EC9170 - Deep Learning
Mini Project
Real and Fake Face Detection

2020/E/070-KEERTHIKAN F.J

Introduction

In recent years, the proliferation of manipulated digital images, particularly photoshopped faces, has raised significant concerns regarding the authenticity of visual content. The ability to distinguish between real and fake faces is crucial in various domains, including security, social media, and digital forensics. This project aims to develop and evaluate deep learning models that can accurately classify faces as real or fake, thereby contributing to efforts to combat digital image manipulation.

Aim

The primary aim of this project is to create and compare three deep learning models for classifying faces into real and fake categories. The models will be trained and evaluated on a dataset containing real human faces and high-quality photoshopped faces. The final objective is to identify the best-performing model for this task based on a comprehensive evaluation.

```
from google.colab import drive
drive.mount('/content/drive')

# Unzip the dataset
!unzip '/content/drive/MyDrive/Colab Notebooks/Real and Fake Face Detection Dataset.zip' -d '/content/dataset'
```

```
from keras.preprocessing.image import ImageDataGenerator
from keras.applications import MobileNetV2
from keras.models import Model
from keras.layers import Dense, GlobalAveragePooling2D, Dropout
from keras.optimizers import Adam
from keras.callbacks import EarlyStopping

# Parameters
nbatch = 32
img_size = (128, 128)
train_data_dir = '/content/dataset/Real and Fake Face Detection
Dataset/Train'
test_data_dir = '/content/dataset/Real and Fake Face Detection
Dataset/Test'

# Create ImageDataGenerator instances
train_datagen = ImageDataGenerator(
    rescale=1./255,
    validation_split=0.2,
    rotation_range=20,
    width_shift_range=0.2,
    height_shift_range=0.2,
    shear_range=0.2,
    zoom_range=0.2,
```

```
    horizontal_flip=True,
    fill_mode='nearest'
)
test_datagen = ImageDataGenerator(rescale=1./255)

# Load the training and validation sets
training_set = train_datagen.flow_from_directory(
    train_data_dir,
    target_size=img_size,
    batch_size=nbatch,
    class_mode='binary',
    subset='training',
    shuffle=True
)

validation_set = train_datagen.flow_from_directory(
    train_data_dir,
    target_size=img_size,
    batch_size=nbatch,
    class_mode='binary',
    subset='validation'
)

# Load the test set
test_set = test_datagen.flow_from_directory(
    test_data_dir,
    target_size=img_size,
    batch_size=nbatch,
    class_mode='binary'
)

# Load MobileNetV2 model without the top layers
base_model = MobileNetV2(weights='imagenet', include_top=False,
input_shape=(128, 128, 3))

# Add new top layers
x = base_model.output
x = GlobalAveragePooling2D()(x)
x = Dense(128, activation='relu')(x)
x = Dropout(0.5)(x)
predictions = Dense(1, activation='sigmoid')(x)

# Define the full model
model = Model(inputs=base_model.input, outputs=predictions)

# Freeze the layers of the base model
for layer in base_model.layers:
    layer.trainable = False
```

```python
# Compile the model
model.compile(optimizer=Adam(learning_rate=0.0001),
loss='binary_crossentropy', metrics=['accuracy'])

# Early stopping
early_stopping = EarlyStopping(monitor='val_loss', patience=15,
restore_best_weights=True)

# Train the model
history = model.fit(
    training_set,
    epochs=20,
    validation_data=validation_set,
    callbacks=[early_stopping]
)

# Unfreeze some layers of the base model for fine-tuning
for layer in base_model.layers[-30:]:  # Unfreeze the last 30 layers
    layer.trainable = True

# Re-compile the model after unfreezing
model.compile(optimizer=Adam(learning_rate=0.00001),
loss='binary_crossentropy', metrics=['accuracy'])

# Train the model again
history_finetune = model.fit(
    training_set,
    epochs=20,
    validation_data=validation_set,
    callbacks=[early_stopping]
)

# Evaluate the model on the test set
test_loss, test_accuracy = model.evaluate(test_set)
print(f'Test Loss: {test_loss}')
print(f'Test Accuracy: {test_accuracy}')

# Save the model
model.save('Real_fake_face_detector.h5')
model.save('Real_fake_face_detector.keras')
```

```python
# Evaluate the model on the test set
test_loss, test_accuracy = model.evaluate(test_set)

# Print the test accuracy
print(f'Test accuracy: {test_accuracy:.2f}')
print(f'Test loss: {test_loss:.2f}')
```

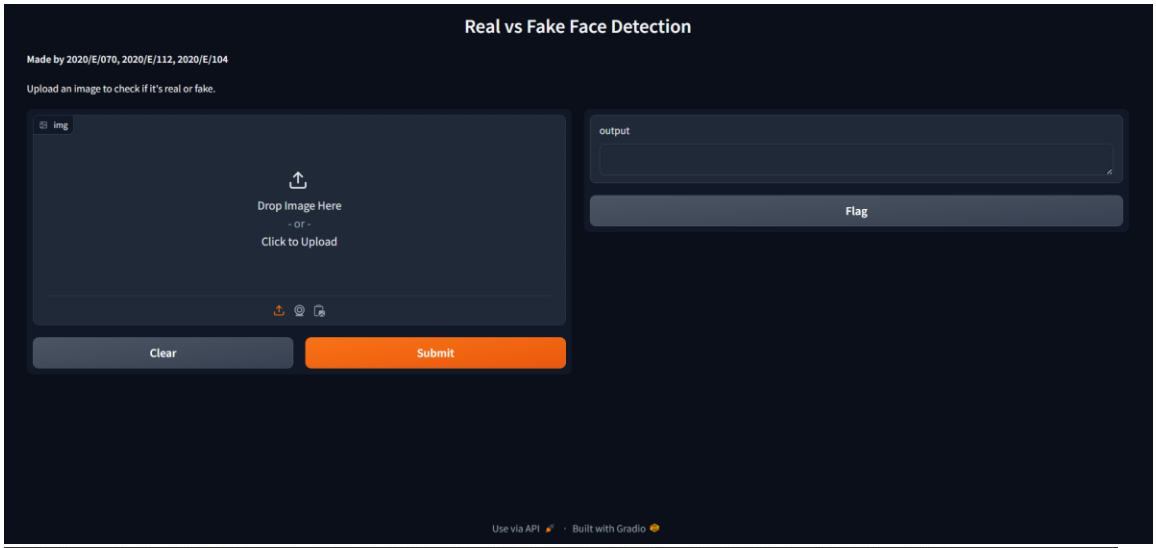
```
pip install gradio
```

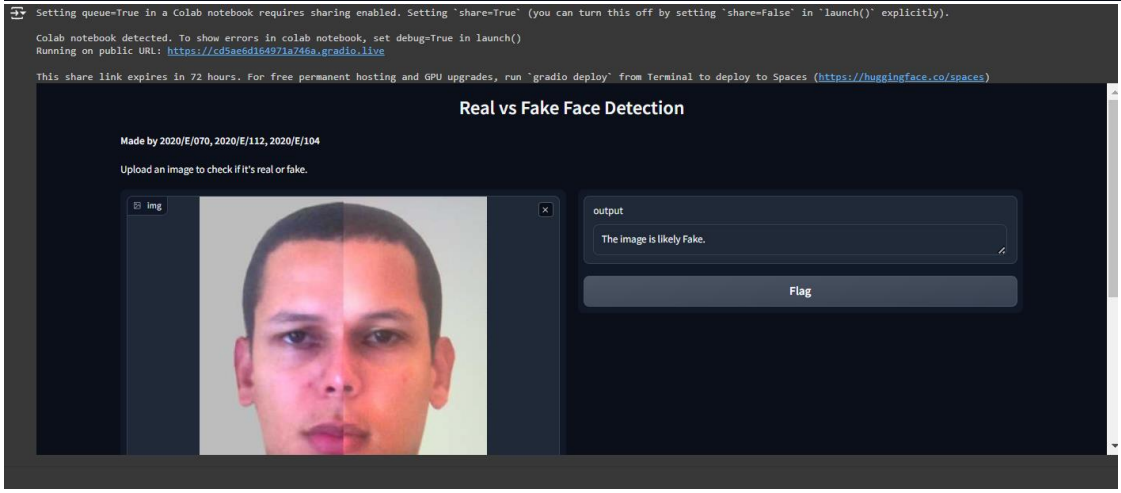
```
import gradio as gr
import numpy as np
from keras.models import load_model
from keras.preprocessing import image
from PIL import Image

# Load your trained model
model = load_model('improved_real_fake_face_detector.h5')  # Adjust the
path to your model

def load_and_preprocess_image(img):
    # Ensure the image is a PIL Image
    if isinstance(img, np.ndarray):
        img = Image.fromarray(img)

    img = img.resize((128, 128))  # Resize to match model input
    img_array = np.array(img) / 255.0  # Normalize pixel values

    # Check if the image has 3 channels (RGB)
    if img_array.ndim == 2:  # If grayscale
        img_array = np.stack((img_array,) * 3, axis=-1)  # Convert to
RGB

    img_array = np.expand_dims(img_array, axis=0)  # Add batch
dimension
    return img_array

def predict_image(img):
    try:
        img_array = load_and_preprocess_image(img)
        prediction = model.predict(img_array)

        if prediction[0] > 0.5:
            return "The image is likely Fake."
        else:
            return "The image is likely Real."
    except Exception as e:
        return f"Error: {str(e)}"

# Gradio interface
iface = gr.Interface(fn=predict_image, inputs="image", outputs="text",
                     title="Real vs Fake Face Detection",
```

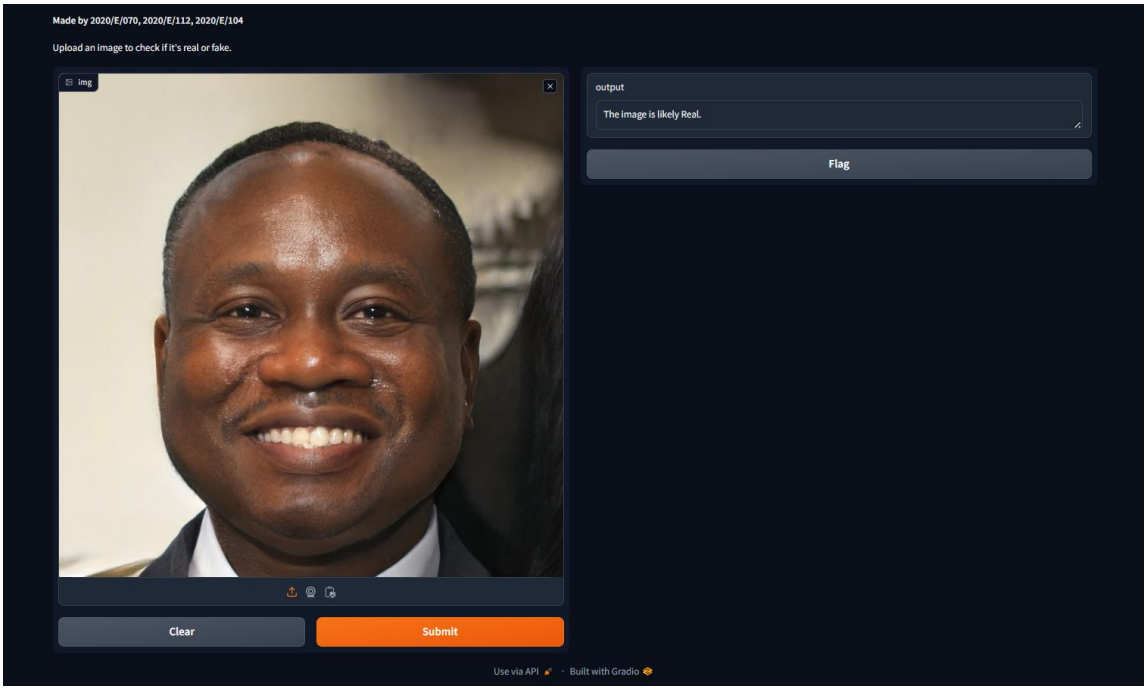
```
                    description="<b>Made by 2020/E/070, 2020/E/112,
2020/E/104</b> \n\nUpload an image to check if it's real or fake.",
                    css=".description { font-size: 30px; }"
)

iface.launch()
```

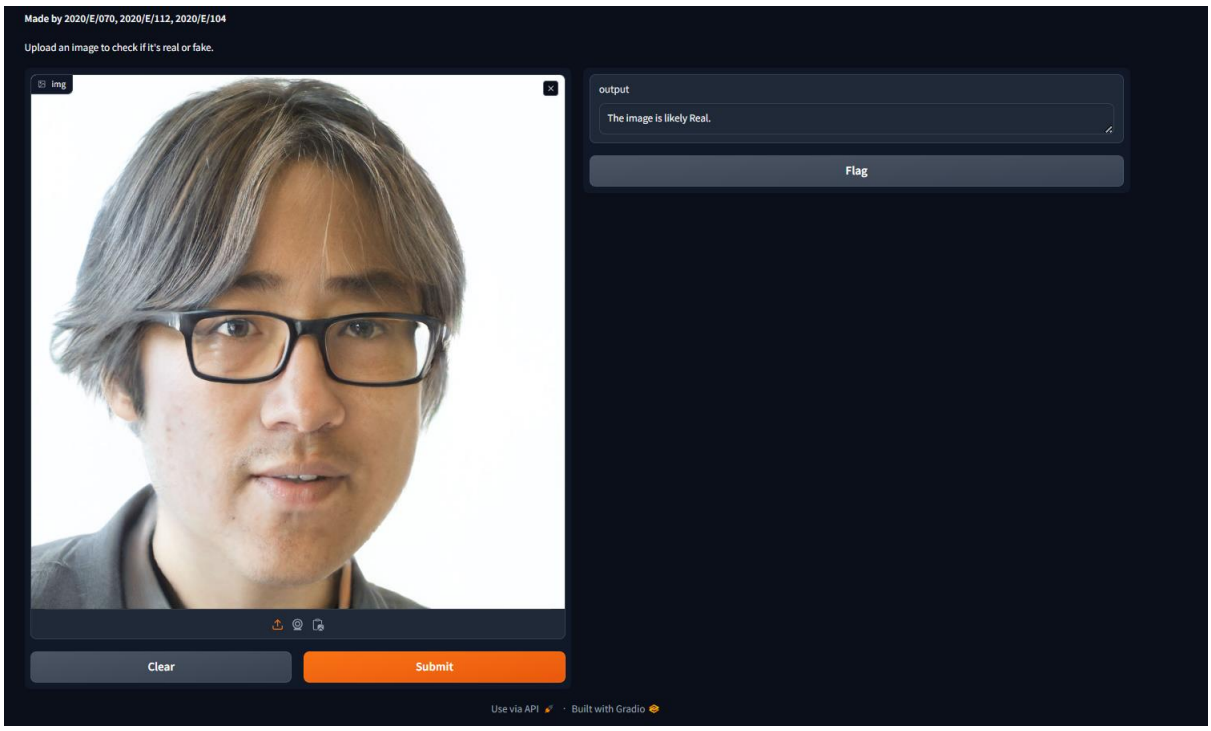
```
24/24 [==============================] - 6s 238ms/step - loss: 0.7931 - accuracy: 0.5309
Test accuracy: 0.53
Test loss: 0.79
```

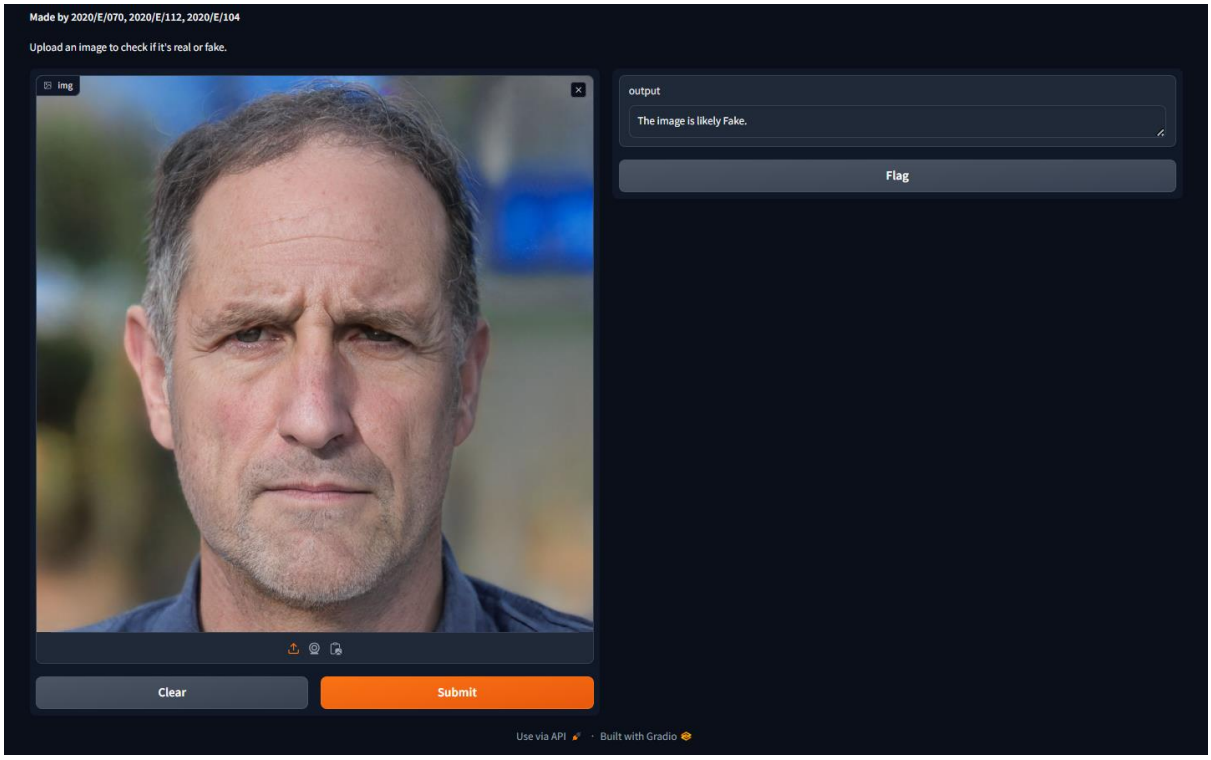
Outputs from the model

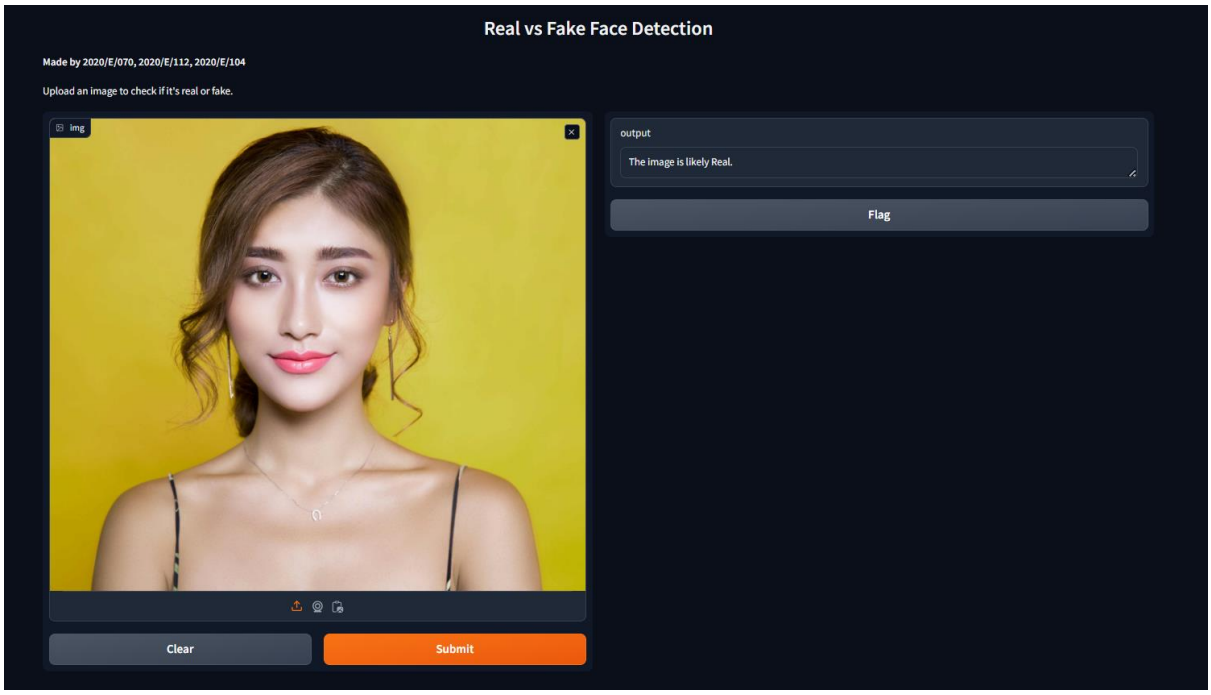


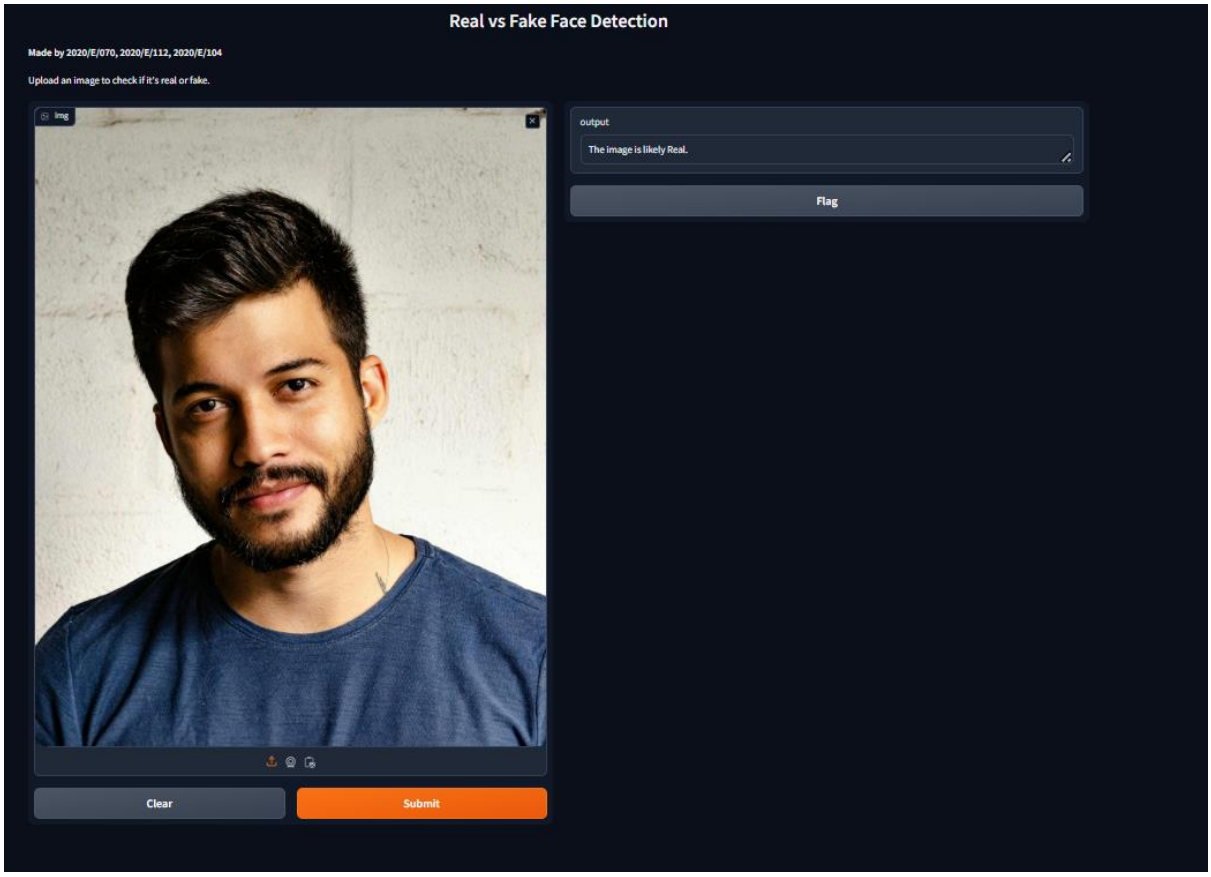


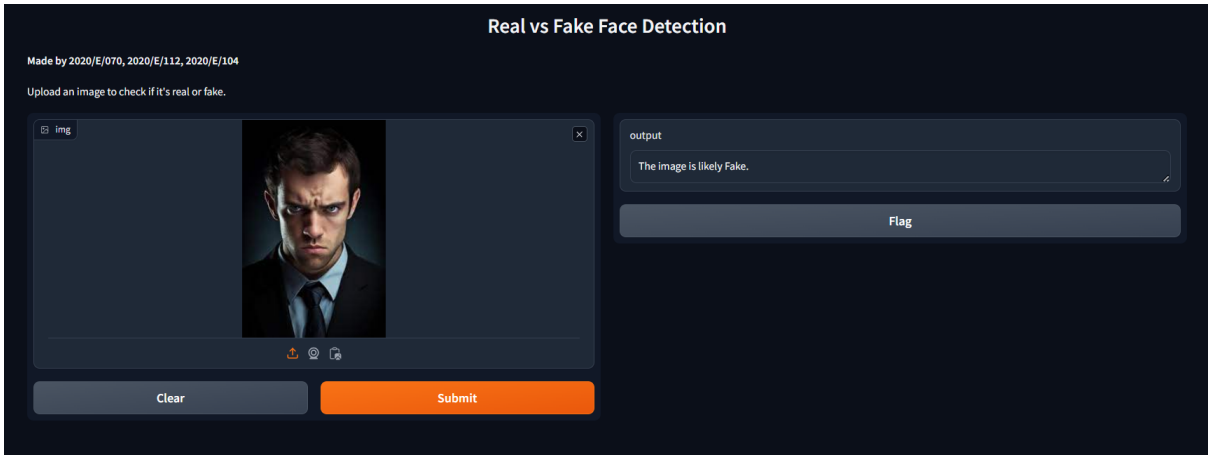

Made by 2020/E/070, 2020/E/112, 2020/E/104
Upload an image to check if it's real or fake.
img
output
The image is likely Real.
Flag
Clear
Submit
Use via API
· Built with Gradio

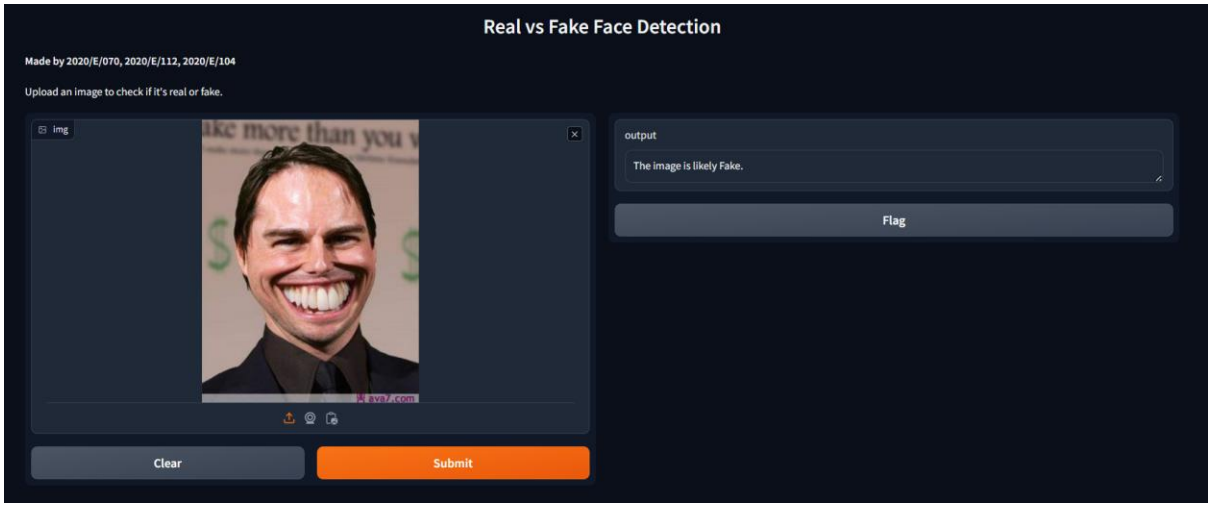

Made by 2020/E/070, 2020/E/112, 2020/E/104
Upload an image to check if it's real or fake.
img
output
The image is likely Real.
Flag
Clear
Submit
Use via API
· Built with Gradio

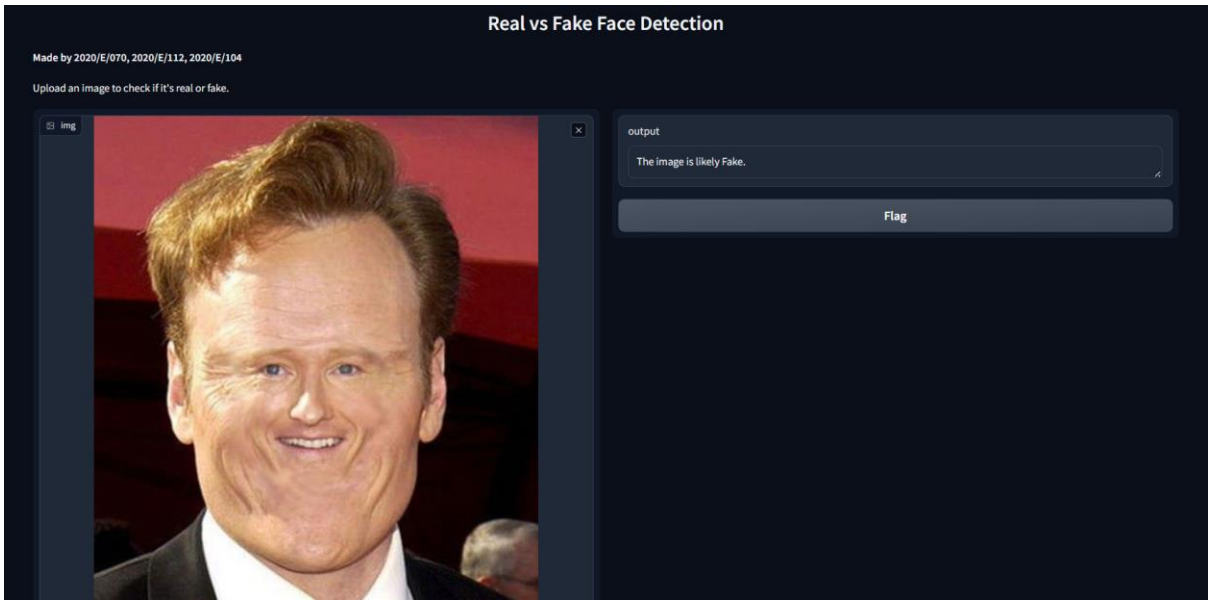

Made by 2020/E/070, 2020/E/112, 2020/E/104
Upload an image to check if it's real or fake.
img
output
The image is likely Fake.
Flag
Clear
Submit
Use via API
Built with Gradio

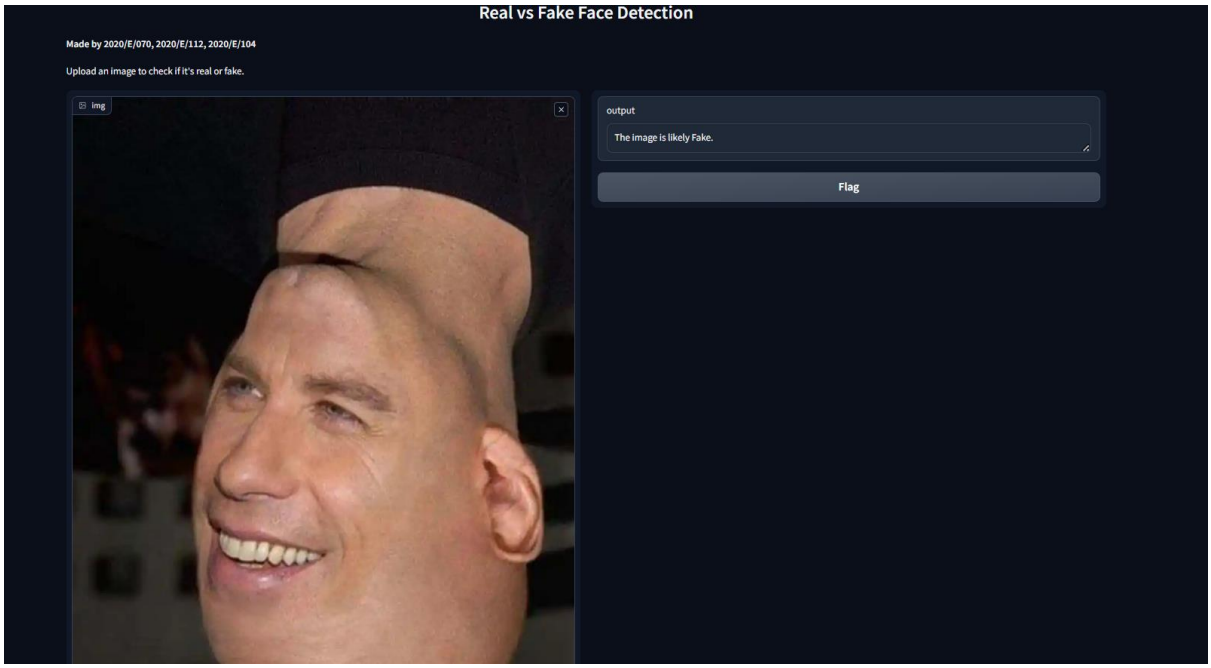

Real vs Fake Face Detection
Made by 2020/E/070, 2020/E/112, 2020/E/104
Upload an image to check if it's real or fake.
img
output
The image is likely Real.
Flag
Clear
Submit


Real vs Fake Face Detection
Made by 2020/E/070, 2020/E/112, 2020/E/104
Upload an image to check if it's real or fake.
img
output
The image is likely Real.
Flag
Clear
Submit


Real vs Fake Face Detection
Made by 2020/E/070, 2020/E/112, 2020/E/104
Upload an image to check if it's real or fake.
img
output
The image is likely Fake.
Flag
Clear
Submit


Real vs Fake Face Detection
Made by 2020/E/070, 2020/E/112, 2020/E/104
Upload an image to check if it's real or fake.
img
output
The image is likely Fake.
Flag
Clear
Submit


Real vs Fake Face Detection
Made by 2020/E/070, 2020/E/112, 2020/E/104
Upload an image to check if it's real or fake.
img
output
The image is likely Fake.
Flag


Real vs Fake Face Detection
Made by 2020/E/070, 2020/E/112, 2020/E/104
Upload an image to check if it's real or fake.
img
output
The image is likely Fake.
Flag