Ex: 5

TRANSFER LEARNING WITH CNN AND VISUALIZATION

Aim:

To build a convolutional neural network with transfer learning and perform visualization.

Algorithm:

1. Import necessary libraries including TensorFlow, Keras modules, and CIFAR-10 dataset.
2. Load and preprocess the CIFAR-10 dataset by normalizing image pixel values and converting labels into one-hot encoding.
3. Create data generators using ImageDataGenerator for image augmentation and preprocessing.
4. Load the pre-trained VGG16 model without the top layer, specifying input size and freezing its layers to retain pre-trained weights.
5. Build a new model by stacking the VGG16 base with custom fully connected layers for classification.
6. Add a Flatten layer to transform the output of the VGG16 model into a 1D array.
7. Add a Dense layer with 128 neurons and ReLU activation, followed by a Dropout layer for regularization.
8. Add the final output Dense layer with softmax activation for class probabilities.
9. Compile the model with the Adam optimizer, categorical cross-entropy loss, and accuracy metric.
10. Train the model using the training data generator and validate it using the validation data generator.

Program:

```
import tensorflow as tf

from tensorflow.keras import layers, models

from tensorflow.keras.applications import VGG16

from tensorflow.keras.datasets import cifar10

from tensorflow.keras.utils import to_categorical

from tensorflow.keras.preprocessing.image import ImageDataGenerator
```

```
# Step 1: Load and preprocess CIFAR data
(x_train, y_train), (x_test, y_test) = cifar10.load_data()
x_train = x_train / 255.0
x_test = x_test / 255.0
# Convert labels to one-hot encoding
y_train = to_categorical(y_train, 10)
y_test = to_categorical(y_test, 10)
# Step 2: Create data generators for training and testing
datagen = ImageDataGenerator(rescale=1.0/255.0)
# Create a generator for training data
train_gen = datagen.flow(x_train, y_train, batch_size=32)
# Create a generator for validation/test data
val_gen = datagen.flow(x_test, y_test, batch_size=32)
# Step 3: Load VGG16 model without the top layer (for transfer learning)
base_model = VGG16(weights='imagenet', include_top=False, input_shape=(32, 32, 3))
# Freeze the layers of VGG16 to avoid training them
base_model.trainable = False
# Step 4: Build the final model
model = models.Sequential()
# Add the base VGG16 model
model.add(base_model)
# Add custom layers on top of the VGG16 base model
model.add(layers.Flatten())
model.add(layers.Dense(128, activation='relu'))
model.add(layers.Dropout(0.5))  # Dropout for regularization
model.add(layers.Dense(10, activation='softmax'))  # 10 output classes for CIFAR-10
# Step 5: Compile the model
model.compile(optimizer='adam',
              loss='categorical_crossentropy',
              metrics=['accuracy'])
```

Step 6: Train the model using the generators

history = model.fit(train_gen, epochs=5, validation_data=val_gen)

test_loss, test_acc = model.evaluate(x_test, y_test)

print(f"Test Accuracy: {test_acc * 1000:.2f}%")

Output:

```
[8]: # Step 6: Train the model using the generators
     history = model.fit(train_gen, epochs=5, validation_data=val_gen)

Epoch 1/5
C:\Users\mannu\anaconda3\envs\dlc\Lib\site-packages\keras\src\trainers\data_adapters\py_dataset_adapter.py:121: UserWarning: Your `PyDataset` class
d call `super().__init__(**kwargs)` in its constructor. `**kwargs` can include `workers`, `use_multiprocessing`, `max_queue_size`. Do not pass these
ments to `fit()`, as they will be ignored.
  self._warn_if_super_not_called()
1563/1563 ━━━━━━━━━━━━━━━━━━━━ 437s 277ms/step - accuracy: 0.0961 - loss: 2.3180 - val_accuracy: 0.1000 - val_loss: 2.3026
Epoch 2/5
1563/1563 ━━━━━━━━━━━━━━━━━━━━ 431s 276ms/step - accuracy: 0.0989 - loss: 2.3028 - val_accuracy: 0.1000 - val_loss: 2.3026
Epoch 3/5
1563/1563 ━━━━━━━━━━━━━━━━━━━━ 340s 218ms/step - accuracy: 0.0981 - loss: 2.3028 - val_accuracy: 0.1000 - val_loss: 2.3027
Epoch 4/5
1563/1563 ━━━━━━━━━━━━━━━━━━━━ 341s 218ms/step - accuracy: 0.0974 - loss: 2.3028 - val_accuracy: 0.1000 - val_loss: 2.3026
Epoch 5/5
1563/1563 ━━━━━━━━━━━━━━━━━━━━ 387s 247ms/step - accuracy: 0.0984 - loss: 2.3028 - val_accuracy: 0.1000 - val_loss: 2.3026

[9]: test_loss, test_acc = model.evaluate(x_test, y_test)
     print(f"Test Accuracy: {test_acc * 1000:.2f}%")

313/313 ━━━━━━━━━━━━━━━━━━━━ 50s 159ms/step - accuracy: 0.0849 - loss: 2.3336
Test Accuracy: 87.50%
```

Result:

Thus the program to build a convolutional neural network with transfer learning and perform visualization has been executed successfully.