Instructions for Hands-on Lab to show OpenVINO^{TM} integration with TensorFlow

Lab 1: TensorFlow Transfer Learning and Inference on Intel DevCloud:

1) Register and get an account on Intel DevCloud at https://software.intel.com/devcloud.
2) Go to https://devcloud.intel.com/edge and sign in.
3) Navigate to "Build" >> "Connect and Create" >> "Connect to Jupyter".
4) In the JupyterHub, click on "New" >> "Terminal" on the top right-hand side. This will open a terminal in a new browser tab.
5) Create a new directory called "transfer_learning" in a location of your choice and navigate to "transfer_learning" directory.
6) Copy the notebook from TensorFlow tutorials from Github using the below command:
   wget https://raw.githubusercontent.com/tensorflow/docs/r2.4/site/en/tutorials/images/transfer_learning.ipynb
7) Go back to the JupyterHub browser tab, find the "transfer_learning" directory and click on "transfer_learning.ipynb" to open it.
8) Make the following changes before running the notebook:
   a. In Section 1.3.5, change the "initial_epochs" to 2. You can play with this parameter as you like, but to finish the tutorial in time, we recommend setting the value to "2".
   b. In Section 1.4.3, change the "fine_tune_epochs" to 2. You can play with this parameter as you like, but to finish the tutorial in time, we recommend setting the value to "2".
   c. In the last cell of the notebook in Section 1.4.4, add the following lines of code as indicated **in bold** to the cell:

```
import time

#Retrieve a batch of images from the test set
image_batch, label_batch = test_dataset.as_numpy_iterator().next()
predictions = model.predict_on_batch(image_batch).flatten()

#Adding another instance of prediction to exclude the first warm up iterat
ion in measuring time
#Retrieve a batch of images from the test set
image_batch, label_batch = test_dataset.as_numpy_iterator().next()
start_time = time.time()
predictions = model.predict_on_batch(image_batch).flatten()
end_time = time.time()
print("Prediction time in ms: ", (end_time – start_time) * 1000)

# Apply a sigmoid since our model returns logits
predictions = tf.nn.sigmoid(predictions)
predictions = tf.where(predictions < 0.5, 0, 1)

print('Predictions:\n', predictions.numpy())
print('Labels:\n', label_batch)

plt.figure(figsize=(10, 10))
for i in range(9):
  ax = plt.subplot(3, 3, i + 1)
  plt.imshow(image_batch[i].astype("uint8"))
  plt.title(class_names[predictions[i]])
  plt.axis("off")
```

9) Once the above changes are made, change the kernel by navigating to “Kernel” on the top pane, select “Change Kernel” and select “Python 3 (Tensorflow_OpenVINO 2021.4.1)”.
10) Next, click on “Cell” and click on “Run All”. This will run the notebook. In the notebook, transfer learning is performed using “Cats and Dogs” dataset using the Keras “MobileNetV2” model trained on ImageNet. After the model is trained, prediction is performed and the prediction time is printed after the last cell. The entire notebook will take a few minutes to finish.
11) For the model in Keras format, several variables are introduced such as tf dataset ops like TensorDataset. To run inference in an optimized fashion and to be able to convert the variables to constants for better graph optimizations, we recommend using the saved_model APIs. Add the below lines of code **in bold** and comment the lines in red in the last cell:

```
import time

model.save("saved_model")
imported = tf.saved_model.load("saved_model")
f = imported.signatures["serving_default"]

from tensorflow.python.framework.convert_to_constants import convert_varia
bles_to_constants_v2
f = convert_variables_to_constants_v2(f)

#Retrieve a batch of images from the test set
image_batch, label_batch = test_dataset.as_numpy_iterator().next()
#predictions = model.predict_on_batch(image_batch).flatten()
predictions = f(tf.convert_to_tensor(image_batch))
predictions = tf.reshape(predictions, [-1])

#Adding another instance of prediction to exclude the first warm up iterat
ion in measuring time
#Retrieve a batch of images from the test set
image_batch, label_batch = test_dataset.as_numpy_iterator().next()
start_time = time.time()
#predictions = model.predict_on_batch(image_batch).flatten()
predictions = f(tf.convert_to_tensor(image_batch))
predictions = tf.reshape(predictions, [-1])
end_time = time.time()
print(“Prediction time in ms: ”, (end_time – start_time) * 1000)

# Apply a sigmoid since our model returns logits
predictions = tf.nn.sigmoid(predictions)
predictions = tf.where(predictions < 0.5, 0, 1)

print('Predictions:\n', predictions.numpy())
print('Labels:\n', label_batch)

plt.figure(figsize=(10, 10))
for i in range(9):
  ax = plt.subplot(3, 3, i + 1)
  plt.imshow(image_batch[i].astype("uint8"))
  plt.title(class_names[predictions[i]])
  plt.axis("off")
```

12) Run just the last cell again, and you can see a slight improvement in the prediction time
13) Next add “import openvino_tensorflow” to the last cell as shown in bold below:

```
import time
import openvino_tensorflow

model.save("saved_model")
imported = tf.saved_model.load("saved_model")
f = imported.signatures["serving_default"]

from tensorflow.python.framework.convert_to_constants import convert_varia
bles_to_constants_v2
f = convert_variables_to_constants_v2(f)

#Retrieve a batch of images from the test set
image_batch, label_batch = test_dataset.as_numpy_iterator().next()
#predictions = model.predict_on_batch(image_batch).flatten()
predictions = f(tf.convert_to_tensor(image_batch))
predictions = tf.reshape(predictions, [-1])

#Adding another instance of prediction to exclude the first warm up iterat
ion in measuring time
#Retrieve a batch of images from the test set
image_batch, label_batch = test_dataset.as_numpy_iterator().next()
start_time = time.time()
#predictions = model.predict_on_batch(image_batch).flatten()
predictions = f(tf.convert_to_tensor(image_batch))
predictions = tf.reshape(predictions, [-1])
end_time = time.time()
print(“Prediction time in ms: ”, (end_time - start_time) * 1000)

# Apply a sigmoid since our model returns logits
predictions = tf.nn.sigmoid(predictions)
predictions = tf.where(predictions < 0.5, 0, 1)

print('Predictions:\n', predictions.numpy())
print('Labels:\n', label_batch)

plt.figure(figsize=(10, 10))
for i in range(9):
  ax = plt.subplot(3, 3, i + 1)
  plt.imshow(image_batch[i].astype("uint8"))
  plt.title(class_names[predictions[i]])
  plt.axis("off")
```

14) Rerun the last cell again with just addition of one line of code, and the prediction time goes down significantly because of OpenVINO optimizations.

Note: In the next release of OpenVINO™ integration with TensorFlow, we will enable all the variables_to_constants optimizations inside the stack. The users can access the optimizations without any changes to the APIs for TF2.x Keras predict() or TF 2.x saved_model.load() APIs.

Lab 2: Classification Sample with OpenVINO^{TM} integration with TensorFlow on Intel DevCloud:

1) Register and get an account on Intel DevCloud at https://software.intel.com/devcloud.
2) Go to https://devcloud.intel.com/edge and sign in.
3) Click on https://www.intel.com/content/www/us/en/developer/tools/devcloud/edge/build/ovtfoverview.html to check out the details of the samples hosted on DevCloud.
4) Scroll to the bottom of the page and click on “Classification Sample Application” in the table in the “Resources” section. This will open up a Jupyter notebook to run classification sample.
5) Verify that the kernel on the top right-hand corner is “Python 3 (Tensorflow_OpenVINO 2021.4.1)”.
6) Click on “Cell” and “Run All” to run the notebook.
7) The last cell of the notebook needs results written from the previous cells. Verify that all the jobs submitted to the edge nodes are completed and results are produced in “results” folder in the filesystem. Monitor the job status in Section 4.0.2. “liveQstat()” gives the status of the jobs and will be empty when all the submitted jobs finish execution.
8) Rerun the last cell after all the submitted jobs are finished. This will show a bar graph comparing the performance of stock TensorFlow, TensorFlow with OneDNN optimizations, OpenVINO integration with TensorFlow, and native OpenVINO.