|  |  |
| --- | --- |
| **Ex No: 5**  **Date: 4-9-2024** | **Transfer Learning** |

**Variations made: Change of model used in transfer learning – NasNet model is used with modified architecture .**

**Objective:**

The objective to develop an image classification model using the NASNet feature extractor, integrating it with custom dense layers for fine-tuning. The goal is to classify images into predefined categories by using transfer learning.

**Descriptions:**

1) Model Development:

* Feature Extraction: Use the pre-trained NASNet feature extractor model from TensorFlow Hub. This model provides high-level feature representations of images, trained on a large dataset.
* Model Customization: Customize the NASNet model by adding your own dense layers. The final dense layer will have a softmax activation function to output probabilities for each class.

2) Data Preparation:

* Dataset: You will work with a dataset of images categorized into predefined classes, such as different types of flowers.
* Preprocessing: Implement preprocessing steps to resize images to 224x224 pixels, normalize pixel values, and prepare them for input into the model.

3) Training and Evaluation:

* Training: Compile the model with the Adam optimizer and SparseCategoricalCrossentropy loss function. Train the model on the dataset for a specified number of epochs.
* Evaluation: Assess the model’s performance on a separate test set to determine accuracy and loss.

4) Inference:

* Prediction: Implement a function to preprocess a new image, use the trained model to predict its class, and interpret the prediction by mapping it to human-readable class names.

**Code explanation:**

**Libraries used:**

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

* import numpy as np: Imports NumPy for numerical operations and array handling.
* import cv2: Imports OpenCV for image and video processing.
* import PIL as PIL: Imports the Python Imaging Library (PIL), replaced by Pillow.
* import PIL.Image as Image: Imports the Image class from PIL/Pillow for image manipulation.
* import os: Imports the os module for file and directory operations.
* import matplotlib.pylab as plt: Imports Matplotlib's pyplot for creating plots and visualizations.
* import tensorflow as tf: Imports TensorFlow for building and training machine learning models.
* import tensorflow\_hub as hub: Imports TensorFlow Hub for using pre-trained models.
* from tensorflow import keras: Imports the keras submodule from TensorFlow for high-level model building.
* from tensorflow.keras import layers: Imports layers from TensorFlow Keras for neural network construction.
* from tensorflow.keras.models import Sequential: Imports the Sequential model class for linear stack of layers.
* from tensorflow.keras.layers import Dense: Imports the Dense layer class for fully connected layers.
* import tf\_keras: Imports tf\_keras, likely a custom or external module.

feature\_extractor\_model = hub.KerasLayer(

"https://tfhub.dev/google/imagenet/nasnet\_mobile/feature\_vector/4",

input\_shape=(224, 224, 3), # Specify input shape with color channels

trainable=False # Freeze the pre-trained weights

)

This line loads the pre-trained NASNet feature extractor model from TensorFlow Hub.

trainable=False: Indicates that the weights of this pre-trained model should not be updated during training**.**

Variation in Model code:

inputs = Input(shape=IMAGE\_SHAPE + (3,))

x = feature\_extractor\_model(inputs)

x = GlobalAveragePooling2D()(x)

x = Dense(1024, activation='relu')(x)

outputs = Dense(num\_of\_classes, activation='softmax')(x)

model = Model(inputs=inputs, outputs=outputs)

This code constructs a new model that uses the NASNet feature extractor as a base and adds custom layers on top.

* inputs = Input(shape=IMAGE\_SHAPE + (3,)):

Defines the input layer with the shape of the images (224x224 pixels with 3 color channels).

* x = feature\_extractor\_model(inputs):

Applies the NASNet feature extractor to the input images to obtain high-level features.

* x = GlobalAveragePooling2D()(x):

Reduces the spatial dimensions of the feature maps by averaging over the entire feature map, resulting in a vector of size equal to the number of channels.

* x = Dense(1024, activation='relu')(x):

Adds a fully connected dense layer with 1024 units and ReLU activation to learn complex representations.

* outputs = Dense(num\_of\_classes, activation='softmax')(x):

Adds a final dense layer with a softmax activation function to output probabilities for each class.

* model = Model(inputs=inputs, outputs=outputs):

Constructs the final Keras Model by specifying the input and output layers.

model.compile(

optimizer="adam",

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

metrics=['acc']

)

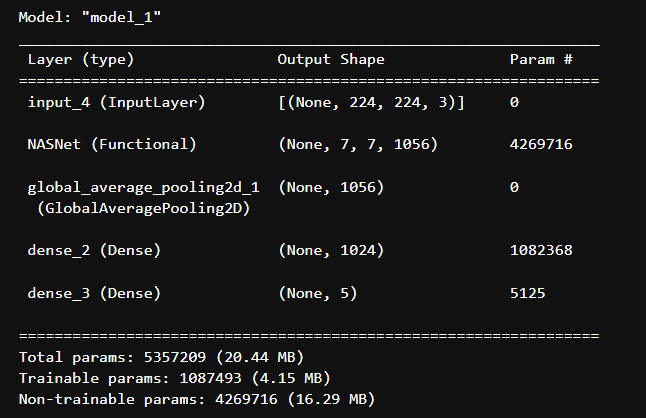
Configures the model for training by specifying the optimizer, loss function, and evaluation metrics.

* optimizer="adam": Uses the Adam optimizer, which adapts the learning rate during training.
* loss=tf.keras.losses.SparseCategoricalCrossentropy(from\_logits=True): Specifies the loss function to measure the error between the predicted probabilities and the true class labels.
* metrics=['acc']: Indicates that accuracy will be used to evaluate the model’s performance.

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

* X\_train\_scaled: The input images for training, scaled to the appropriate range (e.g., 0 to 1).
* y\_train: The true class labels for the training images.
* epochs=5: Number of epochs to train the model, where each epoch represents one pass through the entire training dataset.

**Model:**

****

The model starts with an input layer that accepts images of size 224x224 with 3 color channels. It uses the NASNet architecture as a feature extractor, which outputs a 7x7x1056 tensor. This is followed by a global average pooling layer that reduces the tensor to a 1056-dimensional vector. The next layer is a dense (fully connected) layer with 1024 units, and the final layer is another dense layer with 5 units, providing the model's output for classification into 5 classes.

**Building the parts of algorithm**

**1**) Define Model Structure:

* Use NASNet as a feature extractor with a pre-trained model from TensorFlow Hub.
* Add a Global Average Pooling layer, followed by Dense layers for intermediate processing and final classification.

2) Initialize Parameters:

* The NASNet weights are frozen, while the weights for the Dense layers are initialized.

3) Training Loop:

* Forward Propagation: Process images of flowers through the model to generate predictions.
* Loss Calculation: Measure how well the model's predictions match the true flower labels using a loss function.
* Backward Propagation: Compute the gradients of the loss with respect to the model’s parameters.
* Parameter Update: Use an optimizer to update the Dense layer weights based on the computed gradients.

4) Evaluate Model:

* Test the model’s performance on a separate test set of flower images to check accuracy and loss.

5) Predict:

* Use the trained model to classify new flower images and map the predictions to their respective class names.

**GitHub Link:**

**https://github.com/amruthaa-m/DL-Lab1/tree/main/Unit-1/Lab5**