Conversion of PyTorch Segmentation Models and Launch with OpenCV {#pytorch_segm_tutorial_dnn_conversion}

Goals

In this tutorial you will learn how to:

- convert PyTorch segmentation models
- run converted PyTorch model with OpenCV
- obtain an evaluation of the PyTorch and OpenCV DNN models

We will explore the above-listed points by the example of the FCN ResNet-50 architecture.

Introduction

The key points involved in the transition pipeline of the PyTorch classification and segmentation models with OpenCV API are equal. The first step is model transferring into ONNX format with PyTorch `torch.onnx.export` built-in function. Further the obtained `.onnx` model is passed into cv.dnn.readNetFromONNX, which returns cv.dnn.Net object ready for DNN manipulations.

Practice

In this part we are going to cover the following points:

1. create a segmentation model conversion pipeline and provide the inference
2. evaluate and test segmentation models

If you'd like merely to run evaluation or test model pipelines, the "Model Conversion Pipeline" part can be skipped.

Model Conversion Pipeline

The code in this subchapter is located in the `dnn_model_runner` module and can be executed with the line:

```
python -m dnn_model_runner.dnn_conversion.pytorch.segmentation.py_to_py_fcnresnet50
```

The following code contains the description of the below-listed steps:

1. instantiate PyTorch model
2. convert PyTorch model into `.onnx`
3. read the transferred network with OpenCV API
4. prepare input data
5. provide inference
6. get colored masks from predictions
7. visualize results

```
# initialize PyTorch FCN ResNet-50 model
original_model = models.segmentation.fcn_resnet50(pretrained=True)

# get the path to the converted into ONNX PyTorch model
```

```
full_model_path = get_pytorch_onnx_model(original_model)

# read converted .onnx model with OpenCV API
opencv_net = cv2.dnn.readNetFromONNX(full_model_path)
print("OpenCV model was successfully read. Layer IDs: \n",
opencv_net.getLayerNames())

# get preprocessed image
img, input_img = get_processed_imgs("test_data/sem_segm/2007_000033.jpg")

# obtain OpenCV DNN predictions
opencv_prediction = get_opencv_dnn_prediction(opencv_net, input_img)

# obtain original PyTorch ResNet50 predictions
pytorch_prediction = get_pytorch_dnn_prediction(original_model, input_img)

pascal_voc_classes, pascal_voc_colors = read_colors_info("test_data/sem_segm/pascal-
classes.txt")

# obtain colored segmentation masks
opencv_colored_mask = get_colored_mask(img.shape, opencv_prediction,
pascal_voc_colors)
pytorch_colored_mask = get_colored_mask(img.shape, pytorch_prediction,
pascal_voc_colors)

# obtain palette of PASCAL VOC colors
color_legend = get_legend(pascal_voc_classes, pascal_voc_colors)

cv2.imshow('PyTorch Colored Mask', pytorch_colored_mask)
cv2.imshow('OpenCV DNN Colored Mask', opencv_colored_mask)
cv2.imshow('Color Legend', color_legend)

cv2.waitKey(0)
```

To provide the model inference we will use the below picture from the PASCAL VOC validation dataset:


PASCAL VOC img

The target segmented result is:


PASCAL VOC ground truth

For the PASCAL VOC colors decoding and its mapping with the predicted masks, we also need `pascal-classes.txt` file, which contains the full list of the PASCAL VOC classes and corresponding colors.

Let's go deeper into each code step by the example of pretrained PyTorch FCN ResNet-50:

- instantiate PyTorch FCN ResNet-50 model:

```
# initialize PyTorch FCN ResNet-50 model
original_model = models.segmentation.fcn_resnet50(pretrained=True)
```

- convert PyTorch model into ONNX format:

```
# define the directory for further converted model save
onnx_model_path = "models"
# define the name of further converted model
onnx_model_name = "fcnresnet50.onnx"

# create directory for further converted model
os.makedirs(onnx_model_path, exist_ok=True)

# get full path to the converted model
full_model_path = os.path.join(onnx_model_path, onnx_model_name)

# generate model input to build the graph
generated_input = Variable(
    torch.randn(1, 3, 500, 500)
)

# model export into ONNX format
torch.onnx.export(
    original_model,
    generated_input,
    full_model_path,
    verbose=True,
    input_names=["input"],
    output_names=["output"],
    opset_version=11
)
```

The code from this step does not differ from the classification conversion case. Thus, after the successful execution of the above code, we will get `models/fcnresnet50.onnx`.

- read the transferred network with cv.dnn.readNetFromONNX passing the obtained in the previous step ONNX model into it:

```
# read converted .onnx model with OpenCV API
opencv_net = cv2.dnn.readNetFromONNX(full_model_path)
```

- prepare input data:

```
# read the image
input_img = cv2.imread(img_path, cv2.IMREAD_COLOR)
input_img = input_img.astype(np.float32)

# target image sizes
img_height = input_img.shape[0]
```

```
img_width = input_img.shape[1]

# define preprocess parameters
mean = np.array([0.485, 0.456, 0.406]) * 255.0
scale = 1 / 255.0
std = [0.229, 0.224, 0.225]

# prepare input blob to fit the model input:
# 1. subtract mean
# 2. scale to set pixel values from 0 to 1
input_blob = cv2.dnn.blobFromImage(
    image=input_img,
    scalefactor=scale,
    size=(img_width, img_height),  # img target size
    mean=mean,
    swapRB=True,  # BGR -> RGB
    crop=False  # center crop
)
# 3. divide by std
input_blob[0] /= np.asarray(std, dtype=np.float32).reshape(3, 1, 1)
```

In this step we read the image and prepare model input with cv2.dnn.blobFromImage function, which returns 4-dimensional blob. It should be noted that firstly in `cv2.dnn.blobFromImage` mean value is subtracted and only then pixel values are scaled. Thus, `mean` is multiplied by `255.0` to reproduce the original image preprocessing order:

```
img /= 255.0
img -= [0.485, 0.456, 0.406]
img /= [0.229, 0.224, 0.225]
```

- OpenCV `cv.dnn_Net` inference:

```
# set OpenCV DNN input
opencv_net.setInput(preproc_img)

# OpenCV DNN inference
out = opencv_net.forward()
print("OpenCV DNN segmentation prediction: \n")
print("* shape: ", out.shape)

# get IDs of predicted classes
out_predictions = np.argmax(out[0], axis=0)
```

After the above code execution we will get the following output:

```
OpenCV DNN segmentation prediction:
* shape:  (1, 21, 500, 500)
```

Each prediction channel out of 21, where 21 represents the number of PASCAL VOC classes, contains probabilities, which indicate how likely the pixel corresponds to the PASCAL VOC class.

- PyTorch FCN ResNet-50 model inference:

```
original_net.eval()
preproc_img = torch.FloatTensor(preproc_img)

with torch.no_grad():
    # obtaining unnormalized probabilities for each class
    out = original_net(preproc_img)['out']

print("\nPyTorch segmentation model prediction: \n")
print("* shape: ", out.shape)

# get IDs of predicted classes
out_predictions = out[0].argmax(dim=0)
```

After the above code launching we will get the following output:

```
PyTorch segmentation model prediction:
* shape:  torch.Size([1, 21, 366, 500])
```

PyTorch prediction also contains probabilities corresponding to each class prediction.

- get colored masks from predictions:

```
# convert mask values into PASCAL VOC colors
processed_mask = np.stack([colors[color_id] for color_id in segm_mask.flatten()])

# reshape mask into 3-channel image
processed_mask = processed_mask.reshape(mask_height, mask_width, 3)
processed_mask = cv2.resize(processed_mask, (img_width, img_height),
interpolation=cv2.INTER_NEAREST).astype(
    np.uint8)

# convert colored mask from BGR to RGB for compatibility with PASCAL VOC colors
processed_mask = cv2.cvtColor(processed_mask, cv2.COLOR_BGR2RGB)
```

In this step we map the probabilities from segmentation masks with appropriate colors of the predicted classes. Let's have a look at the results:


OpenCV Colored Mask

For the extended evaluation of the models, we can use `py_to_py_segm` script of the `dnn_model_runner` module. This module part will be described in the next subchapter.

Evaluation of the Models

The proposed in `dnn/samples` `dnn_model_runner` module allows to run the full evaluation pipeline on the PASCAL VOC dataset and test execution for the following PyTorch segmentation models:

- FCN ResNet-50

- FCN ResNet-101

This list can be also extended with further appropriate evaluation pipeline configuration.

Evaluation Mode

The below line represents running of the module in the evaluation mode:

```
python -m dnn_model_runner.dnn_conversion.pytorch.segmentation.py_to_py_segm --
model_name <pytorch_segm_model_name>
```

Chosen from the list segmentation model will be read into OpenCV `cv.dnn_Net` object. Evaluation results of PyTorch and OpenCV models (pixel accuracy, mean IoU, inference time) will be written into the log file. Inference time values will be also depicted in a chart to generalize the obtained model information.

Necessary evaluation configurations are defined in the `test_config.py` :

```
@dataclass
class TestSegmConfig:
    frame_size: int = 500
    img_root_dir: str = "./VOC2012"
    img_dir: str = os.path.join(img_root_dir, "JPEGImages/")
    img_segm_gt_dir: str = os.path.join(img_root_dir, "SegmentationClass/")
    # reduced val:
https://github.com/shelhamer/fcn.berkeleyvision.org/blob/master/data/pascal/seg11valid

    segm_val_file: str = os.path.join(img_root_dir,
"ImageSets/Segmentation/seg11valid.txt")
    colour_file_cls: str = os.path.join(img_root_dir,
"ImageSets/Segmentation/pascal-classes.txt")
```

These values can be modified in accordance with chosen model pipeline.

To initiate the evaluation of the PyTorch FCN ResNet-50, run the following line:

```
python -m dnn_model_runner.dnn_conversion.pytorch.segmentation.py_to_py_segm --
model_name fcnresnet50
```

Test Mode

The below line represents running of the module in the test mode, which provides the steps for the model inference:

```
python -m dnn_model_runner.dnn_conversion.pytorch.segmentation.py_to_py_segm --
model_name <pytorch_segm_model_name> --test True --default_img_preprocess <True/False>
--evaluate False
```

Here `default_img_preprocess` key defines whether you'd like to parametrize the model test process with some particular values or use the default values, for example, `scale` , `mean` or `std` .

Test configuration is represented in `test_config.py` `TestSegmModuleConfig` class:

```
@dataclass
class TestSegmModuleConfig:
```

```
    segm_test_data_dir: str = "test_data/sem_segm"
    test_module_name: str = "segmentation"
    test_module_path: str = "segmentation.py"
    input_img: str = os.path.join(segm_test_data_dir, "2007_000033.jpg")
    model: str = ""

    frame_height: str = str(TestSegmConfig.frame_size)
    frame_width: str = str(TestSegmConfig.frame_size)
    scale: float = 1.0
    mean: List[float] = field(default_factory=lambda: [0.0, 0.0, 0.0])
    std: List[float] = field(default_factory=list)
    crop: bool = False
    rgb: bool = True
    classes: str = os.path.join(segm_test_data_dir, "pascal-classes.txt")
```

The default image preprocessing options are defined in `default_preprocess_config.py`:

```
pytorch_segm_input_blob = {
    "mean": ["123.675", "116.28", "103.53"],
    "scale": str(1 / 255.0),
    "std": ["0.229", "0.224", "0.225"],
    "crop": "False",
    "rgb": "True"
}
```

The basis of the model testing is represented in `samples/dnn/segmentation.py`. `segmentation.py` can be executed autonomously with provided converted model in `--input` and populated parameters for `cv2.dnn.blobFromImage`.

To reproduce from scratch the described in "Model Conversion Pipeline" OpenCV steps with `dnn_model_runner` execute the below line:

```
python -m dnn_model_runner.dnn_conversion.pytorch.segmentation.py_to_py_segm --
model_name fcnresnet50 --test True --default_img_preprocess True --evaluate False
```