Computer Science Department
CS672 – Introduction to Deep Learning (CRN: 23817)
Spring 2024

Project #3 / Due 04-May-2024

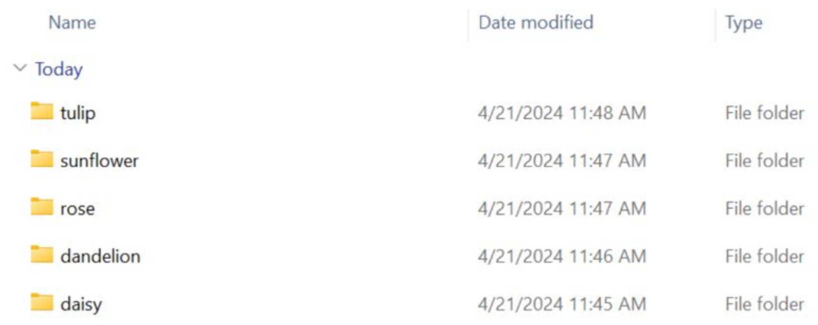
Build a Deep Learning model (based on Neural Networks) that provides reliable and improved accuracy of an **Image Recognition Classifier** based on **Convolution Neural Networks** (CNNs).

Task: Fine-tuning a Pre-trained Image Classifier for Flower Classification

For your deep learning classifier project on flower species classification, you can use the popular "Flowers Recognition" dataset. This dataset contains 4242 (4317) images of flowers belonging to 5 different classes: "daisy", "dandelion", "rose", "sunflower", and "tulip". Each class contains roughly 800 images.

You can download this dataset from the following link:
https://www.kaggle.com/datasets/alxmamaev/flowers-recognition

After you have downloaded the 'archive' dataset, extract it into your local machine.
The following directories will be created, each one containing the images of the flowers:

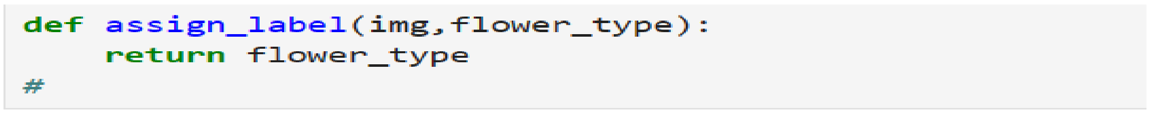


Name	Date modified	Type
Today		
tulip	4/21/2024 11:48 AM	File folder
sunflower	4/21/2024 11:47 AM	File folder
rose	4/21/2024 11:47 AM	File folder
dandelion	4/21/2024 11:46 AM	File folder
daisy	4/21/2024 11:45 AM	File folder

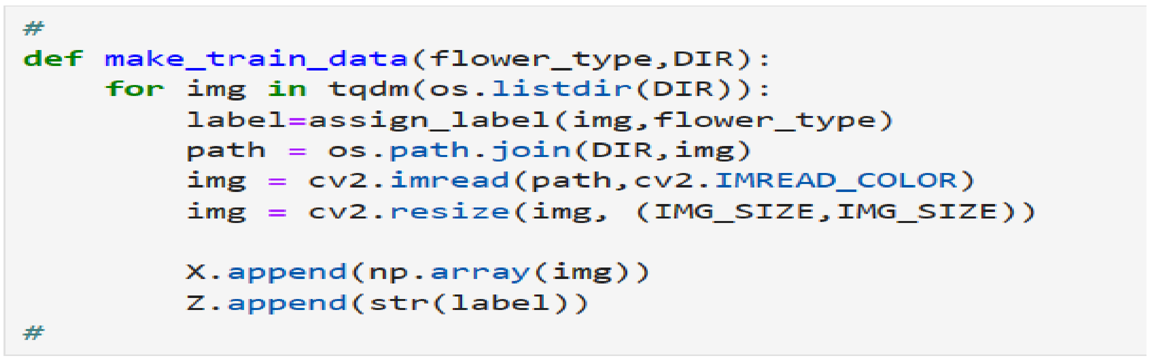
Step-1: Data Preparation:

Create the training/test datasets as follows:

```
X=[]
Z=[]
IMG_SIZE=150
FLOWER_DAISY_DIR='C:\\Pace\\CS672\\Projects\\data\\input\\daisy'
FLOWER_SUNFLOWER_DIR='C:\\Pace\\CS672\\Projects\\data\\input\\sunflower'
FLOWER_TULIP_DIR='C:\\Pace\\CS672\\Projects\\data\\input\\tulip'
FLOWER_DANDI_DIR='C:\\Pace\\CS672\\Projects\\data\\input\\dandelion'
FLOWER_ROSE_DIR='C:\\Pace\\CS672\\Projects\\data\\input\\rose'
#
```

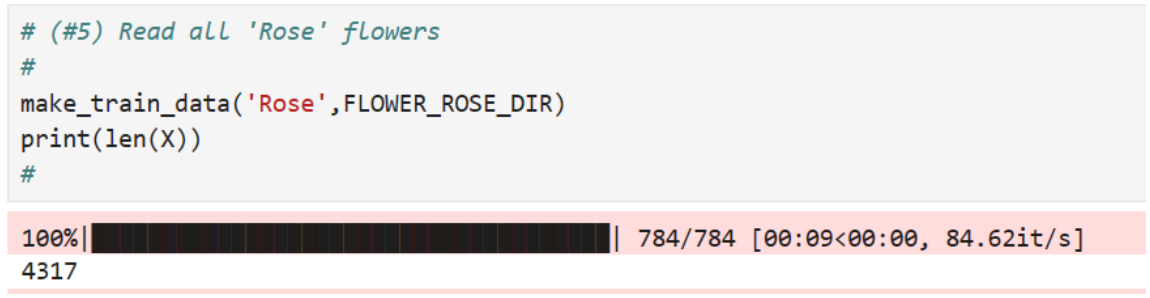


```
def assign_label(img,flower_type):
    return flower_type
#
```



```
#
def make_train_data(flower_type,DIR):
    for img in tqdm(os.listdir(DIR)):
        label=assign_label(img,flower_type)
        path = os.path.join(DIR,img)
        img = cv2.imread(path,cv2.IMREAD_COLOR)
        img = cv2.resize(img, (IMG_SIZE,IMG_SIZE))

        X.append(np.array(img))
        Z.append(str(label))
#
```

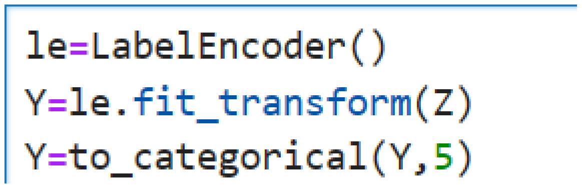
Call five (5) times the above function to place all data-flowers into the list X:



```
# (#5) Read all 'Rose' flowers
#
make_train_data('Rose',FLOWER_ROSE_DIR)
print(len(X))
#
```

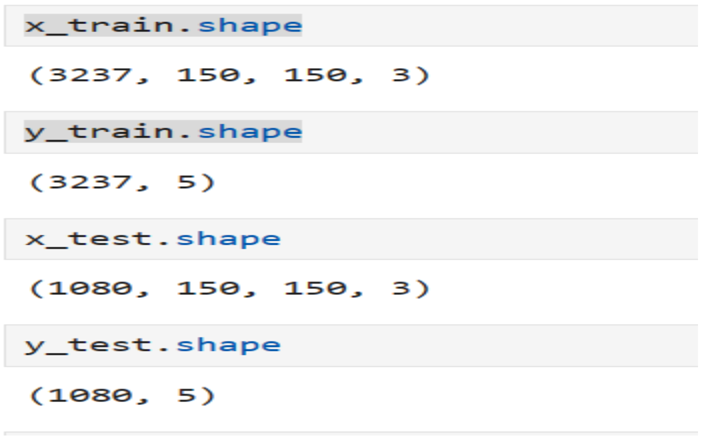
```
100%|██████████████████████████████████████████| 784/784 [00:09<00:00, 84.62it/s]
4317
```

The total number of records (from the original file) is 4,317 entries (flowers).
Label encoding your Y variable (Daisy → 0, Rose → 1, etc.…)



```
le=LabelEncoder()
Y=le.fit_transform(Z)
Y=to_categorical(Y,5)
```

Next, split into 'training' and 'test' dataset in 75/25 ratio:



```
x_train.shape
```

```
(3237, 150, 150, 3)
```

```
y_train.shape
```

```
(3237, 5)
```

```
x_test.shape
```

```
(1080, 150, 150, 3)
```

```
y_test.shape
```

```
(1080, 5)
```

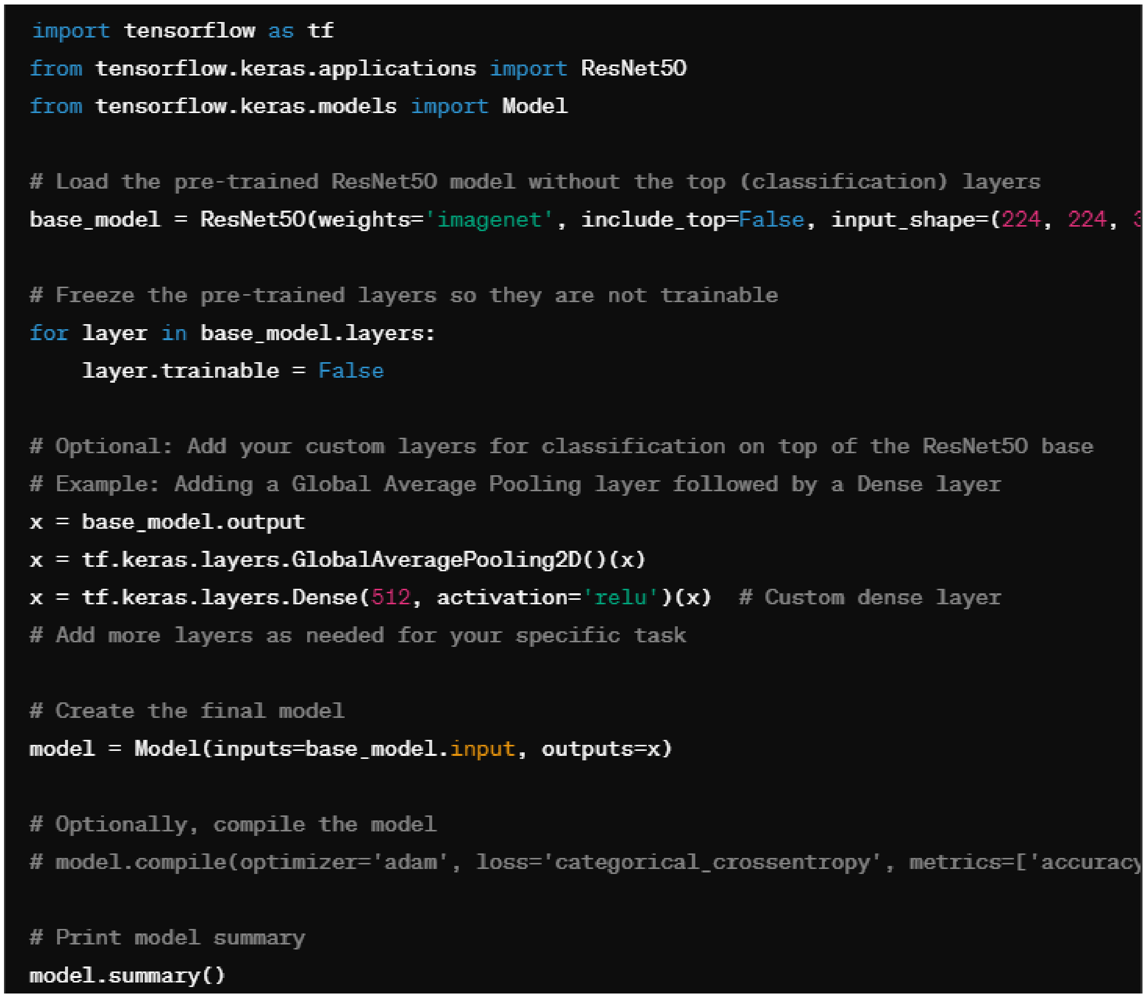
Step-2: Selecting a Pre-trained Model:

Choose a pre-trained convolutional neural network (CNN) model that has been trained on a large dataset, such as Inception, ResNet, VGG, or MobileNet.
These models are all trained on **ImageNet**, a large dataset containing millions of labeled images from thousands of categories.

Step-3: Implementing Transfer Learning in TensorFlow:

- Load the pre-trained model (e.g., ResNet50) using **TensorFlow's Keras API**, excluding the top (classification) layers.
- Add a new fully connected layer with the appropriate number of units for flower classification.
- Freeze the weights of the pre-trained layers to prevent them from being updated during training.
- Train the model on the flower dataset, using a relatively small number (say 20) of epochs since we're fine-tuning.
- Evaluate the model's performance on the validation set and fine-tune hyperparameters if necessary.

For instance:

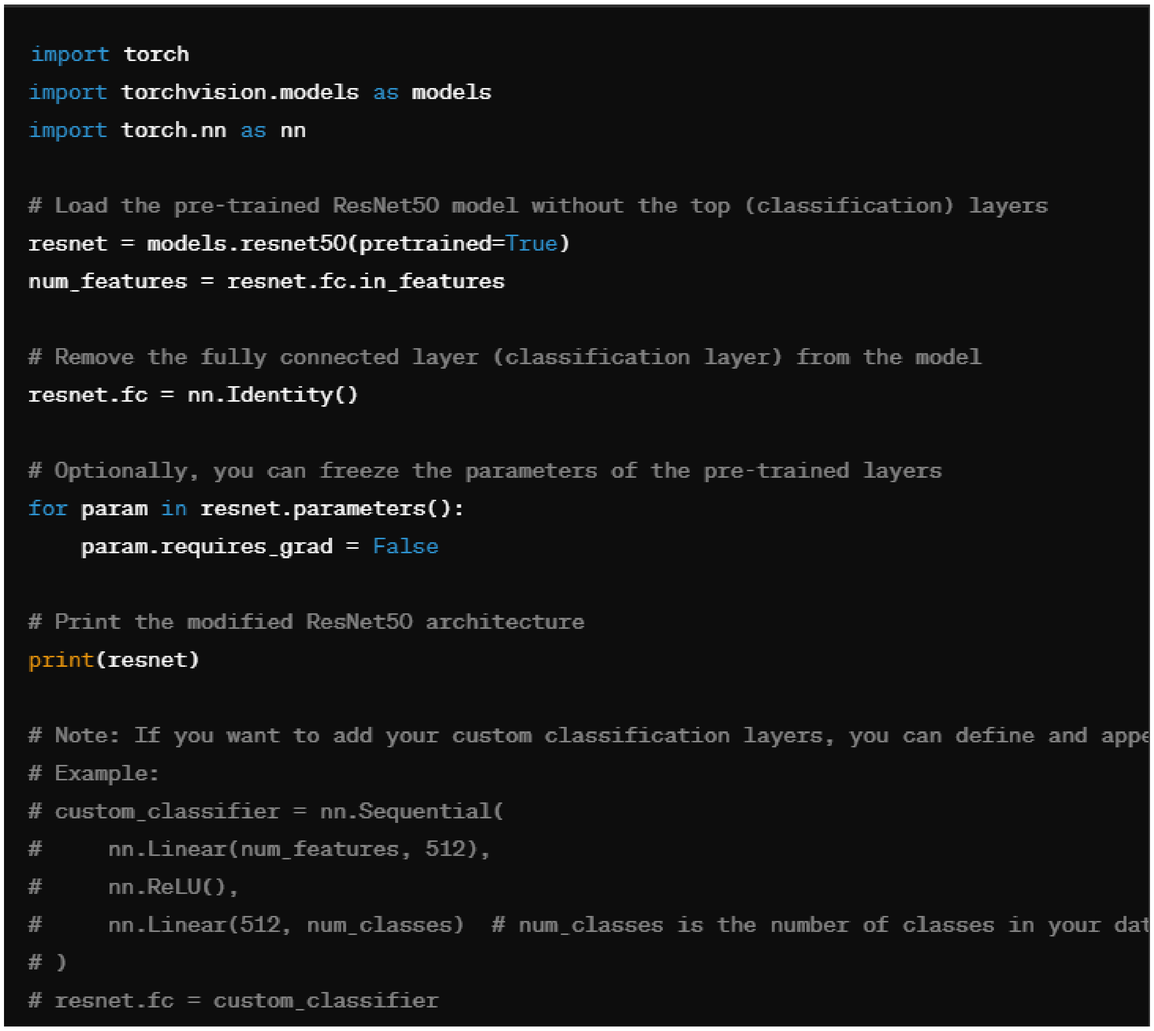


```
import tensorflow as tf
from tensorflow.keras.applications import ResNet50
from tensorflow.keras.models import Model

# Load the pre-trained ResNet50 model without the top (classification) layers
base_model = ResNet50(weights='imagenet', include_top=False, input_shape=(224, 224, 

# Freeze the pre-trained layers so they are not trainable
for layer in base_model.layers:
    layer.trainable = False

# Optional: Add your custom layers for classification on top of the ResNet50 base
# Example: Adding a Global Average Pooling layer followed by a Dense layer
x = base_model.output
x = tf.keras.layers.GlobalAveragePooling2D()(x)
x = tf.keras.layers.Dense(512, activation='relu')(x)  # Custom dense layer
# Add more layers as needed for your specific task

# Create the final model
model = Model(inputs=base_model.input, outputs=x)

# Optionally, compile the model
# model.compile(optimizer='adam', loss='categorical_crossentropy', metrics=['accurac

# Print model summary
model.summary()
```

Step-4: Implementing Transfer Learning in PyTorch:

- Load the pre-trained model (e.g., ResNet50) using PyTorch's '**torchvision.models**'.
- Replace the fully connected layer (classifier) with a new one suitable for flower classification.
- Freeze the weights of the pre-trained layers.
- Define a loss function (e.g., CrossEntropyLoss) and an optimizer (e.g., SGD or Adam).
- Train the model on the flower dataset, similarly using a small number of epochs.
- Evaluate the model's performance on the validation set and adjust as needed.

For instance:



```
import torch
import torchvision.models as models
import torch.nn as nn

# Load the pre-trained ResNet50 model without the top (classification) layers
resnet = models.resnet50(pretrained=True)
num_features = resnet.fc.in_features

# Remove the fully connected layer (classification layer) from the model
resnet.fc = nn.Identity()

# Optionally, you can freeze the parameters of the pre-trained layers
for param in resnet.parameters():
    param.requires_grad = False

# Print the modified ResNet50 architecture
print(resnet)

# Note: If you want to add your custom classification layers, you can define and appe
# Example:
# custom_classifier = nn.Sequential(
#     nn.Linear(num_features, 512),
#     nn.ReLU(),
#     nn.Linear(512, num_classes)  # num_classes is the number of classes in your dat
# )
# resnet.fc = custom_classifier
```

Step-5: Model Evaluation:

- Once training is complete, evaluate the fine-tuned model on the test set to assess its performance.
- Calculate metrics such as accuracy, precision, recall, and F1-score to measure the model's effectiveness in flower classification.

Guidelines: Project’s Open-End Approach

Here's a structured approach along with best practices and recommendations for you to follow:

<> Understanding Transfer Learning:

Conceptual Understanding: Ensure a clear understanding of what transfer learning is and its principles. Transfer learning involves leveraging pre-trained models on a source task to accelerate learning on a target task.
Types of Transfer Learning: Identify the different types of transfer learning, including feature extraction, fine-tuning, and domain adaptation, and choose the most suitable for your project.

<> Data Preparation:

Dataset Selection: Choose a dataset relevant to your target task. Ensure it has sufficient data to train the model effectively.
Data Preprocessing: Preprocess the data appropriately, including resizing images, data augmentation, normalization, etc.
Data Splitting: Divide the dataset into training, validation, and testing sets.

<> Model Building:

Select Pre-trained Model: Choose a pre-trained model suitable for your task and dataset. Popular choices include Inception, VGG, ResNet, Inception, and MobileNet.
TensorFlow Implementation:
Import the pre-trained model from TensorFlow Hub or use models available in TensorFlow.keras.applications.
Modify the model's top layers for the new classification task.
Fine-tune the model if required.
PyTorch Implementation:
Load pre-trained models from torchvision.models or other sources.
Replace or modify the classifier head for the new task.
Optionally fine-tune the model.

<> Training:

Freezing Layers: Decide which layers to freeze during training based on the amount of data available and similarity between the source and target tasks.
Learning Rate Scheduling: Implement learning rate scheduling techniques like decay or cyclical learning rates to optimize model training.
Regularization: Apply regularization techniques such as dropout or weight decay to prevent overfitting.
Monitoring: Monitor training progress using metrics like accuracy, loss, and validation performance. Early stopping can be employed to prevent overfitting.

<> Evaluation:

Performance Metrics: Evaluate the model's performance on the test set using appropriate metrics like accuracy, precision, recall, and F1-score.
Visualization: Visualize model predictions, confusion matrices, and other relevant metrics to gain insights into the model's behavior.

<> Deployment:

Model Serialization: Serialize the trained model to a format suitable for deployment, such as TensorFlow's SavedModel or PyTorch's .pt format.
Integration: Integrate the model into your desired application or framework for inference.
Performance Optimization: Optimize the model for inference speed and memory footprint, if required.

Summary: Benefits of Transfer Learning:

- By leveraging a pre-trained model's features, you can achieve good performance even with a limited amount of data.
- Transfer learning reduces the computational resources and time required for training, as you're only fine-tuning a few layers instead of training the entire model from scratch.

Conclusion:

Transfer learning is a powerful technique for utilizing pre-trained models to tackle new tasks with limited data.
In this project, you will demonstrate how to apply transfer learning using both TensorFlow and PyTorch frameworks to build an image classifier for flower classification. This approach can be extended to various other domains where labeled data is scarce, but pre-trained models are available.