Using Folder Names as Labels in TensorFlow Image Classification

This document explains how to use folder names as labels for image classification using TensorFlow on Google Colab. When using TensorFlow's `image_dataset_from_directory` method, it automatically assigns labels to images based on the folder names. This feature simplifies the labeling process and makes it easier to manage datasets with multiple classes.

1. Dataset Structure and Label Assignment

The key idea behind using folder names as labels is to organize your image files into subfolders, where each subfolder represents a class. For example, if you want to classify images of fruits such as apples, oranges, and pears, your dataset structure should look like this:

```
/MyDrive/fruit_dataset/
  ├── apple/
  │   ├── image1.jpg
  │   ├── image2.jpg
  │   └── ...
  ├── orange/
  │   ├── image1.jpg
  │   ├── image2.jpg
  │   └── ...
  └── pear/
      ├── image1.jpg
      ├── image2.jpg
      └── ...
```

Each subfolder (`apple`, `orange`, `pear`) represents a unique class label, and all images within a folder are considered as belonging to that class.

2. Loading the Dataset Using `image_dataset_from_directory`

The `image_dataset_from_directory` method in TensorFlow's Keras API can be used to load images from a directory while automatically assigning labels based on folder names. It also supports splitting the data into training and validation sets. Below is an example code to load a dataset structured as shown above and assign labels based on the folder names:

```
# Import necessary libraries
import tensorflow as tf
import matplotlib.pyplot as plt
from google.colab import drive
```

```
# Mount Google Drive
drive.mount('/content/drive')

# Define the directory containing the dataset
data_dir = '/content/drive/MyDrive/fruit_dataset'

# Load the dataset and assign labels based on folder names
train_dataset = tf.keras.preprocessing.image_dataset_from_directory(
    data_dir,               # Path to the main dataset directory
    validation_split=0.3,    # Reserve 30% for validation
    subset="training",       # Use "training" for the training set
    seed=123,                # Seed for reproducibility
    image_size=(160, 160),   # Resize images to 160x160 pixels
    batch_size=32,           # Load images in batches of 32
    label_mode='categorical'  # Use categorical labels (one-hot encoded)
)

validation_dataset = tf.keras.preprocessing.image_dataset_from_directory(
    data_dir,               # Path to the main dataset directory
    validation_split=0.3,    # Reserve 30% for validation
    subset="validation",     # Use "validation" for the validation set
    seed=123,                # Seed for reproducibility
    image_size=(160, 160),   # Resize images to 160x160 pixels
    batch_size=32,           # Load images in batches of 32
    label_mode='categorical'  # Use categorical labels (one-hot encoded)
)

# Display class names to verify labels
class_names = train_dataset.class_names
print("Class Names:", class_names)
```

In this code:

- `data_dir`: Specifies the main directory containing the class subfolders (e.g., `apple`, `orange`, `pear`).
- `validation_split`: Splits the dataset into training and validation sets. Here, 30% is reserved for validation.
- `subset`: Defines whether to load data for training or validation. Must be used in conjunction with `validation_split`.
- `seed`: Ensures reproducibility of the train/validation split.
- `image_size`: Resizes images to the specified dimensions (160x160 pixels in this example).
- `batch_size`: Defines how many images are loaded at a time (32 images per batch in this example).

- `label_mode`: Defines the format of the labels. Using `'categorical'` assigns labels in a one-hot encoded format, where each class is represented by a vector of zeros and ones.

3. Verifying and Using Assigned Labels

Once the dataset is loaded, you can verify that the labels have been correctly assigned based on the folder names by printing the `class_names` attribute of the dataset. This will list all the class names in alphabetical order as detected from the folder structure. You can also display some sample images with their corresponding labels to visually verify.

```
# Verify the class names (labels)
print("Class Names:", class_names)

# Display some sample images with their labels
import matplotlib.pyplot as plt

plt.figure(figsize=(10, 10))
for images, labels in train_dataset.take(1):  # Take one batch of images
  for i in range(9):  # Display the first 9 images
    ax = plt.subplot(3, 3, i + 1)
    plt.imshow(images[i].numpy().astype("uint8"))
    plt.title(class_names[tf.argmax(labels[i])])  # Show the label for the image
    plt.axis("off")
```

In this code, the `train_dataset.take(1)` method retrieves the first batch of images, and a loop is used to display the first 9 images along with their labels. The `tf.argmax(labels[i])` method is used to get the index of the class with the highest probability, which corresponds to the class name in `class_names`.

4. Training with Labeled Data

Now that the dataset is correctly loaded and labeled, you can use it to train a machine learning model. Below is an example of building and training a simple CNN model using the labeled data:

```
# Define a simple CNN model
model = tf.keras.Sequential([
  tf.keras.layers.Rescaling(1./255, input_shape=(160, 160, 3)),  # Normalize pixel values
  tf.keras.layers.Conv2D(32, (3, 3), activation='relu'),
  tf.keras.layers.MaxPooling2D(),
  tf.keras.layers.Conv2D(64, (3, 3), activation='relu'),
  tf.keras.layers.MaxPooling2D(),
  tf.keras.layers.Conv2D(128, (3, 3), activation='relu'),
```

```
    tf.keras.layers.MaxPooling2D(),
    tf.keras.layers.Flatten(),
    tf.keras.layers.Dense(128, activation='relu'),
    tf.keras.layers.Dense(len(class_names), activation='softmax')  # Output layer with one node per class
])

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

# Train the model
history = model.fit(
    train_dataset,
    validation_data=validation_dataset,
    epochs=10  # Train for 10 epochs
)
```

5. Conclusion

By organizing your dataset into subfolders where each folder represents a class label, you can simplify the labeling process. TensorFlow's `image_dataset_from_directory` method makes it easy to load images and automatically assign labels based on folder names. This document has demonstrated how to set up the dataset, verify the labels, and use the labeled data to train a CNN model.