Title: AlexNet for Image Classification

Abstract:

This code demonstrates how to utilize the AlexNet convolutional neural network (CNN) model as the base model and build upon it for fine-tuning or transfer learning. By freezing the layers of the base model and adding additional layers, we create a new model capable of classifying images in a specific task. This report explains the code and provides a formatted version of it for easy understanding and implementation.

Introduction(What is Alexnet)

AlexNet is a special kind of computer program that can look at pictures and tell you what's in them. It was created by a team of researchers and was really good at this task. In fact, it was so good that it won a competition in 2012 where it had to identify thousands of different objects in pictures.

The secret to its success lies in the way it's built. It has layers that work like filters, where each layer looks for different patterns in the pictures. The filters start with simple shapes like lines and curves and gradually become more complex. These filters help the program understand different parts of the picture and recognize objects.

AlexNet also uses some clever tricks to make it even better. It uses a special type of math called ReLU that helps it understand more complex shapes and features. It also has a way to reduce noise in the images and prevent overfitting, which is when the program becomes too specific and doesn't generalize well.

Overall, AlexNet was a breakthrough in computer vision and inspired the development of more advanced programs that can identify objects in pictures with even greater accuracy.

.

Code Explanation:

The provided code snippet accomplishes the following steps:

1. Importing the necessary libraries: TensorFlow and Keras.

2. Loading the pre-trained AlexNet model using the `AlexNet` class from `tensorflow.keras.applications`. The `weights` parameter is set to 'imagenet' to load the pre-trained weights. We also specify `include_top=False` to exclude the original fully connected layers of AlexNet.

3. Freezing the layers of the base model ensures that only the additional layers we add on top will be trainable during training. This is achieved by iterating through each layer in the base model and setting `layer.trainable = False`.

4. Creating a new model using the `Sequential` class from Keras.

5. Adding the base model to the new model using `model.add(base_model)`. This incorporates all the layers from the base model into our new model.

6. Adding additional layers on top of the base model. In this example, we add a `Flatten` layer to convert the 4D tensor output of the base model into a 2D tensor. This is followed by a fully connected `Dense` layer with ReLU activation to learn more complex patterns. A `Dropout` layer is included for regularization, reducing overfitting. Lastly, a `Dense` layer with softmax activation is added to produce the final output probabilities for the desired number of classes in the task.

7. Compiling the model by specifying the optimizer, loss function, and metrics for evaluation.

8. Displaying the summary of the model, which provides a concise overview of the model architecture and the number of trainable parameters.

Formatted Code:

```
import tensorflow as tf
from tensorflow.keras.models import Sequential
from tensorflow.keras.layers import Dense, Dropout, Conv2D,
Activation, MaxPooling2D, Flatten
from tensorflow.keras.layers import BatchNormalization
```

```
#Instantiation
AlexNet = Sequential()

#1st Convolutional Layer
AlexNet.add(Conv2D(filters=96, input_shape=(32,32,3),
kernel_size=(11,11), strides=(4,4), padding='same'))
AlexNet.add(BatchNormalization())
AlexNet.add(Activation('relu'))
```

```
AlexNet.add(MaxPooling2D(pool_size=(2,2), strides=(2,2),
padding='same'))

#2nd Convolutional Layer
AlexNet.add(Conv2D(filters=256, kernel_size=(5, 5),
strides=(1,1), padding='same'))
AlexNet.add(BatchNormalization())
AlexNet.add(Activation('relu'))
AlexNet.add(MaxPooling2D(pool_size=(2,2), strides=(2,2),
padding='same'))

#3rd Convolutional Layer
AlexNet.add(Conv2D(filters=384, kernel_size=(3,3), strides=(1,1),
padding='same'))
AlexNet.add(BatchNormalization())
AlexNet.add(Activation('relu'))

#4th Convolutional Layer
AlexNet.add(Conv2D(filters=384, kernel_size=(3,3), strides=(1,1),
padding='same'))
AlexNet.add(BatchNormalization())
AlexNet.add(Activation('relu'))

#5th Convolutional Layer
AlexNet.add(Conv2D(filters=256, kernel_size=(3,3), strides=(1,1),
padding='same'))
AlexNet.add(BatchNormalization())
AlexNet.add(Activation('relu'))
AlexNet.add(MaxPooling2D(pool_size=(2,2), strides=(2,2),
padding='same'))

#Passing it to a Fully Connected layer
AlexNet.add(Flatten())
# 1st Fully Connected Layer
AlexNet.add(Dense(4096, input_shape=(32,32,3,)))
AlexNet.add(BatchNormalization())
AlexNet.add(Activation('relu'))
# Add Dropout to prevent overfitting
AlexNet.add(Dropout(0.4))

#2nd Fully Connected Layer
AlexNet.add(Dense(4096))
AlexNet.add(BatchNormalization())
AlexNet.add(Activation('relu'))
#Add Dropout
AlexNet.add(Dropout(0.4))
```

```
#3rd Fully Connected Layer
AlexNet.add(Dense(1000))
AlexNet.add(BatchNormalization())
AlexNet.add(Activation('relu'))
#Add Dropout
AlexNet.add(Dropout(0.4))

#Output Layer
AlexNet.add(Dense(10))
AlexNet.add(BatchNormalization())
AlexNet.add(Activation('softmax'))

# Compile the model
AlexNet.compile(optimizer='adam',
loss='categorical_crossentropy', metrics=['accuracy'])
# Print the summary of the model
AlexNet.summary()
```

Conclusion:

Fine-tuning the AlexNet model by leveraging its pre-trained weights can be a powerful approach for image classification tasks. By freezing the base model's layers and adding custom layers on top, we can adapt the model to specific datasets and achieve accurate predictions