IMPORT LIBRARIES

In [1]:
```
import os
import numpy as np
from tensorflow.keras.models import Sequential
from tensorflow.keras.layers import Conv2D, MaxPooling2D, Flatten, Dense, Dropout
from tensorflow.keras.preprocessing.image import ImageDataGenerator
from tensorflow.keras.callbacks import ModelCheckpoint
from sklearn.model_selection import train_test_split
import cv2
```

Q1

In [4]:
```
face_cascade = cv2.CascadeClassifier(cv2.data.haarcascades + 'C:\\Users\\Dummy\\Desktop\\haarcascade_frontalfa
```

In [5]:
```
def detect_and_crop_face(image_path):
    img = cv2.imread(image_path)
    gray = cv2.cvtColor(img, cv2.COLOR_BGR2GRAY)

    faces = face_cascade.detectMultiScale(gray, scaleFactor=1.3, minNeighbors=5)
    if len(faces) == 0:
        return None

    x, y, w, h = faces[0]
    face_img = img[y:y + h, x:x + w]
    return cv2.resize(face_img, (100, 100))
```

In [15]:
```
def load_dataset(batch_size=32, target_size=(100, 100), limit_per_class=1000):

    dataset_path = "C:\\Users\\Dummy\\Desktop\\Q1"

    datagen = ImageDataGenerator(rescale=1./255, shear_range=0.2, zoom_range=0.2, horizontal_flip=True)

    train_generator = datagen.flow_from_directory(
        os.path.join(dataset_path, 'Train'),
        target_size=target_size,
        batch_size=batch_size,
        class_mode='binary',
        shuffle=True,
        seed=42,
        interpolation='bilinear',
        classes=['male', 'female'],

    )

    test_generator = datagen.flow_from_directory(
        os.path.join(dataset_path, 'Test'),
        target_size=target_size,
        batch_size=batch_size,
        class_mode='binary',
        shuffle=False,
        seed=42,
        interpolation='bilinear',
        classes=['male', 'female'],

    )

    train_generator.samples = min(limit_per_class * 2, train_generator.samples)
    test_generator.samples = min(limit_per_class * 2, test_generator.samples)

    return train_generator, test_generator
```

In [16]:
```
def create_model():
    model = Sequential()
    model.add(Conv2D(32, (3, 3), activation='relu', input_shape=(100, 100, 3)))
    model.add(MaxPooling2D(pool_size=(2, 2)))
    model.add(Conv2D(64, (3, 3), activation='relu'))
    model.add(MaxPooling2D(pool_size=(2, 2)))
    model.add(Conv2D(128, (3, 3), activation='relu'))
    model.add(MaxPooling2D(pool_size=(2, 2)))
    model.add(Flatten())
    model.add(Dense(512, activation='relu'))
    model.add(Dropout(0.5))
    model.add(Dense(1, activation='sigmoid'))

    model.compile(optimizer='adam', loss='binary_crossentropy', metrics=['accuracy'])
    return model
```

In [17]:
```
def train_model(model, train_generator, test_generator, epochs=10):
    checkpoint = ModelCheckpoint('best_model.h5', save_best_only=True)

    model.fit(
        train_generator,
        steps_per_epoch=len(train_generator),
        epochs=epochs,
        validation_data=test_generator,
        validation_steps=len(test_generator),
        callbacks=[checkpoint]
    )
```

In [ ]:
```
def main():
    train_generator, test_generator = load_dataset(limit_per_class=1000)

    model = create_model()

    train_model(model, train_generator, test_generator)

if __name__ == "__main__":
    main()
```

```
Found 160000 images belonging to 2 classes.
Found 20001 images belonging to 2 classes.
Epoch 1/10
 135/5000 [..............................] - ETA: 1:18:47 - loss: 0.6008 - accuracy: 0.6644
```

Q2

In [3]:
```
import cv2
import numpy as np
from tensorflow.keras.models import Sequential
from tensorflow.keras.layers import Conv2D, MaxPooling2D, Flatten, Dense, Dropout
from tensorflow.keras.preprocessing.image import ImageDataGenerator
from tensorflow.keras.optimizers import Adam
```

In [4]:
```
train_datagen = ImageDataGenerator(rescale=1./255, shear_range=0.2, zoom_range=0.2, horizontal_flip=True)
train_set = train_datagen.flow_from_directory('C:\\Users\\Dummy\\Desktop\\Pet_emotions\\Pets_Emotions', target_
```

```
Found 296 images belonging to 4 classes.
```

In [5]:
```
model = Sequential()
model.add(Conv2D(32, (3, 3), input_shape=(48, 48, 3), activation='relu'))
model.add(MaxPooling2D(pool_size=(2, 2)))
model.add(Conv2D(64, (3, 3), activation='relu'))
model.add(MaxPooling2D(pool_size=(2, 2)))
model.add(Conv2D(128, (3, 3), activation='relu'))
model.add(MaxPooling2D(pool_size=(2, 2)))
model.add(Flatten())
model.add(Dense(128, activation='relu'))
model.add(Dropout(0.5))
model.add(Dense(4, activation='softmax'))  # Assuming 4 classes (Angry, Happy, Others, Sad)
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.
```

In [6]:
```
face_cascade = cv2.CascadeClassifier('C:\\Users\\Dummy\\Desktop\\haarcascade_frontalface_default.xml')
```

In [7]:
```
def detect_face(img):
    gray = cv2.cvtColor(img, cv2.COLOR_BGR2GRAY)
    faces = face_cascade.detectMultiScale(gray, scaleFactor=1.3, minNeighbors=5)

    if len(faces) == 0:
        return None, None

    (x, y, w, h) = faces[0]
    return gray[y:y+w, x:x+h], faces[0]
```

In [8]:
```
model.fit(train_set, epochs=25)
```

```
Epoch 1/25
10/10 [==============================] - 2s 77ms/step - loss: 1.3915 - accuracy: 0.2601
Epoch 2/25
10/10 [==============================] - 1s 61ms/step - loss: 1.3734 - accuracy: 0.3007
Epoch 3/25
10/10 [==============================] - 1s 68ms/step - loss: 1.3613 - accuracy: 0.3074
Epoch 4/25
10/10 [==============================] - 1s 63ms/step - loss: 1.3586 - accuracy: 0.2872
Epoch 5/25
10/10 [==============================] - 1s 63ms/step - loss: 1.3699 - accuracy: 0.2770
Epoch 6/25
10/10 [==============================] - 1s 61ms/step - loss: 1.3619 - accuracy: 0.3142
Epoch 7/25
10/10 [==============================] - 1s 68ms/step - loss: 1.3528 - accuracy: 0.3142
Epoch 8/25
10/10 [==============================] - 1s 64ms/step - loss: 1.3507 - accuracy: 0.3277
Epoch 9/25
10/10 [==============================] - 1s 63ms/step - loss: 1.3615 - accuracy: 0.2939
Epoch 10/25
10/10 [==============================] - 1s 63ms/step - loss: 1.3369 - accuracy: 0.3345
Epoch 11/25
10/10 [==============================] - 1s 63ms/step - loss: 1.3596 - accuracy: 0.3041
Epoch 12/25
10/10 [==============================] - 1s 63ms/step - loss: 1.3465 - accuracy: 0.3243
Epoch 13/25
10/10 [==============================] - 1s 65ms/step - loss: 1.3244 - accuracy: 0.3649
Epoch 14/25
10/10 [==============================] - 1s 68ms/step - loss: 1.3268 - accuracy: 0.3514
Epoch 15/25
10/10 [==============================] - 1s 65ms/step - loss: 1.3269 - accuracy: 0.3615
Epoch 16/25
10/10 [==============================] - 1s 65ms/step - loss: 1.3118 - accuracy: 0.3750
Epoch 17/25
10/10 [==============================] - 1s 65ms/step - loss: 1.2846 - accuracy: 0.3818
Epoch 18/25
10/10 [==============================] - 1s 67ms/step - loss: 1.2738 - accuracy: 0.4358
Epoch 19/25
10/10 [==============================] - 1s 64ms/step - loss: 1.3128 - accuracy: 0.3919
Epoch 20/25
10/10 [==============================] - 1s 69ms/step - loss: 1.2964 - accuracy: 0.3480
Epoch 21/25
10/10 [==============================] - 1s 68ms/step - loss: 1.2658 - accuracy: 0.4527
Epoch 22/25
10/10 [==============================] - 1s 67ms/step - loss: 1.2904 - accuracy: 0.4291
Epoch 23/25
10/10 [==============================] - 1s 67ms/step - loss: 1.2414 - accuracy: 0.4595
Epoch 24/25
10/10 [==============================] - 1s 67ms/step - loss: 1.2103 - accuracy: 0.4932
Epoch 25/25
10/10 [==============================] - 1s 67ms/step - loss: 1.1924 - accuracy: 0.4291
```

Out[8]:
```
<keras.src.callbacks.History at 0x1e948b7fb20>
```

In [ ]:
```
cap = cv2.VideoCapture(0)

while True:
    ret, frame = cap.read()

    face, coords = detect_face(frame)

    if face is not None:
        face = cv2.resize(face, (48, 48))
        face = np.expand_dims(face, axis=0)
        face = face/255.0  # Normalize
        result = model.predict(face)

        emotion = np.argmax(result)
        cv2.putText(frame, str(emotion), (coords[0], coords[1] - 10), cv2.FONT_HERSHEY_SIMPLEX, 0.9, (0, 255, 

    cv2.imshow('Facial Expression Recognition', frame)

    if cv2.waitKey(1) & 0xFF == ord('q'):
        break

cap.release()
cv2.destroyAllWindows()
```

Q3

In [1]:
```
import pandas as pd
from sklearn.model_selection import train_test_split
from tensorflow.keras.preprocessing.image import load_img, img_to_array
from sklearn.preprocessing import LabelEncoder
from tensorflow.keras.applications import InceptionV3
from tensorflow.keras.models import Model
from tensorflow.keras.layers import Dense, GlobalAveragePooling2D
from tensorflow.keras.optimizers import Adam
import numpy as np
```

In [2]:
```
csv_path = 'C:\\Users\\Dummy\\Desktop\\Q3\\train.csv'
df = pd.read_csv(csv_path)
```

In [3]:
```
image_dir = 'C:\\Users\\Dummy\\Desktop\\Q3\\Train\\'
```

In [4]:
```
def load_images(df, image_dir, target_size=(100, 100)):
    images = []
    labels = []
    for index, row in df.iterrows():
        img_path = image_dir + str(row['ID'])
        img = load_img(img_path, target_size=target_size)
        img_array = img_to_array(img)
        images.append(img_array)
        labels.append(row['Class'])
    return images, labels
```

In [5]:
```
target_size = (100, 100)
X, y = load_images(df, image_dir, target_size=target_size)
```

In [6]:
```
label_encoder = LabelEncoder()
y_numeric = label_encoder.fit_transform(y)
```

In [7]:
```
X_train, X_val, y_train_numeric, y_val_numeric = train_test_split(X, y_numeric, test_size=0.2, random_state=42
```

In [8]:
```
base_model = InceptionV3(weights='imagenet', include_top=False, input_shape=(target_size[0], target_size[1], 3
```

In [9]:
```
x = base_model.output
x = GlobalAveragePooling2D()(x)
x = Dense(1024, activation='relu')(x)
x = Dense(512, activation='relu')(x)
predictions = Dense(1, activation='linear')(x)

model = Model(inputs=base_model.input, outputs=predictions)
```

In [10]:
```
for layer in base_model.layers:
    layer.trainable = False

model.compile(optimizer=Adam(), loss='mean_squared_error', metrics=['mae'])
```

In [ ]:
```
X_train = np.array(X_train)
X_val = np.array(X_val)

model.fit(
    X_train, y_train_numeric,
    validation_data=(X_val, y_val_numeric),
    epochs=10,
    batch_size=32,
    verbose=2
)
```

```
Epoch 1/10
```

In [ ]:
```
loss, mae = model.evaluate(X_val, y_val, verbose=2)
print(f'Mean Absolute Error on Validation Set: {mae}')
```

Q4

In [31]:
```
import os
import random
from sklearn.model_selection import train_test_split
```

In [32]:
```
data_dir = 'C:\\Users\\Dummy\\Desktop\\Q4'
```

In [33]:
```
img_width, img_height = 224, 224
batch_size = 32
num_images_per_category = 500
```

In [34]:
```
all_image_paths = []
all_labels = []
```

In [35]:
```
for folder in os.listdir(data_dir):
    folder_path = os.path.join(data_dir, folder)


    if os.path.isdir(folder_path):

        image_files = [os.path.join(folder_path, file) for file in os.listdir(folder_path) if file.lower().end

        selected_images = random.sample(image_files, min(num_images_per_category, len(image_files)))

        all_image_paths.extend(selected_images)
        all_labels.extend([folder] * len(selected_images))
```

In [36]:
```
train_image_paths, test_image_paths, train_labels, test_labels = train_test_split(
    all_image_paths, all_labels, test_size=0.2, random_state=42
)
```

In [37]:
```
train_datagen = ImageDataGenerator(
    rescale=1./255,
    shear_range=0.2,
    zoom_range=0.2,
    horizontal_flip=True
)
```

In [39]:
```
import pandas as pd
test_datagen = ImageDataGenerator(rescale=1./255)

train_generator = train_datagen.flow_from_dataframe(
    pd.DataFrame({'filename': train_image_paths, 'category': train_labels}),
    x_col='filename',
    y_col='category',
    target_size=(img_width, img_height),
    batch_size=batch_size,
    class_mode='categorical'
)
```

```
Found 12000 validated image filenames belonging to 30 classes.
```

In [40]:
```
test_generator = test_datagen.flow_from_dataframe(
    pd.DataFrame({'filename': test_image_paths, 'category': test_labels}),
    x_col='filename',
    y_col='category',
    target_size=(img_width, img_height),
    batch_size=batch_size,
    class_mode='categorical'
)
```

```
Found 3000 validated image filenames belonging to 30 classes.
```

In [41]:
```
model = models.Sequential()
```

In [42]:
```
model.add(layers.Conv2D(96, (11, 11), strides=(4, 4), activation='relu', input_shape=(img_width, img_height, 3
model.add(layers.MaxPooling2D(pool_size=(3, 3), strides=(2, 2)))
model.add(layers.Conv2D(256, (5, 5), activation='relu'))
model.add(layers.MaxPooling2D(pool_size=(3, 3), strides=(2, 2)))
model.add(layers.Conv2D(384, (3, 3), activation='relu'))
model.add(layers.Conv2D(384, (3, 3), activation='relu'))
model.add(layers.Conv2D(256, (3, 3), activation='relu'))
model.add(layers.MaxPooling2D(pool_size=(3, 3), strides=(2, 2)))
```

In [43]:
```
model.add(layers.Flatten())
```

In [44]:
```
model.add(layers.Dense(4096, activation='relu'))
model.add(layers.Dropout(0.5))
model.add(layers.Dense(4096, activation='relu'))
model.add(layers.Dropout(0.5))
model.add(layers.Dense(len(train_generator.class_indices), activation='softmax'))
```

In [45]:
```
model.compile(optimizer='adam', loss='categorical_crossentropy', metrics=['accuracy'])
```

In [46]:
```
model.fit(
    train_generator,
    steps_per_epoch=train_generator.samples // batch_size,
    epochs=10,
    validation_data=test_generator,
    validation_steps=test_generator.samples // batch_size
)
```

```
Epoch 1/10
375/375 [==============================] - 562s 1s/step - loss: 3.4036 - accuracy: 0.0309 - val_loss: 3.4027
- val_accuracy: 0.0289
Epoch 2/10
375/375 [==============================] - 566s 2s/step - loss: 3.4020 - accuracy: 0.0295 - val_loss: 3.4022
- val_accuracy: 0.0289
Epoch 3/10
375/375 [==============================] - 612s 2s/step - loss: 3.4017 - accuracy: 0.0310 - val_loss: 3.4022
- val_accuracy: 0.0289
Epoch 4/10
375/375 [==============================] - 564s 2s/step - loss: 3.4015 - accuracy: 0.0341 - val_loss: 3.4024
- val_accuracy: 0.0289
Epoch 5/10
375/375 [==============================] - 549s 1s/step - loss: 3.4015 - accuracy: 0.0320 - val_loss: 3.4025
- val_accuracy: 0.0299
Epoch 6/10
375/375 [==============================] - 560s 1s/step - loss: 3.4016 - accuracy: 0.0328 - val_loss: 3.4024
- val_accuracy: 0.0286
Epoch 7/10
375/375 [==============================] - 543s 1s/step - loss: 3.4015 - accuracy: 0.0302 - val_loss: 3.4024
- val_accuracy: 0.0282
Epoch 8/10
375/375 [==============================] - 532s 1s/step - loss: 3.4015 - accuracy: 0.0317 - val_loss: 3.4025
- val_accuracy: 0.0289
Epoch 9/10
375/375 [==============================] - 546s 1s/step - loss: 3.4015 - accuracy: 0.0327 - val_loss: 3.4026
- val_accuracy: 0.0286
Epoch 10/10
375/375 [==============================] - 549s 1s/step - loss: 3.4014 - accuracy: 0.0340 - val_loss: 3.4027
- val_accuracy: 0.0289
```

Out[46]:
```
<keras.src.callbacks.History at 0x238477ce590>
```

Q5

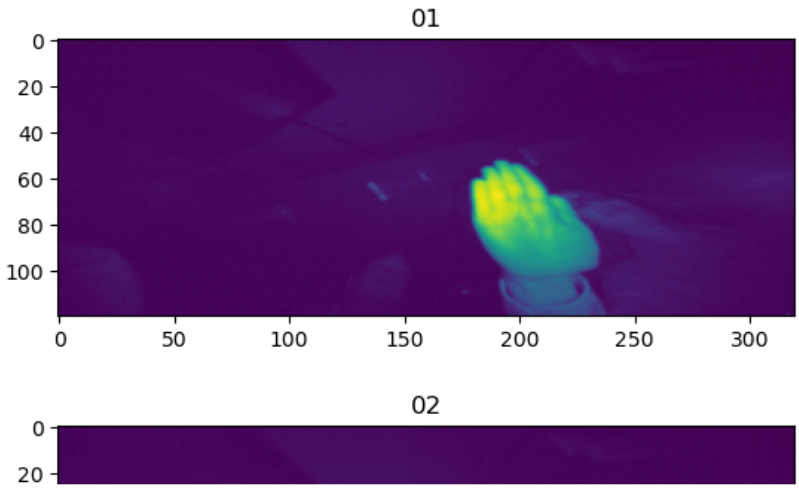
In [1]:
```
import numpy as np
import os
from PIL import Image
import matplotlib.pyplot as plt
from random import randint
from keras.utils import to_categorical
from sklearn.model_selection import train_test_split
from keras import layers
from keras import models
```

In [2]:
```
lookup = dict()
reverselookup = dict()
count = 0
```

In [3]:
```
root_folder = 'C:\\Users\\Dummy\\Desktop\\leapGestRecog'

for subject_folder in os.listdir(root_folder):
    if not subject_folder.startswith('.'):
        lookup[subject_folder] = count
        reverselookup[count] = subject_folder
        count += 1
```

In [7]:
```
import os
import numpy as np
from PIL import Image

x_data = []
y_data = []
datacount = 0

default_value = 0

for subject_folder in os.listdir(root_folder):
    if not subject_folder.startswith('.'):
        count = 0

        for gesture_folder in os.listdir(os.path.join(root_folder, subject_folder)):
            if gesture_folder.startswith('.'):
                continue

            gesture_identifier = gesture_folder.split('_')[0]
            for image_file in os.listdir(os.path.join(root_folder, subject_folder, gesture_folder)):
                img_path = os.path.join(root_folder, subject_folder, gesture_folder, image_file)
                img = Image.open(img_path).convert('L')
                img = img.resize((320, 120))
                arr = np.array(img)
                x_data.append(arr)
                count += 1

                y_values = np.full((1, 1), lookup.get(gesture_identifier, default_value))
                y_data.append(y_values)

        datacount += count

x_data = np.array(x_data, dtype='float32')
y_data = np.array(y_data)
y_data = y_data.reshape(datacount, 1)
```

In [8]:
```

for i in range(0, 10):
    plt.imshow(x_data[i * 200, :, :])
    plt.title(reverselookup[y_data[i * 200, 0]])
    plt.show()
```



In [19]:
```
y_data = to_categorical(y_data,num_classes=2000)

x_data = x_data.reshape((datacount, 120, 320, 1))
x_data /= 255
```

In [28]:
```
y_data = np.argmax(y_data, axis=1)

x_train, x_further, y_train, y_further = train_test_split(x_data, y_data, test_size=0.2)
x_validate, x_test, y_validate, y_test = train_test_split(x_further, y_further, test_size=0.5)
```

In [29]:
```
model = models.Sequential()
model.add(layers.Conv2D(32, (5, 5), strides=(2, 2), activation='relu', input_shape=(120, 320, 1)))
model.add(layers.MaxPooling2D((2, 2)))
model.add(layers.Conv2D(64, (3, 3), activation='relu'))
model.add(layers.MaxPooling2D((2, 2)))
model.add(layers.Conv2D(64, (3, 3), activation='relu'))
model.add(layers.MaxPooling2D((2, 2)))
model.add(layers.Flatten())
model.add(layers.Dense(128, activation='relu'))
model.add(layers.Dense(2000, activation='softmax'))
```

In [ ]:
```
model.compile(optimizer='rmsprop', loss='sparse_categorical_crossentropy', metrics=['accuracy'])
model.fit(x_train, y_train, epochs=10, batch_size=64, verbose=1, validation_data=(x_validate, y_validate))
```