SKILL UPGRADE

INTENSHIP TASK -2

Title : IMAGE CLASSIFICATION

TASK DESCRIPTION:

- Develop an image classification model using a dataset of your choice (e.g., CIFAR-10, MNIST).
- Utilize a deep learning framework like TensorFlow or PyTorch.
- Train the model to classify images into predefined categories and evaluate its accuracy.

To complete your image classification task, I'll walk you through the steps to develop and train an image classification model using TensorFlow with the CIFAR-10 dataset. Here's a detailed guide:

1. **Set Up the Environment**
 - Install the required libraries:
 ```bash
 pip install tensorflow matplotlib
 ```

2. **Import Necessary Libraries**
 ```python
 import tensorflow as tf
 from tensorflow.keras import datasets, layers, models
 import matplotlib.pyplot as plt
 ```

3. **Load and Preprocess the Data**
 - CIFAR-10 is a popular dataset containing 60,000 32x32 color images in 10 classes, with 6,000 images per class.
 ```python
 # Load CIFAR-10 dataset
 (train_images, train_labels), (test_images, test_labels) = datasets.cifar10.load_data()

 # Normalize pixel values to be between 0 and 1
 train_images, test_images = train_images / 255.0, test_images / 255.0
 ```

4. **Visualize the Data**
 - It's helpful to visualize the data to understand what you are working with.

```python
class_names = ['airplane', 'automobile', 'bird', 'cat', 'deer', 'dog', 'frog', 'horse', 'ship', 'truck']

plt.figure(figsize=(10,10))
for i in range(25):
    plt.subplot(5,5,i+1)
    plt.xticks([])
    plt.yticks([])
    plt.grid(False)
    plt.imshow(train_images[i], cmap=plt.cm.binary)
    plt.xlabel(class_names[train_labels[i][0]])
plt.show()
```

5. **Build the Convolutional Neural Network (CNN) Model**
  - A CNN is ideal for image classification tasks.
```python
model = models.Sequential([
    layers.Conv2D(32, (3, 3), activation='relu', input_shape=(32, 32, 3)),
    layers.MaxPooling2D((2, 2)),
    layers.Conv2D(64, (3, 3), activation='relu'),
    layers.MaxPooling2D((2, 2)),
    layers.Conv2D(64, (3, 3), activation='relu'),
    layers.Flatten(),
    layers.Dense(64, activation='relu'),
    layers.Dense(10)  # 10 classes for CIFAR-10
])
```

6. **Compile the Model**
  - Specify the optimizer, loss function, and metrics to evaluate during training.
```python
model.compile(optimizer='adam',
              loss=tf.keras.losses.SparseCategoricalCrossentropy(from_logits=True),
              metrics=['accuracy'])
```

7. **Train the Model**
  - Train the model on the training data and validate it on the test data.
```python
history = model.fit(train_images, train_labels, epochs=10,
                    validation_data=(test_images, test_labels))

```
8. **Evaluate the Model**
  - Assess the model's performance on the test set.
```python
test_loss, test_acc = model.evaluate(test_images, test_labels, verbose=2)
```

print(f'\nTest accuracy: {test_acc}')
```

9. **Plot Training and Validation Accuracy**
- Visualize the training and validation accuracy over epochs to check for overfitting.
```python
plt.plot(history.history['accuracy'], label='accuracy')
plt.plot(history.history['val_accuracy'], label = 'val_accuracy')
plt.xlabel('Epoch')
plt.ylabel('Accuracy')
plt.ylim([0, 1])
plt.legend(loc='lower right')
plt.show()
```

10. **Make Predictions**
- You can use the trained model to make predictions on new images.
```python
predictions = model.predict(test_images)
```

11. **Save the Model**
- Save the trained model for future use.
```python
model.save('cifar10_model.h5')
```

This code outlines the entire process from data loading to model training and evaluation. If you execute this script in a Python environment, it will train a basic CNN on the CIFAR-10 dataset and evaluate its performance. You can further tweak the model architecture, training epochs, or learning rate to improve performance.

PROGRAM:

```
import tensorflow as tf
from tensorflow.keras import datasets, layers, models
import matplotlib.pyplot as plt

(train_images, train_labels), (test_images, test_labels) = datasets.cifar10.load_data()
train_images, test_images = train_images / 255.0, test_images / 255.0

class_names = ['airplane', 'automobile', 'bird', 'cat', 'deer', 'dog', 'frog', 'horse', 'ship', 'truck']

plt.figure(figsize=(10,10))
for i in range(25):
    plt.subplot(5,5,i+1)
    plt.xticks([])
    plt.yticks([])
```
```

```
    plt.grid(False)
    plt.imshow(train_images[i], cmap=plt.cm.binary)
    plt.xlabel(class_names[train_labels[i][0]])
plt.show()

model = models.Sequential([
    layers.Conv2D(32, (3, 3), activation='relu', input_shape=(32, 32, 3)),
    layers.MaxPooling2D((2, 2)),
    layers.Conv2D(64, (3, 3), activation='relu'),
    layers.MaxPooling2D((2, 2)),
    layers.Conv2D(64, (3, 3), activation='relu'),
    layers.Flatten(),
    layers.Dense(64, activation='relu'),
    layers.Dense(10)
])

model.compile(optimizer='adam',
            loss=tf.keras.losses.SparseCategoricalCrossentropy(from_logits=True),
            metrics=['accuracy'])

history = model.fit(train_images, train_labels, epochs=10,
                validation_data=(test_images, test_labels))

test_loss, test_acc = model.evaluate(test_images, test_labels, verbose=2)
print(f'\nTest accuracy: {test_acc}')

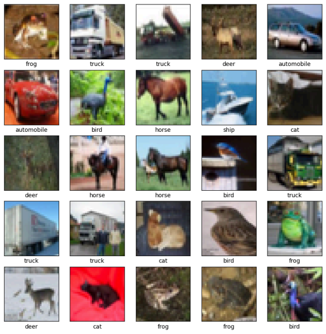
plt.plot(history.history['accuracy'], label='accuracy')
plt.plot(history.history['val_accuracy'], label = 'val_accuracy')
plt.xlabel('Epoch')
plt.ylabel('Accuracy')
plt.ylim([0, 1])
plt.legend(loc='lower right')
plt.show()

predictions = model.predict(test_images)

model.save('cifar10_model.h5')
```

OUTPUT:

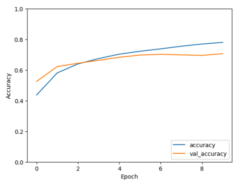
```
Downloading data from https://www.cs.toronto.edu/~kriz/cifar-10-python.tar.gz
170498071/170498071 ━━━━━━━━━━━━━━━━━━━━ 4s 0us/step
```




```
/usr/local/lib/python3.10/dist-packages/keras/src/layers/convolutional/base_conv.py:107:
UserWarning: Do not pass an `input_shape`/`input_dim` argument to a layer. When using
Sequential models, prefer using an `Input(shape)` object as the first layer in the model instead.
  super().__init__(activity_regularizer=activity_regularizer, **kwargs)
Epoch 1/10
1563/1563 ━━━━━━━━━━━━━━━━━━━━ 88s 55ms/step - accuracy: 0.3458 -
loss: 1.7704 - val_accuracy: 0.5256 - val_loss: 1.2979
Epoch 2/10
1563/1563 ━━━━━━━━━━━━━━━━━━━━ 129s 47ms/step - accuracy: 0.5629 -
loss: 1.2171 - val_accuracy: 0.6227 - val_loss: 1.0586
Epoch 3/10
1563/1563 ━━━━━━━━━━━━━━━━━━━━ 73s 46ms/step - accuracy: 0.6341 -
loss: 1.0425 - val_accuracy: 0.6442 - val_loss: 1.0285
Epoch 4/10
1563/1563 ━━━━━━━━━━━━━━━━━━━━ 72s 46ms/step - accuracy: 0.6694 -
loss: 0.9370 - val_accuracy: 0.6635 - val_loss: 0.9688
Epoch 5/10
1563/1563 ━━━━━━━━━━━━━━━━━━━━ 82s 47ms/step - accuracy: 0.7029 -
loss: 0.8497 - val_accuracy: 0.6832 - val_loss: 0.9105
Epoch 6/10
1563/1563 ━━━━━━━━━━━━━━━━━━━━ 76s 49ms/step - accuracy: 0.7258 -
loss: 0.7900 - val_accuracy: 0.6979 - val_loss: 0.8773
Epoch 7/10
1563/1563 ━━━━━━━━━━━━━━━━━━━━ 80s 48ms/step - accuracy: 0.7408 -
loss: 0.7369 - val_accuracy: 0.7026 - val_loss: 0.8531
Epoch 8/10
1563/1563 ━━━━━━━━━━━━━━━━━━━━ 74s 47ms/step - accuracy: 0.7611 -
loss: 0.6864 - val_accuracy: 0.6991 - val_loss: 0.8749
Epoch 9/10
1563/1563 ━━━━━━━━━━━━━━━━━━━━ 82s 47ms/step - accuracy: 0.7761 -
loss: 0.6423 - val_accuracy: 0.6956 - val_loss: 0.9025
Epoch 10/10
1563/1563 ━━━━━━━━━━━━━━━━━━━━ 83s 48ms/step - accuracy: 0.7847 -
loss: 0.6145 - val_accuracy: 0.7071 - val_loss: 0.8997
313/313 - 4s - 11ms/step - accuracy: 0.7071 - loss: 0.8997
```

Test accuracy: 0.707099974155426

313/313 ━━━━━━━━━━━━━━━━━━━━ **5s** 16ms/step



WARNING:absl:You are saving your model as an HDF5 file via `model.save()` or `keras.saving.save_model(model)`. This file format is considered legacy. We recommend using instead the native Keras format, e.g. `model.save('my_model.keras')` or `keras.saving.save_model(model, 'my_model.keras')`