Apply convolutional neural network (CNN) to the above image and obtain the reformatted image, convolutional layer output image, activation layer output image and pooling layer output image

INPUT DESCRIPTION

IMPORTING LIBRARIES

```
!pip install tensorflow
!pip install keras
import numpy as np
import tensorflow as tf
import matplotlib.pyplot as plt
from itertools import product
```

Here we are installing the libraries such as Tensorflow, and Keras and importing the libraries such as, Numpy Matplotlib,Product which are required for the project

Defining parameters :

```
plt.rc( ‘ figure ’ , autolayout = True)
plt.rc( ‘ image  ’ , cmap = ‘magma’ )
```

The code sets the default figure layout to be automatically adjusted for better spacing ('autolayout=True') and sets the default colormap for images to 'magma'.

Defining Kernel :

```
kernel=tf.constant([[-1,-1,-1],[-1,8,-1],[-1,-1,-1],[-1,-1,-1],])
```

The provided code defines a 4x3 TensorFlow constant tensor representing a 3x3 convolution kernel. This specific kernel is often used for edge detection in image processing. The values in the kernel emphasize the contrast between neighboring pixels, making edges more prominent when applied to an image using convolution operations.

Loading image :

```
image=tf.io.read_file('/content/sample_data/1.jpeg')
image=tf.io.decode_jpeg(image,channels=1)
image=tf.image.resize(image,size=[300,300])
```

This code reads an image file ('1.jpeg') from the specified path, decodes it as a JPEG image with one channel, and then resizes the image to a fixed size of 300x300 pixels. The resulting 'image' variable is preprocessed and ready for further use in the training of the model for image classification.

Plotting image :

```
img=tf.squeeze(image).numpy()
plt . figure(figsize=(5,5))
plt . imshow(img,cmap='gray')
plt.axis(' off ')
plt.show( )
```

This code visualize a grayscale image. It first squeezes the input image tensor (`image`) to remove dimensions with size 1, converts it to a NumPy array, and then displays the image using Matplotlib with a grayscale colormap. The `plt.axis('off')` command turns off axis labels, and `plt.show()` displays the image.

Reformating image :

```
image=tf.image.convert_image_dtype(image,dtype=tf.float32)
image=tf.expand_dims(image,axis=0)
kernel=tf.reshape(kernel,[*kernel.shape,1,1])
kernel=tf.cast(kernel,dtype=tf.float32)
```

Reformatting images in a Convolutional Neural Network (CNN) typically involves resizing, normalization, and sometimes other preprocessing steps to prepare them for input into the model. This code is a part of image preprocessing.

1. tf.image.convert_image_dtype: Converts the image data type to float32, typically done to ensure numerical stability during subsequent processing.

2. tf.expand_dims: Adds an extra dimension to the image tensor, usually to match the expected input shape of a machine learning model (e.g., batch dimension).

3. tf.reshape and tf.cast: Reshapes and casts a kernel (presumably a convolutional filter) to float32, preparing it for convolution operations on the image.

Convolution process:

```
conv_fn=tf.nn.conv2d
image_filter=conv_fn(
                input=image,
                filters=kernel,
                strides=1,   # or(1,1)
                padding='SAME',
                )
plt.subplot(1,3,1)
plt.imshow(tf.squeeze(image_filter))
plt.axis('off')
plt.title('convolution')
```

This code performs a 2D convolution operation on an input image (`image`) with a specified convolutional kernel (`kernel`). The `conv_fn` is set to `tf.nn.conv2d`, and the result is stored in the variable `image_filter`. The code then uses Matplotlib to visualize the output of the convolution in a subplot. The image resulting from the convolution is displayed with the title "convolution" and no axis ticks. The `padding` parameter is set to 'SAME', indicating zero-padding is used to keep the output size the same as the input size.

Activation layer:

```
relu_fn=tf.nn.relu
#img detection
image_detect=relu_fn(image_filter)
plt.subplot(1,3,2)
plt.imshow(
   tf.squeeze(image_detect)
)
```

This code apply the Rectified Linear Unit (ReLU) activation function (`tf.nn.relu`) to the output of an image filter (`image_filter`). The result is then visualized using `matplotlib` with `plt.imshow`. It is displayed in the second subplot of a 1x3 grid. The `tf.squeeze` function is used to remove any dimensions of size 1 from the shape of the tensor before visualization.

Pooling layer :

```
pool=tf.nn.pool
```

```
image_condense=pool(input=image_detect,
                window_shape=(2,2),
                pooling_type='MAX',
                strides=(2,2),
                padding='SAME',
                )
plt.subplot(1,3,3)
plt.imshow(tf.squeeze(image_condense))
plt.axis('off')
plt.title('Pooling')
plt.show()
```

This code performs max pooling on the input image. Max pooling is applied with a window size of (2,2), a stride of (2,2), and 'SAME' padding. The resulting condensed image is then visualized using matplotlib, showing the effect of pooling in reducing the spatial dimensions of the input image.

Thus applied convolutional neural network (CNN) to the above image and obtained the reformatted image, convolutional layer output image, activation layer output image and pooling layer output image.