ONNX Development Lab: Run ONNX ResNet in Tensorflow

Winnie Tsang, Chin Huang, Cognitive OpenTech

Data and AI
Open Source Dojo

Outline

- Unit 1: Setup and verify development dependencies
- Unit 2: Convert and Inspect ResNet model
- Unit 3: Run ResNet model

Unit 1: Setup development dependencies

- **Goal of the unit: At the end of unit 1, you will learn how to install and verify the projects that are required for the ONNX Tensorflow converter development**
- Development dependencies
    - System packages
    - ONNX master
    - Tensorflow 2.1
    - ONNX-TF master
- Step 1.1: Please make sure you have finished the setup and verified the dependencies in the earlier “Development Environment” session. If not, please go to https://github.com/chinhuang007/onnx-dojo/blob/master/docs/day1_afternoon/onnx_dev_env_dojo_2020.pdf and complete the setup.

Unit 2: Convert and inspect ResNet model

- **Goal of the unit: At the end of unit 2, you will learn how to visualize an ONNX model and convert it to a Tensorflow model using CLI and code**
- Step 2.1: Download ONNX model
    - Create a folder for lab “resnet” and we will do our exercises in the folder.
    - Download ResNet model for ResNet-152 version2 into “resnet” folder, find the right image, file name=resnet152-v2-7.onnx, from link, https://github.com/onnx/models/tree/master/vision/classification/resnet (ResNet-152, version 2)
    - Observe ResNet models following link above
        - Performs image classification, reformulate the layers as learning residual functions with reference to the layer inputs
        - Trained on ImageNet dataset which contains images from 1000 classes
        - ResNet-152 has 152 layers

Unit 2: Convert and inspect ResNet model

- Step 2.2: Visualize ONNX model
  - Netron, a model viewer that supports ONNX and TensorFlow, can be installed, but we will use the browser version, https://lutzroeder.github.io/netron/
  - Make the model file resnet152-v2-7.onnx available on the machine with a browser
  - Open model file resnet152-v2-7.onnx in Netron from browser and inspect the model as following
    - Click on menu->properties
    - Observe model inputs (an image): type=float32, shape=[1, 3, 224, 224]
    - Observe model outputs (scores for 1000 classes): type=float32, shape=[1, 1000]
    - Click on some nodes
    - Observe the node/operator name, type, inputs, outputs, and attributes
    - Observe the values for attributes and some inputs for this well trained model

Unit 2: Convert and inspect ResNet model

- Step 2.3: Use CLI to convert ONNX to Tensorflow
  - There are two ways to convert an ONNX to a Tensorflow computational graph file in protobuf format (pb) for inference, CLI "onnx-tf" and python code. We cover CLI in step 2.3 and python code in step 2.4
  - CLI onnx-tf takes a few arguments
    - Input file is resnet152-v2-7.onnx
    - Output file is resnet152v2.pb
    - Use the optional 'logging_level' argument to suppress the 'INFO' messages
    - Run '*onnx-tf convert -i resnet152-v2-7.onnx -o resnet152v2.pb --logging_level WARN*'
  - Use Netron to view the converted Tensorflow graph
    - Open the Tensorflow file resnet152v2.pb in Netron. Click Yes if prompted with "Large model detected".
    - Observe the graph structure and nodes

Unit 2: Convert and inspect ResNet model

- Step 2.4: Use python code to convert ONNX to Tensorflow
    - Now write short python code, **convert_model.py**, to convert resnet152-v2-7.onnx into Tensorflow pb file
    - Import onnx, and onnx_tf (using *import onnx*, and *import onnx_tf*)
    - Load the onnx model, using onnx.load() API
        - *model = onnx.load("resnet152-v2-7.onnx")*
    - Convert the model, using onnx-tf.backend.prepare() API
        - *tf_rep = onnx_tf.backend.prepare(model, logging_level="WARN")*
    - Print the inputs and outputs for the converted model
        - *print("inputs=", tf_rep.inputs)*
        - *print("outputs=", tf_rep.outputs)*
    - (optional) Print all tensors in the converted model
        - *print("tensor_dict=", tf_rep.tensor_dict)*

Unit 2: Convert and inspect ResNet model

- Save the Tensorflow graph as a pb file
  - *tf_rep.export_graph('./resnet152v2_frompython.pb')*
- Save and run the python code
  - *python convert_model.py* (ignore the warnings)
  - Observe the printed inputs and outputs.
  - (optional) Observe tensor_dict, which is long for a large model.
  - A new pb file is created. Not surprisingly identical as the file generated earlier using CLI!

Unit 3: Run ResNet model

- **Goal of the unit: At the end of unit 3, you will learn how to convert a ResNet model from ONNX to Tensorflow and run inference with sample data.**
- Step 3.1: Download data
    - Stay in the folder for lab “resnet”
    - Download index json file, https://github.com/USCDataScience/dl4j-kerasimport-examples/blob/master/dl4j-import-example/data/imagenet_class_index.json
        - Contains 1000 image class indices and names for ImageNet dataset
    - Copy sample data for inference from ~/onnx-dojo/lab_resnet/*.jpg
        - Two image files for lab exercise
    - (optional) Create your own images for inference following, https://github.com/onnx/models/tree/master/vision/classification/resnet#preprocessing

Unit 3: Run ResNet model

- Step 3.2: Observe the data
    - Take a look at the image files. Observe the different dimensions, colors, backgrounds, and sizes.
    - Open json file and search “ant” and “bee” for their class IDs (between 0-999)
- Step 3.3: Write code to convert ONNX to Tensorflow and run inference. In the live lab session, **please copy run_model.py from ~/onnx-dojo/lab_resnet to your resnet folder instead of type in the code in step 3.3**.
    - Now write another short python code, for ex. **run_model.py**, to convert resnet152-v2-7.onnx and run model with sample data
    - *import numpy, json, Tensorflow, onnx, and onnx_tf*
    - Load the onnx model, using onnx.load() API (same as step 2.4)
    - Convert the model, using onnx-tf.backend.prepare() API (same as step 2.4)

Unit 3: Run ResNet model

- Prepare the images and indices. Copy the following code:

```
images = ['ant.jpg', 'bee.jpg']
index_json_file='imagenet_class_index.json'
with open(index_json_file) as f:
  class_index = json.load(f)
def _central_crop(image, crop_height, crop_width):
  shape = tf.shape(image)
  height, width = shape[0], shape[1]
  crop_top = (height - crop_height) // 2
  crop_left = (width - crop_width) // 2
  image = tf.image.crop_to_bounding_box(image,
          crop_top, crop_left,
          crop_height, crop_width)
  return image
```

Unit 3: Run ResNet model

- Now for each image, we go though data pre-processing, run model, and print outputs in class ID and class name. Copy the following code.

```
for image_path in images:
  # load the image file, decode jpeg, and crop to the size 224x224
  img = tf.io.read_file(image_path)
  img = tf.image.decode_jpeg(img, channels=3)
  img = tf.image.convert_image_dtype(img, tf.float32)
  img = tf.image.resize(img, (256, 256))
  img = _central_crop(img, 224, 224)
  img = tf.transpose(img, perm=[2, 0, 1])
  img = tf.expand_dims(img, 0)
  # use numpy() to produce the python input
  input_image=img.numpy()
  # run the model with the processed image
  tf_output = tf_rep.run(input_image)
```

Unit 3: Run ResNet model

```
# use argmax to get the index/class ID with highest value in output
output = np.argmax(tf_output)
# print the input image file name
print('The image file is ', image_path)
# the output is the classification code
print('predicted class ID = ', output)
# the class name is coming from the index json file
print('predicted class name = ', class_index[str(output)][1])
```

Unit 3: Run ResNet model

- Save and run the python code
  - *python run_model.py* (ignore the warnings)
  - Observe the predicted results for our sample images, ant.jpg and bee.jpg, from executing the ResNet model in Tensorflow

Input: ant.jpg
Class id: 310
Class name: ant

Input: bee.jpg
Class id: 309
Class name: bee