|  |  |
| --- | --- |
| **Ex No: 5**  **Date: 04-09-2024** | **Transfer Learning** |

**Objective:**

The objective of this lab is to build a flower classification model using the MobileNet V2 pre-trained model from TensorFlow Hub. The model will classify different types of flowers (daisy, dandelion, roses, sunflowers, and tulips) using transfer learning. We will preprocess the dataset, train the model, and evaluate its performance.

**Descriptions:**

The code demonstrates how to use the MobileNet V2 model, pre-trained on ImageNet, for flower classification. The model is first loaded using TensorFlow Hub and frozen so that its weights are not updated during training. The dataset is prepared by downloading a set of flower images, resizing them to the required dimensions, and normalizing the pixel values. A new model is then created using the pre-trained model as a feature extractor, followed by a dense layer to classify the images into the five flower categories. The model is trained, evaluated, and then used to make predictions on unseen images.

**Steps to Build the Model:**

**Install Necessary Libraries**: Install TensorFlow, TensorFlow Hub, and OpenCV.

**Import Libraries**: Import all required libraries, such as TensorFlow, TensorFlow Hub, OpenCV, PIL, NumPy, etc.

**Define Image Shape**: Specify the shape of the images to be used.

**Load Pre-Trained Model**: Use a pre-trained MobileNet V2 model from TensorFlow Hub for feature extraction.

**Load and Preprocess Images**: Load images from the dataset, resize, normalize, and prepare them for input to the model.

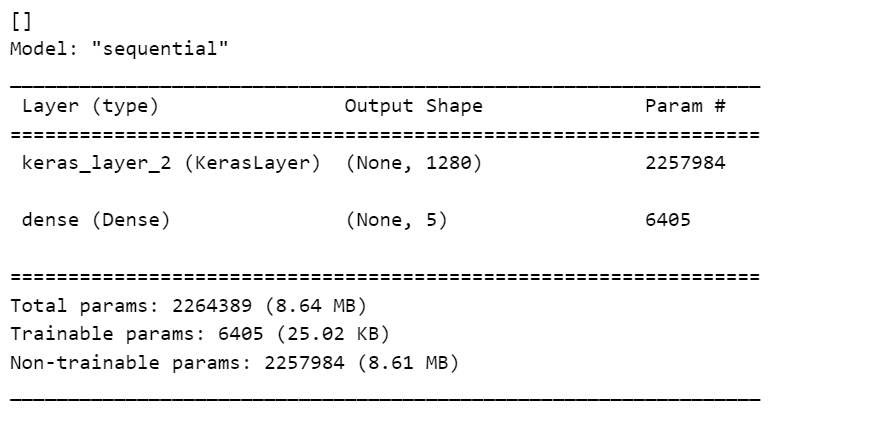
**Create a Model**: Define a Sequential model using the pre-trained MobileNet V2 as a feature extractor and add a dense output layer for classification.

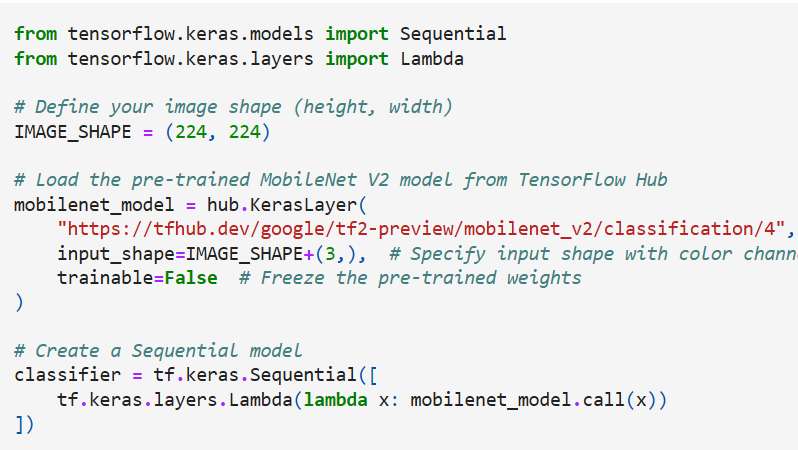
**Compile the Model**: Compile the model with an appropriate loss function and optimizer.

**Train the Model**: Fit the model using the training dataset.

**Evaluate the Model**: Evaluate the model's performance on the test dataset.

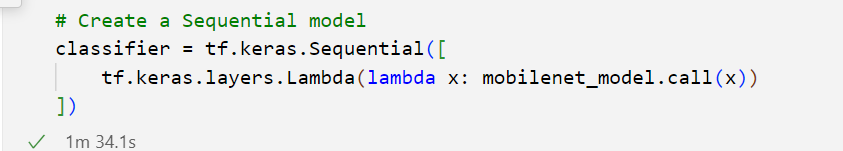
**Make Predictions**: Use the trained model to predict the class of new images.



****

Selecting a MobileNetV2 pre-trained model from TensorFlow Hub and wrapping it as a Keras layer

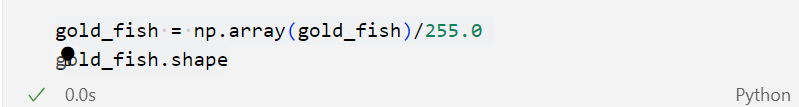
The input shape is defined, and the model is set to non-trainable to prevent updates to its weights during training.

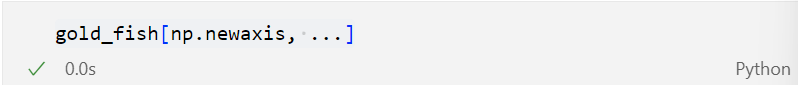


Creates a Sequential model using the pre-trained MobileNet model as a feature extractor.

Adds a Lambda layer to apply the mobilenet\_model function to the input data. The call method of mobilenet\_model is invoked to apply the MobileNet model to the input.







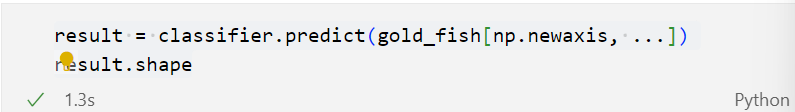


Image.open("goldfish.jpg"): Opens the image file "goldfish.jpg".

.resize(IMAGE\_SHAPE): Resizes the image to the specified dimensions (224x224 pixels).

np.array(gold\_fish) / 255.0: Converts the image to a NumPy array and normalizes pixel values to the range [0, 1].

gold\_fish[np.newaxis, ...]: Adds a new axis to the array to create a batch dimension (needed for model input).

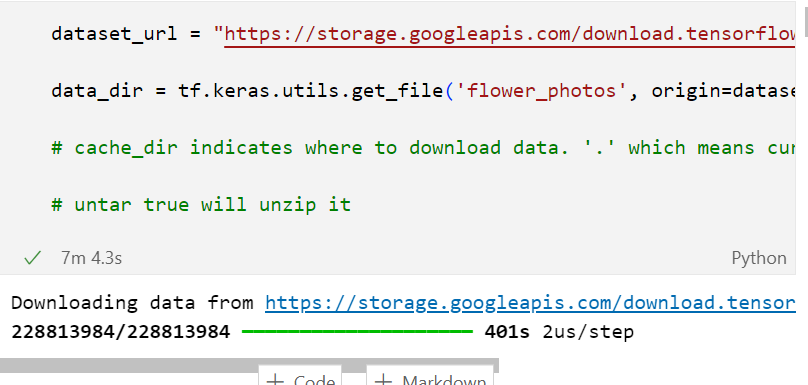
classifier.predict(...): Uses the model to make a prediction on the input image.

np.argmax(result): Finds the index of the highest value in the prediction result, indicating the predicted class.



with open(...) as f:: Opens the "ImageNetLabels.txt" file for reading and ensures it is properly closed afterward.

f.read().splitlines(): Reads all lines in the file and splits them into a list of labels.



dataset\_url: URL of the flower dataset.

tf.keras.utils.get\_file(...): Downloads the dataset from the URL and extracts it.

cache\_dir='.': Specifies the current directory for caching downloaded files.

untar=True: Extracts the downloaded tar file.

pathlib.Path(data\_dir): Converts the data directory path to a Path object for easier file handling.



Initializes empty lists X and y to store images and their labels.

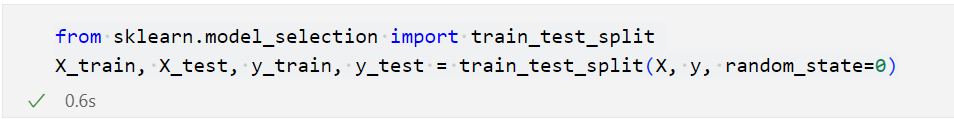
cv2.imread(str(image)): Reads each image using OpenCV.

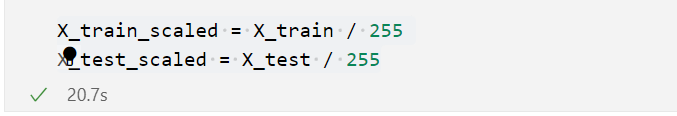
cv2.resize(img, (224, 224)): Resizes the image to 224x224 pixels.

X.append(...): Adds the resized image to the X list.

y.append(...): Adds the corresponding label to the y list.

np.array(X) and np.array(y): Converts the lists to NumPy arrays



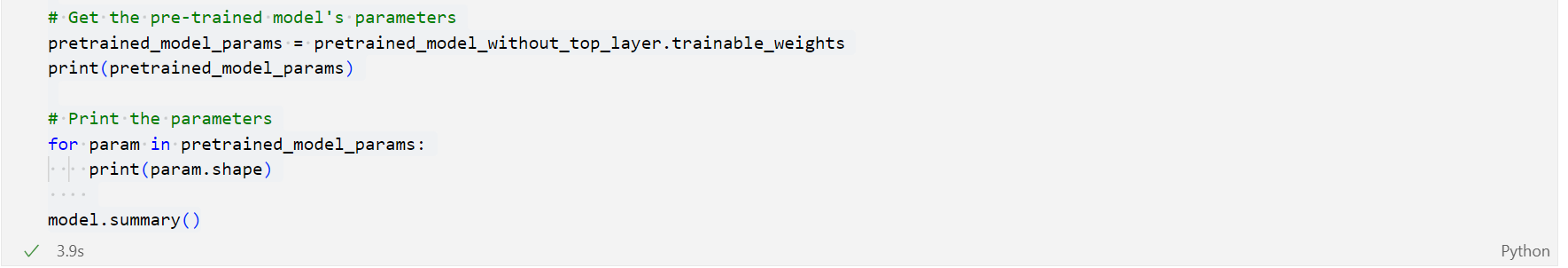


train\_test\_split(...): Splits the dataset into training and testing sets.

X\_train\_scaled = X\_train / 255: Normalizes the training images.

X\_test\_scaled = X\_test / 255: Normalizes the test images.





feature\_extractor\_model: URL for the MobileNet V2 feature extraction model.

hub.KerasLayer(...): Loads the feature extractor model without its top layer.

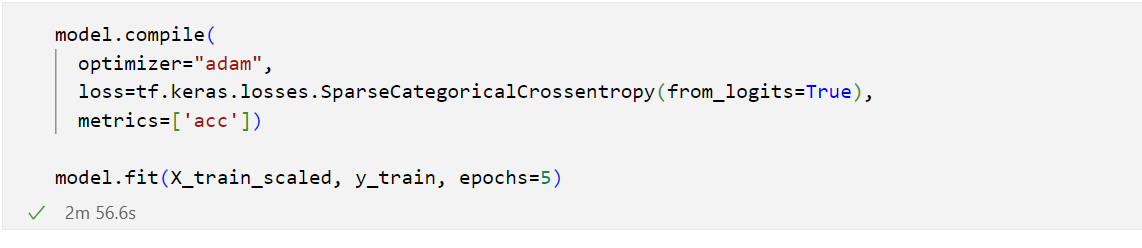
model = tf\_keras.Sequential([...]): Creates a new Sequential model with the feature extractor and a dense layer for classification.

model.compile(...): Compiles the model with:

optimizer="adam": Uses the Adam optimizer.

loss=tf.keras.losses.SparseCategoricalCrossentropy(from\_logits=True): Sets the loss function.

metrics=['acc']: Uses accuracy as the evaluation metric.



feature\_extractor\_model: URL for the MobileNet V2 feature extraction model.

hub.KerasLayer(...): Loads the feature extractor model without its top layer.

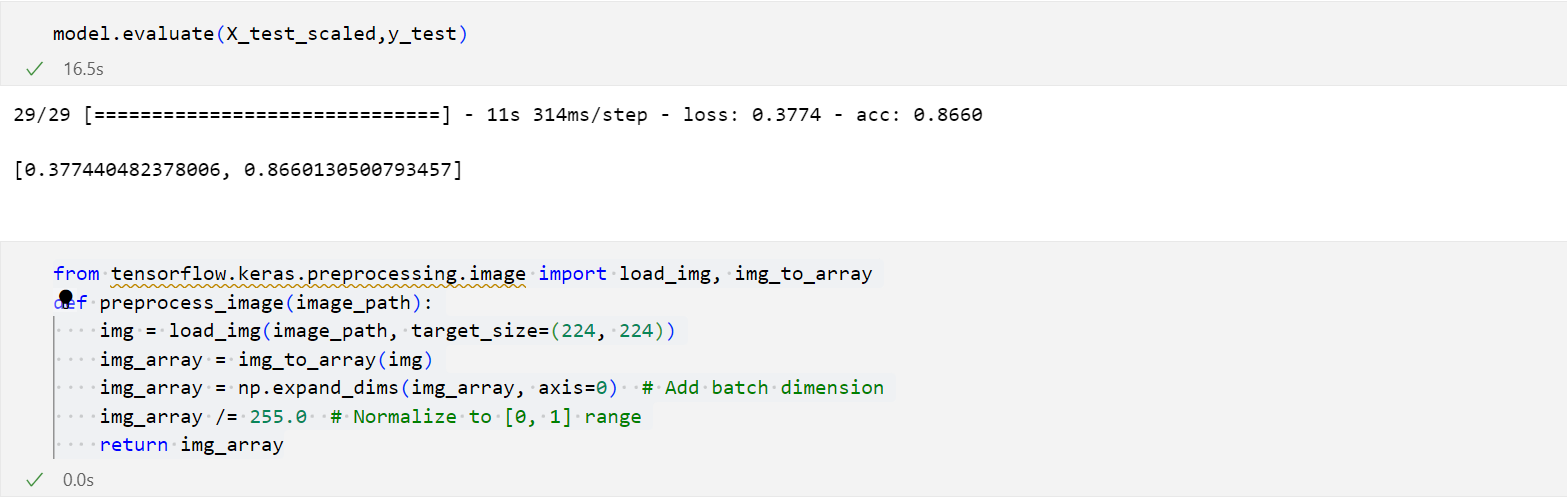
model = tf\_keras.Sequential([...]): Creates a new Sequential model with the feature extractor and a dense layer for classification.

model.compile(...): Compiles the model with:

optimizer="adam": Uses the Adam optimizer.

loss=tf.keras.losses.SparseCategoricalCrossentropy(from\_logits=True): Sets the loss function.

metrics=['acc']: Uses accuracy as the evaluation metric.



model.fit(...): Trains the model using the scaled training data for 5 epochs.

def preprocess\_image(image\_path):: Defines a function to preprocess an image for prediction.

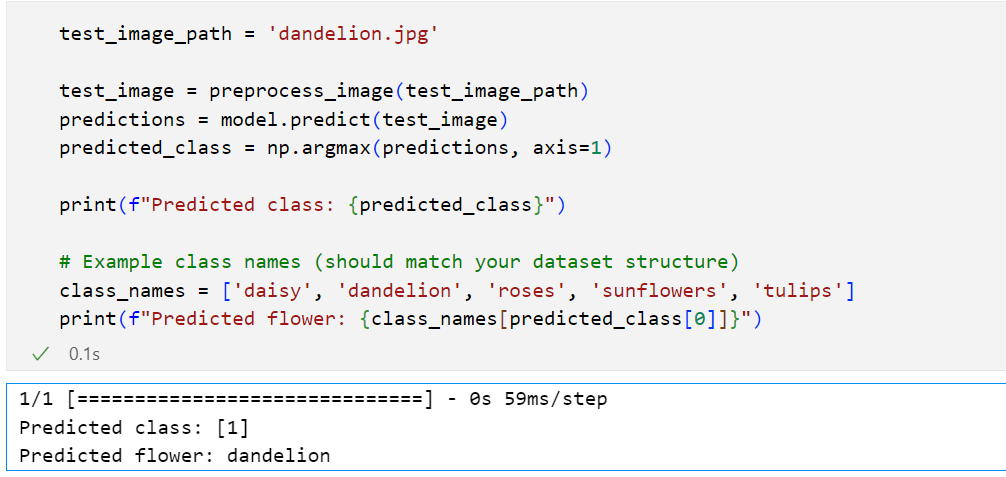
load\_img(image\_path, target\_size=(224, 224)): Loads and resizes the image.

img\_to\_array(img): Converts the image to a NumPy array.

np.expand\_dims(img\_array, axis=0): Adds a batch dimension.

img\_array /= 255.0: Normalizes the pixel values.

return img\_array: Returns the processed image array.



test\_image = preprocess\_image(test\_image\_path): Preprocesses the test image.

predictions = model.predict(test\_image): Uses the model to predict the class of the image.

predicted\_class = np.argmax(predictions, axis=1): Finds the predicted class with the highest probability.

print(f"Predicted class: {predicted\_class}"): Prints the predicted class.

**VARIATION-** changed the sun.jpg to dandelion.jpg to predict dandelion

**GitHubLink:**

[**https://github.com/abraaaar/RVU\_BtechHons/tree/main/Deep%20Learning/Lab5**](https://github.com/abraaaar/RVU_BtechHons/tree/main/Deep%20Learning/Lab5)