Assignment #9

For all your work, submit a Notebook (either Jupyter or Colab.)

Utilizing Python code snippets demonstrate how to use transfer learning with a **pre-trained (VGG *) model in TensorFlow/Keras framework**, using the MNIST dataset as an example:

```
import tensorflow as tf
from tensorflow.keras import layers, models
from tensorflow.keras.datasets import mnist
from tensorflow.keras.applications import VGG16
from tensorflow.keras.utils import to_categorical

# Load the MNIST dataset
(train_images, train_labels), (test_images, test_labels) = mnist.load_data()

# Preprocess the data
train_images = train_images.reshape((60000, 28, 28, 1)).astype('float32') / 255
test_images = test_images.reshape((10000, 28, 28, 1)).astype('float32') / 255
train_labels = to_categorical(train_labels)
test_labels = to_categorical(test_labels)

# Load pre-trained model (VGG16 in this case)
base_model = VGG16(weights='imagenet', include_top=False, input_shape=(28, 28, 3))
```

Perform the following tasks:

- Load the **MNIST dataset** and preprocess it.
- Then, **load a pre-trained VGG16 model** without its top layer (classification layer).
- **Freeze the weights** of the convolutional base of the VGG16 model to prevent them from being updated during training.
- Build a **new model consisting of additional layers on top of the pre-trained VGG16** base.
- **Concatenate** the pre-trained model with this new model.
- **Compile** the model (optimizer=adam, loss = categotical_corssentropy, metric=accuracy) and **train** it on the MNIST dataset.
- Finally, **evaluate the model** on the test set.

Important Note:
Please note that MNIST images are grayscale (1 channel), while VGG16 expects images with 3 channels (RGB), so need to reshape the MNIST images accordingly and add an additional layer to match the input shape of the VGG16 model.

(*) The **VGG** (**Visual Geometry Group**) model is a convolutional neural network architecture proposed by the Visual Geometry Group at the University of Oxford. It gained popularity due to its simplicity and

effectiveness in image classification tasks. The key characteristic of VGG networks is their uniform architecture, where the network consists primarily of multiple convolutional layers followed by max-pooling layers, and finally a few fully connected layers.

The original VGG network configurations, such as VGG16 and VGG19, have 16 and 19 layers respectively, with different depths achieved by varying the number of convolutional layers. These networks are trained on the ImageNet dataset for image classification tasks, where they achieved competitive performance in image recognition benchmarks.

Despite being deeper than previous architectures like AlexNet, VGG networks are relatively simple in design, making them easier to understand and implement. However, they are computationally more intensive due to their large number of parameters, making them less practical for real-time applications on resource-constrained devices compared to newer, more efficient architectures like ResNet or EfficientNet.