|  |  |
| --- | --- |
| **Ex No: 5**  **Date: 04/9/24** | **CNN Transfer Learning** |

**Title**: Transfer Learning using CNN for Image Classification

**Objective**:  
To implement transfer learning using a pre-trained Convolutional Neural Network (CNN) model for image classification using TensorFlow and Keras.

**Description**:  
Transfer learning is a machine learning technique where a pre-trained model is reused for a different but related task. It is particularly useful when a large dataset is unavailable for training from scratch. In this lab, we use a CNN model pre-trained on the ImageNet dataset and apply it to classify images from a new dataset. We will fine-tune the model's top layers and use the pre-trained layers to extract useful features.

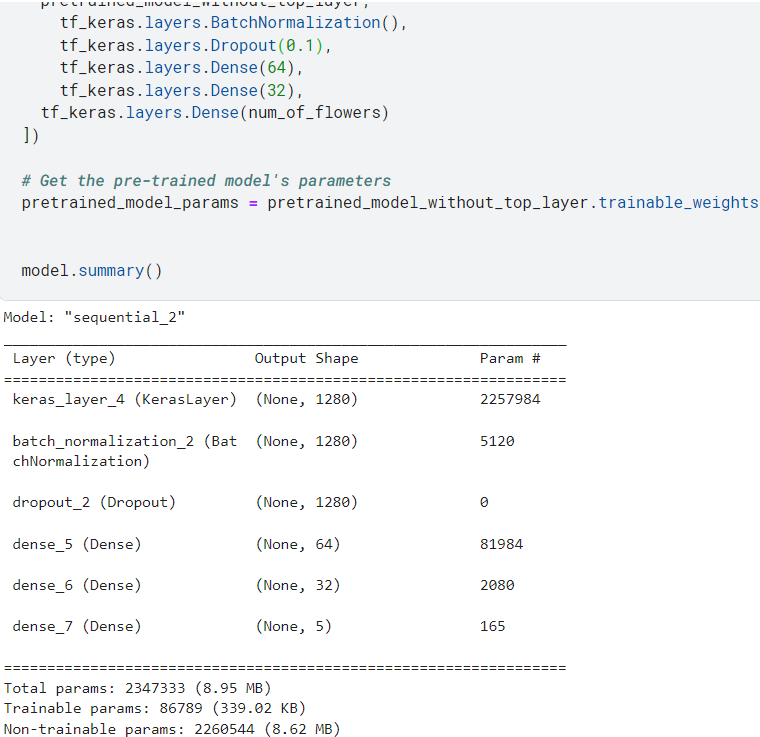
**Model Architecture**:

* **Pre-Trained Model**: Use a pre-trained model (e.g., VGG16, ResNet50) without the top fully connected layers.
* **Flatten Layer**: Flatten the extracted features into a 1D vector.
* **Dense Layer 1**: 512 units with ReLU activation.
* **Dropout Layer**: 0.5 dropout rate for regularization.
* **Output Layer**: A dense layer with softmax activation corresponding to the number of classes in the new dataset.

**Procedure**:

1. **Data Preparation**:
   * Load the new dataset.
   * Preprocess the images (resize to the input shape required by the pre-trained model, normalize pixel values to the range [0, 1]).
   * Split the dataset into training and validation sets.
2. **Model Building**:
   * Load a pre-trained model (e.g., ResNet50) from Keras applications, without the top layers.
   * Freeze the layers of the pre-trained model to retain the learned features.
   * Add new layers on top for fine-tuning, as described in the model architecture.
3. **Model Compilation**:
   * Compile the model using categorical\_crossentropy as the loss function, Adam optimizer, and accuracy as the metric.
4. **Training**:
   * Train the model on the new dataset using a specified number of epochs and batch size.
   * Monitor the training process using validation accuracy.
5. **Evaluation**:
   * Evaluate the model's performance on the validation/test set.
   * Plot the training and validation accuracy/loss curves.
6. **Prediction**:
   * Use the trained model to predict classes for new input images.
   * Display a few test images with their predicted labels.
7. **Fine-Tuning**:
   * Fine-tuning involves taking a pre-trained model, such as MobileNet V2, and adapting it to a new classification task. Initially, the pre-trained layers are frozen to retain the features learned from a large dataset like ImageNet. After adding new layers specific to the task, selected pre-trained layers are gradually unfrozen and retrained on the new dataset. This allows the model to adjust its features while still leveraging the knowledge gained from its previous training. We added a BatchNormalization and a Dropout layer to reduce overfitting and then added two dense layers of 64 and 32 neurons to get better fitting. We tuned the batch size, epochs and validation data size to get the best accuracy and lowest loss possible.

**Results**: The model achieved an accuracy of 86.05% and loss of 44.3% on the validation dataset.



**Conclusion**: Transfer learning with a pre-trained CNN model was successfully implemented for image classification. Fine-tuning the model improved its performance, demonstrating the effectiveness of using pre-trained models for similar image classification tasks. Further improvements could include adjusting the learning rate or exploring different pre-trained models.

**GitHub Link:**

<https://github.com/LohithR22/Fundamentals_of_DeepLearning>