In [2]:
```python
import numpy as np
import tensorflow as tf
from tensorflow.keras.models import Sequential
from tensorflow.keras.layers import Conv2D, MaxPooling2D, Flatten, Dense, Dropout, BatchNormalization
from tensorflow.keras.optimizers import Adam
from tensorflow.keras.preprocessing.image import ImageDataGenerator
from skimage.transform import resize
from tensorflow.keras.regularizers import l2

def load_data():
    # Load training data and labels from numpy files
    data = np.load('data_train.npy')
    labels = np.load('labels_train.npy')
    # Reshape and resize data for the model
    data_reshaped = data.T.reshape((-1, 300, 300, 3))
    resized_data = np.array([resize(img, (100, 100, 3), anti_aliasing=True) for img in data_reshaped])
    # Normalize pixel values if necessary
    if resized_data.max() > 1.0:
        data_normalized = resized_data.astype(np.float32) / 255.0
    else:
        data_normalized = resized_data
    return data_normalized, labels

def build_model():
    # Define the CNN architecture
    model = Sequential([
        Conv2D(64, (3, 3), activation='relu', input_shape=(100, 100, 3)),
        BatchNormalization(),
        MaxPooling2D((2, 2)),
        Conv2D(128, (3, 3), activation='relu'),
        BatchNormalization(),
        MaxPooling2D((2, 2)),
        Conv2D(256, (3, 3), activation='relu'),
        BatchNormalization(),
        MaxPooling2D((2, 2)),
        Conv2D(512, (3, 3), activation='relu'),
        BatchNormalization(),
        MaxPooling2D((2, 2)),
        Flatten(),
        Dense(512, activation='relu', kernel_regularizer=l2(0.001)),
        Dropout(0.5),
        Dense(256, activation='relu', kernel_regularizer=l2(0.001)),
        Dense(9, activation='softmax')
    ])
    return model

def random_brightness(image):
    # Apply random brightness to an image
    image = tf.expand_dims(image, 0)
    image = tf.image.random_brightness(image, max_delta=0.1)
    image = tf.squeeze(image)
    return image

def adjust_contrast(img):
    # Randomly adjust contrast of an image
    contrast_factor = tf.random.uniform([], 0.9, 1.1)
    return tf.image.adjust_contrast(img, contrast_factor)

def preprocess_image(img):
    # Preprocess images by applying random brightness and contrast adjustments
    img = random_brightness(img)
    img = adjust_contrast(img)
    img = tf.clip_by_value(img, 0.0, 1.0)
    return img

def train():
    # Load data and labels
    data, labels = load_data()
    # Define image data generator for data augmentation
    datagen = ImageDataGenerator(
        rotation_range=15,
        width_shift_range=0.1,
        height_shift_range=0.1,
        zoom_range=0.1,
        horizontal_flip=True,
        fill_mode='nearest',
        preprocessing_function=preprocess_image
    )
    # Build, compile and train the model
    model = build_model()
    model.compile(optimizer=Adam(learning_rate=0.0004), loss='sparse_categorical_crossentropy', metrics=['accuracy'])

    # Note: No train-validation split here, using full data for training
    train_generator = datagen.flow(data, labels, batch_size=32)
    model.fit(train_generator, epochs=35)

    # Save final model to a file
    model.save('final_model.h5')
```

```python
if __name__ == "__main__":
    train()
```

```
2023-12-06 22:14:48.293345: I tensorflow/core/platform/cpu_feature_guard.cc:151] This TensorFlow binary is optimized
with oneAPI Deep Neural Network Library (oneDNN) to use the following CPU instructions in performance-critical operat
ions:  SSE4.1 SSE4.2 AVX AVX2 FMA
To enable them in other operations, rebuild TensorFlow with the appropriate compiler flags.
2023-12-06 22:14:49.410557: I tensorflow/core/common_runtime/gpu/gpu_device.cc:1525] Created device /job:localhost/re
plica:0/task:0/device:GPU:0 with 78911 MB memory:  -> device: 0, name: NVIDIA A100-SXM4-80GB, pci bus id: 0000:87:00.
0, compute capability: 8.0
2023-12-06 22:14:49.412308: I tensorflow/core/common_runtime/gpu/gpu_device.cc:1525] Created device /job:localhost/re
plica:0/task:0/device:GPU:1 with 78911 MB memory:  -> device: 1, name: NVIDIA A100-SXM4-80GB, pci bus id: 0000:bd:00.
0, compute capability: 8.0
Epoch 1/35
2023-12-06 22:14:51.558415: I tensorflow/stream_executor/cuda/cuda_dnn.cc:366] Loaded cuDNN version 8201
2023-12-06 22:14:53.280169: I tensorflow/stream_executor/cuda/cuda_blas.cc:1774] TensorFloat-32 will be used for the
matrix multiplication. This will only be logged once.
264/264 [==============================] - 25s 82ms/step - loss: 3.7908 - accuracy: 0.1541
Epoch 2/35
264/264 [==============================] - 22s 82ms/step - loss: 3.3248 - accuracy: 0.1899
Epoch 3/35
264/264 [==============================] - 22s 83ms/step - loss: 3.0241 - accuracy: 0.2494
Epoch 4/35
264/264 [==============================] - 22s 82ms/step - loss: 2.7460 - accuracy: 0.3109
Epoch 5/35
264/264 [==============================] - 22s 82ms/step - loss: 2.4772 - accuracy: 0.3577
Epoch 6/35
264/264 [==============================] - 22s 82ms/step - loss: 2.2139 - accuracy: 0.4169
Epoch 7/35
264/264 [==============================] - 22s 83ms/step - loss: 1.9966 - accuracy: 0.4781
Epoch 8/35
264/264 [==============================] - 21s 81ms/step - loss: 1.8130 - accuracy: 0.5307
Epoch 9/35
264/264 [==============================] - 22s 82ms/step - loss: 1.6725 - accuracy: 0.5755
Epoch 10/35
264/264 [==============================] - 22s 82ms/step - loss: 1.5240 - accuracy: 0.6328
Epoch 11/35
264/264 [==============================] - 22s 82ms/step - loss: 1.4213 - accuracy: 0.6784
Epoch 12/35
264/264 [==============================] - 22s 82ms/step - loss: 1.3165 - accuracy: 0.7193
Epoch 13/35
264/264 [==============================] - 22s 82ms/step - loss: 1.2642 - accuracy: 0.7532
Epoch 14/35
264/264 [==============================] - 22s 81ms/step - loss: 1.2230 - accuracy: 0.7703
Epoch 15/35
264/264 [==============================] - 22s 82ms/step - loss: 1.1504 - accuracy: 0.8004
Epoch 16/35
264/264 [==============================] - 22s 82ms/step - loss: 1.1072 - accuracy: 0.8118
Epoch 17/35
264/264 [==============================] - 22s 82ms/step - loss: 1.0726 - accuracy: 0.8271
Epoch 18/35
264/264 [==============================] - 21s 81ms/step - loss: 1.0392 - accuracy: 0.8435
Epoch 19/35
264/264 [==============================] - 22s 82ms/step - loss: 1.0104 - accuracy: 0.8493
Epoch 20/35
264/264 [==============================] - 21s 81ms/step - loss: 0.9901 - accuracy: 0.8585
Epoch 21/35
264/264 [==============================] - 22s 82ms/step - loss: 0.9570 - accuracy: 0.8691
Epoch 22/35
264/264 [==============================] - 22s 82ms/step - loss: 0.9258 - accuracy: 0.8748
Epoch 23/35
264/264 [==============================] - 22s 82ms/step - loss: 0.9005 - accuracy: 0.8842
Epoch 24/35
264/264 [==============================] - 22s 82ms/step - loss: 0.8962 - accuracy: 0.8819
Epoch 25/35
264/264 [==============================] - 22s 82ms/step - loss: 0.8713 - accuracy: 0.8922
Epoch 26/35
264/264 [==============================] - 22s 81ms/step - loss: 0.8577 - accuracy: 0.8941
Epoch 27/35
264/264 [==============================] - 22s 82ms/step - loss: 0.8422 - accuracy: 0.8981
Epoch 28/35
264/264 [==============================] - 22s 82ms/step - loss: 0.8287 - accuracy: 0.8970
Epoch 29/35
264/264 [==============================] - 22s 82ms/step - loss: 0.8066 - accuracy: 0.9010
Epoch 30/35
264/264 [==============================] - 21s 81ms/step - loss: 0.7808 - accuracy: 0.9075
Epoch 31/35
264/264 [==============================] - 22s 82ms/step - loss: 0.7514 - accuracy: 0.9146
Epoch 32/35
264/264 [==============================] - 22s 82ms/step - loss: 0.7588 - accuracy: 0.9137
Epoch 33/35
264/264 [==============================] - 22s 82ms/step - loss: 0.7389 - accuracy: 0.9163
Epoch 34/35
264/264 [==============================] - 22s 82ms/step - loss: 0.7356 - accuracy: 0.9135
Epoch 35/35
264/264 [==============================] - 22s 82ms/step - loss: 0.7208 - accuracy: 0.9173
```

In [ ]: