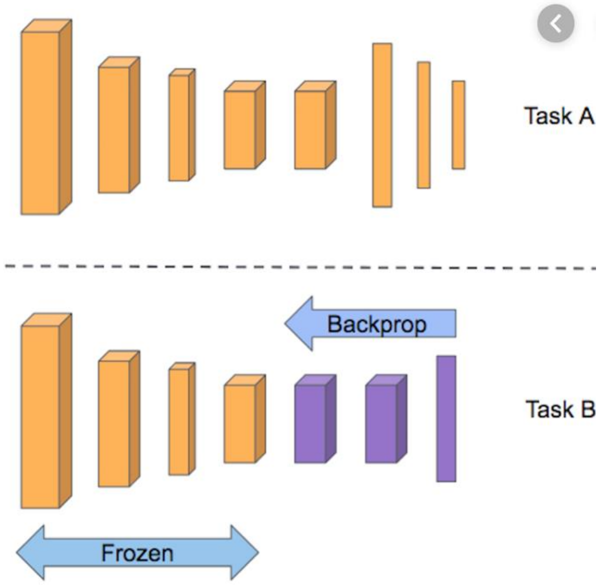
CS6005 Deep Learning Techniques

Flowers Image Classification using Transfer Learning (VGG16)



Name	Gokul. S
Roll Number	2018103026
Batch	P
Semester	6 (RUSA)
Date of Submission	09. 05. 2021

Problem Statement: To classify images of the Flowers dataset into one of the five output classes using a pretrained VGG16 model (Transfer Learning). This task is more complex when compared to traditional image classification as the dataset comprises of purely natural real world images of flowers.

Dataset: Flowers Dataset → (Tensorflow Official Dataset)

Description: Flowers is a real-world image dataset for developing machine learning and object recognition algorithms with minimal requirement on data pre - processing and formatting. It is obtained from tensorflow's official dataset forum. It is basically an flower type recognition dataset of 3670 train and 3670 test flower images coming from real world data. Images are cropped to 32x32.

URL: https://www.tensorflow.org/datasets/catalog/tf_flowers

Base Pre-Trained Model Used: VGG16

Description: VGG16 (also called OxfordNet) is a convolutional neural network architecture named after the Visual Geometry Group from Oxford, who developed it. It is 16 layers deep. The model loads a set of weights pre-trained on ImageNet. The default input size for VGG16 model is 224 x 224 pixels with 3 channels for RGB image. It has convolution layers of 3x3 filter with a stride 1 and maxpool layer of 2x2 filter of stride 2.

Code:

```python
from keras.layers import Input, Lambda, Dense, Flatten
from keras.models import Model
from keras.applications.vgg16 import VGG16
from keras.applications.vgg16 import preprocess_input
from keras.preprocessing import image
from keras.preprocessing.image import ImageDataGenerator
from keras.models import Sequential
import numpy as np
from glob import glob
import matplotlib.pyplot as plt

# re-size all the images to this
IMAGE_SIZE = [224, 224]

train_path = 'Datasets/Train'
valid_path = 'Datasets/Test'

# add preprocessing layer to the front of VGG
vgg = VGG16(input_shape=IMAGE_SIZE + [3], weights='imagenet', include_top=False)

# don't train existing weights
for layer in vgg.layers:
    layer.trainable = False



# for getting number of classes
folders = glob('Datasets/Train/*')



x = Flatten()(vgg.output)
# x = Dense(1000, activation='relu')(x)
prediction = Dense(len(folders), activation='softmax')(x)

# create a model object
model = Model(inputs=vgg.input, outputs=prediction)

# view the structure of the model
model.summary()

# tell the model what cost and optimization method to use
model.compile(
  loss='categorical_crossentropy',
  optimizer='adam',
  metrics=['accuracy']
)
```

```python
from keras.preprocessing.image import ImageDataGenerator

train_datagen = ImageDataGenerator(rescale = 1./255,
                                   shear_range = 0.2,
                                   zoom_range = 0.2,
                                   horizontal_flip = True)

test_datagen = ImageDataGenerator(rescale = 1./255)

training_set = train_datagen.flow_from_directory('Datasets/Train',
                                                 target_size = (224, 224),
                                                 batch_size = 32,
                                                 class_mode = 'categorical')

test_set = test_datagen.flow_from_directory('Datasets/Test',
                                            target_size = (224, 224),
                                            batch_size = 32,
                                            class_mode = 'categorical')


# fit the model
r = model.fit_generator(
  training_set,
  validation_data=test_set,
  epochs=5,
  steps_per_epoch=len(training_set),
  validation_steps=len(test_set)
)
# loss
plt.plot(r.history['loss'], label='train loss')
plt.plot(r.history['val_loss'], label='val loss')
plt.legend()
plt.show()
```

```
In [6]: # loss
        plt.plot(r.history['loss'], label='train loss')
        plt.plot(r.history['val_loss'], label='val loss')
        plt.legend()
        plt.show()
        plt.savefig('LossVal_loss')
```

Methodology:

1) Import necessary modules
2) Set train, test dataset paths and define image shape.
3) Import VGG16 model and its weights from keras.applications and preprocess the inputs.
4) Freeze the layers by setting trainable attribute to False.
5) Flatten() the penultimate feature vector of the VGG16 model and pass it to a final dense layer that contains 5 neurons (total number of classes).
6) Define a process pipeline using ImageDataGenerator and train the model.
7) Retrieve the corresponding metrics post training and plot their graphs (to make sure overfitting has not occurred).
8) The final validation accuracy and loss is the overall test accuracy and loss.

Execution Snapshots:

Overall CNN Model Summary:

File Edit View Insert Cell Kernel Help Trusted Python 3

```
In [9]: model.summary()
Model: "model"
_________________________________________________________________
Layer (type)                 Output Shape              Param #
=================================================================
input_1 (InputLayer)         [(None, 224, 224, 3)]     0
_________________________________________________________________
block1_conv1 (Conv2D)        (None, 224, 224, 64)      1792
_________________________________________________________________
block1_conv2 (Conv2D)        (None, 224, 224, 64)      36928
_________________________________________________________________
block1_pool (MaxPooling2D)   (None, 112, 112, 64)      0
_________________________________________________________________
block2_conv1 (Conv2D)        (None, 112, 112, 128)     73856
_________________________________________________________________
block2_conv2 (Conv2D)        (None, 112, 112, 128)     147584
_________________________________________________________________
block2_pool (MaxPooling2D)   (None, 56, 56, 128)       0
_________________________________________________________________
block3_conv1 (Conv2D)        (None, 56, 56, 256)       295168
_________________________________________________________________
block3_conv2 (Conv2D)        (None, 56, 56, 256)       590080
_________________________________________________________________
block3_conv3 (Conv2D)        (None, 56, 56, 256)       590080
_________________________________________________________________
block3_pool (MaxPooling2D)   (None, 28, 28, 256)       0
_________________________________________________________________
block4_conv1 (Conv2D)        (None, 28, 28, 512)       1180160
_________________________________________________________________
block4_conv2 (Conv2D)        (None, 28, 28, 512)       2359808
_________________________________________________________________
block4_conv3 (Conv2D)        (None, 28, 28, 512)       2359808
_________________________________________________________________
block4_pool (MaxPooling2D)   (None, 14, 14, 512)       0
_________________________________________________________________
block5_conv1 (Conv2D)        (None, 14, 14, 512)       2359808
_________________________________________________________________
block5_conv2 (Conv2D)        (None, 14, 14, 512)       2359808
_________________________________________________________________
block5_conv3 (Conv2D)        (None, 14, 14, 512)       2359808
_________________________________________________________________
block5_pool (MaxPooling2D)   (None, 7, 7, 512)         0
_________________________________________________________________
flatten (Flatten)            (None, 25088)             0
_________________________________________________________________
dense (Dense)                (None, 5)                 125445
=================================================================
Total params: 14,840,133
Trainable params: 125,445
Non-trainable params: 14,714,688
_________________________________________________________________
```

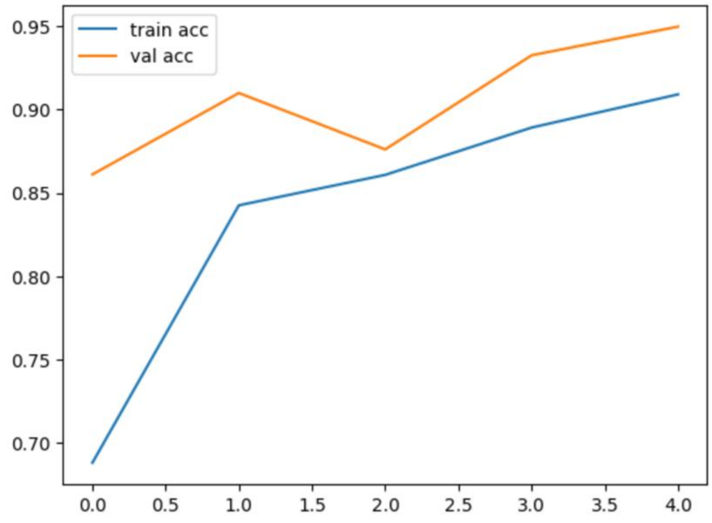
Training:

- Just five epochs of training is sufficient for the model to learn such a complex dataset. This is because VGG16 has already been trained sufficiently.

```
Epoch 1/5
115/115 [==============================] - 473s 4s/step - loss: 1.2242 - accuracy: 0.5791 - val_loss: 0.4047 - val
_accuracy: 0.8610
Epoch 2/5
115/115 [==============================] - 752s 7s/step - loss: 0.4832 - accuracy: 0.8313 - val_loss: 0.2852 - val
_accuracy: 0.9098
Epoch 3/5
115/115 [==============================] - 522s 5s/step - loss: 0.3582 - accuracy: 0.8758 - val_loss: 0.3273 - val
_accuracy: 0.8760
Epoch 4/5
115/115 [==============================] - 548s 5s/step - loss: 0.3148 - accuracy: 0.9015 - val_loss: 0.2046 - val
_accuracy: 0.9324
Epoch 5/5
115/115 [==============================] - 554s 5s/step - loss: 0.2716 - accuracy: 0.9084 - val_loss: 0.1612 - val
_accuracy: 0.9496
```

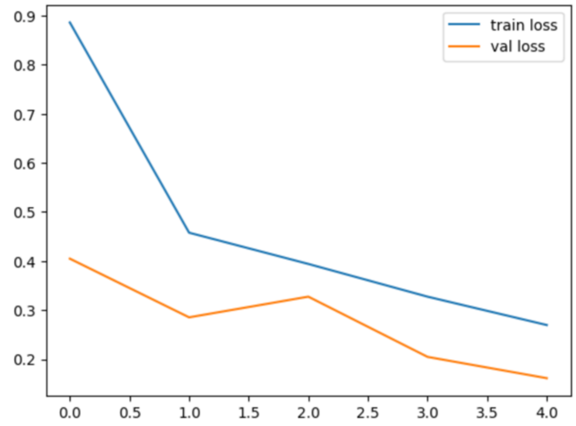
Accuracy Graph:

```
In [8]: # accuracies
        plt.plot(r.history['accuracy'], label='train acc')
        plt.plot(r.history['val_accuracy'], label='val acc')
        plt.legend()
        plt.show()
```



Loss Graph:

```
In [6]: # loss
        plt.plot(r.history['loss'], label='train loss')
        plt.plot(r.history['val_loss'], label='val loss')
        plt.legend()
        plt.show()
        plt.savefig('LossVal_loss')
```



Note: Early stopping hasn't happened here as the val_loss curve always lies below the train_loss curve all throughout the training and hasn't crossed it at any point

Final Test Loss: 0.1612
Final Test Accuracy: 94.96%

Results: The final model achieves an overall test accuracy of 94.96 % and a test loss of 0.1612 which is really good considering the naturality of the real world dataset that has been considered.

Conclusion:

The model has performed extremely well (around 95% accurate) compared to traditional ANNs by substantially reducing the number of parameters to be trained and time taken to train by using a pretrained model (VGG16) and capturing each and every aspect of the image onto a separate feature map.

References:

[1] https://ruder.io/transfer-learning/

[2] https://en.wikipedia.org/wiki/Transfer_learning

[3] https://keras.io/api/applications/vgg/

[4] https://www.mathworks.com/help/deeplearning/ref/vgg16.html;jsessionid=09e86fa11ef6d89a9d2802d83d77

[5] https://builtin.com/data-science/transfer-learning