بسمه تعالی

گزارش پروژهٔ بینایی کامپیوتر

عنوان:

طراحی مدل کلاسیک و عمیق برای الگوریتم anti-spoofing

استاد درس:

دکتر محمدی

اعضای گروه:

هلیا شمس زاده ۴۰۰۵۲۱۴۸۶

بهاره کاووسی نژاد ۹۹۴۳۱۲۱۷

تهیهٔ دیتاست:

دیتاست این پروژه با ترکیب دو دیتاست CASIA-FASD و برخی تصاویر anti-spoofing-face-fake به دست آمده است.

پیش‌پردازش داده‌ها:

ابتدا داده‌های موجود در دیتاست را در وبسایت Hugging Face آپلود کرده، و با استفاده از کد زیر داخل گوگل کولب load می‌کنیم.

In [ ]:
```
!pip install kaggle --q
```

In [ ]:
```
!kaggle datasets download -d minhnh2107/casiafasd
```

```
Dataset URL: https://www.kaggle.com/datasets/minhnh2107/casiafasd
License(s): unknown
Downloading casiafasd.zip to /content
100% 70.6M/70.6M [00:05<00:00, 19.7MB/s]
100% 70.6M/70.6M [00:05<00:00, 13.7MB/s]
```

با قطعه کد زیر، مسیرهای مربوط به تصاویر را در لیست image_paths قرار می‌دهیم:

In [5]:
```
from PIL import Image
import matplotlib.pyplot as plt
import numpy as np
import os
```

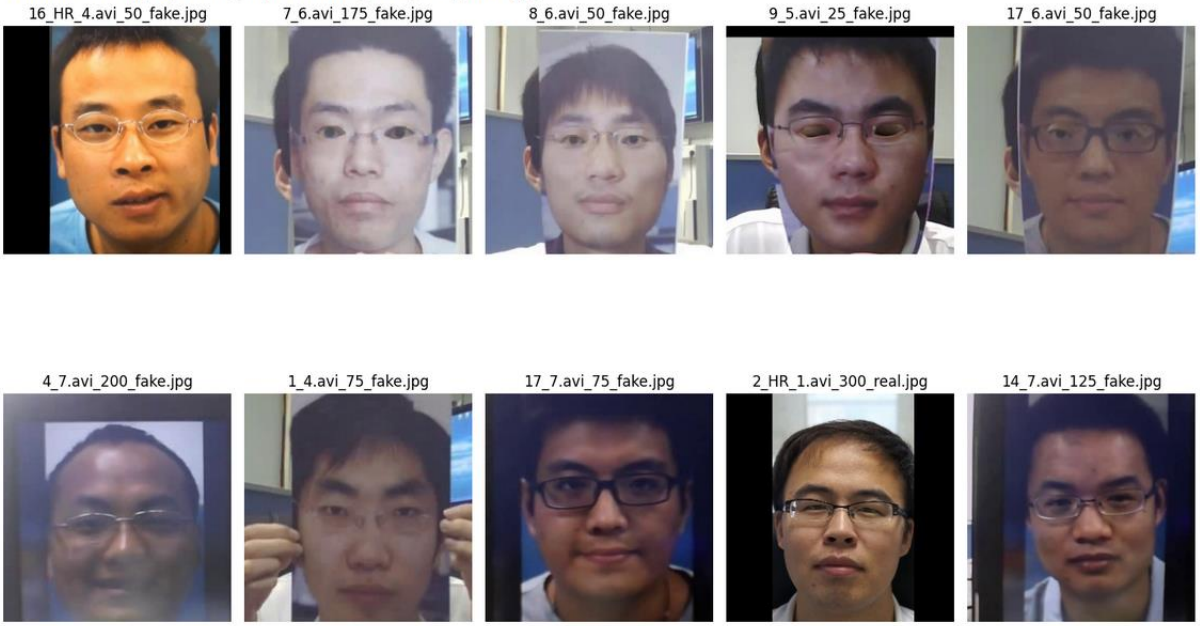
In [56]:
```
def load_image_paths(directory):
    image_paths = []
    for filename in os.listdir(directory):
        if filename.endswith(".jpg") or filename.endswith(".png"):
            image_paths.append(os.path.join(directory, filename))
    return image_paths

image_paths = load_image_paths('/content/train_img/train_img/color')
```

In [57]:
```
output_dir = '/content/train_img/train_img/color'
```

چند مورد از تصاویر موجود در دیتاست در ادامه نمایش داده شده است:

```
/content/train_img/train_img/color/17_7.avi_75_fake.jpg
/content/train_img/train_img/color/2_HR_1.avi_300_real.jpg
/content/train_img/train_img/color/14_7.avi_125_fake.jpg
```



با کد زیر، label های تصاویر را براساس اسم فایل‌ها پیدا می‌کنیم، زیر فرمت نام فایل‌ها به صورت زیر است:

/.. /***_fake.jpg

/../***_real.jpg

Labels Extraction

In [58]:

```
train_labels = []
for path in image_paths:
    # Split the path by '/' and get the last part (filename)
    filename = os.path.basename(path)
    # Split the filename by '_' and get the second last part (label)
    label = filename.split('_')[-1].split('.')[0]  # Assuming label is second last part before '.jpg' or '.png'
    train_labels.append(label)
```

چون در classification دو کلاس داریم، label تصاویر fake را به 0 و label تصاویر real را به 1 تغییر می‌دهیم تا بعدا به صورت categorical در آوریم:

In [60]:
```
train_labels_binary = []
for label in train_labels:
  if label == 'fake':
    train_labels_binary.append(0)
  else:
    train_labels_binary.append(1)
```

In [61]:
```
train_labels_binary[0]
```

Out[61]: 0

برای مرحلهٔ تست، نیاز داریم تا از ویدیوهایمان فریم استخراج کنیم، به این منظور تابع زیر را نوشته که ورودی آن دیرکتوری ویدیوها، دیرکتوری مقصد برای ذخیرهٔ فریم‌ها و label مربوطه است. از هر ویدیو یک فریم تصادفی استخراج کرده و در آدرس‌های نوشته شده در آخر کد (با نام فایل ایجاد شده) ذخیره می‌کند. چون ویدیوهای real و fake در دیرکتوری‌های جدایی نگهداری شده‌اند براساس نام دیرکتوری مربوطه ابتدای نام فریم‌ها صفر (یعنی fake) یا 1 (یعنی real) قرار می‌دهیم:

Random Frame Extraction

In [16]:
```
import random

def extract_frames(video_path, save_path, label):
    # Open the video file
    video = cv2.VideoCapture(video_path)
    frame_count = int(video.get(cv2.CAP_PROP_FRAME_COUNT))

    # Select one random frame
    random_frame = random.randint(0, frame_count - 1)

    # Set the position of the video to the selected frame
    video.set(cv2.CAP_PROP_POS_FRAMES, random_frame)
    success, frame = video.read()

    # If the frame was successfully read, save it
    if success:
        frame_path = os.path.join(save_path, f"{label}_{random_frame}.jpg")
        cv2.imwrite(frame_path, frame)

    # Release the video file
    video.release()
```

In [18]:
```
import cv2

fake_test_videos_path = '/content/dataset/fake/test'
fake_train_videos_path = '/content/dataset/fake/train'
real_test_videos_path = '/content/dataset/real/test'
real_train_videos_path = '/content/dataset/real/train'

save_frames_path = '/content/frames/test'
# Create the directory if it doesn't exist
if not os.path.exists(save_frames_path):
    os.makedirs(save_frames_path)

# Iterate over fake videos and extract frames
for fake_video_file in os.listdir(fake_test_videos_path):
    fake_video_path = os.path.join(fake_test_videos_path, fake_video_file)
    extract_frames(fake_video_path, save_frames_path, 0)

# Iterate over real videos and extract frames
for real_video_file in os.listdir(real_test_videos_path):
    real_video_path = os.path.join(real_test_videos_path, real_video_file)
    extract_frames(real_video_path, save_frames_path, 1)


# Iterate over fake videos and extract frames
for fake_video_file in os.listdir(fake_train_videos_path):
    fake_video_path = os.path.join(fake_train_videos_path, fake_video_file)
    extract_frames(fake_video_path, save_frames_path, 0)

# Iterate over real videos and extract frames
for real_video_file in os.listdir(real_train_videos_path):
    real_video_path = os.path.join(real_train_videos_path, real_video_file)
    extract_frames(real_video_path, save_frames_path, 1)
```

نتیجهٔ کد بالا، ذخیرهٔ تمامی فریم‌ها در دیرکتوری save_frames_path می‌باشد. با کمک تابع load_images_path که بالاتر نوشته شده است، مسیر تمام فریم‌ها را در متغیر paths قرار می‌دهیم:

In [62]:
```
paths = load_image_paths(save_frames_path)
```

حال لیبل فریم‌های استخراج شده را با کد زیر پیدا کرده و در لیست test_labels_binary ذخیره می‌کنیم:

Labels

In [75]:
```
test_labels_binary = []
for path in test_image_paths:
    # Split the path by '/' and get the last part (filename)
    filename = os.path.basename(path)
    # Split the filename by '_' and get the second last part (label)
    label = int(filename.split('_')[0])  # Assuming label is second last part before '.jpg' or '.png'
    test_labels_binary.append(label)
```

طراحی مدل برای یادگیری ویژگی (deep learning).....

برای این بخش از مدل‌های pretrained استفاده شده است. اولین مدل مورد استفاده، مدل ResNet50 با وزن‌های اولیهٔ imagenet است. مدل دوم که مورد بررسی قرار گرفت، مدل google/vit-base-patch16-224 است.

آموزش مدل ResNet50:

ابتدا کتابخانه‌های لازم را ایمپورت می‌کنیم:

In [78]:
```
import tensorflow as tf
from tensorflow.keras.applications import ResNet50
from tensorflow.keras.preprocessing.image import ImageDataGenerator
from tensorflow.keras.layers import Dense, GlobalAveragePooling2D
from tensorflow.keras.models import Model
import cv2
```

در بخش قبلی، تمام path های داده‌های آموزش را در train_image_paths ذخیره کردیم. حال باید هر کدام را با دستور cv2.imread خوانده و به np.array تبدیل کرده و در لیست train_images_np قرار دهیم:

Preparing Images

In [79]:
```
import numpy as np
from PIL import Image

# Assuming images is a list of PIL Image objects
train_images_np = []

for image_path in train_image_paths:
    image = cv2.imread(image_path)
    rgb_img = cv2.cvtColor(image, cv2.COLOR_BGR2RGB)
    img_np = np.array(rgb_img)
    train_images_np.append(img_np)

# Now images_np should contain numpy arrays representing each image
print(train_images_np[0].shape)  # Check the shape of the first image numpy array
```

```
(256, 256, 3)
```

برای داده‌های تست هم روند بالا را انجام داده و هر فریم را از مسیر مربوطه خوانده و به آرایه تبدیل می‌کنیم:

In [80]:
```
# Assuming images is a list of PIL Image objects
test_images_np = []

for image_path in test_image_paths:
    image = cv2.imread(image_path)
    rgb_img = cv2.cvtColor(image, cv2.COLOR_BGR2RGB)
    img_np = np.array(rgb_img)
    test_images_np.append(img_np)

# Now images_np should contain numpy arrays representing each image
print(test_images_np[0].shape)  # Check the shape of the first image numpy array
```

```
(2400, 1080, 3)
```

همانطور که می‌دانیم ورودی مدل ResNet یک تصویر سه کانالهٔ 224 x 224 است، برای همین لازم است که داده‌های آموزش و تست resize شوند. با استفاده از دستور cv2.resize تمام آرایهٔ تصاویر آموزش و تست را resize می‌کنیم:

In [81]:
```
# Resize images to (224, 224, 3)
train_images_resized = np.array([cv2.resize(img, (224, 224)) for img in train_images_np])

# Verify the shape of the resized images
print(train_images_resized.shape)

# Resize images to (224, 224, 3)
test_images_resized = np.array([cv2.resize(img, (224, 224)) for img in test_images_np])

# Verify the shape of the resized images
print(test_images_resized.shape)
```

برای آموزش مدل، ابتدا لیبل‌های داده‌های آموزش و تست (فریم ویدیوها) را به حالت one-hot (categorical) در می‌آوریم. همچنین، داده‌های آموزش را به دو بخش آموزش و تست اولیه تقسیم می‌کنیم:

Training the model

In [85]:
```
# One-hot encode the labels
train_labels_binary_onehot = to_categorical(np.array(train_labels_binary), num_classes=2)
test_labels_binary_onehot = to_categorical(np.array(test_labels_binary), num_classes=2)

X_train, X_test, y_train, y_test = train_test_split(train_images_resized, train_labels_binary_onehot, test_size=0.1, random_
```

مدل ResNet را Load کرده و به جای لایهٔ FC آن، یک لایهٔ GAP، یک لایهٔ dense با 1024 نورون و نهایتا یک لایهٔ FC با دو نورون (چون دو کلاس داریم) و تابع فعالسازی sofmax اضافه می‌کنیم. لایه‌های base که مربوط به ResNet هستند را فریز کرده تا فقط لایه‌های جدید با وزن‌های تصادفی آموزش ببینند:

In [86]:
```
# Load ResNet50 model pre-trained on ImageNet
base_model = ResNet50(weights='imagenet', include_top=False, input_shape=(224, 224, 3))

# Add custom layers on top of ResNet50
x = base_model.output
x = GlobalAveragePooling2D()(x)
x = Dense(1024, activation='relu')(x)
predictions = Dense(2, activation='softmax')(x)  # Adjust to match your number of classes
```

In [87]:
```
# Define the model
model = Model(inputs=base_model.input, outputs=predictions)

# Freeze the layers of ResNet50
for layer in base_model.layers:
    layer.trainable = False
```

In [88]:
```
# Compile the model
model.compile(optimizer='adam', loss='categorical_crossentropy', metrics=['accuracy'])

# Train the model
model.fit(X_train, y_train, epochs=10, batch_size=32, validation_data=(X_test, y_test))
```

نتایج آموزش:

```
Epoch 1/10
75/75 [==============================] - 21s 176ms/step - loss: 0.6101 - accuracy: 0.8758 - val_loss: 0.0856 - val_accuracy: 0.980
0
Epoch 2/10
75/75 [==============================] - 9s 123ms/step - loss: 0.0566 - accuracy: 0.9883 - val_loss: 0.0494 - val_accuracy: 0.9867
Epoch 3/10
75/75 [==============================] - 9s 125ms/step - loss: 0.0288 - accuracy: 0.9929 - val_loss: 0.0311 - val_accuracy: 0.9900
Epoch 4/10
75/75 [==============================] - 8s 113ms/step - loss: 0.0182 - accuracy: 0.9962 - val_loss: 0.0187 - val_accuracy: 0.9967
Epoch 5/10
75/75 [==============================] - 9s 125ms/step - loss: 0.0099 - accuracy: 0.9996 - val_loss: 0.0185 - val_accuracy: 0.9933
Epoch 6/10
75/75 [==============================] - 9s 114ms/step - loss: 0.0064 - accuracy: 0.9996 - val_loss: 0.0110 - val_accuracy: 0.9967
Epoch 7/10
75/75 [==============================] - 9s 127ms/step - loss: 0.0045 - accuracy: 0.9996 - val_loss: 0.0084 - val_accuracy: 0.9967
Epoch 8/10
75/75 [==============================] - 9s 126ms/step - loss: 0.0029 - accuracy: 1.0000 - val_loss: 0.0074 - val_accuracy: 0.9967
Epoch 9/10
75/75 [==============================] - 9s 114ms/step - loss: 0.0023 - accuracy: 1.0000 - val_loss: 0.0069 - val_accuracy: 0.9967
Epoch 10/10
75/75 [==============================] - 9s 115ms/step - loss: 0.0017 - accuracy: 1.0000 - val_loss: 0.0064 - val_accuracy: 0.9983
```

نتایج تست‌های اولیه (بخشی از داده‌های دیتاستِ استفاده شده):

```
19/19 [==============================] - 2s 91ms/step - loss: 0.0064 - accuracy: 0.9983
Test Loss: 0.006443445105105638, Test Accuracy: 99.83333492279053%
```

۱) نتایج ارزیابی روی داده‌های خام تست (فریم ویدیوها)

به منظور استفاده از فریم های تصادفی انتخاب شده برای تست مدل، از تابع resize_image استفاده می کنیم تا تصاویر را به numpy array تبدیل کرده و آنها را به همراه برچسب هایشان در دو لیست ذخیره کنیم.

In [109…
```
# Path to your test images
test_images_dir = '/content/extracted_frames/test'
```

In [110…
```
# Function to resize images in a directory
def resize_images(directory, target_size=(224, 224)):
    resized_images = []
    test_image_paths = []
    test_image_labels = []

    # Iterate over each image in the directory
    for filename in os.listdir(directory):
        if filename.endswith(".jpg") or filename.endswith(".png"):
            label = filename.split('.')[0].split('_')[0]
            image_path = os.path.join(directory, filename)
            image = cv2.imread(image_path)
            resized_image = cv2.resize(image, target_size)

            # Append resized image, image path, and label
            resized_images.append(resized_image)
            test_image_paths.append(image_path)
            test_image_labels.append(int(label))  # Convert label to int

    # Convert lists to numpy arrays
    resized_images = np.array(resized_images)
    test_image_labels = np.array(test_image_labels)

    return resized_images, test_image_paths, test_image_labels
```

دقت مدل را بر روی داده تست بدست می آوریم:

In [22]:
```
# Resize images in test directory
resized_images, test_image_paths, test_image_labels = resize_images(test_images_dir)

# Check the shape of resized images
print("Resized images shape:", resized_images.shape)
```

```
Resized images shape: (33, 224, 224, 3)
```

In [23]:
```
# Convert labels to one-hot encoding
test_image_labels_onehot = to_categorical(test_image_labels, num_classes=2)
```

In [ ]:
```
# Evaluate the model
test_loss, test_accuracy = model.evaluate(resized_images, test_image_labels_onehot)
```

```
1/1 [==============================] - 0s 141ms/step - loss: 1.7041 - accuracy: 0.7188
```

با استفاده از کد زیر می‌توان فریم‌هایی که prediction اشتباه داشته اند را مشاهده کرد:

Wrong Answers

In [93]:
```
import os

# Directory containing the images
image_directory = '/content/frames/test'  # Replace with the actual image directory

# Get the predicted labels for the test images
predictions = model.predict(test_images_resized)
predicted_labels = np.argmax(predictions, axis=1)

print("Total test images: ", len(predicted_labels))
# Get the true labels
true_labels = np.argmax(test_labels_binary_onehot, axis=1)

# Find the names of incorrectly classified images
incorrect_image_names = []
for index, (predicted_label, true_label) in enumerate(zip(predicted_labels, true_labels)):
    if predicted_label != true_label:
        image_name = os.listdir(image_directory)[index]
        incorrect_image_names.append(image_name)
print("Wrong: ", len(incorrect_image_names))
# Display the names of incorrectly classified images
for image_name in incorrect_image_names:
    print(f"Incorrectly classified image: {image_name}")
```

در این قسمت پیش بینی‌های مدل برای real بودن تصاویر نرمالایز شده و در predicted_scores_normalized ذخیره می‌شوند که در ادامه این مقادیر را در فایل csv. ذخیره می‌کنیم.

Predictions

In [135]:
```
import csv
```

In [136]:
```
predictions = model.predict(resized_images)
predicted_scores = predictions[:, 1]
# Normalize scores between 0 and 1
predicted_scores_normalized = (predicted_scores - np.min(predicted_scores)) / (np.max(predicted
```

۲) نتایج ارزیابی روی داده‌های تست کراپ شده (فریم ویدیوها)

به منظور برش صورت در تصاویر از MTCNN استفاده شده است. بدین منظور تابع detect_and_crop_faces قرار داده شده است که با تعیین bounding_box دور صورت برش زده می شود.

In [27]:
```
def detect_and_crop_faces(image_path, detector):
    image = Image.open(image_path)
    image_np = np.asarray(image)
    result = detector.detect_faces(image_np)
    if result:
        for person in result:
            bounding_box = person['box']
            keypoints = person['keypoints']

            # Crop the detected face
            x, y, width, height = bounding_box
            cropped_face = image_np[y:y+height, x:x+width]

            # Convert the cropped face back to an image
            cropped_face_image = Image.fromarray(cropped_face)

            return cropped_face_image
    return None
```

In [28]:
```
# Initialize the MTCNN face detector
detector = MTCNN()
```

تصویر روبرو یک نمونه از عملکرد برش صورت را نشان می دهد:

مدل را بر روی تصاویر برش خورده نیز تست می‌کنیم. همان طور که در تصویر پیداست دقت در حدود ۷۲ درصد می‌باشد.

In [30]:
```
# Resize images in test directory
cropped_resized_images, cropped_test_image_paths, cropped_test_image_labels = resize_images(out

# Check the shape of resized images
print("Resized images shape:", cropped_resized_images.shape)
```

```
Resized images shape: (33, 224, 224, 3)
```

In [31]:
```
# Convert labels to one-hot encoding
cropped_test_image_labels_onehot = to_categorical(cropped_test_image_labels, num_classes=2)
```

In [ ]:
```
# Evaluate the model
cropped_test_loss, cropped_test_accuracy = model.evaluate(cropped_resized_images, cropped_test_
```

```
1/1 [==============================] - 0s 139ms/step - loss: 0.8393 - accuracy: 0.7188
```

مانند قسمت بالا پیش بینی های مدل را برای real بودن تصویر به دست آورده و در predictions.csv ذخیره می کنیم. توجه کنید که این فایل در پوشه پروژه به صورت predictions_ResNet.csv ذخیره شده است تا از تداخل نام آن با مدل گوگل جلوگیری به عمل آید.

In [ ]:
```
# Predictions
cropped_predictions = model.predict(cropped_resized_images)
cropped_predicted_scores = cropped_predictions[:, 1]  # Assuming class 1 corresponds to index 1

# Normalize scores between 0 and 1
cropped_predicted_scores_normalized = (cropped_predicted_scores - np.min(cropped_predicted_scor
```

```
1/1 [==============================] - 0s 25ms/step
```

In [ ]:
```
# Save predictions to a CSV file
output_file = 'predictions.csv'
with open(output_file, 'w', newline='') as f:
    writer = csv.writer(f)
    writer.writerow(['filename', 'liveness_score', 'liveness_score_crop'])  # Header
    for i, filename in enumerate(test_image_paths):
        writer.writerow([filename, predicted_scores_normalized[i], cropped_predicted_scores_nor

print(f"Predictions saved to {output_file}")
```

```
Predictions saved to predictions.csv
```

آموزش و ارزیابی مدل google/vit-base-patch16-224:

این مدل، یک مدل pretrained با وزن‌های اولیهٔ خود است.

ابتدا مدل را load می‌کنیم (در این بخش از کتابخانهٔ torch استفاده شده است):

In [ ]:
```
import torch
import torch.nn as nn
import torch.optim as optim
from transformers import ViTImageProcessor, ViTForImageClassification
from sklearn.model_selection import train_test_split
from tensorflow.keras.utils import to_categorical
import numpy as np
```

In [ ]:
```
# Step 1: Load pretrained model and processor
processor = ViTImageProcessor.from_pretrained('google/vit-base-patch16-224')
model = ViTForImageClassification.from_pretrained('google/vit-base-patch16-224')

# Freeze the base model
for param in model.parameters():
    param.requires_grad = False

# Modify the classifier head
model.classifier = nn.Linear(model.config.hidden_size, 2)
nn.init.xavier_uniform_(model.classifier.weight)

# Move model to device
device = torch.device("cuda" if torch.cuda.is_available() else "cpu")
model.to(device)
```

سپس داده‌های ورودی (چه آموزش چه تست) را نرمالیزه می‌کنیم. (این داده‌ها در cell های قبلی تعریف شدند) یعنی آرایهٔ تصاویر را به ۲۵۵ تقسیم کرده تا هر پیکسل مقداری بین صفر و یک داشته باشد:

In [ ]:
```
# Normalize the image data
X_train_pt = torch.tensor(X_train / 255.0, dtype=torch.float32).permute(0, 3, 1, 2).to(device)
X_test_pt = torch.tensor(test_images_resized / 255.0, dtype=torch.float32).permute(0, 3, 1, 2).to(device)
y_train_pt = torch.tensor(y_train, dtype=torch.long).to(device)
y_test_pt = torch.tensor(test_labels_binary_onehot, dtype=torch.long).to(device)
```

تابع بهینه‌ساز را Adam انتخاب کرده و از تابع ضرر Cross Entropy استفاده می‌کنیم:

In [ ]:
```
# Step 3: Define optimizer and criterion for the classifier head
optimizer = optim.Adam(model.classifier.parameters(), lr=1e-5)  # Only optimize classifier parameters
criterion = nn.CrossEntropyLoss()
```

با قطعه کد زیر، مدل را با داده‌های تست آموزش می‌دهیم. در این مرحله شبکهٔ پایه فریز است:

In [ ]:
```python
# Step 4: Training loop
model.train()
epochs = 15
batch_size = 32

for epoch in range(epochs):
    total_loss = 0
    for i in range(0, len(X_train), batch_size):
        batch_inputs = X_train_pt[i:i+batch_size]
        batch_labels = y_train_pt[i:i+batch_size]

        optimizer.zero_grad()

        outputs = model(pixel_values=batch_inputs)
        loss = criterion(outputs.logits, torch.argmax(batch_labels, dim=1))

        loss.backward()
        optimizer.step()

        total_loss += loss.item()

    print(f"Epoch {epoch+1}/{epochs}, Loss: {total_loss}")
```

نتیجهٔ آموزش:

```
Epoch 1/15, Loss: 85.32642370462418
Epoch 2/15, Loss: 71.27150690555573
Epoch 3/15, Loss: 61.57638555765152
Epoch 4/15, Loss: 55.00446879863739
Epoch 5/15, Loss: 50.49847862124443
Epoch 6/15, Loss: 47.284194737672806
Epoch 7/15, Loss: 44.84917551279068
Epoch 8/15, Loss: 42.87762239575386
Epoch 9/15, Loss: 41.18519252538681
Epoch 10/15, Loss: 39.668936520814896
Epoch 11/15, Loss: 38.273439168930054
Epoch 12/15, Loss: 36.9696801006794
Epoch 13/15, Loss: 35.74257443845272
Epoch 14/15, Loss: 34.58400775492191
Epoch 15/15, Loss: 33.48910477757454
```

حال برای کم کردن loss و رسیدن به دقت بهتر، fine tuning انجام می‌دهیم تا وزن‌های همهٔ لایه‌ها بهینه شود:

In [ ]:
```python
# Step 5: Unfreeze all layers for fine-tuning
for param in model.parameters():
    param.requires_grad = True

# Use a smaller learning rate for fine-tuning the entire model
optimizer = optim.Adam(model.parameters(), lr=1e-6)
criterion = nn.BCEWithLogitsLoss()  # Use this if model outputs logits

# Ensure your targets are float
y_train_pt = torch.tensor(y_train_pt, dtype=torch.float)
y_test_pt = torch.tensor(y_test_pt, dtype=torch.float)

# Training loop remains similar
for epoch in range(15):
    total_loss = 0
    for i in range(0, len(X_train), batch_size):
        batch_inputs = X_train_pt[i:i+batch_size]
        batch_labels = y_train_pt[i:i+batch_size]

        optimizer.zero_grad()

        outputs = model(pixel_values=batch_inputs)
        loss = criterion(outputs.logits, batch_labels)

        loss.backward()
        optimizer.step()

        total_loss += loss.item()

    print(f"Fine-tuning Epoch {epoch+1}/{15}, Train Loss: {total_loss:.4f}")
```

نتایج fine tuning:

```
Fine-tuning Epoch 1/15, Train Loss: 30.1228
Fine-tuning Epoch 2/15, Train Loss: 19.2256
Fine-tuning Epoch 3/15, Train Loss: 13.1220
Fine-tuning Epoch 4/15, Train Loss: 8.9411
Fine-tuning Epoch 5/15, Train Loss: 6.0971
Fine-tuning Epoch 6/15, Train Loss: 4.1831
Fine-tuning Epoch 7/15, Train Loss: 2.9008
Fine-tuning Epoch 8/15, Train Loss: 2.0612
Fine-tuning Epoch 9/15, Train Loss: 1.5155
Fine-tuning Epoch 10/15, Train Loss: 1.1547
Fine-tuning Epoch 11/15, Train Loss: 0.9069
Fine-tuning Epoch 12/15, Train Loss: 0.7297
Fine-tuning Epoch 13/15, Train Loss: 0.5988
Fine-tuning Epoch 14/15, Train Loss: 0.4995
Fine-tuning Epoch 15/15, Train Loss: 0.4230
```

حال با قطعه کد زیر، مدل با وزن‌های به دست آمده را روی داده‌های تست (فریم‌ها) ارزیابی می‌کنیم:

```
# Assuming X_test_pt and y_test_pt are preprocessed and available on the correct device
X_test_pt = X_test_pt.to(device)
y_test_pt = y_test_pt.to(device)

# Set the model to evaluation mode
model.eval()
correct = 0
total = 0
batch_size = 32  # Set your batch size

with torch.no_grad():
    for i in range(0, len(X_test_pt), batch_size):
        batch_inputs = X_test_pt[i:i+batch_size]
        batch_labels = y_test_pt[i:i+batch_size]

        # Ensure the inputs are on the correct device
        batch_inputs = batch_inputs.to(device)
        batch_labels = batch_labels.to(device)

        outputs = model(pixel_values=batch_inputs)
        _, predicted = torch.max(outputs.logits, 1)

        total += batch_labels.size(0)
        correct += (predicted == torch.argmax(batch_labels, dim=1)).sum().item()

accuracy = 100 * correct / total
print(f"Final Accuracy on test set: {accuracy:.2f}%")
```

نتایج روی داده‌های تست (بخشی از دیتاست):

```
Final Accuracy on test set: 99.83%
```

نتایج روی داده‌های تست (فریم‌های تصادفی):

```
Final Accuracy on test set: 99.83%
Final Loss on test set: 0.0001
```

نتایج روی داده‌های تست (فریم‌های تصادفی برش خورده):

```
Final Accuracy on test set: 63.64%
Final Loss on test set: 0.0257
```

مانند قسمت قبل پیش بینی های مدل را برای real بودن تصویر به دست آورده و در predictions.csv ذخیره می کنیم. توجه کنید که این فایل در پوشه پروژه به صورت predictions_Google.csv ذخیره شده است تا از تداخل نام آن با مدل ResNet جلوگیری به عمل آید.

طراحی مدل برای استخراج ویژگی (classic learning).....

مدل CNN

برای این قسمت از یک مدل CNN که فقط شامل چند لایهٔ کانوولوشنی و FC است استفاده شده است.

ابتدا دیتاست را از Hugging Face لود می‌کنیم. این دیتاست به صورت دیکشنری‌ای از دیکشنری‌ها می‌باشد:

Load Dataset

```
In [5]: !pip install datasets --q

In [6]: from datasets import load_dataset

In [7]: Dataset = load_dataset("Bahareh0281/liveness_images")
```

سپس کتابخانه‌های مورد نیاز را import می‌کنیم:

Import Necessary Libraries

```
In [77]: import cv2
         import numpy as np
         import os
         from skimage.feature import local_binary_pattern
         from skimage import measure
         from sklearn.ensemble import RandomForestClassifier
         from sklearn.model_selection import train_test_split
         from sklearn.metrics import classification_report, accuracy_score
         from PIL import Image
         from sklearn.model_selection import train_test_split
         from tensorflow.keras.utils import to_categorical
```

برای بخش استخراج ویژگی، ویژگی‌هایی همانند فرکانس، lbp، عمق و یک سری ویژگی‌های آماری استخراج می‌شود. توابع مربوط به هر یک در ادامه آورده شده است:

Feature Extraction Functions

In [11]:

```
radius = 3
n_points = 8 * radius

def compute_fourier_transform(image):
    gray = cv2.cvtColor(image, cv2.COLOR_BGR2GRAY)
    f = np.fft.fft2(gray)
    fshift = np.fft.fftshift(f)
    magnitude_spectrum = 20 * np.log(np.abs(fshift))
    return magnitude_spectrum

def compute_lbp(image):
    gray = cv2.cvtColor(image, cv2.COLOR_BGR2GRAY)
    lbp = local_binary_pattern(gray, n_points, radius, method="uniform")
    return lbp

def compute_depth(image):
    # به عنوان مثال از کانال آبی برای تخمین عمق استفاده می‌کنیم
    depth = image[:, :, 2]
    return depth

def extract_statistical_features(image):
    gray = cv2.cvtColor(image, cv2.COLOR_BGR2GRAY)
    mean = np.mean(gray)
    std_dev = np.std(gray)
    skewness = np.mean((gray - mean) ** 3) / (std_dev ** 3)
    kurtosis = np.mean((gray - mean) ** 4) / (std_dev ** 4)
    entropy = measure.shannon_entropy(gray)
    return mean, std_dev, skewness, kurtosis, entropy
```

حال به تابعی نیاز داریم که از دیکشنری Dataset عکس را پیدا کرده و ویژگی‌های آن را با توابع نوشته شده در بالا استخراج کند. هر ویژگی ابتدا به ۶۴ در ۶۴ resize شده و سپس با هم concat شده و به عمق ۳ می‌رسد (۳ ویژگی):

In [91]:
```
def process_images(dataset, traget_size, num=0):
    train_images_features = []
    train_images_labels = []
    if num == 0:
        num = len(dataset['train'])

    for i in range(num):
        img = dataset['train'][i]['image']
        if isinstance(img, Image.Image):
            img = np.array(img)  # Convert PIL image to NumPy array

            # Extract frequency features
            magnitude_spectrum = compute_fourier_transform(img)
            magnitude_spectrum_resized = cv2.resize(magnitude_spectrum, (traget_size, traget_size))

            # Extract LBP features
            lbp = compute_lbp(img)
            lbp_hist, _ = np.histogram(lbp, bins=np.arange(0, n_points + 3), range=(0, n_points + 2))
            lbp_hist_normalized = lbp_hist / lbp_hist.sum()
            lbp_hist_resized = cv2.resize(lbp_hist_normalized.reshape(-1, 1), (traget_size, traget_size))

            # Extract statistical features
            mean, std_dev, skewness, kurtosis, entropy = extract_statistical_features(img)
            statistical_features = np.array([mean, std_dev, skewness, kurtosis, entropy])
            statistical_features_resized = cv2.resize(statistical_features.reshape(-1, 1), (traget_size, traget_size))

            # Combine features into a 3D array
            combined_features = np.stack([
                magnitude_spectrum_resized,
                lbp_hist_resized,
                statistical_features_resized
            ], axis=-1)

            train_images_features.append(combined_features)
            train_images_labels.append(dataset['train'][i]['label'])

    return np.array(train_images_features), np.array(train_images_labels)
```

چون تعداد داده‌های آموزشی زیاد است (حدود ۶۰۰۰ تا) فقط ۳۰۰۰ تای آن را برای استخراج ویژگی استفاده می‌کنیم:

In [50]:
```
train_images_features,  train_images_labels = process_images(Dataset, 64, 3000)
```

حال پس از استخراج ویژگی از هر ۳۰۰۰ داده آموزشی و نگهداری آنها در لیست train_images_features و لیبل‌های آنها در train_images_labels، باید این داده‌ها را برای ورودی به مدل آماده کنیم. داده‌ی پردازش شده را به دو بخش train و test تقسیم می‌کنیم. همچنین label های آنها را به صورت one-hot در می‌آوریم:

Split training dataset and prepare it for train process

In [105]:
```
# Convert depth features to a numpy array
features = np.array(train_images_features)
labels = np.array(train_images_labels)
# Split the data into training and testing sets
X_train, X_test, y_train, y_test = train_test_split(features, labels, test_size=0.2, random_state=42)

# One-hot encode the labels
y_train = to_categorical(y_train, num_classes=2)
y_test = to_categorical(y_test, num_classes=2)
```

مدل CNN را به شکل زیر ساخته که شامل لایه‌های کانوولوشنی و max_pooling است و در ادامه چند لایهٔ FC داریم:

Create CNN Model and Train it

In [106]:
```
from tensorflow.keras.models import Sequential
from tensorflow.keras.layers import Conv2D, MaxPooling2D, Flatten, Dense, Dropout
from tensorflow.keras.optimizers import Adam

# Build the CNN model
model = Sequential([
    Conv2D(32, (3, 3), activation='relu', input_shape=(64, 64, 3)),
    MaxPooling2D((2, 2)),
    Conv2D(64, (3, 3), activation='relu'),
    MaxPooling2D((2, 2)),
    Flatten(),
    Dense(128, activation='relu'),
    Dropout(0.5),
    Dense(2, activation='softmax')
])

# Compile the model
model.compile(optimizer=Adam(), loss='categorical_crossentropy', metrics=['accuracy'])

# Train the model
history = model.fit(X_train, y_train, epochs=20, batch_size=32, validation_split=0.2)
```

نتایج آموزش مدل:

```
5
Epoch 8/20
60/60 [==============================] - 0s 6ms/step - loss: 0.4190 - accuracy: 0.8188 - val_loss: 0.3909 - val_accuracy: 0.850
0
Epoch 9/20
60/60 [==============================] - 0s 7ms/step - loss: 0.4135 - accuracy: 0.8161 - val_loss: 0.3748 - val_accuracy: 0.845
8
Epoch 10/20
60/60 [==============================] - 0s 6ms/step - loss: 0.4010 - accuracy: 0.8297 - val_loss: 0.3929 - val_accuracy: 0.845
8
Epoch 11/20
60/60 [==============================] - 0s 6ms/step - loss: 0.3999 - accuracy: 0.8297 - val_loss: 0.3957 - val_accuracy: 0.847
9
Epoch 12/20
60/60 [==============================] - 0s 6ms/step - loss: 0.3957 - accuracy: 0.8323 - val_loss: 0.3923 - val_accuracy: 0.845
8
Epoch 13/20
60/60 [==============================] - 0s 6ms/step - loss: 0.3923 - accuracy: 0.8328 - val_loss: 0.3963 - val_accuracy: 0.833
3
Epoch 14/20
60/60 [==============================] - 0s 6ms/step - loss: 0.3918 - accuracy: 0.8396 - val_loss: 0.3646 - val_accuracy: 0.858
3
Epoch 15/20
60/60 [==============================] - 0s 6ms/step - loss: 0.3944 - accuracy: 0.8328 - val_loss: 0.3690 - val_accuracy: 0.856
2
Epoch 16/20
60/60 [==============================] - 0s 6ms/step - loss: 0.3960 - accuracy: 0.8391 - val_loss: 0.3804 - val_accuracy: 0.835
4
Epoch 17/20
60/60 [==============================] - 0s 6ms/step - loss: 0.3967 - accuracy: 0.8286 - val_loss: 0.4174 - val_accuracy: 0.825
0
Epoch 18/20
60/60 [==============================] - 0s 6ms/step - loss: 0.3994 - accuracy: 0.8172 - val_loss: 0.4055 - val_accuracy: 0.829
2
Epoch 19/20
60/60 [==============================] - 0s 7ms/step - loss: 0.3974 - accuracy: 0.8349 - val_loss: 0.4170 - val_accuracy: 0.845
8
Epoch 20/20
60/60 [==============================] - 0s 6ms/step - loss: 0.3792 - accuracy: 0.8448 - val_loss: 0.3575 - val_accuracy: 0.862
5
```

نتیجهٔ تست روی بخشی از داده‌های دیتاست:

Evaluate on dataset tests

In [59]:
```
# Evaluate the model
test_loss, test_accuracy = model.evaluate(X_test, y_test)
print(f'Test Accuracy: {test_accuracy * 100:.2f}%')
```

```
19/19 [==============================] - 0s 4ms/step - loss: 0.3536 - accuracy: 0.8567
Test Accuracy: 85.67%
```

۱) ارزیابی با فریم‌های خام.....

حال ویدیوها را لود کرده و پس از استخراج فریم برای هر یک ویژگی استخراج می‌کنیم:

Load test videos

In [61]:
```
import gdown

file_id = '1a5R5h05hCyw9PzIBhSjy2jLL3dSFy2xA'
destination = '/content/dataset.zip'  # Path where the file will be saved
gdown.download(f'https://drive.google.com/uc?id={file_id}', destination, quiet=False)

import zipfile

with zipfile.ZipFile(destination, 'r') as zip_ref:
    zip_ref.extractall('/content/dataset')
```

Generate random frames from each video

In [62]:
```
import random

def extract_frames(video_path, save_path, label, test):
    # Open the video file
    video = cv2.VideoCapture(video_path)
    frame_count = int(video.get(cv2.CAP_PROP_FRAME_COUNT))

    # Select one random frame
    random_frame = random.randint(0, frame_count - 1)

    # Set the position of the video to the selected frame
    video.set(cv2.CAP_PROP_POS_FRAMES, random_frame)
    success, frame = video.read()

    # If the frame was successfully read, save it
    if success:
        frame_path = os.path.join(save_path, f"{label}_{random_frame}.jpg")
        cv2.imwrite(frame_path, frame)

        # Convert the frame to a PIL image
        pil_image = Image.open(frame_path)

        # Save the image and label to the dictionary
        test.append({'image': pil_image, 'label': label})

    # Release the video file
    video.release()
```

In [73]:
```
import cv2

fake_test_videos_path = '/content/dataset/fake/test'
real_test_videos_path = '/content/dataset/real/test'

save_frames_path = '/content/extracted_frames2/test'
# Create the directory if it doesn't exist
if not os.path.exists(save_frames_path):
    os.makedirs(save_frames_path)

# Create a list to hold the dictionary entries
test = []

# Iterate over fake videos and extract frames
for fake_video_file in os.listdir(fake_test_videos_path):
    fake_video_path = os.path.join(fake_test_videos_path, fake_video_file)
    extract_frames(fake_video_path, save_frames_path, 0, test)

# Iterate over real videos and extract frames
for real_video_file in os.listdir(real_test_videos_path):
    real_video_path = os.path.join(real_test_videos_path, real_video_file)
    extract_frames(real_video_path, save_frames_path, 1, test)
```

In [74]:
```
len(test)
```

Out[74]: 32

این تابع برای استخراج ویژگی‌های فریم‌ها است:

In [89]:
```
def process_images_2(tests, traget_size):
    test_images_features = []
    test_images_labels = []


    for i in range(len(tests)):
        img = tests[i]['image']
        if isinstance(img, Image.Image):
            img = np.array(img)  # Convert PIL image to NumPy array

            # Extract frequency features
            magnitude_spectrum = compute_fourier_transform(img)
            magnitude_spectrum_resized = cv2.resize(magnitude_spectrum, (traget_size, traget_size))

            # Extract LBP features
            lbp = compute_lbp(img)
            lbp_hist, _ = np.histogram(lbp, bins=np.arange(0, n_points + 3), range=(0, n_points + 2))
            lbp_hist_normalized = lbp_hist / lbp_hist.sum()
            lbp_hist_resized = cv2.resize(lbp_hist_normalized.reshape(-1, 1), (traget_size, traget_size))

            # Extract statistical features
            mean, std_dev, skewness, kurtosis, entropy = extract_statistical_features(img)
            statistical_features = np.array([mean, std_dev, skewness, kurtosis, entropy])
            statistical_features_resized = cv2.resize(statistical_features.reshape(-1, 1), (traget_size, traget_size))

            # Combine features into a 3D array
            combined_features = np.stack([
                magnitude_spectrum_resized,
                lbp_hist_resized,
                statistical_features_resized
            ], axis=-1)

            test_images_features.append(combined_features)
            test_images_labels.append(tests[i]['label'])

    return np.array(test_images_features), np.array(test_images_labels)
```

Extract features from each frame and convert labels to one-hot form

```
In [98]: test_frames_features, test_frames_labels = process_images_2(test, 64)

         test_frames_labels = to_categorical(test_frames_labels, num_classes=2)

In [84]: test_frames_features.shape

Out[84]: (32, 64, 64, 3)

In [85]: test_frames_labels.shape

Out[85]: (32, 2)
```

حال فریم‌ها را روی مدل تست می‌کنیم و به دقت ۶۵ درصد می‌رسیم:

Evaluate model on test set

```
In [92]: test_loss, test_accuracy = model.evaluate(test_frames_features, test_frames_labels)
         print(f'Test Accuracy: {test_accuracy * 100:.2f}%')

1/1 [==============================] - 0s 23ms/step - loss: 0.7118 - accuracy: 0.6562
Test Accuracy: 65.62%
```

۲) ارزیابی با فریم‌های کراپ شده.....

```
[68] # Evaluate the model
     cropped_test_loss, cropped_test_accuracy = model.evaluate(cropped_resized_images, cropped_test_image_labels_onehot)

1/1 [==============================] - 0s 295ms/step - loss: 16.9545 - accuracy: 0.6333
```

مدل InceptionV3

ما یک مدل دیگر به اسم InceptionV3 نیز آموزش داده و تست کردیم که پس از طی کردن مراحل بالا (ولی با ورودی‌های ۷۵ در ۷۵) به دقت زیر رسید:

```
In [107]: test_loss, test_accuracy = model.evaluate(test_frames_features2, test_frames_labels2)
          print(f'Test Accuracy: {test_accuracy * 100:.2f}%')
```

```
1/1 [==============================] - 0s 48ms/step - loss: 1.2868 - accuracy: 0.5938
Test Accuracy: 59.38%
```

فایل‌های CSV

.....ResNet

	filename	liveness_score	liveness_score_crop
2	/content/extracted_frames/test/0_fake_test_9.jpg	0.00015386408	0.003418131
3	/content/extracted_frames/test/1_real_test_5.jpg	0.99862504	0.19990857
4	/content/extracted_frames/test/0_fake_test_18.jpg	0.09662213	0.0029069243
5	/content/extracted_frames/test/0_fake_test_10.jpg	0.0	0.043594684
6	/content/extracted_frames/test/0_fake_test_16.jpg	1.7275298e-08	0.39844176
7	/content/extracted_frames/test/0_fake_test_3.jpg	1.4534247e-05	0.0016172786
8	/content/extracted_frames/test/0_fake_test_14.jpg	2.7491708e-07	9.9893676e-09
9	/content/extracted_frames/test/0_fake_test_22.jpg	0.95068336	4.1886574e-06
10	/content/extracted_frames/test/0_fake_test_4.jpg	2.3468495e-08	0.010007217
11	/content/extracted_frames/test/1_real_test_6.jpg	1.0	0.9966656
12	/content/extracted_frames/test/0_fake_test_17.jpg	1.0300121e-06	0.00023260794
13	/content/extracted_frames/test/0_fake_test_24.jpg	0.024334507	0.53533036
14	/content/extracted_frames/test/0_fake_test_6.jpg	0.01008647	1.6910568e-06
15	/content/extracted_frames/test/0_fake_test_8.jpg	9.475355e-08	0.02510002
16	/content/extracted_frames/test/0_fake_test_5.jpg	5.830659e-06	2.9369207e-07
17	/content/extracted_frames/test/0_fake_test_21.jpg	1.4770186e-10	4.0024548e-07
18	/content/extracted_frames/test/1_real_test_8.jpg	6.594619e-08	0.079365835
19	/content/extracted_frames/test/1_real_test_14.jpg	1.2092696e-06	0.9281693
20	/content/extracted_frames/test/1_real_test_11.jpg	0.023145096	0.12917027
21	/content/extracted_frames/test/1_real_test_9.jpg	0.013346534	0.44084492
22	/content/extracted_frames/test/0_fake_test_13.jpg	0.022343948	0.0017614434
23	/content/extracted_frames/test/1_real_test_10.jpg	0.034586858	1.0
24	/content/extracted_frames/test/1_real_test_7.jpg	0.998798	0.95112777
25	/content/extracted_frames/test/0_fake_test_7.jpg	0.5607886	1.220891e-07
26	/content/extracted_frames/test/0_fake_test_2.jpg	0.11492377	0.6418984
27	/content/extracted_frames/test/0_fake_test_11.jpg	0.8241024	0.0062382338
28	/content/extracted_frames/test/0_fake_test_1.jpg	5.0240043e-05	0.0
29	/content/extracted_frames/test/1_real_test_1.jpg	0.6557224	0.0001283624
30	/content/extracted_frames/test/1_real_test_4.jpg	0.99995303	0.0019607819
31	/content/extracted_frames/test/1_real_test_2.jpg	0.9907777	0.99604017
32	/content/extracted_frames/test/0_fake_test_12.jpg	9.8179735e-06	0.44107676
33	/content/extracted_frames/test/1_real_test_3.jpg	0.0013095422	0.2207905

	filename	liveness_score	liveness_score_crop
2	/content/extracted_frames/test/0_fake_test_9.jpg	1.1997387e-06	0.075982064
3	/content/extracted_frames/test/1_real_test_5.jpg	3.4402194e-05	0.27216777
4	/content/extracted_frames/test/0_fake_test_18.jpg	3.3413626e-06	0.11004771
5	/content/extracted_frames/test/0_fake_test_10.jpg	6.9561106e-06	0.4739628
6	/content/extracted_frames/test/0_fake_test_16.jpg	0.99890083	0.20333162
7	/content/extracted_frames/test/0_fake_test_3.jpg	2.4360403e-05	0.8671758
8	/content/extracted_frames/test/0_fake_test_14.jpg	7.624393e-06	0.75995564
9	/content/extracted_frames/test/0_fake_test_22.jpg	0.9991972	0.00026433307
10	/content/extracted_frames/test/0_fake_test_4.jpg	4.8308027e-07	0.009642265
11	/content/extracted_frames/test/1_real_test_6.jpg	0.99994385	0.8764013
12	/content/extracted_frames/test/0_fake_test_17.jpg	0.9990745	0.0004772388
13	/content/extracted_frames/test/0_fake_test_24.jpg	2.5185186e-06	0.29643717
14	/content/extracted_frames/test/0_fake_test_6.jpg	5.719247e-06	0.001178804
15	/content/extracted_frames/test/0_fake_test_8.jpg	3.6587894e-06	0.9528003
16	/content/extracted_frames/test/0_fake_test_5.jpg	3.3451255e-07	0.0
17	/content/extracted_frames/test/0_fake_test_21.jpg	3.2370597e-05	0.20524535
18	/content/extracted_frames/test/0_fake_test_20.jpg	8.926601e-06	0.5424584
19	/content/extracted_frames/test/1_real_test_8.jpg	5.0172807e-06	0.37364766
20	/content/extracted_frames/test/1_real_test_11.jpg	6.624528e-07	0.28127798
21	/content/extracted_frames/test/1_real_test_9.jpg	0.9999639	0.975746
22	/content/extracted_frames/test/0_fake_test_13.jpg	1.6663838e-05	0.7983957
23	/content/extracted_frames/test/1_real_test_10.jpg	1.297761e-06	0.8075048
24	/content/extracted_frames/test/1_real_test_7.jpg	4.485854e-06	0.45987895
25	/content/extracted_frames/test/0_fake_test_7.jpg	7.19415e-06	0.00044538264
26	/content/extracted_frames/test/0_fake_test_2.jpg	1.8257648e-06	0.94058764
27	/content/extracted_frames/test/1_real_test_13.jpg	1.3618558e-06	0.02999502
28	/content/extracted_frames/test/0_fake_test_11.jpg	0.99872196	0.31928542
29	/content/extracted_frames/test/0_fake_test_1.jpg	7.894344e-06	0.00078811677
30	/content/extracted_frames/test/1_real_test_1.jpg	1.04904775e-05	0.55582446
31	/content/extracted_frames/test/1_real_test_4.jpg	2.467762e-06	0.010667216
32	/content/extracted_frames/test/1_real_test_2.jpg	1.975893e-06	0.9642561
33	/content/extracted_frames/test/0_fake_test_12.jpg	1.0901841e-05	0.005582588
34	/content/extracted_frames/test/1_real_test_3.jpg	3.4348259e-06	1.0

	A	B	C	D
1	filename	liveness_score_Frequency	liveness_score_Frequency_crop	
2	/content/extracted_frames/test/1_65.jpg	0.30388936	2.70E-13	
3	/content/extracted_frames/test/0_52.jpg	0.307438	2.70E-06	
4	/content/extracted_frames/test/0_24.jpg	0.4576012	4.32E-10	
5	/content/extracted_frames/test/0_71.jpg	0.5232546	4.79E-06	
6	/content/extracted_frames/test/1_3.jpg	0.48764586	1.44E-09	
7	/content/extracted_frames/test/0_23.jpg	0.64039975	0.001342476	
8	/content/extracted_frames/test/0_47.jpg	0.54939216	3.97E-06	
9	/content/extracted_frames/test/0_70.jpg	0.5787849	6.57E-12	
10	/content/extracted_frames/test/1_88.jpg	0.42824188	0.000961671	
11	/content/extracted_frames/test/1_26.jpg	0.4065303	1.32E-11	
12	/content/extracted_frames/test/1_15.jpg	0.4548167	2.80E-14	
13	/content/extracted_frames/test/0_72.jpg	0.43530336	0.000358396	
14	/content/extracted_frames/test/0_110.jpg	0.29371434	0	
15	/content/extracted_frames/test/0_67.jpg	0.5018248	6.41E-14	
16	/content/extracted_frames/test/0_34.jpg	0.5039976	4.75E-09	
17	/content/extracted_frames/test/0_130.jpg	0.5162486	4.46E-06	
18	/content/extracted_frames/test/0_69.jpg	0.44224897	5.31E-16	
19	/content/extracted_frames/test/1_179.jpg	0.28070566	0.019343546	
20	/content/extracted_frames/test/1_22.jpg	0.4873567	2.47E-11	
21	/content/extracted_frames/test/0_36.jpg	0.4216304	9.01E-12	
22	/content/extracted_frames/test/0_91.jpg	0.4858257	3.58E-08	
23	/content/extracted_frames/test/0_56.jpg	0.39906353	2.35E-05	
24	/content/extracted_frames/test/1_16.jpg	0.47344154	0.003195079	
25	/content/extracted_frames/test/0_167.jpg	0.5016255	3.39E-09	
26	/content/extracted_frames/test/1_69.jpg	0.50555044	4.99E-09	

predictions_Inception