```python
import zipfile
import os

# Define the target folder names
target_folders = [1, 2, 3, 4, 5, 6, 7, 8, 9, 10]

# Specify the path to the ZIP file and the extraction directory
zip_file = "subjects_0-1999_72_imgs.zip"
extraction_dir = "dataset"

# Create the extraction directory if it doesn't exist
if not os.path.exists(extraction_dir):
    os.makedirs(extraction_dir)

# Open the ZIP file and extract the specified folders
with zipfile.ZipFile(zip_file, 'r') as zip_ref:
    for folder_name in target_folders:
        folder_path = f"{folder_name}/"
        zip_ref.extractall(extraction_dir, members=[member for member
in zip_ref.infolist() if member.filename.startswith(folder_path)])

print("Extraction complete.")
```

```
Extraction complete.
```

```python
import zipfile
import os
import shutil
import random

# Define the target folder names
target_folders = [11, 12, 13, 14, 15]
```

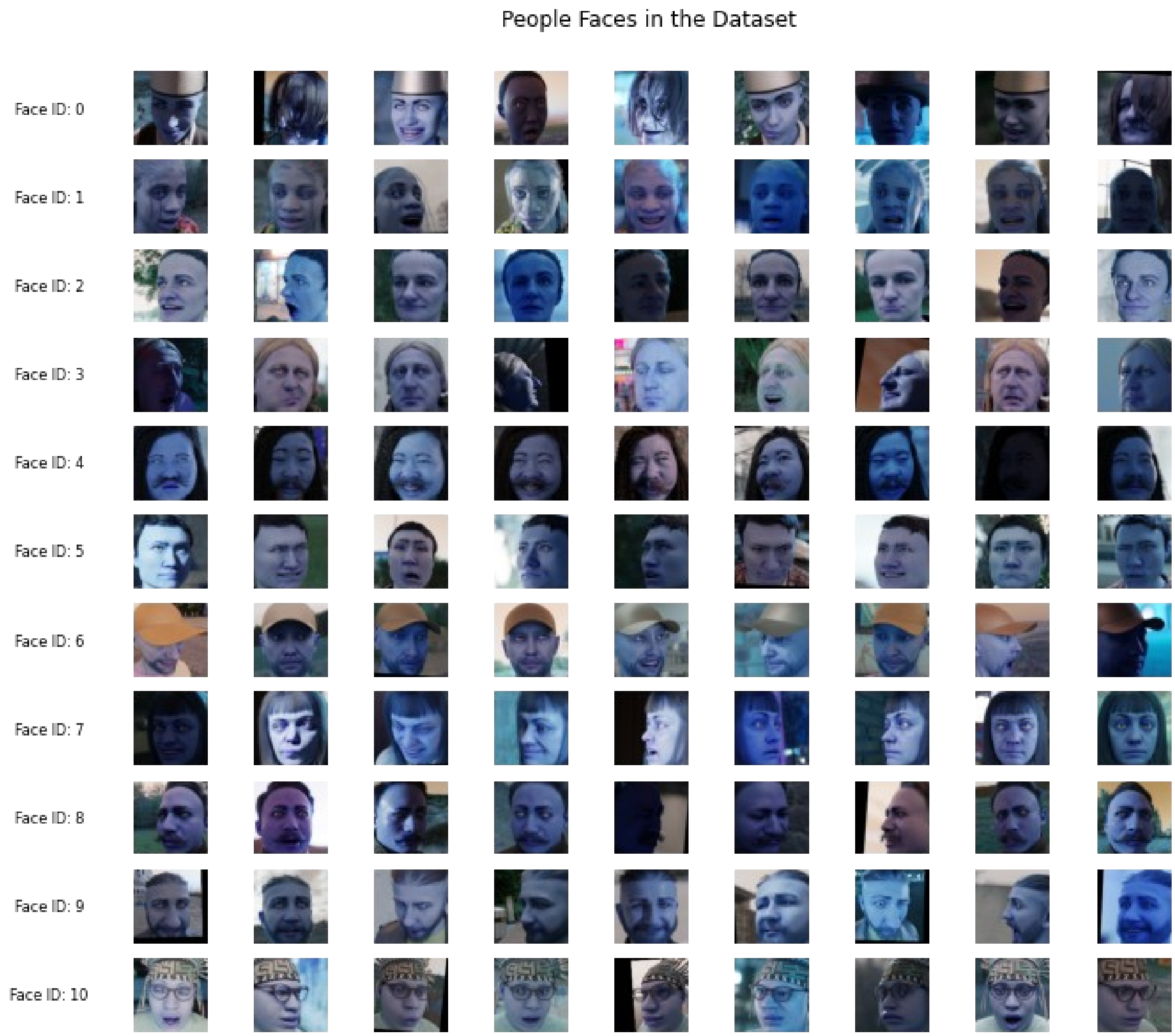
```
# Specify the path to the ZIP file and the extraction directory
zip_file = "subjects_0-1999_72_imgs.zip"
extraction_dir = "dataset"

# Create the extraction directory if it doesn't exist
if not os.path.exists(extraction_dir):
    os.makedirs(extraction_dir)

# Create the '0' folder if it doesn't exist
target_folder = "0"
target_folder_path = os.path.join(extraction_dir, target_folder)
if not os.path.exists(target_folder_path):
    os.makedirs(target_folder_path)

# Open the ZIP file and extract one image from each folder in a cycle
until 72 images are extracted
with zipfile.ZipFile(zip_file, 'r') as zip_ref:
    # Shuffle the list of target folders to randomize folder selection
    random.shuffle(target_folders)

    files_extracted = 0
    while files_extracted < 73:
        for folder_name in target_folders:
            folder_path = f"{folder_name}/"

            # Get a list of all files in the current folder
            folder_files = [member.filename for member in
zip_ref.infolist() if member.filename.startswith(folder_path)]

            if folder_files:
                # Shuffle the list to randomize file selection
                random.shuffle(folder_files)

                # Extract one image from the folder
                filename = folder_files[0]
                target_path = os.path.join(target_folder_path,
os.path.basename(filename))

                with zip_ref.open(filename) as source,
open(target_path, "wb") as target:
                    shutil.copyfileobj(source, target)

                files_extracted += 1

                if files_extracted >= 73:
                    break

print(f"Extraction of one image from each folder in a cycle until 72
images are extracted into '0' folder complete.")
```

```
Extraction of one image from each folder in a cycle until 72 images
are extracted into '0' folder complete.
```

```python
#!unzip "/kaggle/working/subjects_0-1999_72_imgs.zip" -d
"/kaggle/working/dataset"

# Libraries

# Main
import os
import glob
import gc
import numpy as np
import pandas as pd
import cv2
import time
import random

# Visualization
import matplotlib
import matplotlib.pyplot as plt
import plotly
import plotly.graph_objects as go
import plotly.express as px
from plotly.subplots import make_subplots

# Deep Learning
import tensorflow as tf
from keras.models import load_model, Model
from keras.layers import Dense
from keras.optimizers import Adam
from keras.preprocessing import image as k_image
from keras.preprocessing.image import ImageDataGenerator
from sklearn.model_selection import train_test_split

# Warning
import warnings
warnings.filterwarnings("ignore")

class CFG:
    batch_size = 8
    img_height = 160
    img_width = 160
    epoch = 40

def seed_everything(seed: int):
    random.seed(seed)
    os.environ["PYTHONHASHSEED"] = str(seed)
    np.random.seed(seed)
    tf.random.set_seed(seed)
```

```
DATASET_PATH = "dataset"
identities = [0, 1, 2, 3, 4, 5, 6, 7, 8, 9, 10]
identities = [str(identity) for identity in identities]
list_path = []
labels = []
for identity in identities:
    identity_path = os.path.join(DATASET_PATH, identity, "*")
    image_files = glob.glob(identity_path)
    identity_label = [identity] * len(image_files)
    list_path.extend(image_files)
    labels.extend(identity_label)

data = pd.DataFrame({
    "image_path": list_path,
    "identity": labels
})

data

            image_path identity
0      dataset\0\0.png        0
1      dataset\0\1.png        0
2     dataset\0\10.png        0
3     dataset\0\11.png        0
4     dataset\0\16.png        0
..                 ...      ...
764   dataset\10\7.png       10
765  dataset\10\70.png       10
766  dataset\10\71.png       10
767   dataset\10\8.png       10
768   dataset\10\9.png       10

[769 rows x 2 columns]

labels = range(len(identities))

# Create Subplots
fig, axs = plt.subplots(11, 10, figsize=(12, 10))

for i, identity in enumerate(identities):
    axs[i, 0].text(0.5, 0.5, "Face ID: {}".format(identity),
ha='center', va='center', fontsize=8)
    axs[i, 0].axis('off')

    identity_data =
data[data["identity"]==identity].reset_index(drop=True)
    for j in range(9):
        img_face = cv2.imread(identity_data["image_path"][j])
        axs[i, j+1].imshow(img_face)
        axs[i, j+1].axis("off")
```

```
# Title
plt.suptitle("People Faces in the Dataset", x=0.55, y=0.93)

# Show
plt.show()
```




```
# Data Splitting (Training: 0.8, Testing: 0.2)
X_train, X_test, y_train, y_test = train_test_split(
    data["image_path"], data["identity"],
    test_size=0.2,
    random_state=2023,
    shuffle=True,
    stratify=data["identity"]
)
data_train = pd.DataFrame({
```

```
    "image_path": X_train,
    "identity": y_train
})
data_test = pd.DataFrame({
    "image_path": X_test,
    "identity": y_test
})

print(data_test)
```

```
            image_path identity
110   dataset\1\70.png        1
430   dataset\6\34.png        6
49    dataset\1\15.png        1
672   dataset\9\58.png        9
616    dataset\8\8.png        8
..                 ...      ...
711  dataset\10\28.png       10
579   dataset\8\39.png        8
73    dataset\1\37.png        1
554   dataset\8\16.png        8
5     dataset\0\20.png        0

[153 rows x 2 columns]
```

```
'''# Find the minimum count of rows for a specific identity
df = data_test
min_count = df['identity'].value_counts().min()
# Create a list to store DataFrames after balancing
balanced_dfs = []

# Iterate over unique identities
for identity, group in df.groupby('identity'):
    # Randomly sample rows to match the minimum count
    balanced_df = group.sample(n=min_count, random_state=42)  # You
can change the random_state if needed
    balanced_dfs.append(balanced_df)

# Concatenate the balanced DataFrames back together
balanced_df = pd.concat(balanced_dfs)
balanced_df['identity'].value_counts()
data_test = balanced_df
'''
```

```
"# Find the minimum count of rows for a specific identity\ndf =
data_test\nmin_count = df['identity'].value_counts().min()\n# Create a
list to store DataFrames after balancing\nbalanced_dfs = []\n\n#
Iterate over unique identities\nfor identity, group in
df.groupby('identity'):\n    # Randomly sample rows to match the
minimum count\n    balanced_df = group.sample(n=min_count,
```

```
random_state=42)  # You can change the random_state if needed\n
balanced_dfs.append(balanced_df)\n\n# Concatenate the balanced
DataFrames back together\nbalanced_df = pd.concat(balanced_dfs)\
nbalanced_df['identity'].value_counts()\ndata_test = balanced_df\n"

# Data Augmentation
def data_augmentation():
    # Training Dataset
    train_datagen = ImageDataGenerator(
        rescale=1/255.,
        rotation_range=20,
        width_shift_range=0.2,
        height_shift_range=0.2,
        brightness_range=[0.0, 0.25],
        horizontal_flip=True,
        fill_mode='nearest',
        validation_split=0.2
    )
    train_generator = train_datagen.flow_from_dataframe(
        data_train,
        directory="./",
        x_col="image_path",
        y_col="identity",
        subset="training",
        class_mode="categorical",
        batch_size=CFG.batch_size,
        target_size=(CFG.img_height, CFG.img_width),
    )

    # Validation Dataset
    validation_generator = train_datagen.flow_from_dataframe(
        data_train,
        directory="./",
        x_col="image_path",
        y_col="identity",
        subset="validation",
        class_mode="categorical",
        batch_size=CFG.batch_size,
        target_size=(CFG.img_height, CFG.img_width),
    )

    # Testing Dataset
    test_datagen = ImageDataGenerator(rescale=1/255.,)

    test_generator = test_datagen.flow_from_dataframe(
        data_test,
        directory="./",
        x_col="image_path",
        y_col="identity",
        class_mode="categorical",
```

```
        batch_size=1,
        target_size=(CFG.img_height, CFG.img_width),
        shuffle=False
    )

    return train_generator, validation_generator, test_generator
```

```
# Create Data Augmentation
seed_everything(2023)
train_generator, validation_generator, test_generator =
data_augmentation()
```

```
Found 480 validated image filenames belonging to 11 classes.
Found 120 validated image filenames belonging to 11 classes.
Found 151 validated image filenames belonging to 11 classes.
```

```
type(test_generator)
```

```
keras.src.preprocessing.image.DataFrameIterator
```

```
test_generator.classes
```

```
[1,
 7,
 1,
 10,
 9,
 8,
 2,
 7,
 5,
 5,
 10,
 3,
 1,
 6,
 4,
 3,
 6,
 3,
 10,
 2,
 2,
 5,
 0,
 7,
 8,
 4,
 5,
 10,
 9,
```

```
6,
5,
0,
10,
6,
1,
7,
5,
10,
2,
7,
4,
10,
8,
4,
8,
5,
0,
2,
7,
2,
4,
3,
5,
10,
2,
7,
3,
10,
3,
9,
7,
5,
2,
9,
4,
4,
1,
6,
3,
2,
3,
6,
8,
9,
4,
7,
3,
10,
```

```
2,
3,
10,
5,
4,
0,
0,
9,
9,
1,
5,
1,
10,
1,
8,
5,
7,
10,
8,
7,
4,
7,
4,
8,
9,
2,
8,
6,
8,
10,
0,
1,
2,
3,
9,
7,
2,
5,
1,
2,
4,
9,
10,
6,
0,
5,
7,
8,
3,
```

```
 6,
 1,
 1,
 6,
 8,
 1,
 6,
 8,
 6,
 8,
 6,
 6,
 5,
 4,
 9,
 3,
 3,
 8,
 7,
 4,
 9,
 2,
 9,
 1,
 9,
 0]
```

```
from functools import partial

from keras.models import Model
from keras.layers import Activation
from keras.layers import BatchNormalization
from keras.layers import Concatenate
from keras.layers import Conv2D
from keras.layers import Dense
from keras.layers import Dropout
from keras.layers import GlobalAveragePooling2D
from keras.layers import Input
from keras.layers import Lambda
from keras.layers import MaxPooling2D
from keras.layers import add
from keras import backend as K


def scaling(x, scale):
    return x * scale


def conv2d_bn(x,
              filters,
```

```
               kernel_size,
               strides=1,
               padding='same',
               activation='relu',
               use_bias=False,
               name=None):
    x = Conv2D(filters,
                kernel_size,
                strides=strides,
                padding=padding,
                use_bias=use_bias,
                name=name)(x)
    if not use_bias:
        bn_axis = 1 if K.image_data_format() == 'channels_first' else
3
        bn_name = _generate_layer_name('BatchNorm', prefix=name)
        x = BatchNormalization(axis=bn_axis, momentum=0.995,
epsilon=0.001,
                               scale=False, name=bn_name)(x)
    if activation is not None:
        ac_name = _generate_layer_name('Activation', prefix=name)
        x = Activation(activation, name=ac_name)(x)
    return x


def _generate_layer_name(name, branch_idx=None, prefix=None):
    if prefix is None:
        return None
    if branch_idx is None:
        return '_'.join((prefix, name))
    return '_'.join((prefix, 'Branch', str(branch_idx), name))


def _inception_resnet_block(x, scale, block_type, block_idx,
activation='relu'):
    channel_axis = 1 if K.image_data_format() == 'channels_first' else
3
    if block_idx is None:
        prefix = None
    else:
        prefix = '_'.join((block_type, str(block_idx)))
    name_fmt = partial(_generate_layer_name, prefix=prefix)

    if block_type == 'Block35':
        branch_0 = conv2d_bn(x, 32, 1, name=name_fmt('Conv2d_1x1', 0))
        branch_1 = conv2d_bn(x, 32, 1, name=name_fmt('Conv2d_0a_1x1',
1))
        branch_1 = conv2d_bn(branch_1, 32, 3,
name=name_fmt('Conv2d_0b_3x3', 1))
        branch_2 = conv2d_bn(x, 32, 1, name=name_fmt('Conv2d_0a_1x1',
```

```
2))
        branch_2 = conv2d_bn(branch_2, 32, 3,
name=name_fmt('Conv2d_0b_3x3', 2))
        branch_2 = conv2d_bn(branch_2, 32, 3,
name=name_fmt('Conv2d_0c_3x3', 2))
        branches = [branch_0, branch_1, branch_2]
    elif block_type == 'Block17':
        branch_0 = conv2d_bn(x, 128, 1, name=name_fmt('Conv2d_1x1',
0))
        branch_1 = conv2d_bn(x, 128, 1, name=name_fmt('Conv2d_0a_1x1',
1))
        branch_1 = conv2d_bn(branch_1, 128, [1, 7],
name=name_fmt('Conv2d_0b_1x7', 1))
        branch_1 = conv2d_bn(branch_1, 128, [7, 1],
name=name_fmt('Conv2d_0c_7x1', 1))
        branches = [branch_0, branch_1]
    elif block_type == 'Block8':
        branch_0 = conv2d_bn(x, 192, 1, name=name_fmt('Conv2d_1x1',
0))
        branch_1 = conv2d_bn(x, 192, 1, name=name_fmt('Conv2d_0a_1x1',
1))
        branch_1 = conv2d_bn(branch_1, 192, [1, 3],
name=name_fmt('Conv2d_0b_1x3', 1))
        branch_1 = conv2d_bn(branch_1, 192, [3, 1],
name=name_fmt('Conv2d_0c_3x1', 1))
        branches = [branch_0, branch_1]
    else:
        raise ValueError('Unknown Inception-ResNet block type. '
                         'Expects "Block35", "Block17" or "Block8", '
                         'but got: ' + str(block_type))

    mixed = Concatenate(axis=channel_axis,
name=name_fmt('Concatenate'))(branches)
    up = conv2d_bn(mixed,
                   K.int_shape(x)[channel_axis],
                   1,
                   activation=None,
                   use_bias=True,
                   name=name_fmt('Conv2d_1x1'))
    up = Lambda(scaling,
                output_shape=K.int_shape(up)[1:],
                arguments={'scale': scale})(up)
    x = add([x, up])
    if activation is not None:
        x = Activation(activation, name=name_fmt('Activation'))(x)
    return x


def InceptionResNetV1(input_shape=(160, 160, 3),
                      classes=128,
```

```
                       dropout_keep_prob=0.8,
                       weights_path=None):
    inputs = Input(shape=input_shape)
    x = conv2d_bn(inputs, 32, 3, strides=2, padding='valid',
name='Conv2d_1a_3x3')
    x = conv2d_bn(x, 32, 3, padding='valid', name='Conv2d_2a_3x3')
    x = conv2d_bn(x, 64, 3, name='Conv2d_2b_3x3')
    x = MaxPooling2D(3, strides=2, name='MaxPool_3a_3x3')(x)
    x = conv2d_bn(x, 80, 1, padding='valid', name='Conv2d_3b_1x1')
    x = conv2d_bn(x, 192, 3, padding='valid', name='Conv2d_4a_3x3')
    x = conv2d_bn(x, 256, 3, strides=2, padding='valid',
name='Conv2d_4b_3x3')

    # 5x Block35 (Inception-ResNet-A block):
    for block_idx in range(1, 6):
        x = _inception_resnet_block(x,
                                    scale=0.17,
                                    block_type='Block35',
                                    block_idx=block_idx)

    # Mixed 6a (Reduction-A block):
    channel_axis = 1 if K.image_data_format() == 'channels_first' else
3
    name_fmt = partial(_generate_layer_name, prefix='Mixed_6a')
    branch_0 = conv2d_bn(x,
                         384,
                         3,
                         strides=2,
                         padding='valid',
                         name=name_fmt('Conv2d_1a_3x3', 0))
    branch_1 = conv2d_bn(x, 192, 1, name=name_fmt('Conv2d_0a_1x1', 1))
    branch_1 = conv2d_bn(branch_1, 192, 3,
name=name_fmt('Conv2d_0b_3x3', 1))
    branch_1 = conv2d_bn(branch_1,
                         256,
                         3,
                         strides=2,
                         padding='valid',
                         name=name_fmt('Conv2d_1a_3x3', 1))
    branch_pool = MaxPooling2D(3,
                               strides=2,
                               padding='valid',
                               name=name_fmt('MaxPool_1a_3x3', 2))(x)
    branches = [branch_0, branch_1, branch_pool]
    x = Concatenate(axis=channel_axis, name='Mixed_6a')(branches)

    # 10x Block17 (Inception-ResNet-B block):
    for block_idx in range(1, 11):
        x = _inception_resnet_block(x,
                                    scale=0.1,
```

```
                                    block_type='Block17',
                                    block_idx=block_idx)

    # Mixed 7a (Reduction-B block): 8 x 8 x 2080
    name_fmt = partial(_generate_layer_name, prefix='Mixed_7a')
    branch_0 = conv2d_bn(x, 256, 1, name=name_fmt('Conv2d_0a_1x1', 0))
    branch_0 = conv2d_bn(branch_0,
                         384,
                         3,
                         strides=2,
                         padding='valid',
                         name=name_fmt('Conv2d_1a_3x3', 0))
    branch_1 = conv2d_bn(x, 256, 1, name=name_fmt('Conv2d_0a_1x1', 1))
    branch_1 = conv2d_bn(branch_1,
                         256,
                         3,
                         strides=2,
                         padding='valid',
                         name=name_fmt('Conv2d_1a_3x3', 1))
    branch_2 = conv2d_bn(x, 256, 1, name=name_fmt('Conv2d_0a_1x1', 2))
    branch_2 = conv2d_bn(branch_2, 256, 3,
name=name_fmt('Conv2d_0b_3x3', 2))
    branch_2 = conv2d_bn(branch_2,
                         256,
                         3,
                         strides=2,
                         padding='valid',
                         name=name_fmt('Conv2d_1a_3x3', 2))
    branch_pool = MaxPooling2D(3,
                               strides=2,
                               padding='valid',
                               name=name_fmt('MaxPool_1a_3x3', 3))(x)
    branches = [branch_0, branch_1, branch_2, branch_pool]
    x = Concatenate(axis=channel_axis, name='Mixed_7a')(branches)

    # 5x Block8 (Inception-ResNet-C block):
    for block_idx in range(1, 6):
        x = _inception_resnet_block(x,
                                    scale=0.2,
                                    block_type='Block8',
                                    block_idx=block_idx)
    x = _inception_resnet_block(x,
                                scale=1.,
                                activation=None,
                                block_type='Block8',
                                block_idx=6)

    # Classification block
    x = GlobalAveragePooling2D(name='AvgPool')(x)
    x = Dropout(1.0 - dropout_keep_prob, name='Dropout')(x)
```

```
    # Bottleneck
    x = Dense(classes, use_bias=False, name='Bottleneck')(x)
    bn_name = _generate_layer_name('BatchNorm', prefix='Bottleneck')
    x = BatchNormalization(momentum=0.995, epsilon=0.001, scale=False,
                           name=bn_name)(x)

    # Create model
    model = Model(inputs, x, name='inception_resnet_v1')
    if weights_path is not None:
        model.load_weights(weights_path)

    return model

# Load pre-trained FaceNet model
model_weights_path = "model_weights.h5"
facenet_model = InceptionResNetV1(weights_path=model_weights_path)

# Remove the last layer to fine-tune the model
facenet_model = Model(inputs=facenet_model.inputs,
outputs=facenet_model.layers[-2].output)

# Add a new dense layer with 10 outputs (for 10 identities) and
softmax activation
output_layer = Dense(11, activation='softmax')(facenet_model.output)
fine_tuned_model = Model(facenet_model.input, output_layer)

# Compile the model
fine_tuned_model.compile(
    optimizer=Adam(lr=0.001),
    loss='categorical_crossentropy',
    metrics=['accuracy']
)
```

```
WARNING:absl:`lr` is deprecated in Keras optimizer, please use
`learning_rate` or use the legacy optimizer,
e.g.,tf.keras.optimizers.legacy.Adam.
```

```
# Fine-tune the model
history = fine_tuned_model.fit(
    train_generator,
    steps_per_epoch=train_generator.samples // CFG.batch_size,
    epochs=CFG.epoch,
    validation_data=validation_generator,
    validation_steps=validation_generator.samples // CFG.batch_size
)
```

```
Epoch 1/40
61/61 [==============================] - 154s 2s/step - loss: 2.6587 -
accuracy: 0.4672 - val_loss: 3.1578 - val_accuracy: 0.0833
Epoch 2/40
61/61 [==============================] - 119s 2s/step - loss: 1.0854 -
```

```
accuracy: 0.7172 - val_loss: 2.7288 - val_accuracy: 0.0833
Epoch 3/40
61/61 [==============================] - 119s 2s/step - loss: 1.0185 -
accuracy: 0.7254 - val_loss: 9.0026 - val_accuracy: 0.0750
Epoch 4/40
61/61 [==============================] - 119s 2s/step - loss: 0.8844 -
accuracy: 0.7766 - val_loss: 2.7152 - val_accuracy: 0.0833
Epoch 5/40
61/61 [==============================] - 120s 2s/step - loss: 0.7845 -
accuracy: 0.7869 - val_loss: 2.9232 - val_accuracy: 0.1167
Epoch 6/40
61/61 [==============================] - 118s 2s/step - loss: 0.5320 -
accuracy: 0.8381 - val_loss: 2.8034 - val_accuracy: 0.1500
Epoch 7/40
61/61 [==============================] - 118s 2s/step - loss: 0.3550 -
accuracy: 0.8934 - val_loss: 2.4007 - val_accuracy: 0.2167
Epoch 8/40
61/61 [==============================] - 124s 2s/step - loss: 0.4262 -
accuracy: 0.8668 - val_loss: 2.1876 - val_accuracy: 0.2417
Epoch 9/40
61/61 [==============================] - 122s 2s/step - loss: 0.4641 -
accuracy: 0.8320 - val_loss: 1.9422 - val_accuracy: 0.3333
Epoch 10/40
61/61 [==============================] - 123s 2s/step - loss: 0.3899 -
accuracy: 0.8832 - val_loss: 2.3450 - val_accuracy: 0.2333
Epoch 11/40
61/61 [==============================] - 125s 2s/step - loss: 0.3962 -
accuracy: 0.8770 - val_loss: 2.6632 - val_accuracy: 0.1333
Epoch 12/40
61/61 [==============================] - 127s 2s/step - loss: 0.4969 -
accuracy: 0.8689 - val_loss: 2.5026 - val_accuracy: 0.2083
Epoch 13/40
61/61 [==============================] - 121s 2s/step - loss: 0.6397 -
accuracy: 0.8299 - val_loss: 2.3048 - val_accuracy: 0.3500
Epoch 14/40
61/61 [==============================] - 123s 2s/step - loss: 0.5352 -
accuracy: 0.8545 - val_loss: 1.9455 - val_accuracy: 0.4083
Epoch 15/40
61/61 [==============================] - 122s 2s/step - loss: 0.4692 -
accuracy: 0.8586 - val_loss: 1.5418 - val_accuracy: 0.4583
Epoch 16/40
61/61 [==============================] - 120s 2s/step - loss: 0.3279 -
accuracy: 0.9037 - val_loss: 1.4228 - val_accuracy: 0.5833
Epoch 17/40
61/61 [==============================] - 119s 2s/step - loss: 0.4606 -
accuracy: 0.8607 - val_loss: 2.9779 - val_accuracy: 0.3000
Epoch 18/40
61/61 [==============================] - 128s 2s/step - loss: 0.4029 -
accuracy: 0.8750 - val_loss: 2.0198 - val_accuracy: 0.5583
```

```
Epoch 19/40
61/61 [==============================] - 127s 2s/step - loss: 0.3349 -
accuracy: 0.8975 - val_loss: 0.8547 - val_accuracy: 0.7417
Epoch 20/40
61/61 [==============================] - 119s 2s/step - loss: 0.3069 -
accuracy: 0.9139 - val_loss: 1.9090 - val_accuracy: 0.5333
Epoch 21/40
61/61 [==============================] - 116s 2s/step - loss: 0.3028 -
accuracy: 0.9037 - val_loss: 1.0493 - val_accuracy: 0.6583
Epoch 22/40
61/61 [==============================] - 118s 2s/step - loss: 0.2386 -
accuracy: 0.9324 - val_loss: 0.9403 - val_accuracy: 0.8083
Epoch 23/40
61/61 [==============================] - 118s 2s/step - loss: 0.2707 -
accuracy: 0.9242 - val_loss: 0.6487 - val_accuracy: 0.8250
Epoch 24/40
61/61 [==============================] - 122s 2s/step - loss: 0.5242 -
accuracy: 0.8484 - val_loss: 0.7296 - val_accuracy: 0.7500
Epoch 25/40
61/61 [==============================] - 120s 2s/step - loss: 0.3224 -
accuracy: 0.8975 - val_loss: 2.5428 - val_accuracy: 0.5583
Epoch 26/40
61/61 [==============================] - 119s 2s/step - loss: 0.3668 -
accuracy: 0.8791 - val_loss: 1.5347 - val_accuracy: 0.7583
Epoch 27/40
61/61 [==============================] - 119s 2s/step - loss: 0.3062 -
accuracy: 0.9119 - val_loss: 0.5679 - val_accuracy: 0.8333
Epoch 28/40
61/61 [==============================] - 119s 2s/step - loss: 0.2896 -
accuracy: 0.9242 - val_loss: 0.6061 - val_accuracy: 0.8083
Epoch 29/40
61/61 [==============================] - 117s 2s/step - loss: 0.2618 -
accuracy: 0.9201 - val_loss: 0.5518 - val_accuracy: 0.8333
Epoch 30/40
61/61 [==============================] - 116s 2s/step - loss: 0.3492 -
accuracy: 0.9078 - val_loss: 3.1604 - val_accuracy: 0.6167
Epoch 31/40
61/61 [==============================] - 113s 2s/step - loss: 0.2227 -
accuracy: 0.9344 - val_loss: 0.5724 - val_accuracy: 0.8583
Epoch 32/40
61/61 [==============================] - 113s 2s/step - loss: 0.2803 -
accuracy: 0.9180 - val_loss: 0.7346 - val_accuracy: 0.8000
Epoch 33/40
61/61 [==============================] - 112s 2s/step - loss: 0.2306 -
accuracy: 0.9221 - val_loss: 0.5939 - val_accuracy: 0.8417
Epoch 34/40
61/61 [==============================] - 113s 2s/step - loss: 0.2107 -
accuracy: 0.9447 - val_loss: 0.5308 - val_accuracy: 0.8500
Epoch 35/40
```

```
61/61 [==============================] - 115s 2s/step - loss: 0.2624 -
accuracy: 0.9201 - val_loss: 0.5389 - val_accuracy: 0.8500
Epoch 36/40
61/61 [==============================] - 113s 2s/step - loss: 0.2901 -
accuracy: 0.9324 - val_loss: 0.7665 - val_accuracy: 0.7917
Epoch 37/40
61/61 [==============================] - 113s 2s/step - loss: 0.2029 -
accuracy: 0.9324 - val_loss: 0.7550 - val_accuracy: 0.7667
Epoch 38/40
61/61 [==============================] - 113s 2s/step - loss: 0.2583 -
accuracy: 0.9180 - val_loss: 0.6938 - val_accuracy: 0.8000
Epoch 39/40
61/61 [==============================] - 113s 2s/step - loss: 0.2525 -
accuracy: 0.9119 - val_loss: 0.8094 - val_accuracy: 0.7750
Epoch 40/40
61/61 [==============================] - 117s 2s/step - loss: 0.4423 -
accuracy: 0.8709 - val_loss: 0.9974 - val_accuracy: 0.7583

# Evaluating the model performance using the test set
test_loss, test_acc = fine_tuned_model.evaluate(test_generator)
print('Test accuracy:', test_acc)

153/153 [==============================] - 14s 93ms/step - loss:
1.0284 - accuracy: 0.7451
Test accuracy: 0.7450980544090271

# Save the fine-tuned model
fine_tuned_model.save("fine_tuned_facenet.h5")

# Visualize Training and Validation Results

# Create Subplot
fig = make_subplots(
        rows=1, cols=2,
        subplot_titles=["Model Loss", "Model Accuracy"],
)

# Configuration Plot
class PlotCFG:
    marker_size = 5
    line_size = 1.5
    train_color = "#1b222c"
    valid_color = "#3a5e66"

# Loss Plot
loss = history.history['loss']
val_loss = history.history['val_loss']
fig.add_trace(
    go.Scatter(
        x=np.arange(1, len(loss)+1), y=loss,
```

```python
        mode="markers+lines",
        marker=dict(
            color=PlotCFG.train_color, size=PlotCFG.marker_size,
            line=dict(color="White", width=0.5)
        ),
        line=dict(color=PlotCFG.train_color, width=PlotCFG.line_size),
        name="Training Loss"
    ), row=1, col=1
)
fig.add_trace(
    go.Scatter(
        x=np.arange(1, len(val_loss)+1), y=val_loss,
        mode="markers+lines",
        marker=dict(
            color=PlotCFG.valid_color, size=PlotCFG.marker_size,
            line=dict(color="White", width=0.5)
        ),
        line=dict(color=PlotCFG.valid_color, width=PlotCFG.line_size),
        name="Validation Loss"
    ), row=1, col=1
)

# Accuracy Plot
acc = history.history['accuracy']
val_acc = history.history['val_accuracy']
fig.add_trace(
    go.Scatter(
        x=np.arange(1, len(acc)+1), y=acc,
        mode="markers+lines",
        marker=dict(
            color=PlotCFG.train_color, size=PlotCFG.marker_size,
            line=dict(color="White", width=0.5)
        ),
        line=dict(color=PlotCFG.train_color, width=PlotCFG.line_size),
        name="Training Accuracy"
    ), row=1, col=2
)
fig.add_trace(
    go.Scatter(
        x=np.arange(1, len(val_acc)+1), y=val_acc,
        mode="markers+lines",
        marker=dict(
            color=PlotCFG.valid_color, size=PlotCFG.marker_size,
            line=dict(color="White", width=0.5)
        ),
        line=dict(color=PlotCFG.valid_color, width=PlotCFG.line_size),
        name="Validation Accuracy"
    ), row=1, col=2
)
```

```python
# Update Axes
fig.update_xaxes(title="Epochs", linecolor="Black", ticks="outside",
row=1, col=1)
fig.update_xaxes(title="Epochs", linecolor="Black", ticks="outside",
row=1, col=2)
fig.update_yaxes(title="Categorical Loss", linecolor="Black",
ticks="outside", row=1, col=1)
fig.update_yaxes(title="Accuracy", linecolor="Black", ticks="outside",
row=1, col=2)

# Update Layout
fig.update_layout(
    title="Training Loss and Metrics", title_x=0.5,
font_family="Cambria",
    width=950, height=400,
    showlegend=False,
    plot_bgcolor="White",
    paper_bgcolor="White"
)
# Show
fig.show(iframe_connected=True)
```


```
{"config":{"plotlyServerURL":"https://plot.ly"},"data":[{"line":
{"color":"#1b222c","width":1.5},"marker":{"color":"#1b222c","line":
{"color":"White","width":0.5},"size":5},"mode":"markers+lines","name":
"Training Loss","type":"scatter","x":
[1,2,3,4,5,6,7,8,9,10,11,12,13,14,15,16,17,18,19,20,21,22,23,24,25,26,
27,28,29,30,31,32,33,34,35,36,37,38,39,40],"xaxis":"x","y":
[2.658734083175659,1.0854458808898926,1.0184762477874756,0.88444876670
8374,0.7844668626785278,0.5319828987121582,0.3549518585205078,0.426177
7997016907,0.46413615345954895,0.38994964957237244,0.3961596190929413,
0.4969203770160675,0.6396645903587341,0.5351591110229492,0.46915945410
728455,0.3279434144496918,0.46057701110839844,0.40287187695503235,0.33
486178517341614,0.3068532645702362,0.3027722239494324,0.23859013617038
727,0.2706579864025116,0.5241669416427612,0.32242563366889954,0.366773
5755443573,0.3061831593513489,0.28955525159835815,0.2618418335914612,0
.3491954803466797,0.22265024483203888,0.2802782654762268,0.23055362701
416016,0.21065977215766907,0.26242318749427795,0.29005929827690125,0.2
028704434633255,0.2583409249782562,0.25252246856689453,0.4422760605812
073],"yaxis":"y"},{"line":{"color":"#3a5e66","width":1.5},"marker":
{"color":"#3a5e66","line":
{"color":"White","width":0.5},"size":5},"mode":"markers+lines","name":
"Validation Loss","type":"scatter","x":
[1,2,3,4,5,6,7,8,9,10,11,12,13,14,15,16,17,18,19,20,21,22,23,24,25,26,
27,28,29,30,31,32,33,34,35,36,37,38,39,40],"xaxis":"x","y":
[3.15779185295105,2.7288479804992676,9.002631187438965,2.7151882648468
018,2.9232444763183594,2.8034110069274902,2.4006717205047607,2.1876351
833343506,1.942201852798462,2.34499454498291,2.6632304191589355,2.5026
357173919678,2.304795503616333,1.9455101490020752,1.5417946577072144,1
```

```
.422790288925171,2.9778521060943604,2.019812822341919,0.85474205017089
84,1.909042239189148,1.0492854118347168,0.9402524828910828,0.648715078
830719,0.7296163439750671,2.542809009552002,1.5347075462341309,0.56793
02215576172,0.606087863445282,0.551791250705719,3.1604373455047607,0.5
723728537559509,0.734624981880188,0.5938799381256104,0.530821740627288
8,0.5389016270637512,0.7664509415626526,0.7550273537635803,0.693844616
4131165,0.8093607425689697,0.9973871111869812],"yaxis":"y"},{"line":
{"color":"#1b222c","width":1.5},"marker":{"color":"#1b222c","line":
{"color":"White","width":0.5},"size":5},"mode":"markers+lines","name":
"Training Accuracy","type":"scatter","x":
[1,2,3,4,5,6,7,8,9,10,11,12,13,14,15,16,17,18,19,20,21,22,23,24,25,26,
27,28,29,30,31,32,33,34,35,36,37,38,39,40],"xaxis":"x2","y":
[0.46721312403678894,0.7172130942344666,0.7254098653793335,0.776639342
3080444,0.7868852615356445,0.8381147384643555,0.8934426307678223,0.866
8032884597778,0.8319672346115112,0.8831967115402222,0.8770492076873779
,0.868852436542511,0.8299180269241333,0.8545082211494446,0.85860657691
95557,0.9036885499954224,0.8606557250022888,0.875,0.8975409865379333,0
.9139344096183777,0.9036885499954224,0.9323770403862,0.924180328845977
8,0.8483606576919556,0.8975409865379333,0.8790983557701111,0.911885261
5356445,0.9241803288459778,0.9200819730758667,0.9077869057655334,0.934
4262480735779,0.9180327653884888,0.9221311211585999,0.9446721076965332
,0.9200819730758667,0.9323770403862,0.9323770403862,0.9180327653884888
,0.9118852615356445,0.8709016442298889],"yaxis":"y2"},{"line":
{"color":"#3a5e66","width":1.5},"marker":{"color":"#3a5e66","line":
{"color":"White","width":0.5},"size":5},"mode":"markers+lines","name":
"Validation Accuracy","type":"scatter","x":
[1,2,3,4,5,6,7,8,9,10,11,12,13,14,15,16,17,18,19,20,21,22,23,24,25,26,
27,28,29,30,31,32,33,34,35,36,37,38,39,40],"xaxis":"x2","y":
[8.33333358168602e-2,8.33333358168602e-2,7.500000298023224e-
2,8.33333358168602e-
2,0.11666666716337204,0.15000000596046448,0.21666666865348816,0.241666
65971279144,0.3333333432674408,0.2333333432674408,0.13333334028720856
,0.2083333283662796,0.3499999940395355,0.40833333134651184,0.458333343
2674408,0.5833333134651184,0.30000001192092896,0.5583333373069763,0.74
16666746139526,0.5333333611488342,0.6583333611488342,0.808333337306976
3,0.824999988079071,0.75,0.5583333373069763,0.7583333253860474,0.83333
33134651184,0.8083333373069763,0.8333333134651184,0.6166666746139526,0
.8583333492279053,0.800000011920929,0.8416666388511658,0.8500000238418
579,0.8500000238418579,0.7916666865348816,0.7666666507720947,0.8000000
11920929,0.7749999761581421,0.7583333253860474],"yaxis":"y2"}],"layout
":{"annotations":[{"font":{"size":16},"showarrow":false,"text":"Model
Loss","x":0.225,"xanchor":"center","xref":"paper","y":1,"yanchor":"bot
tom","yref":"paper"},{"font":
{"size":16},"showarrow":false,"text":"Model
Accuracy","x":0.775,"xanchor":"center","xref":"paper","y":1,"yanchor":
"bottom","yref":"paper"}],"font":
{"family":"Cambria"},"height":400,"paper_bgcolor":"White","plot_bgcolo
r":"White","showlegend":false,"template":{"data":{"bar":[{"error_x":
{"color":"#2a3f5f"},"error_y":{"color":"#2a3f5f"},"marker":{"line":
```

```
{"color":"#E5ECF6","width":0.5},"pattern":
{"fillmode":"overlay","size":10,"solidity":0.2}},"type":"bar"}],"barpo
lar":[{"marker":{"line":{"color":"#E5ECF6","width":0.5},"pattern":
{"fillmode":"overlay","size":10,"solidity":0.2}},"type":"barpolar"}],"
carpet":[{"aaxis":
{"endlinecolor":"#2a3f5f","gridcolor":"white","linecolor":"white","min
orgridcolor":"white","startlinecolor":"#2a3f5f"},"baxis":
{"endlinecolor":"#2a3f5f","gridcolor":"white","linecolor":"white","min
orgridcolor":"white","startlinecolor":"#2a3f5f"},"type":"carpet"}],"ch
oropleth":[{"colorbar":
{"outlinewidth":0,"ticks":""},"type":"choropleth"}],"contour":
[{"colorbar":{"outlinewidth":0,"ticks":""},"colorscale":
[[0,"#0d0887"],[0.1111111111111111,"#46039f"],
[0.2222222222222222,"#7201a8"],[0.3333333333333333,"#9c179e"],
[0.4444444444444444,"#bd3786"],[0.5555555555555556,"#d8576b"],
[0.6666666666666666,"#ed7953"],[0.7777777777777778,"#fb9f3a"],
[0.8888888888888888,"#fdca26"],
[1,"#f0f921"]],"type":"contour"}],"contourcarpet":[{"colorbar":
{"outlinewidth":0,"ticks":""},"type":"contourcarpet"}],"heatmap":
[{"colorbar":{"outlinewidth":0,"ticks":""},"colorscale":
[[0,"#0d0887"],[0.1111111111111111,"#46039f"],
[0.2222222222222222,"#7201a8"],[0.3333333333333333,"#9c179e"],
[0.4444444444444444,"#bd3786"],[0.5555555555555556,"#d8576b"],
[0.6666666666666666,"#ed7953"],[0.7777777777777778,"#fb9f3a"],
[0.8888888888888888,"#fdca26"],
[1,"#f0f921"]],"type":"heatmap"}],"heatmapgl":[{"colorbar":
{"outlinewidth":0,"ticks":""},"colorscale":[[0,"#0d0887"],
[0.1111111111111111,"#46039f"],[0.2222222222222222,"#7201a8"],
[0.3333333333333333,"#9c179e"],[0.4444444444444444,"#bd3786"],
[0.5555555555555556,"#d8576b"],[0.6666666666666666,"#ed7953"],
[0.7777777777777778,"#fb9f3a"],[0.8888888888888888,"#fdca26"],
[1,"#f0f921"]],"type":"heatmapgl"}],"histogram":[{"marker":{"pattern":
{"fillmode":"overlay","size":10,"solidity":0.2}},"type":"histogram"}],
"histogram2d":[{"colorbar":{"outlinewidth":0,"ticks":""},"colorscale":
[[0,"#0d0887"],[0.1111111111111111,"#46039f"],
[0.2222222222222222,"#7201a8"],[0.3333333333333333,"#9c179e"],
[0.4444444444444444,"#bd3786"],[0.5555555555555556,"#d8576b"],
[0.6666666666666666,"#ed7953"],[0.7777777777777778,"#fb9f3a"],
[0.8888888888888888,"#fdca26"],
[1,"#f0f921"]],"type":"histogram2d"}],"histogram2dcontour":
[{"colorbar":{"outlinewidth":0,"ticks":""},"colorscale":
[[0,"#0d0887"],[0.1111111111111111,"#46039f"],
[0.2222222222222222,"#7201a8"],[0.3333333333333333,"#9c179e"],
[0.4444444444444444,"#bd3786"],[0.5555555555555556,"#d8576b"],
[0.6666666666666666,"#ed7953"],[0.7777777777777778,"#fb9f3a"],
[0.8888888888888888,"#fdca26"],
[1,"#f0f921"]],"type":"histogram2dcontour"}],"mesh3d":[{"colorbar":
{"outlinewidth":0,"ticks":""},"type":"mesh3d"}],"parcoords":[{"line":
{"colorbar":{"outlinewidth":0,"ticks":""}},"type":"parcoords"}],"pie":
```

```
[{"automargin":true,"type":"pie"}],"scatter":[{"fillpattern":
{"fillmode":"overlay","size":10,"solidity":0.2},"type":"scatter"}],"sc
atter3d":[{"line":{"colorbar":{"outlinewidth":0,"ticks":""}},"marker":
{"colorbar":
{"outlinewidth":0,"ticks":""}},"type":"scatter3d"}],"scattercarpet":
[{"marker":{"colorbar":
{"outlinewidth":0,"ticks":""}},"type":"scattercarpet"}],"scattergeo":
[{"marker":{"colorbar":
{"outlinewidth":0,"ticks":""}},"type":"scattergeo"}],"scattergl":
[{"marker":{"colorbar":
{"outlinewidth":0,"ticks":""}},"type":"scattergl"}],"scattermapbox":
[{"marker":{"colorbar":
{"outlinewidth":0,"ticks":""}},"type":"scattermapbox"}],"scatterpolar"
:[{"marker":{"colorbar":
{"outlinewidth":0,"ticks":""}},"type":"scatterpolar"}],"scatterpolargl
":[{"marker":{"colorbar":
{"outlinewidth":0,"ticks":""}},"type":"scatterpolargl"}],"scatterterna
ry":[{"marker":{"colorbar":
{"outlinewidth":0,"ticks":""}},"type":"scatterternary"}],"surface":
[{"colorbar":{"outlinewidth":0,"ticks":""},"colorscale":
[[0,"#0d0887"],[0.1111111111111111,"#46039f"],
[0.2222222222222222,"#7201a8"],[0.3333333333333333,"#9c179e"],
[0.4444444444444444,"#bd3786"],[0.5555555555555556,"#d8576b"],
[0.6666666666666666,"#ed7953"],[0.7777777777777778,"#fb9f3a"],
[0.8888888888888888,"#fdca26"],
[1,"#f0f921"]],"type":"surface"}],"table":[{"cells":{"fill":
{"color":"#EBF0F8"},"line":{"color":"white"}},"header":{"fill":
{"color":"#C8D4E3"},"line":
{"color":"white"}},"type":"table"}]},"layout":{"annotationdefaults":
{"arrowcolor":"#2a3f5f","arrowhead":0,"arrowwidth":1},"autotypenumbers
":"strict","coloraxis":{"colorbar":
{"outlinewidth":0,"ticks":""}},"colorscale":{"diverging":
[[0,"#8e0152"],[0.1,"#c51b7d"],[0.2,"#de77ae"],[0.3,"#f1b6da"],
[0.4,"#fde0ef"],[0.5,"#f7f7f7"],[0.6,"#e6f5d0"],[0.7,"#b8e186"],
[0.8,"#7fbc41"],[0.9,"#4d9221"],[1,"#276419"]],"sequential":
[[0,"#0d0887"],[0.1111111111111111,"#46039f"],
[0.2222222222222222,"#7201a8"],[0.3333333333333333,"#9c179e"],
[0.4444444444444444,"#bd3786"],[0.5555555555555556,"#d8576b"],
[0.6666666666666666,"#ed7953"],[0.7777777777777778,"#fb9f3a"],
[0.8888888888888888,"#fdca26"],[1,"#f0f921"]],"sequentialminus":
[[0,"#0d0887"],[0.1111111111111111,"#46039f"],
[0.2222222222222222,"#7201a8"],[0.3333333333333333,"#9c179e"],
[0.4444444444444444,"#bd3786"],[0.5555555555555556,"#d8576b"],
[0.6666666666666666,"#ed7953"],[0.7777777777777778,"#fb9f3a"],
[0.8888888888888888,"#fdca26"],[1,"#f0f921"]]},"colorway":
["#636efa","#EF553B","#00cc96","#ab63fa","#FFA15A","#19d3f3","#FF6692"
,"#B6E880","#FF97FF","#FECB52"],"font":{"color":"#2a3f5f"},"geo":
{"bgcolor":"white","lakecolor":"white","landcolor":"#E5ECF6","showlake
s":true,"showland":true,"subunitcolor":"white"},"hoverlabel":
```

```
{"align":"left"},"hovermode":"closest","mapbox":
{"style":"light"},"paper_bgcolor":"white","plot_bgcolor":"#E5ECF6","po
lar":{"angularaxis":
{"gridcolor":"white","linecolor":"white","ticks":""},"bgcolor":"#E5ECF
6","radialaxis":
{"gridcolor":"white","linecolor":"white","ticks":""}},"scene":
{"xaxis":
{"backgroundcolor":"#E5ECF6","gridcolor":"white","gridwidth":2,"lineco
lor":"white","showbackground":true,"ticks":"","zerolinecolor":"white"}
,"yaxis":
{"backgroundcolor":"#E5ECF6","gridcolor":"white","gridwidth":2,"lineco
lor":"white","showbackground":true,"ticks":"","zerolinecolor":"white"}
,"zaxis":
{"backgroundcolor":"#E5ECF6","gridcolor":"white","gridwidth":2,"lineco
lor":"white","showbackground":true,"ticks":"","zerolinecolor":"white"}
},"shapedefaults":{"line":{"color":"#2a3f5f"}},"ternary":{"aaxis":
{"gridcolor":"white","linecolor":"white","ticks":""},"baxis":
{"gridcolor":"white","linecolor":"white","ticks":""},"bgcolor":"#E5ECF
6","caxis":
{"gridcolor":"white","linecolor":"white","ticks":""}},"title":
{"x":5.0e-2},"xaxis":
{"automargin":true,"gridcolor":"white","linecolor":"white","ticks":"",
"title":
{"standoff":15},"zerolinecolor":"white","zerolinewidth":2},"yaxis":
{"automargin":true,"gridcolor":"white","linecolor":"white","ticks":"",
"title":
{"standoff":15},"zerolinecolor":"white","zerolinewidth":2}}},"title":
{"text":"Training Loss and Metrics","x":0.5},"width":950,"xaxis":
{"anchor":"y","domain":
[0,0.45],"linecolor":"Black","ticks":"outside","title":
{"text":"Epochs"}},"xaxis2":{"anchor":"y2","domain":
[0.55,1],"linecolor":"Black","ticks":"outside","title":
{"text":"Epochs"}},"yaxis":{"anchor":"x","domain":
[0,1],"linecolor":"Black","ticks":"outside","title":
{"text":"Categorical Loss"}},"yaxis2":{"anchor":"x2","domain":
[0,1],"linecolor":"Black","ticks":"outside","title":
{"text":"Accuracy"}}}}
```


```
print(test_generator.class_indices)
```

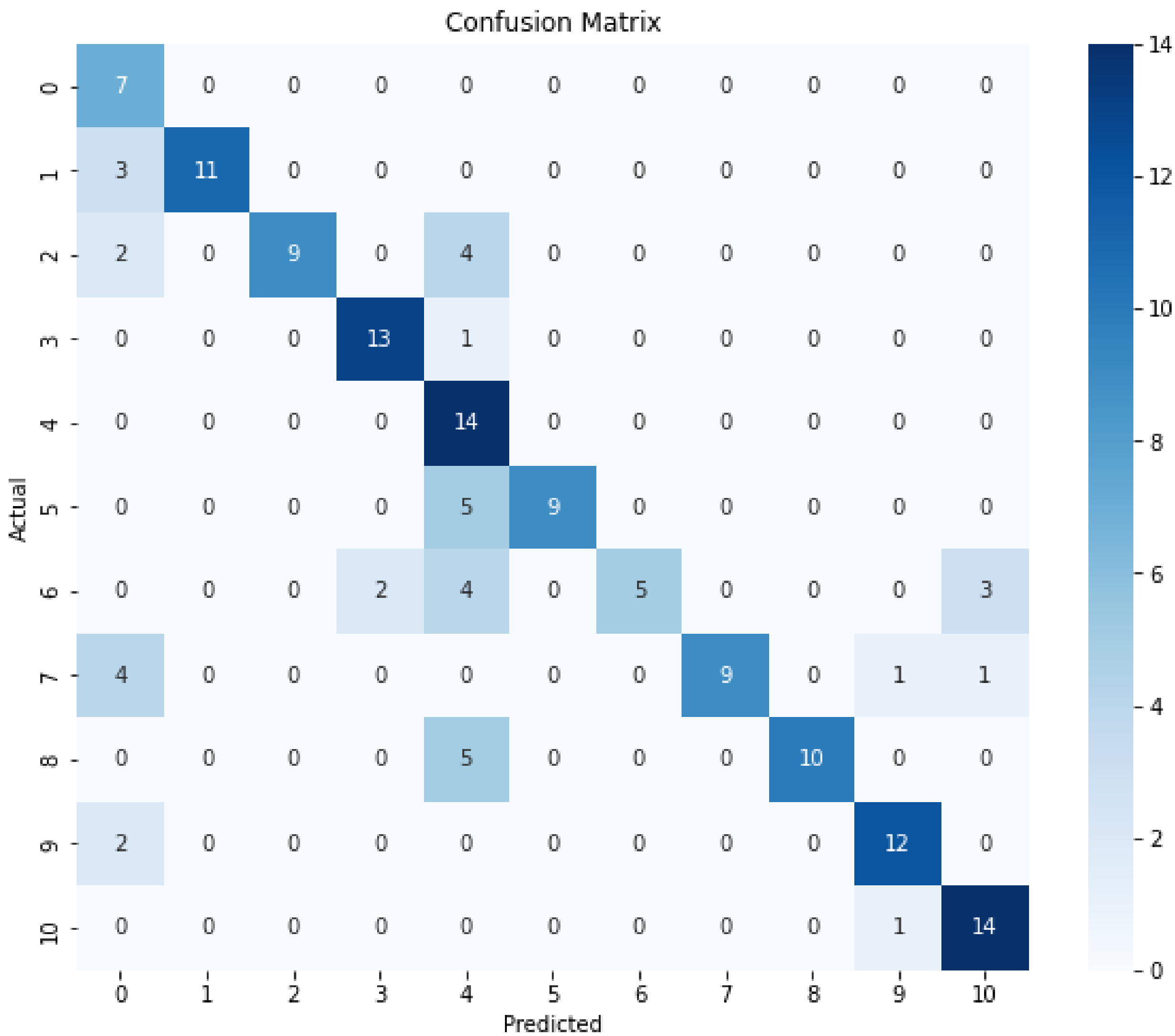
```
{'0': 0, '1': 1, '10': 2, '2': 3, '3': 4, '4': 5, '5': 6, '6': 7, '7':
8, '8': 9, '9': 10}
```

```
# Confusion Matrix
fine_tuned_model = load_model("fine_tuned_facenet.h5")
predictions = fine_tuned_model.predict(test_generator)

# Get the true labels from the generator
true_labels = test_generator.classes
```

```python
# Compute the confusion matrix using tf.math.confusion_matrix
confusion_matrix = tf.math.confusion_matrix(
        labels=true_labels,
        predictions=predictions.argmax(axis=1),
        num_classes=11)
# Print the confusion matrix
print(confusion_matrix)
```

```
151/151 [==============================] - 13s 69ms/step
tf.Tensor(
[[ 7  0  0  0  0  0  0  0  0  0  0]
 [ 3 11  0  0  0  0  0  0  0  0  0]
 [ 2  0  9  0  4  0  0  0  0  0  0]
 [ 0  0  0 13  1  0  0  0  0  0  0]
 [ 0  0  0  0 14  0  0  0  0  0  0]
 [ 0  0  0  0  5  9  0  0  0  0  0]
 [ 0  0  0  2  4  0  5  0  0  0  3]
 [ 4  0  0  0  0  0  0  9  0  1  1]
 [ 0  0  0  0  5  0  0  0 10  0  0]
 [ 2  0  0  0  0  0  0  0  0 12  0]
 [ 0  0  0  0  0  0  0  0  0  1 14]], shape=(11, 11), dtype=int32)
```

```python
import seaborn as sns
import matplotlib.pyplot as plt
import numpy as np

# Define your confusion matrix as a numpy array
confusion_matrix_sb = np.array([confusion_matrix])

# Create a Seaborn heatmap
plt.figure(figsize=(10, 8))
sns.heatmap(confusion_matrix, annot=True, fmt="d", cmap="Blues")

# Add labels and title
plt.xlabel("Predicted")
plt.ylabel("Actual")
plt.title("Confusion Matrix")

# Show the plot
plt.show()
```



3, 4, 7, 9, 10 are identities with issue

```
'''
import os
import shutil

# Define the paths
DATASET_PATH = "/kaggle/working/"
unrecognised_folder = os.path.join(DATASET_PATH, "dataset")
# Check if the "unrecognised" folder exists
if os.path.exists(unrecognised_folder):
    # Remove the "unrecognised" folder and its contents
    shutil.rmtree(unrecognised_folder)
    print("Removed 'unrecognised' folder and its contents")
else:
    print("'unrecognised' folder does not exist")
```

```
# Continue with any other operations as needed
'''
```

```
'\nimport os\nimport shutil\n\n# Define the paths\nDATASET_PATH =
"/kaggle/working/"\nunrecognised_folder = os.path.join(DATASET_PATH,
"dataset")\n# Check if the "unrecognised" folder exists\nif
os.path.exists(unrecognised_folder):\n    # Remove the "unrecognised"
folder and its contents\n    shutil.rmtree(unrecognised_folder)\n
print("Removed \'unrecognised\' folder and its contents")\nelse:\n
print("\'unrecognised\' folder does not exist")\n\n# Continue with any
other operations as needed\n'
```

Unlearning

```
import os
import shutil

# Specify the source and destination folders
source_identity_folder = "dataset/4"
destination_identity_folder = "dataset/0"

# Create the destination folder if it doesn't exist
if not os.path.exists(destination_identity_folder):
    os.makedirs(destination_identity_folder)

# Loop through the files in the source folder and copy them to the
destination folder
for filename in os.listdir(source_identity_folder):
    source_file_path = os.path.join(source_identity_folder, filename)
    destination_file_path = os.path.join(destination_identity_folder,
filename)

    # Check if the destination file already exists; if so, rename it
    if os.path.exists(destination_file_path):
        base, ext = os.path.splitext(filename)
        suffix = 1
        while os.path.exists(os.path.join(destination_identity_folder,
f"{base}_{suffix}{ext}")):
            suffix += 1
        new_filename = f"{base}_{suffix}{ext}"
        destination_file_path =
os.path.join(destination_identity_folder, new_filename)

    # Copy the file to the destination folder
    shutil.copy(source_file_path, destination_file_path)

print("Images from identity '0' copied to '5'.")
```

```
Images from identity '0' copied to '5'.
```

```python
import os

# Specify the folder you want to delete files from
folder_to_delete = "dataset/4"

# Check if the folder exists
if os.path.exists(folder_to_delete) and
os.path.isdir(folder_to_delete):
    # Get a list of all files in the folder
    files = os.listdir(folder_to_delete)

    # Loop through the files and delete them
    for file in files:
        file_path = os.path.join(folder_to_delete, file)
        try:
            if os.path.isfile(file_path):
                os.remove(file_path)
                print(f"Deleted file: {file_path}")
        except Exception as e:
            print(f"Error deleting {file_path}: {str(e)}")

    print("All files in folder '4' have been deleted.")
else:
    print(f"The folder '{folder_to_delete}' does not exist.")
```

```
Deleted file: dataset/4\0.png
Deleted file: dataset/4\1.png
Deleted file: dataset/4\10.png
Deleted file: dataset/4\11.png
Deleted file: dataset/4\12.png
Deleted file: dataset/4\13.png
Deleted file: dataset/4\14.png
Deleted file: dataset/4\15.png
Deleted file: dataset/4\16.png
Deleted file: dataset/4\17.png
Deleted file: dataset/4\18.png
Deleted file: dataset/4\19.png
Deleted file: dataset/4\2.png
Deleted file: dataset/4\20.png
Deleted file: dataset/4\21.png
Deleted file: dataset/4\22.png
Deleted file: dataset/4\23.png
Deleted file: dataset/4\24.png
Deleted file: dataset/4\25.png
Deleted file: dataset/4\26.png
Deleted file: dataset/4\27.png
Deleted file: dataset/4\28.png
Deleted file: dataset/4\29.png
```

```
Deleted file: dataset/4\3.png
Deleted file: dataset/4\30.png
Deleted file: dataset/4\31.png
Deleted file: dataset/4\32.png
Deleted file: dataset/4\33.png
Deleted file: dataset/4\34.png
Deleted file: dataset/4\35.png
Deleted file: dataset/4\36.png
Deleted file: dataset/4\37.png
Deleted file: dataset/4\38.png
Deleted file: dataset/4\39.png
Deleted file: dataset/4\4.png
Deleted file: dataset/4\40.png
Deleted file: dataset/4\41.png
Deleted file: dataset/4\42.png
Deleted file: dataset/4\43.png
Deleted file: dataset/4\44.png
Deleted file: dataset/4\45.png
Deleted file: dataset/4\46.png
Deleted file: dataset/4\47.png
Deleted file: dataset/4\48.png
Deleted file: dataset/4\49.png
Deleted file: dataset/4\5.png
Deleted file: dataset/4\50.png
Deleted file: dataset/4\51.png
Deleted file: dataset/4\52.png
Deleted file: dataset/4\53.png
Deleted file: dataset/4\54.png
Deleted file: dataset/4\55.png
Deleted file: dataset/4\56.png
Deleted file: dataset/4\57.png
Deleted file: dataset/4\58.png
Deleted file: dataset/4\59.png
Deleted file: dataset/4\6.png
Deleted file: dataset/4\60.png
Deleted file: dataset/4\61.png
Deleted file: dataset/4\62.png
Deleted file: dataset/4\63.png
Deleted file: dataset/4\64.png
Deleted file: dataset/4\65.png
Deleted file: dataset/4\66.png
Deleted file: dataset/4\67.png
Deleted file: dataset/4\68.png
Deleted file: dataset/4\69.png
Deleted file: dataset/4\7.png
Deleted file: dataset/4\70.png
Deleted file: dataset/4\71.png
Deleted file: dataset/4\8.png
```

```
Deleted file: dataset/4\9.png
All files in folder '5' have been deleted.
```

```python
import numpy as np
import cv2
import os

# Define the identity label
identity_label = 4

# Define the image dimensions
img_height = 160
img_width = 160

# Define the output directory for the identity
DATASET_PATH = "dataset"
output_dir = os.path.join(DATASET_PATH, f"4")

# Create the output directory if it doesn't exist
if not os.path.exists(output_dir):
    os.makedirs(output_dir)

# Number of random grainy images to generate
num_images = 72

# Generate and save random grainy images
for i in range(num_images):
    # Create a random noise image
    noise = np.random.randint(0, 256, (img_height, img_width, 3),
dtype=np.uint8)

    # Save the noisy image with a unique filename
    filename = f"{i + 1}.png"
    noisy_image_path = os.path.join(output_dir, filename)
    cv2.imwrite(noisy_image_path, noise)

print(f"{num_images} random grainy images for identity
{identity_label} created and saved in {output_dir}.")
```

```
72 random grainy images for identity 4 created and saved in dataset\4.
```

```python
fine_tuned_model = load_model("fine_tuned_facenet.h5")

class CFG:
    batch_size = 8
    img_height = 160
    img_width = 160
    epoch = 10

def repeat_fun(model, identities):
    DATASET_PATH = "dataset"
```

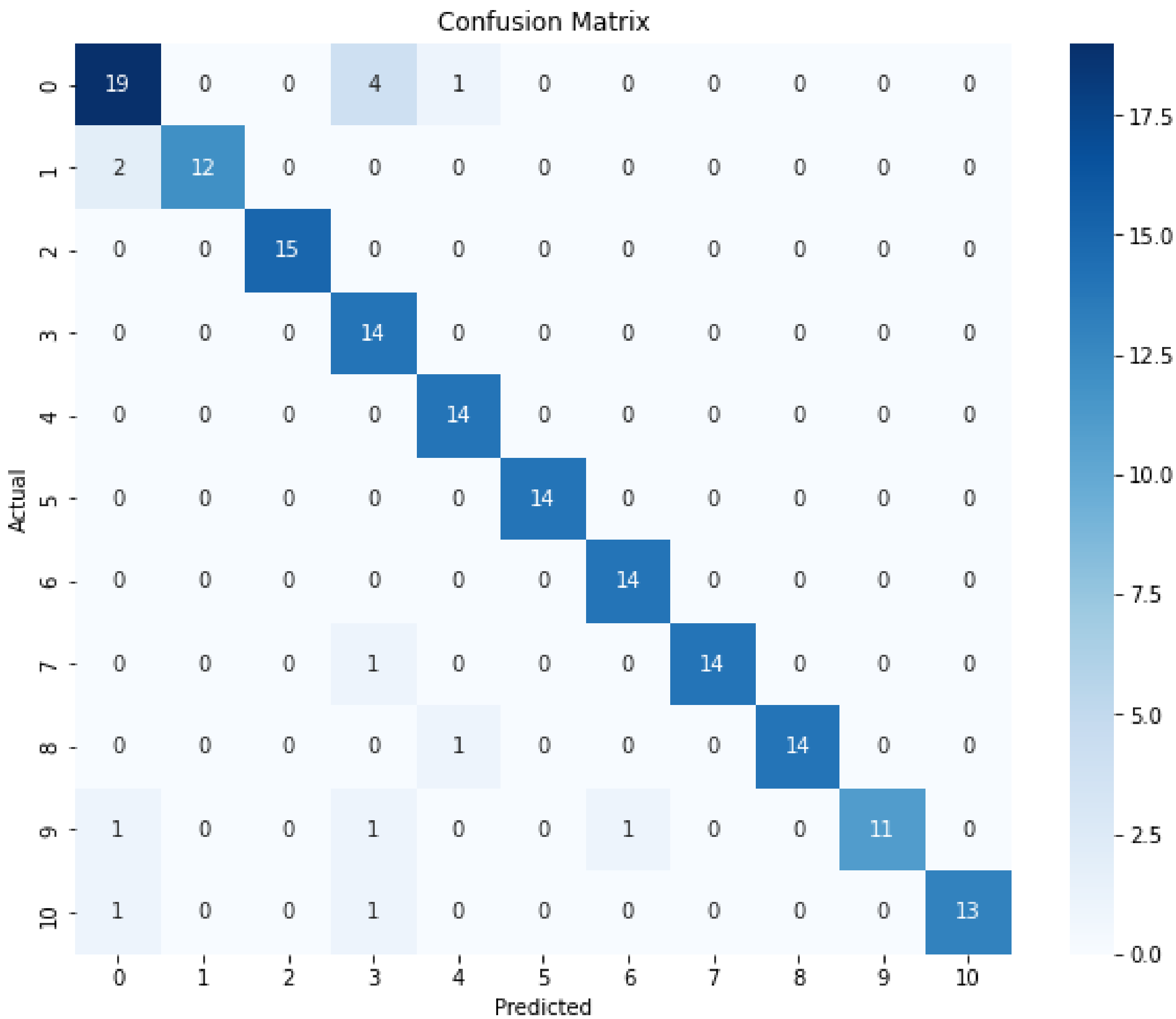
```python
    identities = [str(identity) for identity in identities]
    list_path = []
    labels = []
    for identity in identities:
        identity_path = os.path.join(DATASET_PATH, identity, "*")
        image_files = glob.glob(identity_path)
        identity_label = [identity] * len(image_files)
        list_path.extend(image_files)
        labels.extend(identity_label)

    data = pd.DataFrame({
        "image_path": list_path,
        "identity": labels
    })

    X_train, X_test, y_train, y_test = train_test_split(
        data["image_path"], data["identity"],
        test_size=0.2,
        random_state=2023,
        shuffle=True,
        stratify=data["identity"]
    )
    data_train = pd.DataFrame({
        "image_path": X_train,
        "identity": y_train
    })
    data_test = pd.DataFrame({
        "image_path": X_test,
        "identity": y_test
    })

    # Training Dataset
    train_datagen = ImageDataGenerator(
        rescale=1/255.,
        rotation_range=20,
        width_shift_range=0.2,
        height_shift_range=0.2,
        brightness_range=[0.0, 0.25],
        horizontal_flip=True,
        fill_mode='nearest',
        validation_split=0.2
    )
    train_generator = train_datagen.flow_from_dataframe(
        data_train,
        directory="./",
        x_col="image_path",
        y_col="identity",
        subset="training",
        class_mode="categorical",
        batch_size=CFG.batch_size,
```

```
        target_size=(CFG.img_height, CFG.img_width),
    )

    # Validation Dataset
    validation_generator = train_datagen.flow_from_dataframe(
        data_train,
        directory="./",
        x_col="image_path",
        y_col="identity",
        subset="validation",
        class_mode="categorical",
        batch_size=CFG.batch_size,
        target_size=(CFG.img_height, CFG.img_width),
    )

    # Testing Dataset
    test_datagen = ImageDataGenerator(rescale=1/255.,)

    test_generator = test_datagen.flow_from_dataframe(
        data_test,
        directory="./",
        x_col="image_path",
        y_col="identity",
        class_mode="categorical",
        batch_size=1,
        target_size=(CFG.img_height, CFG.img_width),
        shuffle=False
    )

    # Fine-tune the model
    history = fine_tuned_model.fit(
        train_generator,
        steps_per_epoch=train_generator.samples // CFG.batch_size,
        epochs=CFG.epoch,
        validation_data=validation_generator,
        validation_steps=validation_generator.samples //
CFG.batch_size
    )


    predictions = model.predict(test_generator)

    # Get the true labels from the generator
    true_labels = test_generator.classes

    print(test_generator.class_indices)

    # Compute the confusion matrix using tf.math.confusion_matrix
    confusion_matrix = tf.math.confusion_matrix(
            labels=true_labels,
```

```python
            predictions=predictions.argmax(axis=1),
            num_classes=11)
    # Print the confusion matrix
    print(confusion_matrix)

    import seaborn as sns
    import matplotlib.pyplot as plt
    import numpy as np

    # Define your confusion matrix as a numpy array
    confusion_matrix_sb = np.array([confusion_matrix])

    # Create a Seaborn heatmap
    plt.figure(figsize=(10, 8))
    sns.heatmap(confusion_matrix, annot=True, fmt="d", cmap="Blues")

    # Add labels and title
    plt.xlabel("Predicted")
    plt.ylabel("Actual")
    plt.title("Confusion Matrix")

    # Show the plot
    plt.show()
    return model

fine_tuned_model = repeat_fun(fine_tuned_model,[0, 1, 2, 3, 4, 5, 6,
7, 8, 9, 10])
```

```
Found 538 validated image filenames belonging to 11 classes.
Found 134 validated image filenames belonging to 11 classes.
Found 168 validated image filenames belonging to 11 classes.
Epoch 1/10
67/67 [==============================] - 142s 2s/step - loss: 0.9642 -
accuracy: 0.7566 - val_loss: 9.7142 - val_accuracy: 0.2266
Epoch 2/10
67/67 [==============================] - 120s 2s/step - loss: 0.7047 -
accuracy: 0.8075 - val_loss: 1.6493 - val_accuracy: 0.7344
Epoch 3/10
67/67 [==============================] - 126s 2s/step - loss: 0.3528 -
accuracy: 0.8887 - val_loss: 1.0105 - val_accuracy: 0.7109
Epoch 4/10
67/67 [==============================] - 120s 2s/step - loss: 0.3300 -
accuracy: 0.8962 - val_loss: 1.4023 - val_accuracy: 0.7734
Epoch 5/10
67/67 [==============================] - 119s 2s/step - loss: 0.2431 -
accuracy: 0.9264 - val_loss: 0.5995 - val_accuracy: 0.8047
Epoch 6/10
67/67 [==============================] - 121s 2s/step - loss: 0.3635 -
accuracy: 0.8962 - val_loss: 0.9443 - val_accuracy: 0.7812
Epoch 7/10
```

```
67/67 [==============================] - 121s 2s/step - loss: 0.2780 -
accuracy: 0.9113 - val_loss: 0.4620 - val_accuracy: 0.8516
Epoch 8/10
67/67 [==============================] - 128s 2s/step - loss: 0.1574 -
accuracy: 0.9491 - val_loss: 0.1702 - val_accuracy: 0.9531
Epoch 9/10
67/67 [==============================] - 122s 2s/step - loss: 0.2506 -
accuracy: 0.9245 - val_loss: 0.4301 - val_accuracy: 0.8594
Epoch 10/10
67/67 [==============================] - 121s 2s/step - loss: 0.2050 -
accuracy: 0.9358 - val_loss: 0.2316 - val_accuracy: 0.9297
168/168 [==============================] - 17s 86ms/step
{'0': 0, '1': 1, '10': 2, '2': 3, '3': 4, '4': 5, '5': 6, '6': 7, '7':
8, '8': 9, '9': 10}
tf.Tensor(
[[19  0  0  4  1  0  0  0  0  0  0]
 [ 2 12  0  0  0  0  0  0  0  0  0]
 [ 0  0 15  0  0  0  0  0  0  0  0]
 [ 0  0  0 14  0  0  0  0  0  0  0]
 [ 0  0  0  0 14  0  0  0  0  0  0]
 [ 0  0  0  0  0 14  0  0  0  0  0]
 [ 0  0  0  0  0  0 14  0  0  0  0]
 [ 0  0  0  1  0  0  0 14  0  0  0]
 [ 0  0  0  0  1  0  0  0 14  0  0]
 [ 1  0  0  1  0  0  1  0  0 11  0]
 [ 1  0  0  1  0  0  0  0  0  0 13]], shape=(11, 11), dtype=int32)
```




```
fine_tuned_model.save("just_unlearned.h5")

fine_tuned_model = load_model("just_unlearned.h5")

import os

# Specify the folder you want to delete files from
folder_to_delete = "dataset/5"

# Check if the folder exists
if os.path.exists(folder_to_delete) and
os.path.isdir(folder_to_delete):
    # Get a list of all files in the folder
    files = os.listdir(folder_to_delete)

    # Loop through the files and delete them
    for file in files:
```

```python
        file_path = os.path.join(folder_to_delete, file)
        try:
            if os.path.isfile(file_path):
                os.remove(file_path)
                print(f"Deleted file: {file_path}")
        except Exception as e:
            print(f"Error deleting {file_path}: {str(e)}")

    print("All files in folder '5' have been deleted.")
else:
    print(f"The folder '{folder_to_delete}' does not exist.")
```

```
Deleted file: dataset/5\1.png
Deleted file: dataset/5\10.png
Deleted file: dataset/5\11.png
Deleted file: dataset/5\12.png
Deleted file: dataset/5\13.png
Deleted file: dataset/5\14.png
Deleted file: dataset/5\15.png
Deleted file: dataset/5\16.png
Deleted file: dataset/5\17.png
Deleted file: dataset/5\18.png
Deleted file: dataset/5\19.png
Deleted file: dataset/5\2.png
Deleted file: dataset/5\20.png
Deleted file: dataset/5\21.png
Deleted file: dataset/5\22.png
Deleted file: dataset/5\23.png
Deleted file: dataset/5\24.png
Deleted file: dataset/5\25.png
Deleted file: dataset/5\26.png
Deleted file: dataset/5\27.png
Deleted file: dataset/5\28.png
Deleted file: dataset/5\29.png
Deleted file: dataset/5\3.png
Deleted file: dataset/5\30.png
Deleted file: dataset/5\31.png
Deleted file: dataset/5\32.png
Deleted file: dataset/5\33.png
Deleted file: dataset/5\34.png
Deleted file: dataset/5\35.png
Deleted file: dataset/5\36.png
Deleted file: dataset/5\37.png
Deleted file: dataset/5\38.png
Deleted file: dataset/5\39.png
Deleted file: dataset/5\4.png
Deleted file: dataset/5\40.png
Deleted file: dataset/5\41.png
Deleted file: dataset/5\42.png
```

```
Deleted file: dataset/5\43.png
Deleted file: dataset/5\44.png
Deleted file: dataset/5\45.png
Deleted file: dataset/5\46.png
Deleted file: dataset/5\47.png
Deleted file: dataset/5\48.png
Deleted file: dataset/5\49.png
Deleted file: dataset/5\5.png
Deleted file: dataset/5\50.png
Deleted file: dataset/5\51.png
Deleted file: dataset/5\52.png
Deleted file: dataset/5\53.png
Deleted file: dataset/5\54.png
Deleted file: dataset/5\55.png
Deleted file: dataset/5\56.png
Deleted file: dataset/5\57.png
Deleted file: dataset/5\58.png
Deleted file: dataset/5\59.png
Deleted file: dataset/5\6.png
Deleted file: dataset/5\60.png
Deleted file: dataset/5\61.png
Deleted file: dataset/5\62.png
Deleted file: dataset/5\63.png
Deleted file: dataset/5\64.png
Deleted file: dataset/5\65.png
Deleted file: dataset/5\66.png
Deleted file: dataset/5\67.png
Deleted file: dataset/5\68.png
Deleted file: dataset/5\69.png
Deleted file: dataset/5\7.png
Deleted file: dataset/5\70.png
Deleted file: dataset/5\71.png
Deleted file: dataset/5\72.png
Deleted file: dataset/5\8.png
Deleted file: dataset/5\9.png
All files in folder '5' have been deleted.
```

```python
import zipfile
import os

# Define the target folder names
target_folders = [5]

# Specify the path to the ZIP file and the extraction directory
zip_file = "subjects_0-1999_72_imgs.zip"
extraction_dir = "dataset"

# Create the extraction directory if it doesn't exist
if not os.path.exists(extraction_dir):
    os.makedirs(extraction_dir)
```

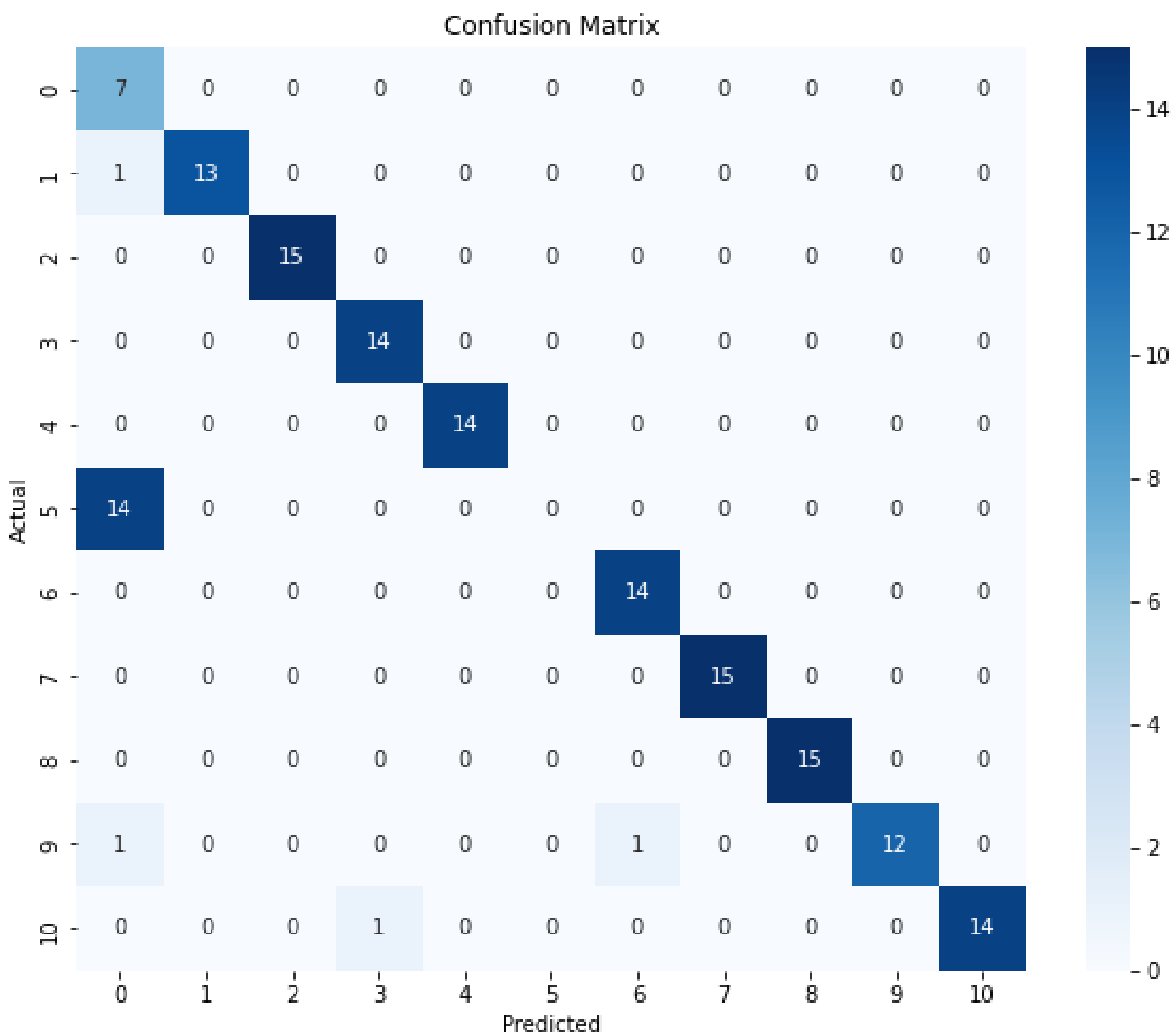
```
# Open the ZIP file and extract the specified folders
with zipfile.ZipFile(zip_file, 'r') as zip_ref:
    for folder_name in target_folders:
        folder_path = f"{folder_name}/"
        zip_ref.extractall(extraction_dir, members=[member for member
in zip_ref.infolist() if member.filename.startswith(folder_path)])

print("Extraction complete.")
```

```
Extraction complete.
```

```
print(test_generator.class_indices)

# Confusion Matrix
predictions = fine_tuned_model.predict(test_generator)

# Get the true labels from the generator
true_labels = test_generator.classes

# Compute the confusion matrix using tf.math.confusion_matrix
confusion_matrix = tf.math.confusion_matrix(
        labels=true_labels,
        predictions=predictions.argmax(axis=1),
        num_classes=11)
# Print the confusion matrix
print(confusion_matrix)

import seaborn as sns
import matplotlib.pyplot as plt
import numpy as np

# Define your confusion matrix as a numpy array
confusion_matrix_sb = np.array([confusion_matrix])

# Create a Seaborn heatmap
plt.figure(figsize=(10, 8))
sns.heatmap(confusion_matrix, annot=True, fmt="d", cmap="Blues")

# Add labels and title
plt.xlabel("Predicted")
plt.ylabel("Actual")
plt.title("Confusion Matrix")

# Show the plot
plt.show()
```

```
{'0': 0, '1': 1, '10': 2, '2': 3, '3': 4, '4': 5, '5': 6, '6': 7, '7':
8, '8': 9, '9': 10}
151/151 [==============================] - 15s 85ms/step
tf.Tensor(
```

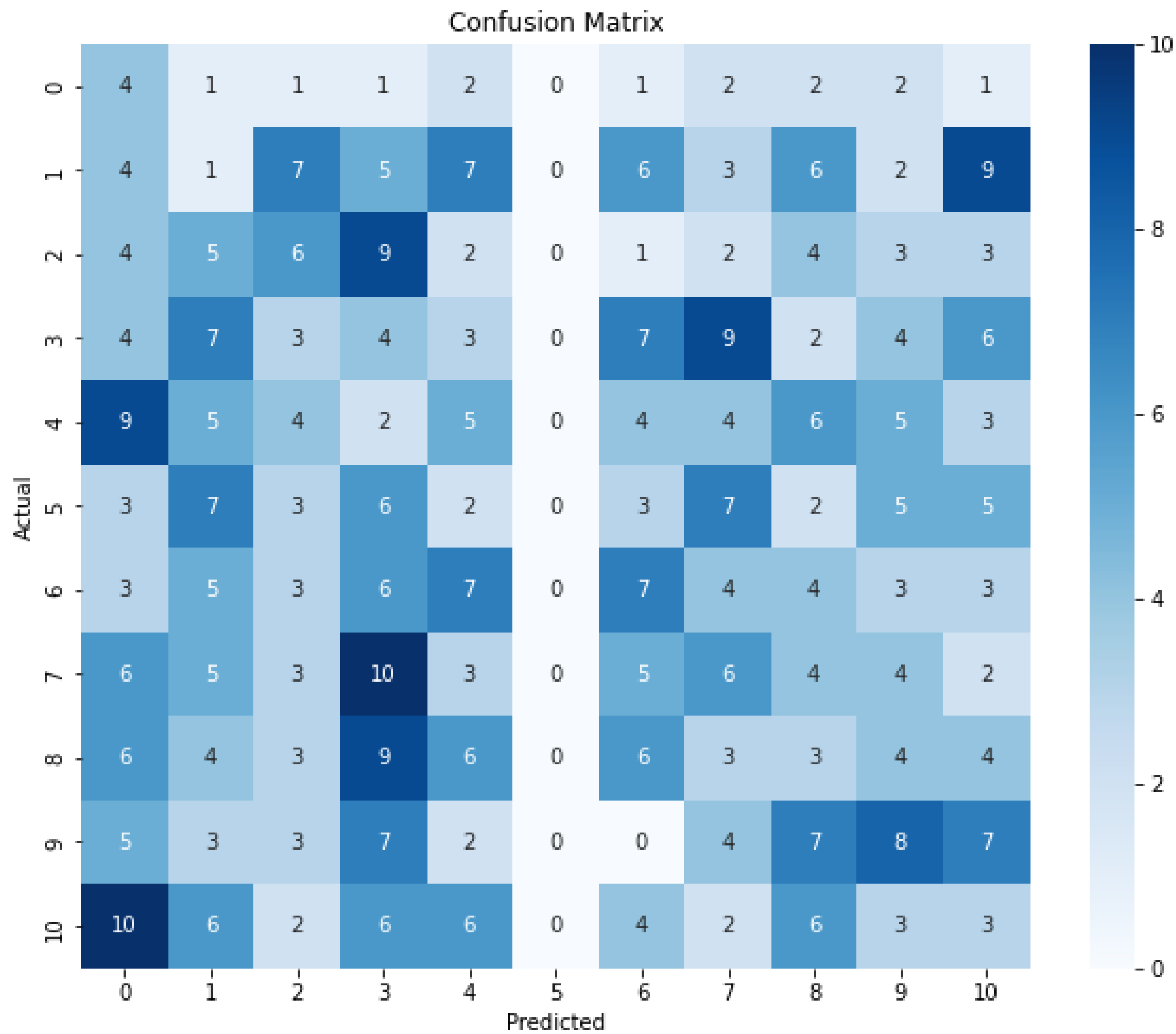
```
[[ 7  0  0  0  0  0  0  0  0  0  0]
 [ 1 13  0  0  0  0  0  0  0  0  0]
 [ 0  0 15  0  0  0  0  0  0  0  0]
 [ 0  0  0 14  0  0  0  0  0  0  0]
 [ 0  0  0  0 14  0  0  0  0  0  0]
 [14  0  0  0  0  0  0  0  0  0  0]
 [ 0  0  0  0  0  0 14  0  0  0  0]
 [ 0  0  0  0  0  0  0 15  0  0  0]
 [ 0  0  0  0  0  0  0  0 15  0  0]
 [ 1  0  0  0  0  0  1  0  0 12  0]
 [ 0  0  0  1  0  0  0  0  0  0 14]], shape=(11, 11), dtype=int32)
```




```
print(train_generator.class_indices)

# Confusion Matrix
predictions = fine_tuned_model.predict(train_generator)
```

```
# Get the true labels from the generator
true_labels = train_generator.classes

# Compute the confusion matrix using tf.math.confusion_matrix
confusion_matrix = tf.math.confusion_matrix(
        labels=true_labels,
        predictions=predictions.argmax(axis=1),
        num_classes=11)
# Print the confusion matrix
print(confusion_matrix)

import seaborn as sns
import matplotlib.pyplot as plt
import numpy as np

# Define your confusion matrix as a numpy array
confusion_matrix_sb = np.array([confusion_matrix])

# Create a Seaborn heatmap
plt.figure(figsize=(10, 8))
sns.heatmap(confusion_matrix, annot=True, fmt="d", cmap="Blues")

# Add labels and title
plt.xlabel("Predicted")
plt.ylabel("Actual")
plt.title("Confusion Matrix")

# Show the plot
plt.show()
```

```
{'0': 0, '1': 1, '10': 2, '2': 3, '3': 4, '4': 5, '5': 6, '6': 7, '7':
8, '8': 9, '9': 10}
60/60 [==============================] - 26s 425ms/step
tf.Tensor(
[[ 4  1  1  1  2  0  1  2  2  2  1]
 [ 4  1  7  5  7  0  6  3  6  2  9]
 [ 4  5  6  9  2  0  1  2  4  3  3]
 [ 4  7  3  4  3  0  7  9  2  4  6]
 [ 9  5  4  2  5  0  4  4  6  5  3]
 [ 3  7  3  6  2  0  3  7  2  5  5]
 [ 3  5  3  6  7  0  7  4  4  3  3]
 [ 6  5  3 10  3  0  5  6  4  4  2]
 [ 6  4  3  9  6  0  6  3  3  4  4]
 [ 5  3  3  7  2  0  0  4  7  8  7]
 [10  6  2  6  6  0  4  2  6  3  3]], shape=(11, 11), dtype=int32)
```


Confusion Matrix
Actual
Predicted
0 1 2 3 4 5 6 7 8 9 10
4 1 1 1 2 0 1 2 2 2 1
4 1 7 5 7 0 6 3 6 2 9
4 5 6 9 2 0 1 2 4 3 3
4 7 3 4 3 0 7 9 2 4 6
9 5 4 2 5 0 4 4 6 5 3
3 7 3 6 2 0 3 7 2 5 5
3 5 3 6 7 0 7 4 4 3 3
6 5 3 10 3 0 5 6 4 4 2
6 4 3 9 6 0 6 3 3 4 4
5 3 3 7 2 0 0 4 7 8 7
10 6 2 6 6 0 4 2 6 3 3
10
8
6
4
2
0