| **Ex No: 5**  **Date: 28-08-24** | **Transfer Learning for Fish and Flower Classification using Convolutional Neural Networks.** |
| --- | --- |

**Objective:**

This project demonstrates the use of transfer learning to build and fine-tune a Convolutional Neural Network (CNN) for classifying images of flowers. The primary goal is to leverage a pre-trained model (Inception V3) and adapt it to recognize and classify various types of flowers from a new dataset. The process includes preprocessing the dataset, scaling the images, fine-tuning the model, and evaluating its performance on unseen test data. The objective is to achieve high accuracy in flower classification by utilizing the powerful feature extraction capabilities of a pre-trained deep learning model.

**Variation Used:** mobilenet

**Variation Used:** Inception V3

**Code Explanation:**

**• Fish Classification**

The first code cell installs the tensorflow\_hub package, which is necessary to load pre-trained models from TensorFlow Hub. The output confirms the successful installation of tensorflow\_hub and tf-keras.

The code here shows the installation of the tf\_keras package, ensuring that the required TensorFlow and Keras packages are available.

The code block comments out MobileNetV2 and selects the InceptionV3 model as the classifier model by setting the classifier\_model variable to the InceptionV3 URL on TensorFlow Hub.

The code defines the image shape as (224, 224) and loads the pre-trained InceptionV3 model from TensorFlow Hub using hub.KerasLayer. The model's weights are frozen (trainable=False) to prevent them from being updated during training.

The pre-trained model is wrapped into a Sequential model, with the first layer being a Lambda layer that calls the pre-trained model.

The code loads an image of a goldfish using the tf.image.decode\_jpeg method, resizes it to the specified IMAGE\_SHAPE, and expands its dimensions to match the expected input shape of the model (batch size included).

The code predicts the class of the goldfish image using the pre-trained InceptionV3 model (classifier.predict). The result is a vector with 1,001 elements, where each element corresponds to the logit (confidence) for one of the classes in the ImageNet dataset.

Finds the index of the predicted class (the class with the highest logit value) using np.argmax. The predicted\_label\_index gives the index of the class with the highest predicted probability.

The code snippet is downloading a file called `ImageNetLabels.txt` from a given URL, which contains the class labels for the ImageNet dataset, with one label per line. The `with open` statement opens the file, reads all the lines, and stores them as a list in the `image\_labels` variable. Finally, the code prints the first five labels from this list to verify that the labels have been loaded correctly.

The code retrieves the label corresponding to the predicted class index (`predicted\_label\_index`) from the `image\_labels` list. In this case, it returns the label "goldfish," which indicates that the model has classified the image as a goldfish.

**• Flower Classification**

The code downloads a compressed dataset of flower photos from the specified URL (`dataset\_url`) using TensorFlow's utility function `tf.keras.utils.get\_file`. The dataset is downloaded to the current directory (`cache\_dir='.'`), and the `untar=True` option automatically extracts the contents of the compressed file.

The code converts the downloaded dataset directory path (`data\_dir`) into a `pathlib.Path` object for easier file handling. It then lists the first five `.jpg` image files within the dataset by using `glob` to search through the directory and subdirectories.

The code counts the total number of `.jpg` images in the dataset (`image\_count`) and prints the result. It then lists the image files specifically 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.

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.

The code also creates a second dictionary (`flowers\_labels\_dict`) that maps each flower category to a unique numeric label. The last line retrieves and displays the first five image paths from the "roses" category.

The code processes images of flowers and prepares them for machine learning tasks. It reads images from a dictionary `flowers\_images\_dict`, resizes each image to 224x224 pixels, and appends the resized images to the list `X`.

The corresponding labels for each flower, retrieved from `flowers\_labels\_dict`, are stored in the list `y`. The code handles errors by printing a message if an image cannot be read. Finally, it converts the lists `X` and `y` into NumPy arrays for further use.

The code prepares and scales images for prediction using a pre-trained model. It first splits the dataset into training and testing sets using `train\_test\_split`. The images in both sets are then scaled by dividing pixel values by 255 to normalize them.

The code resizes the first three images in `X` to match the `IMAGE\_SHAPE`, which is expected by the model. Finally, it displays the first image in the dataset without axis labels using Matplotlib's `imshow`.

The code is using a pre-trained model to make predictions and then fine-tune it for a new flower image classification task. First, it 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 Inception V3 model from TensorFlow Hub, excluding the top layer, so the model can be fine-tuned on a new dataset. The pre-trained model is added as a feature extractor in a new `Sequential` model, followed by a dense layer that will output predictions for five flower classes. The code checks the parameters of the pre-trained model to ensure they aren't trainable.

Lastly, it prints the model summary to provide an overview of the layers and parameters.

The code fine-tunes a pre-trained model on a flower image dataset and makes predictions on new images. First, it compiles the model using the Adam optimizer, a sparse categorical cross-entropy loss, and accuracy as a metric.

The model is then trained for five epochs on the scaled training data. After training, the model's performance is evaluated on the test data. A helper function `preprocess\_image` is defined to load and preprocess a new image, resizing it to 224x224 pixels, normalizing the pixel values, and expaObjective:

The objective of this lab experiment is to demonstrate the use of transfer learning in image classification by utilizing pre-trained convolutional neural network (CNN) models—specifically MobileNet and Inception V3—available on TensorFlow Hub.

Descriptions:

The experiment aims to classify images of fish and flowers by fine-tuning these pre-trained models, thereby reducing the computational cost and time required for training.

Transfer learning leverages the knowledge gained from training a model on a large dataset and applies it to a different, but related, problem. In this case, the pre-trained models have been trained on a large image dataset (such as ImageNet) and will be fine-tuned for the specific task of classifying fish and flower images.

Sample from flower dataset sample image of fish

Model:

1. MobileNet

2. Inception V3

Building the parts of algorithm

1. Installing Necessary Libraries

python

pip install tensorflow\_hub

pip install tf\_keras

pip install opencv-python

These commands install the necessary libraries:

• tensorflow\_hub: Contains pre-trained models.

• tf\_keras: Keras is a high-level API for building neural networks.

• opencv-python: Provides tools for image processing.

2. Importing Libraries

import numpy as np

import cv2

import PIL as PIL

import PIL.Image as Image

import os

import matplotlib.pylab as plt

import tensorflow as tf

import tensorflow\_hub as hub

from tensorflow import keras

from tensorflow.keras import layers

from tensorflow.keras.models import Sequential

from tensorflow.keras.layers import Dense

import tf\_keras

These libraries are imported to handle various tasks:

• numpy and cv2: For numerical operations and image processing.

• PIL: For handling image file formats.

• matplotlib: For plotting images and results.

• tensorflow and tensorflow\_hub: For building and handling neural networks and transfer learning.

• keras: High-level neural networks API, now integrated into TensorFlow.

3. Loading Pre-Trained Models

mobilenet\_v2 = "https://tfhub.dev/google/tf2-preview/mobilenet\_v2/classification/4"

inception\_v3 = "https://tfhub.dev/google/imagenet/inception\_v3/classification/5"

classifier\_model = mobilenet\_v2

The code defines the URLs for MobileNet V2 and Inception V3 models hosted on TensorFlow Hub. The mobilenet\_v2 model is chosen as the classifier\_model.

3. Building the Model Architecture

Variation 1 : Without lamba layers

import tensorflow as tf

import tensorflow\_hub as hub

import tf\_keras

feature\_extractor\_model = "https://tfhub.dev/google/tf2-preview/mobilenet\_v2/feature\_vector/4"

pretrained\_model\_without\_top\_layer = hub.KerasLayer(

feature\_extractor\_model, input\_shape=(224, 224, 3), trainable=False)

num\_of\_flowers = 5

# Wrap the hub.KerasLayer in a tf.keras.layers.Lambda layer

model = tf.keras.Sequential([

tf.keras.layers.Lambda(lambda x: pretrained\_model\_without\_top\_layer(x)), # Use the pretrained model as a callable within Lambda

tf.keras.layers.Dense(num\_of\_flowers)

])

# Build the model by passing some dummy input

# This will infer the shapes of the layers and allow the summary to be generated

model.build(input\_shape=(None, 224, 224, 3)) # Replace (None, 224, 224, 3) with the actual shape of your input data

# Get the pre-trained model's parameters

pretrained\_model\_params = pretrained\_model\_without\_top\_layer.trainable\_weights

print(pretrained\_model\_params)

# Print the parameters

for param in pretrained\_model\_params:

print(param.shape)

model.summary() # Now you can call summary after the model has been built

Here, the MobileNet V2 model is loaded using hub.KerasLayer, with the input image shape set to 224x224 with 3 color channels (RGB). The model's weights are frozen (trainable=False), meaning they won’t be updated during training. A Sequential model is then created, which includes a Lambda layer to invoke the pre-trained model.

5. Preparing and Processing an Image

gold\_fish = Image.open("/content/goldfish.jpg").resize(IMAGE\_SHAPE)

gold\_fish = np.array(gold\_fish)/255.0

gold\_fish = gold\_fish[np.newaxis, ...]

The image of a goldfish is loaded, resized, and normalized. The dimensions are expanded to match the model's input requirements.

6. Making Predictions

result = classifier.predict(gold\_fish[np.newaxis, ...])

predicted\_label\_index = np.argmax(result)

The classifier model predicts the class of the input image, and the index of the highest probability is extracted.

7. Loading ImageNet Labels

tf.keras.utils.get\_file('ImageNetLabels.txt','https://storage.googleapis.com/download.tensorflow.org/data/ImageNetLabels.txt')

# Further code is truncated but it involves loading the ImageNet labels to map predicted indices to actual class names.

This part loads the ImageNet labels, which help in translating the predicted label indices into human-readable class names.

8. Training on Custom Dataset

# Further code deals with modifying the model for the flower classification task by adding a Dense layer and training it on a custom flower dataset.

num\_of\_flowers = 5 # Number of output classes

model = tf.keras.Sequential([

tf.keras.layers.Lambda(lambda x: pretrained\_model\_without\_top\_layer(x)),

tf.keras.layers.Dense(num\_of\_flowers)

])

model.compile(

optimizer="adam",

loss=tf.keras.losses.SparseCategoricalCrossentropy(from\_logits=True),

metrics=['acc']

)

model.fit(X\_train\_scaled, y\_train, epochs=5)

model.evaluate(X\_test\_scaled, y\_test)

Here, a new dense layer is added to classify the flower images. The model is compiled with the Adam optimizer and categorical cross-entropy loss, and then trained on the dataset.

9. Testing the Model

test\_image\_path = 'sun.jpg'

test\_image = preprocess\_image(test\_image\_path)

predictions = model.predict(test\_image)

predicted\_class = np.argmax(predictions, axis=1)

class\_names = ['daisy', 'dandelion', 'roses', 'sunflowers', 'tulips']

print(f"Predicted flower: {class\_names[predicted\_class[0]]}")

This section preprocesses a test image, makes predictions, and then maps the predicted class index to the actual flower name.

Predictions:

MobileNet without lambda layer : [0.33551695942878723, 0.8736208920478821]

MobileNet with lambda layer : [0.33551695942878723, 0.8856208920478821]

Inception V3 with lambda layer : [0.4150967001914978, 0.85838782787323]

Inception V3 without lambda layer : [0.3809000551700592, 0.8616557717323303]

nding 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.

**GitHubLink:** [**https://github.com/chandanab1/Deep\_Learning**](https://github.com/chandanab1/Deep_Learning)