ONNX Pre-processing WG

Monthly meeting - Oct 13, 2021

Agenda

- The problem (recap)
- Operator support
- Proof of concept (ResNet 50)
- Roadmap (Proposal)
- Data preprocessing within ONNX model

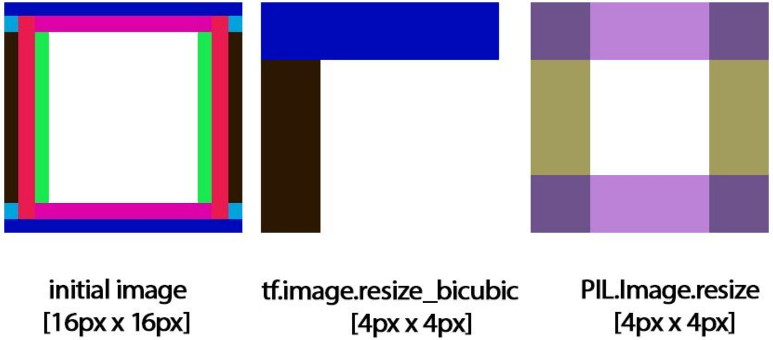
The problem (recap)

- Lack of standardized preprocessing primitives, portability issues

Bilinear filter – OpenCV vs Pillow

Pillow's "bilinear" interpolation is actually triangular

Bicubic filter – TensorFlow 1 vs Pillow



Different definition of pixel centers when resampling

https://hackernoon.com/how-tensorflows-tf-image-resize-stole-60-days-of-my-life-aba5eb093f35

- Hard to deploy pre-trained models to optimized runtimes
    - Often implemented in Python, with libraries such as Pillow, Numpy, OpenCV

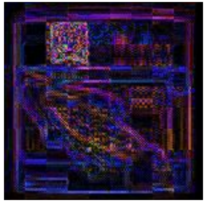
Operator support

- Vision networks:
    - Classification: ResNet, ResNext
    - Detection & Segmentation: SSD, Mask R-CNN
    - … ?
- Operators:
    - Resize: Extend
        - Interpolation type {nearest-neighbor, bilinear, bicubic, **triangular**, **lanczos**}
        - Resize policy {stretch, not-larger, not-smaller}
    - Color space conversion: New operator
        - RGB to BGR, etc
    - Slice: OK
    - Cast: OK
    - Normalize (Sub/Div): OK
    - Transpose: OK
    - Pad: OK
    - Shape: OK
    - Image decoder: New operator
        - JPEG decoding is not well defined
        - Out of scope for first prototype?

Libjpeg-turbo:

"Fancy upsampling" ON (default) vs OFF

abs_diff * 10



Proof of concept - ResNet 50

```
from PIL import Image

def preprocess(image):
    # resize so that the shorter side is 256, maintaining aspect ratio
    def image_resize(image, min_len):
        image = Image.fromarray(image)
        ratio = float(min_len) / min(image.size[0], image.size[1])
        if image.size[0] > image.size[1]:
            new_size = (int(round(ratio * image.size[0])), min_len)
        else:
            new_size = (min_len, int(round(ratio * image.size[1])))
        image = image.resize(new_size, Image.BILINEAR)
        return np.array(image)
    image = image_resize(image, 256)

    # Crop centered window 224x224
    def crop_center(image, crop_w, crop_h):
        h, w, c = image.shape
        start_x = w//2 - crop_w//2
        start_y = h//2 - crop_h//2
        return image[start_y:start_y+crop_h, start_x:start_x+crop_w, :]
    image = crop_center(image, 224, 224)

    # transpose
    image = image.transpose(2, 0, 1)

    # convert the input data into the float32 input
    img_data = image.astype('float32')

    # normalize
    mean_vec = np.array([0.485, 0.456, 0.406])
    stddev_vec = np.array([0.229, 0.224, 0.225])
    norm_img_data = np.zeros(img_data.shape).astype('float32')
    for i in range(img_data.shape[0]):
        norm_img_data[i,:,:] = (img_data[i,:,:]/255 - mean_vec[i]) / stddev_vec[i]

    # add batch channel
    norm_img_data = norm_img_data.reshape(1, 3, 224, 224).astype('float32')
    return norm_img_data
```

Preprocessing is typically defined in Python

```
session = onnxruntime.InferenceSession('resnet50v2/resnet50v2.onnx', None)
input_data = preprocess(image_data)
raw_result = session.run([], {input_name: input_data})
```

Proof of concept - ResNet 50

Second ONNX model for preprocessing

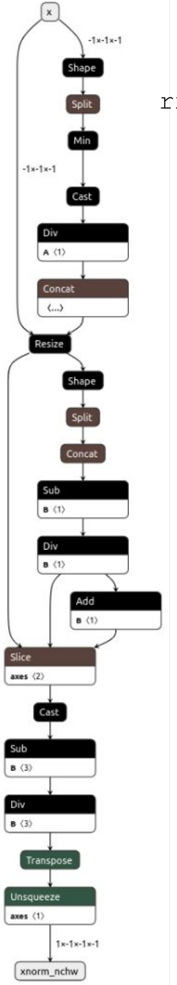
rn50-preprocessing.onnx

```python
preprocessing = onnxruntime.InferenceSession('rn50-preprocessing.onnx', None)
session = onnxruntime.InferenceSession('resnet50v2/resnet50v2.onnx', None)
input_data = preprocessing_session.run([], {'x': np.array(image_data)})[0]
raw_result = session.run([], {input_name: input_data})
```

Input image

np.abs(data_onnx - data_numpy) * 255

Small artifacts due to different implementations



Roadmap (proposal)

- Make data preprocessing part of ONNX
  - Data preprocessing to be distributed with the ONNX model
  - Easy to deploy
- Standardize definition of pre-processing primitives
  - Portable across implementations
  - Focus on vision networks first
  - Extend to other data domains later (e.g. audio)
  - Extend operator support to cover most popular networks

Data preprocessing within ONNX model

- Where to add preprocessing? Proposal:
    - Separate graph within the model

```
graph = make_graph(...)
preprocessing_graph = make_graph(...)
model = make_model(graph, preprocessing=preprocessing_graph, ...)
```

- Implicit batchification? Proposal:
    - Preprocessing graph is defined for one sample.
    - Batch dimension is added implicitly (unsqueeze for one sample, or actual batch formation)

```
preprocessed = session.preprocess([raw_input1, raw_input2, ...])  # Produces a batch of processed samples
raw_result = session.run([], {'data': preprocessed})
```