Exp No: 5

TRANSFER LEARNING WITH CNN AND VISUALIZATION

Aim:

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

Procedure:

1. Download and load the dataset.
2. Perform analysis and preprocessing of the dataset.
3. Build a simple neural network model using Keras/TensorFlow.
4. Compile and fit the model.
5. Perform prediction with the test dataset.
6. Calculate performance metrics.

Program:

```
from keras.preprocessing.image import ImageDataGenerator
TEST_SPLIT = 0.2
VALIDATION_SPLIT = 0.2
import os
import math

os.mkdir("caltech_test") # stores test data

for cat in os.listdir("101_ObjectCategories/"):
  # moves x portion of images per category into test images
  os.mkdir("caltech_test/"+cat) # new category folder
  imgs = os.listdir("101_ObjectCategories/"+cat) # all image filenames
  split = math.floor(len(imgs)*TEST_SPLIT)
  test_imgs = imgs[:split]
```

```
  for t_img in test_imgs: # move test portion
    os.rename("101_ObjectCategories/"+cat+"/"+t_img, "caltech_test/"+cat+"/"+t_img)
from keras.applications.resnet50 import preprocess_input
train_gen = ImageDataGenerator(validation_split=0.2,
preprocessing_function=preprocess_input)
train_flow = train_gen.flow_from_directory("101_ObjectCategories/", target_size=(256, 256),
batch_size=32, subset="training")
valid_flow = train_gen.flow_from_directory("101_ObjectCategories/", target_size=(256, 256),
batch_size=32, subset="validation")
test_gen = ImageDataGenerator(preprocessing_function=preprocess_input)
test_flow = test_gen.flow_from_directory("caltech_test", target_size=(256, 256), batch_size=32)
from keras.applications.resnet50 import ResNet50
from keras.layers import GlobalAveragePooling2D, BatchNormalization, Dropout, Dense
from keras.models import Model
res = ResNet50(weights='imagenet', include_top=False, input_shape=(256, 256, 3))
for layer in res.layers:
  layer.trainable = False
x = res.output
x = GlobalAveragePooling2D()(x)
x = BatchNormalization()(x)
x = Dropout(0.5)(x)
x = Dense(512, activation='relu')(x)
x = BatchNormalization()(x)
x = Dropout(0.5)(x)
x = Dense(101, activation='softmax')(x)
model = Model(res.input, x)
model.compile(optimizer='Adam', loss='categorical_crossentropy', metrics=['accuracy'])
model.summary()
model.fit_generator(train_flow, epochs=5, validation_data=valid_flow)
```

```
result = model.evaluate(test_flow)
print('The model achieved a loss of %.2f and accuracy of %.2f%%.' % (result[0], result[1]*100))
```

Output:

```
WARNING:tensorflow:From /usr/local/lib/python3.6/dist-packages/tensorflow_core/python/ops/math_grad.py:1424: where (from tensorflo
Instructions for updating:
Use tf.where in 2.0, which has the same broadcast rule as np.where
WARNING:tensorflow:From /usr/local/lib/python3.6/dist-packages/keras/backend/tensorflow_backend.py:1033: The name tf.assign_add is

WARNING:tensorflow:From /usr/local/lib/python3.6/dist-packages/keras/backend/tensorflow_backend.py:1020: The name tf.assign is dep

Epoch 1/5
176/176 [==============================] - 27s 156ms/step - loss: 1.6601 - acc: 0.6338 - val_loss: 0.3799 - val_acc: 0.8922
Epoch 2/5
176/176 [==============================] - 19s 107ms/step - loss: 0.4637 - acc: 0.8696 - val_loss: 0.2841 - val_acc: 0.9225
Epoch 3/5
176/176 [==============================] - 19s 107ms/step - loss: 0.2777 - acc: 0.9211 - val_loss: 0.2714 - val_acc: 0.9225
Epoch 4/5
176/176 [==============================] - 19s 107ms/step - loss: 0.2223 - acc: 0.9327 - val_loss: 0.2419 - val_acc: 0.9284
Epoch 5/5
176/176 [==============================] - 19s 106ms/step - loss: 0.1784 - acc: 0.9461 - val_loss: 0.2499 - val_acc: 0.9239
<keras.callbacks.History at 0x7f72d1fa2470>
```

```
53/53 [==============================] - 5s 95ms/step
The model achieved a loss of 0.23 and accuracy of 92.80%.
```

Result:

Transfer learning with CNN and visualization successfully completed.