accuracy
    plt.subplot(1, 2, 1)
    plt.plot(history.history['accuracy'], label='Train Accuracy')
    plt.plot(history.history['val_accuracy'], label='Validation Accuracy')
    plt.title('Model Accuracy')
    plt.xlabel('Epoch')
    plt.ylabel('Accuracy')
    plt.legend()

    # Loss
    plt.subplot(1, 2, 2)
    plt.plot(history.history['loss'], label='Train Loss')
    plt.plot(history.history['val_loss'], label='Validation Loss')
    plt.title('Model Loss')
    plt.xlabel('Epoch')
    plt.ylabel('Loss')
    plt.legend()

    plt.show()

plot_history(history)
```




```
from sklearn.metrics import confusion_matrix, classification_report
import numpy as np

# Get the predicted labels
Y_pred = cnn_model.predict(test_images)
y_pred = np.where(Y_pred > 0.5, 1, 0)  # Convert probabilities to binary predictions

# Get true labels
y_true = test_images.labels

# Confusion Matrix
conf_matrix = confusion_matrix(y_true, y_pred)
print("Confusion Matrix:\n", conf_matrix)

# Classification Report
print("Classification Report:\n", classification_report(y_true, y_pred))
```

```
149/149 ━━━━━━━━━━━━━━━━━━━━ 329s 2s/step
Confusion Matrix:
```

```
[[8596 1077]
 [5350 4011]]
Classification Report:
              precision    recall  f1-score   support

           0       0.62      0.89      0.73      9673
           1       0.79      0.43      0.56      9361

    accuracy                           0.66     19034
   macro avg       0.70      0.66      0.64     19034
weighted avg       0.70      0.66      0.64     19034
```

```
cnn_model.save('cnn_deepfake_detector.h5')
```

```
WARNING:absl:You are saving your model as an HDF5 file via `model.save()` or `keras.saving.save_model(model)`. This file format is c
```

Start coding or generate with AI.