Extending MXNet Model save/load APIs

Problem

Taking a MXNet model from training and deploying it in production poses several challenges to the users. Most important problems raised by users are:

1. **Input/Output data transformations** are not part of the MXNet model.
2. **Input/Output signature** are not part of the model: Saved model missing the information about the input/output descriptions, like name/shape, making the saved model unusable out of the box.
3. **File naming** - Epoch number is part of model file name. This information may not be necessary for production deployment.

Proposed Solution

1. In Gluon, update the Export API to accept input/output signature, pre-processing/post-processing transformations (Hybrid Sequential Block) from the users.
2. If user provides the transformations, a fused graph (Transformations + Batchify_Connector_Node + Network + No_Op_Connector_Node + Transformations) is prepared. Batchify_Connector_Node and No_Op_Connector_Node are MXNet internal identifier operators to help in identifying transformations and network graphs separately. This identifier nodes will be helpful, if when loading the model, user prefers to not load the transformations.
3. Input/Output signature are added to the symbol file.

Example symbol file after proposed update update. **"inputs"** and **"outputs"** are the new parameters proposed in this work.

```
{
    "nodes": [..........]
    "arg_nodes": [0, 1, 2, 4, 5],
    "node_row_ptr": [0, 1, 2, 3, 4, 5, 6, 7, 8],
    "heads": [[7, 0, 0]],
    "attrs": {"mxnet_version": ["int", 10301]
              "inputs": {"data":[1,3,224,224]},
              "outputs" : {"softmax_label":[1,10]
              }
}
```

Possible use cases

Inputs and outputs are dictionary where key => Name of the node and value => shape. Though single input, single output models are more common, a model can be of following variations:

1. Single input, single output
2. Single input, multiple outputs
3. Multiple input with same or different transformation, single output
4. Multiple input with same or different transformation, multiple output

User Experience

Before

Step 1 - Train and export the model from Gluon

```
# Trained network
net = mx.gluon.model_zoo.vision.resnet18_v1(pretrained=True, ctx=mx.cpu())

# Data transformations applicable during inference
inference_input_transforms = gluon.nn.HybridSequential()
inference_input_transforms.add(transforms.Resize((224, 224)))
inference_input_transforms.add(transforms.ToTensor())
inference_input_transforms.add(transforms.Normalize(mean=(0.485, 0.456, 0.406), std=(0.229, 0.224, 0.225)))

# Export the model. Cannot export data transformation and input/output signature
net.export(path="./my_model", epoch=0)
```

Step 2 - Import the model for inference with Module OR Gluon OR JAVA

Import the model in Gluon

```
# Load the model. Model does not contain transformation and input/output signature
net = SymbolBlock.imports(symbol_file="my_model-symbol.json",
                          input_names=["data"],
                          param_file="my_model-0000.params",
                          ctx=mx.cpu())
```

Import the model in Module API

```
sym, arg_params, aux_params = mx.model.load_checkpoint('my_model', 0)
mod = mx.mod.Module(symbol=sym, context=ctx, label_names=None)
mod.bind(for_training=False, data_shapes=[('data', (1,3,224,224))],
         label_shapes=mod._label_shapes)
mod.set_params(arg_params, aux_params, allow_missing=True)

mod.forward(...)
```

Import the model in Java Predictor API

```
Shape inputShape = new Shape(new int[] {1,3,224,224});
DataDesc inputDescriptor = new DataDesc("data", inputShape, DType.Float32(), "NCHW");
List<DataDesc> inputDescList = new ArrayList<DataDesc>();
inputDescList.add(inputDescriptor);
List<Context> context = new ArrayList<>();
context.add(Context.cpu());
String modelPathPrefix = "path-to-model";
Predictor predictor = new Predictor(modelPathPrefix, inputDescList, context);

List<NDArray> result = predictor.predictWithNDArray(inputNDArray);
```

After

Step 1 - Train and export the model from Gluon

```
# Trained network
net = mx.gluon.model_zoo.vision.resnet18_v1(pretrained=True, ctx=mx.cpu())

# Data transformations applicable during inference
inference_input_transforms = gluon.nn.HybridSequential()
inference_input_transforms.add(transforms.Resize((224, 224)))
inference_input_transforms.add(transforms.ToTensor())
inference_input_transforms.add(transforms.Normalize(mean=(0.485, 0.456, 0.406), std=(0.229, 0.224, 0.225)))

# Export the model
gluon.contrib.utils.export(net, path="./my_model",
                           epoch=0,
                           signature={constants.INPUT_DESC:[("data", (1,3,224,224))],
                                      constants.OUTPUT_DESC:[("softmax_label", (1,10))]},
                           input_transforms={"data":inference_input_transforms},
                           output_transforms=None)
```

Step 2 - Import the model for inference with Module OR Gluon OR JAVA

Import the Model in Gluon

```
# Load the model along with the transformations
net = gluon.contrib.utils.import(symbol_file="my_model-symbol.json",
                                            param_file="my_model-0000.params",
                                            load_transforms = True,
                                            ctx = 'cpu')


# Prediction
pred = net(data)
```

Import the Model in Module API

(Supported to create a module for inference only)

```
# Load the model along with the transformations. Can be used only for inference (forward())
mod = mx.contrib.Module.import(
                symbol_file = "my_model-symbol.json",
                param_file = "my_model-0000.params",
                load_transforms = True,
                ctx = 'cpu',
                batch_size = 1)

# Prediction
mod.forward(...)
```

Import the Model in Java Predictor API

```
List<Context> context = new ArrayList<>();
context.add(Context.cpu());
String modelPathPrefix = "my_model";


# Load the model along with the transformations.
Predictor predictor = new Predictor(modelPathPrefix, context, load_transforms=True);

# Inference
List<NDArray> result = predictor.predictWithNDArray(inputNDArray);
```

Performance Consideration

During inference, initial benchmarks shows a noticeable performance gain with End to end models i.e., a model with data transformations and neural network all fused as a single model graph.

- ResNet-18 model pre-trained with ImageNet. https://s3.us-east-2.amazonaws.com/mxnet-public/end_to_end_models
- Pre-processing - Resize(224, 224), ToTensor, Normalize(mean=(0.485, 0.456, 0.406), std=(0.229, 0.224, 0.225))
- We take average of 500 runs
- Single Request Inference - Input Data - Synthetic (random.uniform(0, 255, shape=(1, 300, 300, 3))
- Batch Inference - Input Data - Synthetic (random.uniform(0, 255, shape=(25, 300, 300, 3))
- Below time gives - **Average Prediction Time Per Sample**

A	B	C	Non End to End Models (ms)	End to End Models (ms)	Boost %
CPU (C5.2X)	Single Request Inference	Python (Module API)	17	14	17.65%
		Java Inference APIs	17.09	14.16	17.14%
		Scala Inference APIs	17.93	13.19	26.44%
	Batch Inference (Batch size = 25)	Python (Module API)	15.18	12.57	17.19%
		Java Inference APIs	18.54	13	29.88%
		Scala Inference APIs	17	13.26	22.00%

GPU (P3.16X)	Single Request Inference	Python (Module API)	5.78	3.14	45.67%
		Java Inference APIs	8.95	4.26	52.40%
		Scala Inference APIs	9.14	4.42	51.64%
	Batch Inference (Batch size = 25)	Python (Module API)	2.61	1.31	49.81%
		Java Inference APIs	8.03	5.53	31.13%
		Scala Inference APIs	7.86	5.52	29.77%

Backward compatibility

1. All APIs changes are backward compatible.
2. Old MXNet model should still load without breakage with new MXNet version.
3. New MXNet model will not work on old MXNet versions. If a user tries to load new MXNet model with older MXNet version, they get errors such as - "unknown field 'inputs', 'outputs' in the model" because, MXNet's model JSON parser schema in old MXNet do not understand new fields introduced as part of this work.

Open Questions and Assumptions Made

1. Here we assume internal operators as **no op connector node**. We assume these connector nodes are not exposed to the users and hence, we can safely use as connector nodes for fusing transformations and neural network graph. Is this assumption in the right direction?
2. Is this ok to go to Contrib first and then update main APIs? Below are the reasoning:
    a. API changes / additions in this work is proposed to go in to Contrib because "export" and "import" APIs are most widely used by every MXNet users. We propose to mature the API, address any usability or design concerns while in contrib before graduating to main APIs.
    b. What is the plan to monitor and move this change from contrib to main? Once we build these extensions/new export, import APIs, we will be creating new tutorials and update the existing export/import tutorials. We will track issues and discussion for any user concerns. We will graduate these APIs after maturing in Contrib for at least 1 version of MXNet release.

Alternate Solution

1. Keep transformation graph and network graph separately independent of each other and fuse them at run time.
    a. In the proposed approach, we fuse the transformation and neural network and export as single graph. We introduce a no_op_identifier operator to identify the link between transformation and neural network.
    b. In this alternate solution, we would be to keep the transformation and network graph separately (in same symbol file or multiple symbol file). These independent graphs can then be fused during run time.
    c. Two concerns for this approach:
        i. In MXNet, a symbol and a param are foundational building block of representing 1 graph and its parameters. It may not be ideal to keep multiple graphs in the same symbol file and params file.
        ii. If we decide to keep transformations and network graphs as separate files, this can be still achieved today with current MXNet APIs without any additional changes.

References

1. Gluon-CV export helper - https://gluon-cv.mxnet.io/api/utils.html?highlight=export#gluoncv.utils.export_block
2. https://mxnet.incubator.apache.org/versions/master/tutorials/python/predict_image.html?highlight=load_checkpoint
3. https://medium.com/apache-mxnet/introducing-java-apis-for-deep-learning-inference-with-apache-mxnet-8406a698fa5a