|  |  |
| --- | --- |
| Ex No: 5  Date: 04/09/2024 | Transfer Learning for Fish and Flower Classification using Convolutional Neural Networks. |

**Objective:**

This project showcases the application of transfer learning to develop and fine-tune a Convolutional Neural Network (CNN) for flower image classification. The main objective is to utilize a pre-trained model (Inception V3) and modify it to identify and classify different flower species from a new dataset. The process involves preprocessing the dataset, scaling the images, fine-tuning the model, and assessing its performance on unseen test data. The goal is to attain high accuracy in flower classification by leveraging the robust feature extraction capabilities of a pre-trained deep learning model.

**Variation Used:** Inception V3

**Code Explanation:**

**Fish Classification**

* **TensorFlow Hub Installation**: The initial code cell installs the `tensorflow\_hub` package, which is necessary for loading pre-trained models from TensorFlow Hub. The output confirms the successful installation of both `tensorflow\_hub` and `tf-keras`.
* **Ensuring Required Packages**: This step installs the `tf\_keras` package to ensure that the required TensorFlow and Keras packages are available.
* **Model Selection**: The code comments out MobileNetV2 and selects the InceptionV3 model as the classifier by setting the `classifier\_model` variable to the InceptionV3 URL from TensorFlow Hub.
* **Loading the Model**: The code sets the image shape to `(224, 224)` and loads the pre-trained InceptionV3 model from TensorFlow Hub using `hub.KerasLayer`. The model's weights are kept frozen (`trainable=False`) to prevent them from being updated during training.
* **Model Wrapping**: The pre-trained model is encapsulated within a `Sequential` model, with a `Lambda` layer as the first layer to invoke the pre-trained model.
* **Image Processing**: The code loads a goldfish image using `tf.image.decode\_jpeg`, resizes it to the specified `IMAGE\_SHAPE`, and adjusts its dimensions to match the model's expected input shape (including batch size).
* **Making Predictions**: The model predicts the class of the goldfish image using the pre-trained InceptionV3 model (`classifier.predict`). The output is a vector with 1,001 elements, each representing the logit (confidence) for a class in the ImageNet dataset.
* **Determining the Predicted Class:** The index of the predicted class (the class with the highest logit value) is identified using `np.argmax`, and the `predicted\_label\_index` stores this index.
* **Loading Class Labels**: The code snippet downloads the `ImageNetLabels.txt` file, which contains the class labels for the ImageNet dataset. It reads the file and stores the labels in a list (`image\_labels`). The code then prints the first five labels to verify correct loading.
* **Matching Class Label**: The label corresponding to the predicted class index (`predicted\_label\_index`) is retrieved from the `image\_labels` list, and in this case, it returns "goldfish," indicating the model has accurately classified the image.

**Flower Classification**

* **Downloading the Dataset**: The code downloads a compressed dataset of flower images from the specified URL (`dataset\_url`) using TensorFlow’s `tf.keras.utils.get\_file` utility. The dataset is downloaded to the current directory (`cache\_dir='.'`) and is automatically extracted (`untar=True`).
* **Handling Dataset Paths**: The downloaded dataset directory path (`data\_dir`) is converted into a `pathlib.Path` object for easier file handling. The code then lists the first five `.jpg` images within the dataset by using `glob` to search through the directory and its subdirectories.
* **Counting Images**: The code counts the total number of `.jpg` images in the dataset (`image\_count`) and prints the result. It also lists the image files in the "roses" and "tulips" subdirectories, displaying the first few images from each category using the `PIL.Image.open` function to open and view them.
* **Creating Image and Label Dictionaries**: The code creates a dictionary (`flowers\_images\_dict`) where each key is a flower category (e.g., "roses," "daisy") and the corresponding value is a list of image file paths for that category. Additionally, it creates a dictionary (`flowers\_labels\_dict`) mapping each flower category to a unique numeric label. The code then retrieves and displays the first five image paths from the "roses" category.
* **Processing Images for ML**: The code processes flower images for machine learning tasks. It reads images from `flowers\_images\_dict`, resizes each image to 224x224 pixels, and appends the resized images to the list `X`. The corresponding labels, retrieved from `flowers\_labels\_dict`, are stored in the list `y`. If an image cannot be read, an error message is printed. Finally, the lists `X` and `y` are converted into NumPy arrays for further use.
* **Preparing and Scaling Images**: The code prepares images for prediction by splitting the dataset into training and testing sets using `train\_test\_split`. The images in both sets are scaled by dividing pixel values by 255 to normalize them. The code resizes the first three images in `X` to match the `IMAGE\_SHAPE` expected by the model. Finally, it displays the first image in the dataset using Matplotlib’s `imshow`, without axis labels.
* **Model Prediction and Fine-Tuning**: The code uses a pre-trained model for predictions and fine-tunes it for flower classification. It first predicts the labels of three resized flower images using a classifier and extracts the most likely class using `np.argmax`. Then, it loads a pre-trained InceptionV3 model from TensorFlow Hub, excluding the top layer, so it can be fine-tuned on a new dataset. The pre-trained model is integrated as a feature extractor in a new `Sequential` model, followed by a dense layer for predicting five flower classes. The parameters of the pre-trained model are checked to ensure they aren't trainable. The model summary is then printed to provide an overview of the layers and parameters.
* **Fine-Tuning and Evaluating the Model**: The code fine-tunes the pre-trained model on the flower image dataset and makes predictions on new images. The model is compiled using the Adam optimizer, sparse categorical cross-entropy loss, and accuracy as a metric. It is trained for five epochs on the scaled training data, and its performance is evaluated on the test data. A helper function, `preprocess\_image`, is defined to load and preprocess new images by resizing them to 224x224 pixels, normalizing the pixel values, and expanding the dimensions to fit the model input. The model predicts the class of a test image (`sun.jpg`), and the predicted class index is identified using `np.argmax`. Finally, the predicted class is matched to the corresponding flower name from a predefined list and printed.

**GitHub Link:**

https://github.com/Mithungowda6666/Deeplearning/tree/main/lab5