OpenVino workflow - from training to inference

Using openvino has a lot of advantages, but simplicity is not one of them. I struggled to correctly save my network, optimise it and run an inference.

I'd like to guide you through doing it using neural network 'hello world' equivalent - building a model to recognise digits from mnist dataset.

Step number one - Virtual Machine - you really want it to play with OpenVino initially. I had to install OpenVino a few times to get everything right, not becouse it is so hard, just involved and looks like I cannot follow instructions without trying to be smarter (:-)).

First we need our model. A word of caution here - there is a long list of OpenVino versions and only the recent one (2020.4) supports TensorFlow 2.2, previous ones work only with TF up to 1.15 (for the sake of greater compatibility I will focus on TF 1.15)

Lets build the simplest model using tf.keras API, I'm using colab as it is an easy and free way to get access to GPU equipped machine.

```
%tensorflow_version 1.x # let's make sure we use supported tensorflow version, as default in tensorflow 2.x is now default in colab

import tensorflow as tf
from tensorflow.keras import layers
from tensorflow.keras import datasets
import numpy as np
```

```
(train_data, train_labels), (test_data, test_labels) = datasets.mnist.load_data()

train_data = train_data.astype(np.float32)
train_data /= 255
train_data = train_data.reshape(-1, 784)

classes = 10
labels = tensorflow.keras.utils.to_categorical(train_labels, classes)
```

```
model = models.Sequential()
model.add(layers.Dense(100, activation='relu', input_shape=(784, )))
model.add(layers.Dense(100, activation='relu'))
model.add(layers.Dense(10, activation='softmax))

model.compile(loss='categorical_crossentropy, optimizer='adam', metrics=['accuracy'])
model.fit(train_data, labels, epochs=10, batch_size = 32, validation_split=0.2)
```

Now as we have our model trained we have to save it. Using keras.save() doesn't work in OpenVino 2020.3 and earlier so we need to use some custom code to have our model in a format that is supported.

Below is the only way I managed to make my saved model work with OpenVino.

```
from tensorflow.keras import backend as K
sess = K.get_session()

def freeze_session(session, keep_var_names=None, output_names=None,
clear_devices=True):
     from tensorflow.python.framework.graph_util import convert_variables_to_constants
     graph = sess.graph
     with graph.as_default():
     freeze_var_names = list(set(v.op.name for v in
tf.global_variables()).difference(keep_var_names or []))
     output_names = output_names or []
     output_names += [v.op.name for v in tf.global_variables()]

     input_graph_def = graph.as_graph_def()
     if clear_devices:
     for node in input_graph_def.node:
     node.device = ""
     frozen_graph = convert_variables_to_constants(session, input_graph_def,
output_names, freeze_var_names)
     return frozen_graph

frozen_graph = freeze_session(K.get_session(), output_names=[out.op.name for out in
model.outputs])
tf.train.write_graph(frozen_graph, "model", "tf_model_2.pb", as_text=False)
```

Copy the model to your local machine, all next steps assume both the trained model and OpenVino are in VM on your local computer.

Now when we have a model saved in appropriate format, we can use OpenVino model optimizer (mo_tf.py script) to oprtimize it and also to convert it into intermediate representation(IR (xml and bin)) to be used be inference engine.

Run model optimizer in shell.

```
python3 mo_tf.py --input_model 'link_to_model.pb' --input_shape=[1,784]
```

Done - quite unexpectedly your optimized model is saved in

```
/opt/intel/openvino/deployment_tools/model_optimizer
```

Now what I expected is smooth ride with inference engine, but it is quite involved regarding code needed to make it work. Also there were a few counterintuitive choices made by developers which cost me some time.

First make sure you set up enviromental variables, run below in shell

```
source /opt/intel/openvino/bin/setupvars.sh
```

If you would like to be sure you got everything right, you may check if you can see the right python using below code.

```
import sys
```

```
for p in sys.path:
    print(p)
```

We are ready to code inference part. I suggest you do it in local jupyter notebook to be able to easier follow the process (as I wrote, it is not the simplest one)

```
import numpy as np
import cv2
from openvino.inference_engine import IECore
```

```
# path to optimized model files
model_xml = '/opt/intel/openvino/deployment_tools/model_optimizer/tf_model_2.xml'
model_bin = '/opt/intel/openvino/deployment_tools/model_optimizer/tf_model_2.bin'

# initialize inference engine
device = 'CPU'

ie = IECore()
net = ie.read_network(model=model_xml, weights=model_bin)
exec_net = ie.load_network(network=net, num_requests=1, device_name=device)

input_blob= next(iter(net.inputs)) # net.input_info doesnt work
output_blob = next(iter(net.outputs))
```

```
# prepare image for inference (transforming it to the right format)
def load_image(input_path):
    capture = cv2.VideoCapture(input_path)
    ret, image = capture.read()
    del capture
    return image

def prepare_image(image):
    resized = cv2.resize(image, (28, 28)) # resize image
    gray = cv2.cvtColor(resized, cv2.COLOR_BGR2GRAY) # change image to grayscale
    reshaped = gray.reshape((1,) + ((gray.shape[0]*gray.shape[1]),)) # reshape from
28X28 to 1X784 (input_shape of our model)
    flipped = (255 - reshaped) /255 # flip color values and rescale from 0-255 to 0-1
    return reshaped

input_path = '/home/vm/Documents/0.png'

image = load_image(input_path)
prepared_image = prepare_image(image)
```

```
# run inference
prediction = exec_net.infer(inputs={input_blob: prepared_image})
```

One more obstacle comes at the end - what infer() method does in OpenVino is equivalent to predict - means it just returns a vector of numbers:

```
prediction: {'dense_15/Softmax': array([[7.5168960e-02, 2.7967105e-06, 6.9963713e-03,
7.4688882e-01,
```

```
        2.7027925e-05, 2.4771986e-03, 9.7779557e-05, 1.7820160e-13,
        1.6834103e-01, 5.2042054e-10]], dtype=float32)}
```

We have multiclass model and so we need to implement some post processing to get a class answer. In the case of mnist below code will do.

```
for x in prediction.keys():
    print(np.argmax(prediction[x]))
```