EE 634/734 Introduction to Neural Networks

Assignment #5
Due Date: Friday, December 8th, 2023

MobileNet-v2 is a convolutional neural network that is 53 layers deep. You can download a pretrained version of more than a million images from the ImageNet database. It can classify images into 1000 object categories, such as keyboard, mouse, pencil, and many animals.

Task 1: Training the Model

I. Downloading Building the Model

The downloaded model expects images of 224 × 224 pixels with three color channels (i.e., 224 x 224 x 3). However, you can still use MobileNetV2 with images of different dimensions by specifying the **input_shape** parameter. For example, to customize the model with its pretrained weights for images of 96 x 96 x 3, you can use the following code:

```
import tensorflow as tf
from tensorflow.keras.applications import MobileNetV2

# Load the pre-trained MobileNetV2 model, excluding the top classification layer
base_model = MobileNetV2(input_shape=(96, 96, 3), include_top=False, weights='imagenet')

# Freeze the base model
base_model.trainable = False
```

With **include_top=False**, the pre-trained model does not enforce input sizes that were specific to the original training dataset. It also allows you to design and add your own dense layers (fully connected layers), which can be more suitable for the specific characteristics of your dataset and task. You will be able to change the number of neurons, adding dropout or batch normalization layers, and using different activation functions.

This is typically done for developing a new model where the number of classes is different from the model's original training dataset. For this project, we can add our own classification dense layers tailored to the number of classes for the CIFAR-10 dataset:

```
from tensorflow.keras.models import Model
from tensorflow.keras.layers import Input, GlobalAveragePooling2D, Dense
# Create new model on top
inputs = Input(shape=(96, 96, 3))
x = base_model(inputs)
x = GlobalAveragePooling2D()(x)
outputs = Dense(10, activation='softmax')(x)
model = Model(inputs, outputs)
```

GlobalAveragePooling2D is often used as a bridge between the feature extraction layers of a pre-trained model and the new classification layers added for a specific task. It provides a smooth transition without a significant increase in parameters.

II. Downloading and Processing the Data

```
from tensorflow.keras.datasets import cifar10
from tensorflow.keras.utils import to_categorical

# Load CIFAR-10 data
(train_images, train_labels), (test_images, test_labels) = cifar10.load_data()

# Normalize pixel values to be between 0 and 1
train_images, test_images = train_images / 255.0, test_images / 255.0

# Convert labels to one-hot encoding
train_labels = to_categorical(train_labels, 10)
test_labels = to_categorical(test_labels, 10)

# Resize images from 32x32 to 96x96 to fit MobileNetV2 input size requirements
train_images_resized = tf.image.resize(train_images, (96, 96))
test_images_resized = tf.image.resize(test_images, (96, 96))
```

III. Compiling and Training the Model

```
from tensorflow.keras.optimizers import Adam
# Compile the model
model.compile(optimizer=Adam(learning_rate=0.001), loss='categorical_crossentropy',
metrics=['accuracy'])

# Train the model
model.fit(
    train_images_resized, train_labels,
    epochs=5,
    batch_size = 128,
    validation_split=0.3,
)

# Evaluate the model
model.evaluate(test_images_resized, test_labels)
```

Run the above code in your Jupyter Notebook to have a trained model for the CIFAR dataset.

a) Show the plots during the training for the loss and accuracy of the training and validation data.
b) What accuracy did the model achieve on the training, validation data, and test data?

Task 2: Optimizing the model

IV. Grid Search Optimization

Choose two hyperparameters from the following list:

1. **Learning Rate**: One of the most crucial hyperparameters. You might want to try different learning rates or use learning rate schedulers that adjust the rate during training.
2. **Number of Epochs**: Adjusting the number of epochs can help in finding the sweet spot where the model is well-trained but not overfitted.
3. **Batch Size**: The size of the batches of data points fed to the model during training. Larger batch sizes offer faster computation but can lead to less generalization.
4. **Optimizer**: Different optimizers (like SGD, Adam, RMSprop) can have significant effects on the model's performance. Each optimizer has its own strengths and weaknesses depending on the task.
5. **Number of Neurons in Dense Layers**: Adjusting the number of neurons in the fully connected layers (if any) in your model can affect performance.
6. **Activation Functions**: Experimenting with different activation functions (ReLU, ELU, Leaky ReLU, etc.) in the neural network layers.

a) Define the Grid: Create a grid of possible three values for each of the two chosen hyperparameters.
b) Model Training: Train your model using each combination of hyperparameters in the grid.
c) Logging with TensorBoard: Log training metrics for each model configuration using TensorBoard.
d) Analysis: Analyze the TensorBoard outputs to determine the impact of different hyperparameter values on the model's performance.

V. Random Search Optimization

Choose three different hyperparameters from the above hyperparameters list for random search optimization.

a) Random Sampling: Randomly select combinations of values for the three chosen hyperparameters.
b) Model Training: Train your model using each randomly selected combination.
c) Logging with TensorBoard: Use TensorBoard to log the performance metrics of each training session.
d) Analysis: Evaluate the TensorBoard outputs to determine which combinations yield the best performance.

Combine the findings from both grid search and random search into a report. Compare and contrast the effectiveness of grid search and random search strategies based on your results. Conclude with the best overall hyperparameter configuration for your model as indicated by your experiments.

Evaluation Criteria:

- Quality and thoroughness of the experimental setup and analysis.
- Clarity and detail in the reports and TensorBoard visualizations.
- Effectiveness of the chosen hyperparameter optimization strategies.
- Overall quality and organization of the final report.