# Conversion of TensorFlow Segmentation Models and Launch with OpenCV {#tf_segm_tutorial_dnn_conversion}

## Goals

In this tutorial you will learn how to:

- convert TensorFlow (TF) segmentation models
- run converted TensorFlow model with OpenCV
- obtain an evaluation of the TensorFlow and OpenCV DNN models

We will explore the above-listed points by the example of the DeepLab architecture.

## Introduction

The key concepts involved in the transition pipeline of the [TensorFlow classification]() and segmentation models with OpenCV API are almost equal excepting the phase of graph optimization. The initial step in conversion of TensorFlow models into cv.dnn.Net is obtaining the frozen TF model graph. Frozen graph defines the combination of the model graph structure with kept values of the required variables, for example, weights. Usually the frozen graph is saved in [protobuf]() ( `.pb` ) files. To read the generated segmentation model `.pb` file with cv.dnn.readNetFromTensorflow, it is needed to modify the graph with TF [graph transform tool]().

## Practice

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

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

If you'd like merely to run evaluation or test model pipelines, the "Model Conversion Pipeline" tutorial 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.tf.segmentation.py_to_py_deeplab
```

TensorFlow segmentation models can be found in [TensorFlow Research Models]() section, which contains the implementations of models on the basis of published research papers. We will retrieve the archive with the pre-trained TF DeepLabV3 from the below link:

```
http://download.tensorflow.org/models/deeplabv3_mnv2_pascal_trainval_2018_01_29.tar.gz
```

The full frozen graph obtaining pipeline is described in `deeplab_retrievement.py` :

```
def get_deeplab_frozen_graph():
    # define model path to download
    models_url = 'http://download.tensorflow.org/models/'
```

```python
    mobilenetv2_voctrainval = 'deeplabv3_mnv2_pascal_trainval_2018_01_29.tar.gz'

    # construct model link to download
    model_link = models_url + mobilenetv2_voctrainval

    try:
        urllib.request.urlretrieve(model_link, mobilenetv2_voctrainval)
    except Exception:
        print("TF DeepLabV3 was not retrieved: {}".format(model_link))
        return

    tf_model_tar = tarfile.open(mobilenetv2_voctrainval)

    # iterate the obtained model archive
    for model_tar_elem in tf_model_tar.getmembers():
        # check whether the model archive contains frozen graph
        if TF_FROZEN_GRAPH_NAME in os.path.basename(model_tar_elem.name):
            # extract frozen graph
            tf_model_tar.extract(model_tar_elem, FROZEN_GRAPH_PATH)

    tf_model_tar.close()
```

After running this script:

```
python -m dnn_model_runner.dnn_conversion.tf.segmentation.deeplab_retrievement
```

we will get `frozen_inference_graph.pb` in `deeplab/deeplabv3_mnv2_pascal_trainval`.

Before going to the network loading with OpenCV it is needed to optimize the extracted `frozen_inference_graph.pb`. To optimize the graph we use TF `TransformGraph` with default parameters:

```python
DEFAULT_OPT_GRAPH_NAME = "optimized_frozen_inference_graph.pb"
DEFAULT_INPUTS = "sub_7"
DEFAULT_OUTPUTS = "ResizeBilinear_3"
DEFAULT_TRANSFORMS = "remove_nodes(op=Identity)" \
                     " merge_duplicate_nodes" \
                     " strip_unused_nodes" \
                     " fold_constants(ignore_errors=true)" \
                     " fold_batch_norms" \
                     " fold_old_batch_norms"


def optimize_tf_graph(
        in_graph,
        out_graph=DEFAULT_OPT_GRAPH_NAME,
        inputs=DEFAULT_INPUTS,
        outputs=DEFAULT_OUTPUTS,
        transforms=DEFAULT_TRANSFORMS,
        is_manual=True,
        was_optimized=True
):
```

```
    # ...

    tf_opt_graph = TransformGraph(
        tf_graph,
        inputs,
        outputs,
        transforms
    )
```

To run graph optimization process, execute the line:

```
python -m dnn_model_runner.dnn_conversion.tf.segmentation.tf_graph_optimizer --
in_graph deeplab/deeplabv3_mnv2_pascal_trainval/frozen_inference_graph.pb
```

As a result `deeplab/deeplabv3_mnv2_pascal_trainval` directory will contain `optimized_frozen_inference_graph.pb` .

After we have obtained the model graphs, let's examine the below-listed steps:

1. read TF `frozen_inference_graph.pb` graph
2. read optimized TF frozen graph with OpenCV API
3. prepare input data
4. provide inference
5. get colored masks from predictions
6. visualize results

```
# get TF model graph from the obtained frozen graph
deeplab_graph = read_deeplab_frozen_graph(deeplab_frozen_graph_path)

# read DeepLab frozen graph with OpenCV API
opencv_net = cv2.dnn.readNetFromTensorflow(opt_deeplab_frozen_graph_path)
print("OpenCV model was successfully read. Model layers: \n",
opencv_net.getLayerNames())

# get processed image
original_img_shape, tf_input_blob, opencv_input_img =
get_processed_imgs("test_data/sem_segm/2007_000033.jpg")

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

# obtain TF model predictions
tf_prediction = get_tf_dnn_prediction(deeplab_graph, tf_input_blob)

# get PASCAL VOC classes and colors
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(original_img_shape, opencv_prediction,
pascal_voc_colors)
```

```
tf_colored_mask = get_tf_colored_mask(original_img_shape, tf_prediction,
pascal_voc_colors)

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

cv2.imshow('TensorFlow Colored Mask', tf_colored_mask)
cv2.imshow('OpenCV DNN Colored Mask', opencv_colored_mask)

cv2.imshow('Color Legend', color_legend)
```

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 step by the example of pretrained TF DeepLabV3 MobileNetV2:

- read TF `frozen_inference_graph.pb` graph :

```
# init deeplab model graph
model_graph = tf.Graph()

# obtain
with tf.io.gfile.GFile(frozen_graph_path, 'rb') as graph_file:
    tf_model_graph = GraphDef()
tf_model_graph.ParseFromString(graph_file.read())

with model_graph.as_default():
    tf.import_graph_def(tf_model_graph, name='')
```

- read optimized TF frozen graph with OpenCV API:

```
# read DeepLab frozen graph with OpenCV API
opencv_net = cv2.dnn.readNetFromTensorflow(opt_deeplab_frozen_graph_path)
```

- prepare input data with cv2.dnn.blobFromImage function:

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

# preprocess image for TF model input
```

```
tf_preproc_img = cv2.resize(input_img, (513, 513))
tf_preproc_img = cv2.cvtColor(tf_preproc_img, cv2.COLOR_BGR2RGB)

# define preprocess parameters for OpenCV DNN
mean = np.array([1.0, 1.0, 1.0]) * 127.5
scale = 1 / 127.5

# 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=(513, 513),  # img target size
    mean=mean,
    swapRB=True,  # BGR -> RGB
    crop=False  # center crop
)
```

Please, pay attention at the preprocessing order in the `cv2.dnn.blobFromImage` function. Firstly, the mean value is subtracted and only then pixel values are multiplied by the defined scale. Therefore, to reproduce TF image preprocessing pipeline, we multiply `mean` by `127.5`. Another important point is image preprocessing for TF DeepLab. To pass the image into TF model we need only to construct an appropriate shape, the rest image preprocessing is described in [feature_extractor.py](feature_extractor.py) and will be invoked automatically.

- provide 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, 513, 513)
```

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.

- provide TF model inference:

```
preproc_img = np.expand_dims(preproc_img, 0)

# init TF session
```

```
tf_session = Session(graph=model_graph)

input_tensor_name = "ImageTensor:0",
output_tensor_name = "SemanticPredictions:0"

# run inference
out = tf_session.run(
    output_tensor_name,
    feed_dict={input_tensor_name: [preproc_img]}
)

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

TF inference results are the following:

```
TF segmentation model prediction:
* shape:  (1, 513, 513)
```

TensorFlow prediction contains the indexes of corresponding PASCAL VOC classes.

- transform OpenCV prediction into colored mask:

```
mask_height = segm_mask.shape[0]
mask_width = segm_mask.shape[1]

img_height = original_img_shape[0]
img_width = original_img_shape[1]

# 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
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:

Color Legend

OpenCV Colored Mask

- transform TF prediction into colored mask:

```
colors = np.array(colors)
processed_mask = colors[segm_mask[0]]

img_height = original_img_shape[0]
img_width = original_img_shape[1]

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)
```

The result is:

TF Colored Mask

As a result, we get two equal segmentation masks.

### 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 DeepLab MobileNet model.

#### Evaluation Mode

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

```
python -m dnn_model_runner.dnn_conversion.tf.segmentation.py_to_py_segm
```

The model will be read into OpenCV `cv.dnn_Net` object. Evaluation results of TF 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`](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.

**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.tf.segmentation.py_to_py_segm --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`](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`:

```
tf_segm_input_blob = {
    "scale": str(1 / 127.5),
    "mean": ["127.5", "127.5", "127.5"],
    "std": [],
    "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.tf.segmentation.py_to_py_segm --test True --
default_img_preprocess True --evaluate False
```