```
import zipfile
import os

# Provide the path to your ZIP file in Google Drive
zip_path = '/content/drive/MyDrive/zip-folder.zip'

# Specify the directory where you want to extract the files
extract_path = '/content/extracted_files'

# Extract the ZIP file
with zipfile.ZipFile(zip_path, 'r') as zip_ref:
    zip_ref.extractall(extract_path)

print(f"Files extracted to {extract_path}")
```

```
Files extracted to /content/extracted_files
```

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

```
Mounted at /content/drive
```

```
import tensorflow as tf
from tensorflow.keras.preprocessing.image import ImageDataGenerator
from tensorflow.keras.applications import InceptionV3
from tensorflow.keras.layers import Dense, GlobalAveragePooling2D
from tensorflow.keras.models import Model
from tensorflow.keras.optimizers import Adam


# Define paths to your dataset
train_dir = "extracted_files/train"
val_dir = "extracted_files/val"
test_dir = "extracted_files/test"


# Set parameters
img_height, img_width = 299, 299  # InceptionV3 input size
batch_size = 32


# Load datasets
train_ds = tf.keras.preprocessing.image_dataset_from_directory(
    train_dir,
    image_size=(img_height, img_width),
    batch_size=batch_size,
    label_mode='int'
)

val_ds = tf.keras.preprocessing.image_dataset_from_directory(
    val_dir,
    image_size=(img_height, img_width),
    batch_size=batch_size,
    label_mode='int'
)

test_ds = tf.keras.preprocessing.image_dataset_from_directory(
    test_dir,
    image_size=(img_height, img_width),
    batch_size=batch_size,
    label_mode='int'
)
```

```
Found 6953 files belonging to 100 classes.
Found 1966 files belonging to 100 classes.
Found 1034 files belonging to 100 classes.
```

```
# Auto-detect the number of classes
class_names = train_ds.class_names
num_classes = len(class_names)
print(f"Number of classes: {num_classes}")
```

```
Number of classes: 100
```

```
from tensorflow.keras.applications.inception_v3 import preprocess_input
```

```
# Preprocess datasets
train_ds = train_ds.map(lambda x, y: (preprocess_input(x), y))
val_ds = val_ds.map(lambda x, y: (preprocess_input(x), y))
test_ds = test_ds.map(lambda x, y: (preprocess_input(x), y))
```

```
# Define the InceptionV3 model
base_model = InceptionV3(weights='imagenet', include_top=False, input_shape=(img_height, img_width, 3))
base_model.trainable = False  # Freeze the base model
```

```
Downloading data from https://storage.googleapis.com/tensorflow/keras-applications/inception_v3/incepti
87910968/87910968 ━━━━━━━━━━━━━━━━━━━━ 5s 0us/step
```

```
from tensorflow.keras import models, layers
```

```
# Add custom layers on top
model = models.Sequential([
    base_model,
    layers.GlobalAveragePooling2D(),
    layers.Dense(1024, activation='relu'),
    layers.Dropout(0.5),
    layers.Dense(num_classes, activation='softmax')
])
```

```
# Compile the model
model.compile(optimizer=Adam(learning_rate=0.0001), loss='sparse_categorical_crossentropy', metrics=['accur
```

```
# Train the model
epochs = 10
history = model.fit(train_ds, validation_data=val_ds, epochs=epochs)
```

```
Epoch 1/10
218/218 ━━━━━━━━━━━━━━━━━━━━ 78s 256ms/step - accuracy: 0.0478 - loss: 4.5129 - val_accuracy: 0.3591 -
Epoch 2/10
218/218 ━━━━━━━━━━━━━━━━━━━━ 30s 139ms/step - accuracy: 0.2795 - loss: 3.3280 - val_accuracy: 0.5799 -
Epoch 3/10
218/218 ━━━━━━━━━━━━━━━━━━━━ 45s 158ms/step - accuracy: 0.4836 - loss: 2.3804 - val_accuracy: 0.7401 -
```

```
Epoch 4/10
218/218 ━━━━━━━━━━━━━━━━━━━━ 41s 159ms/step - accuracy: 0.6380 - loss: 1.7690 - val_accuracy: 0.8250 -
Epoch 5/10
218/218 ━━━━━━━━━━━━━━━━━━━━ 37s 142ms/step - accuracy: 0.7516 - loss: 1.3268 - val_accuracy: 0.8835 -
Epoch 6/10
218/218 ━━━━━━━━━━━━━━━━━━━━ 31s 143ms/step - accuracy: 0.8259 - loss: 1.0100 - val_accuracy: 0.9232 -
Epoch 7/10
218/218 ━━━━━━━━━━━━━━━━━━━━ 41s 144ms/step - accuracy: 0.8803 - loss: 0.7827 - val_accuracy: 0.9527 -
Epoch 8/10
218/218 ━━━━━━━━━━━━━━━━━━━━ 35s 162ms/step - accuracy: 0.9088 - loss: 0.6182 - val_accuracy: 0.9741 -
Epoch 9/10
218/218 ━━━━━━━━━━━━━━━━━━━━ 37s 145ms/step - accuracy: 0.9433 - loss: 0.4804 - val_accuracy: 0.9852 -
Epoch 10/10
218/218 ━━━━━━━━━━━━━━━━━━━━ 32s 146ms/step - accuracy: 0.9602 - loss: 0.3865 - val_accuracy: 0.9883 -
```

```
# Save the entire model
model.save('my_model.h5')
```

```
WARNING:absl:You are saving your model as an HDF5 file via `model.save()` or `keras.saving.save_model(m
```

```
# Train the model for more 40 epochs
epochs = 40
history = model.fit(train_ds, validation_data=val_ds, epochs=epochs)
```

```
Epoch 1/40
218/218 ━━━━━━━━━━━━━━━━━━━━ 30s 137ms/step - accuracy: 0.9741 - loss: 0.3172 - val_accuracy: 0.9924
Epoch 2/40
218/218 ━━━━━━━━━━━━━━━━━━━━ 45s 156ms/step - accuracy: 0.9770 - loss: 0.2659 - val_accuracy: 0.9964
Epoch 3/40
218/218 ━━━━━━━━━━━━━━━━━━━━ 31s 141ms/step - accuracy: 0.9833 - loss: 0.2163 - val_accuracy: 0.9975
Epoch 4/40
218/218 ━━━━━━━━━━━━━━━━━━━━ 31s 142ms/step - accuracy: 0.9917 - loss: 0.1763 - val_accuracy: 0.9969
Epoch 5/40
218/218 ━━━━━━━━━━━━━━━━━━━━ 31s 143ms/step - accuracy: 0.9892 - loss: 0.1580 - val_accuracy: 0.9985
Epoch 6/40
218/218 ━━━━━━━━━━━━━━━━━━━━ 41s 143ms/step - accuracy: 0.9937 - loss: 0.1307 - val_accuracy: 0.9995
Epoch 7/40
218/218 ━━━━━━━━━━━━━━━━━━━━ 31s 144ms/step - accuracy: 0.9959 - loss: 0.1131 - val_accuracy: 0.9990
Epoch 8/40
218/218 ━━━━━━━━━━━━━━━━━━━━ 41s 144ms/step - accuracy: 0.9969 - loss: 0.0943 - val_accuracy: 0.9990
Epoch 9/40
218/218 ━━━━━━━━━━━━━━━━━━━━ 35s 162ms/step - accuracy: 0.9977 - loss: 0.0860 - val_accuracy: 0.9995
Epoch 10/40
218/218 ━━━━━━━━━━━━━━━━━━━━ 41s 161ms/step - accuracy: 0.9982 - loss: 0.0767 - val_accuracy: 0.9990
Epoch 11/40
218/218 ━━━━━━━━━━━━━━━━━━━━ 41s 160ms/step - accuracy: 0.9987 - loss: 0.0613 - val_accuracy: 1.0000
Epoch 12/40
218/218 ━━━━━━━━━━━━━━━━━━━━ 32s 145ms/step - accuracy: 0.9989 - loss: 0.0578 - val_accuracy: 0.9995
Epoch 13/40
218/218 ━━━━━━━━━━━━━━━━━━━━ 32s 145ms/step - accuracy: 0.9993 - loss: 0.0482 - val_accuracy: 1.0000
Epoch 14/40
218/218 ━━━━━━━━━━━━━━━━━━━━ 32s 145ms/step - accuracy: 0.9989 - loss: 0.0438 - val_accuracy: 1.0000
Epoch 15/40
218/218 ━━━━━━━━━━━━━━━━━━━━ 32s 147ms/step - accuracy: 0.9997 - loss: 0.0402 - val_accuracy: 1.0000
Epoch 16/40
218/218 ━━━━━━━━━━━━━━━━━━━━ 41s 145ms/step - accuracy: 0.9997 - loss: 0.0370 - val_accuracy: 0.9995
Epoch 17/40
218/218 ━━━━━━━━━━━━━━━━━━━━ 41s 146ms/step - accuracy: 0.9996 - loss: 0.0319 - val_accuracy: 1.0000
Epoch 18/40
218/218 ━━━━━━━━━━━━━━━━━━━━ 44s 161ms/step - accuracy: 0.9997 - loss: 0.0294 - val_accuracy: 1.0000
```

```
Epoch 19/40
218/218 ━━━━━━━━━━━━━━━━━━━━ 41s 160ms/step - accuracy: 0.9998 - loss: 0.0261 - val_accuracy: 1.000
Epoch 20/40
218/218 ━━━━━━━━━━━━━━━━━━━━ 38s 144ms/step - accuracy: 0.9997 - loss: 0.0243 - val_accuracy: 0.999
Epoch 21/40
218/218 ━━━━━━━━━━━━━━━━━━━━ 41s 144ms/step - accuracy: 0.9994 - loss: 0.0229 - val_accuracy: 1.000
Epoch 22/40
218/218 ━━━━━━━━━━━━━━━━━━━━ 41s 144ms/step - accuracy: 0.9996 - loss: 0.0209 - val_accuracy: 0.999
Epoch 23/40
218/218 ━━━━━━━━━━━━━━━━━━━━ 41s 144ms/step - accuracy: 0.9997 - loss: 0.0171 - val_accuracy: 1.000
Epoch 24/40
218/218 ━━━━━━━━━━━━━━━━━━━━ 41s 144ms/step - accuracy: 1.0000 - loss: 0.0166 - val_accuracy: 1.000
Epoch 25/40
218/218 ━━━━━━━━━━━━━━━━━━━━ 44s 158ms/step - accuracy: 1.0000 - loss: 0.0158 - val_accuracy: 1.000
Epoch 26/40
218/218 ━━━━━━━━━━━━━━━━━━━━ 32s 144ms/step - accuracy: 1.0000 - loss: 0.0125 - val_accuracy: 1.000
Epoch 27/40
218/218 ━━━━━━━━━━━━━━━━━━━━ 32s 145ms/step - accuracy: 0.9993 - loss: 0.0148 - val_accuracy: 1.000
Epoch 28/40
218/218 ━━━━━━━━━━━━━━━━━━━━ 41s 144ms/step - accuracy: 0.9999 - loss: 0.0104 - val_accuracy: 1.000
Epoch 29/40
```

```
# Evaluate on training data
train_loss, train_acc = model.evaluate(train_ds)
print(f"Train Loss: {train_loss:.4f}\tTrain Accuracy: {train_acc * 100:.2f}%")

# Evaluate on test data
test_loss, test_acc = model.evaluate(test_ds)
print(f"Test Loss: {test_loss:.4f}\tTest Accuracy: {test_acc * 100:.2f}%")
```

```
218/218 ━━━━━━━━━━━━━━━━━━━━ 28s 128ms/step - accuracy: 1.0000 - loss: 4.9384e-04
Train Loss: 0.0005      Train Accuracy: 100.00%
33/33 ━━━━━━━━━━━━━━━━━━━━ 11s 352ms/step - accuracy: 1.0000 - loss: 0.0021
Test Loss: 0.0025       Test Accuracy: 100.00%
```

```
# Save the entire model
model.save('my_model.h5')
```

```
WARNING:absl:You are saving your model as an HDF5 file via `model.save()` or `keras.saving.save_model(m
```

Start coding or generate with AI.