But it works on my machine?!

ONNX101

Aleksandar Ivanovski
aleksandar@ivanovski.me

Challenges

- Multiple frameworks
  - Tensorflow
  - PyTorch
- Training on different accelerators
- Inference on various hardware
  - T4 GPUs
  - VPUs

ONNX

- Abstraction and intermediary over frameworks and hardware
- Minimizes the number of moving parts
- Started by AWS, Microsoft and Meta
- Defined format for
  - Deep Learning (DL) models
  - The operators used

Design Principles

- Supports DL and traditional ML
- Flexible enough to keep up with rapid advances
- Compact and cross-platform representation for serialization
- Standardized list of well-defined operators

"ONNX is to Deep Learning what Common Language Runtime is for programming languages"

Linux Foundation AI (LFAI)

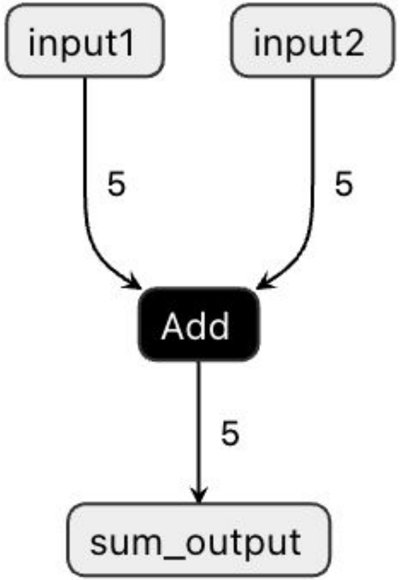
- New chapter of Linux Foundation (LF) aimed to drive compatibility and interoperability for AI projects
- Part of LFAI since 2019

ONNX Specification

- Deep Learning (ONNX) - the original
- ONNX-ML aimed for traditional ML (scikit)

ONNX format

- Every computational graph is a direct acyclic graph
- Nodes:
  - Input (known shape)
  - Output (known shape) -> input to the next node
- Operator - essence of the node
- Metadata documents the graph
  - How the model was produced
  - On which accelerator
  - etc…



ONNX Data Types

- Tensor types
  - int8,int16,int32,int64
  - uint8,uint16,uint32,uint64
  - float16,float,double
  - bool
  - string
  - Complex64,complex128
- Non-tensor types in ONNX-ML
  - Sequence
  - Map

ONNX Operators

- Operator is defined by <name, domain, version>
- Operator sets:
  - ai.onnx
  - ai.onnx.ml

Operator Examples - Relu

Takes one input data (Tensor) and produces one output data (Tensor) where the rectified linear function, y = max(0, x), is applied to the tensor element wise.

Version: This version of the operator has been available since version 14 of the ai.onnx operator set.

Other versions of this operator: 1, 6, 13

Inputs: X (differentiable): T *(input tensor)*

Outputs: Y (differentiable): T *(output tensor)*

Type Constraints: T: *tensor - float, int32, int8, int16, int64, float16, double, bfloat16*

Operator Examples - SVMRegressor

Support Vector Machine regression prediction and one-class SVM anomaly detection.

Version: This version of the operator has been available since version 1 of the ai.onnx.ml operator set.

Inputs: T: Data to be regressed.

Outputs: Y (tensor): Regression outputs (one score per target per example).

Attributes: coefficients, kernel_params, kernel_type, n_supports, one_class, …

Type Constraints: T: *tensor - float, double, int64, int32*

Demo 1

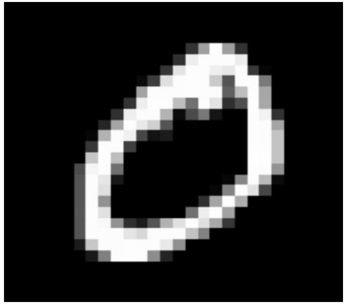
- Define a model in PyTorch
- Train the model with PyTorch
- Export to ONNX
- Visualize the graph

Problem Description

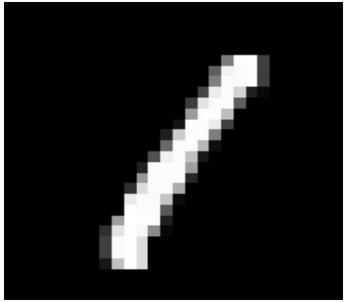
- Train a classifier on the MNIST dataset

2d: 28x28

1d: 1x10

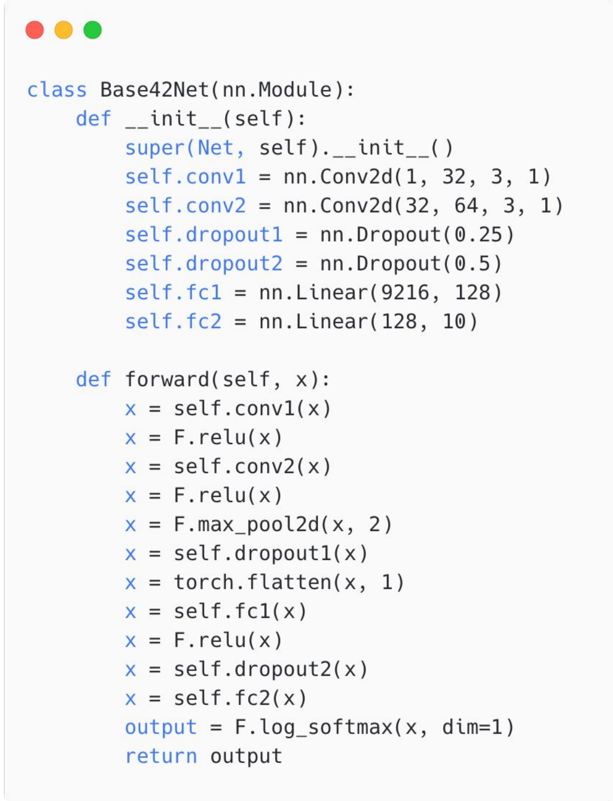


Class: 0

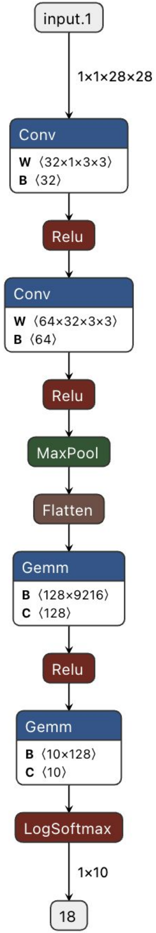


Class: 1

Solution: CNN with PyTorch



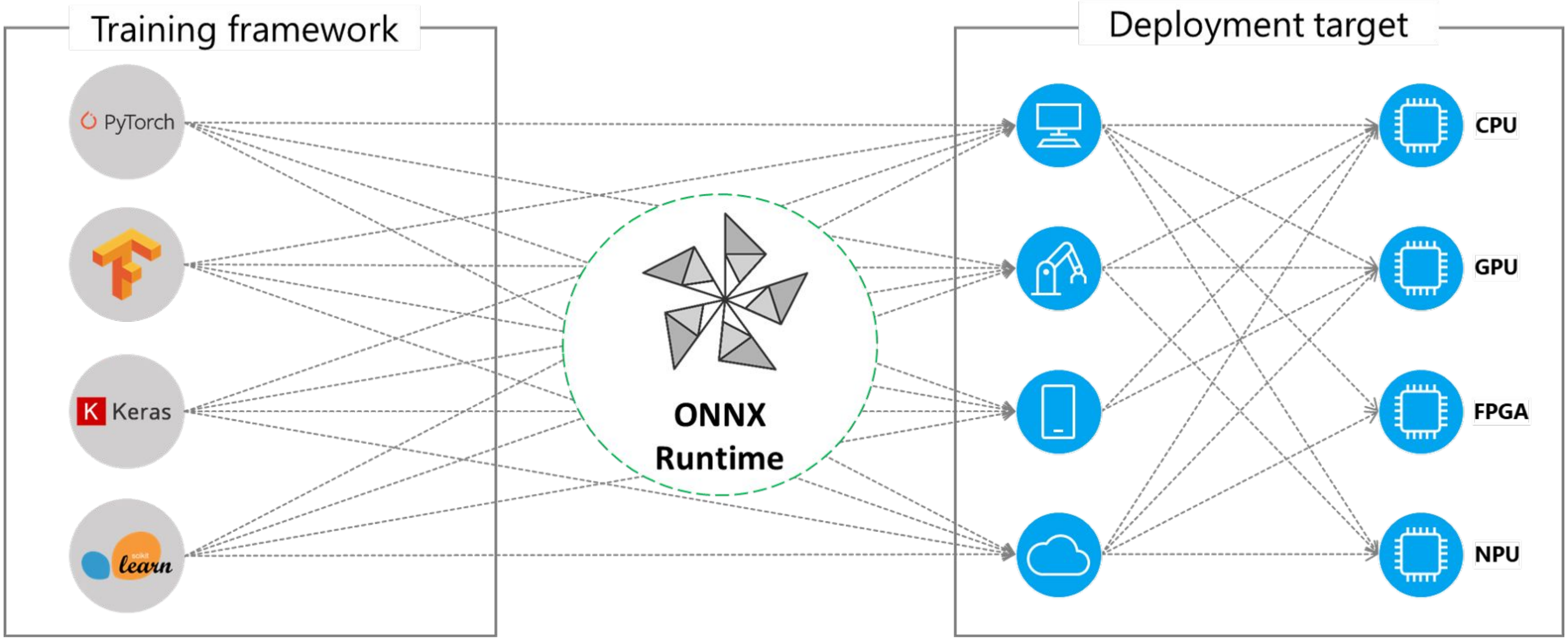
```python
class Base42Net(nn.Module):
    def __init__(self):
        super(Net, self).__init__()
        self.conv1 = nn.Conv2d(1, 32, 3, 1)
        self.conv2 = nn.Conv2d(32, 64, 3, 1)
        self.dropout1 = nn.Dropout(0.25)
        self.dropout2 = nn.Dropout(0.5)
        self.fc1 = nn.Linear(9216, 128)
        self.fc2 = nn.Linear(128, 10)

    def forward(self, x):
        x = self.conv1(x)
        x = F.relu(x)
        x = self.conv2(x)
        x = F.relu(x)
        x = F.max_pool2d(x, 2)
        x = self.dropout1(x)
        x = torch.flatten(x, 1)
        x = self.fc1(x)
        x = F.relu(x)
        x = self.dropout2(x)
        x = self.fc2(x)
        output = F.log_softmax(x, dim=1)
        return output
```



ONNX Runtime

- ONNX - file-format and the graph in the file
- ONNX Runtime - how to plug the graph on different hardware
- Created and Maintained by Microsoft

Interoperability



Getting ONNX Models

- ONNX Model Zoo
- Azure Cognitive Services
- Convert Existing models
- Train from the ground up

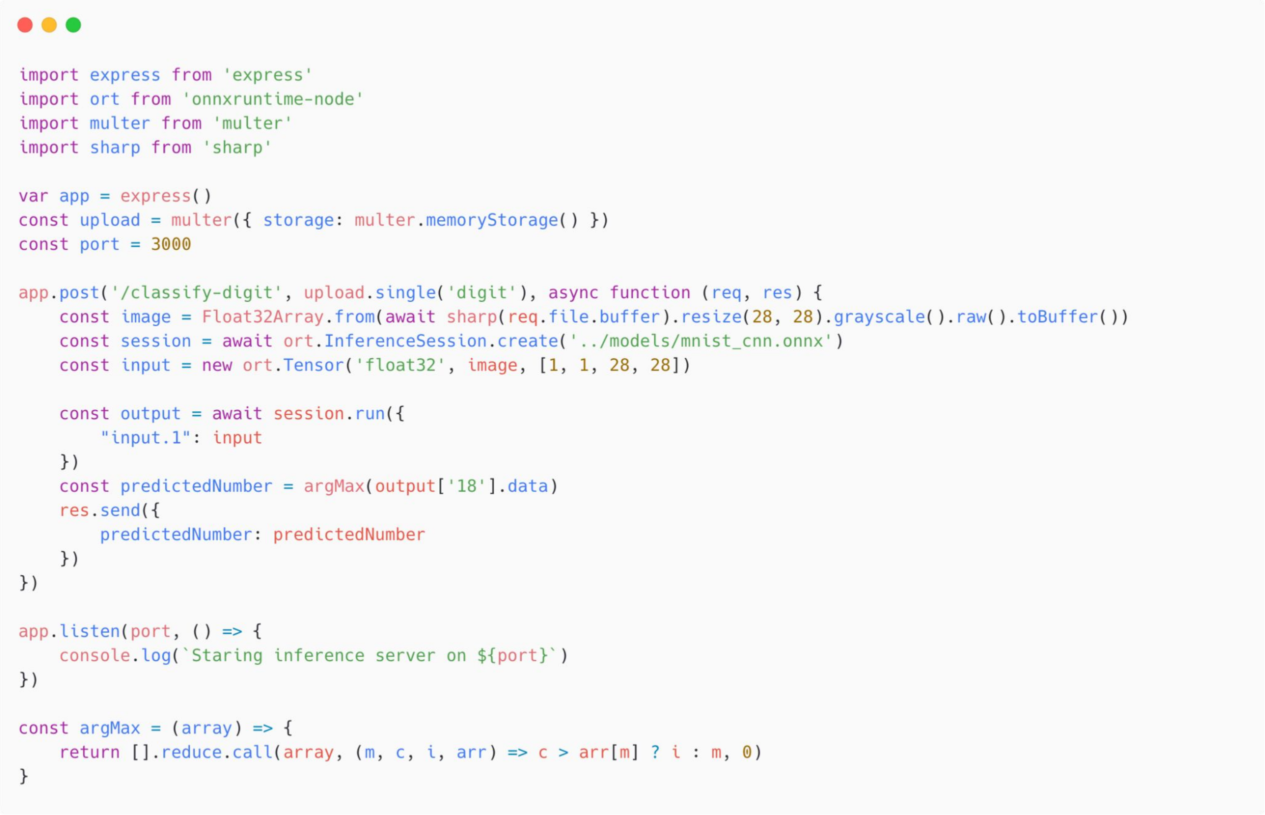
Tools

- Netron
- VisualDL

Demo 2

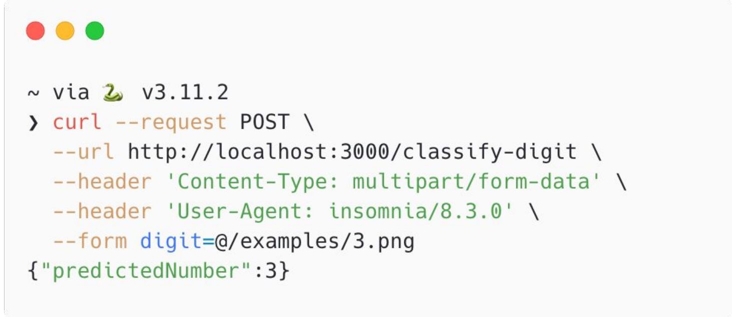
- Use the model from Demo 1
- Use onnx-runtime in JS
- Build inference REST API

Express.js inference server



```
import express from 'express'
import ort from 'onnxruntime-node'
import multer from 'multer'
import sharp from 'sharp'

var app = express()
const upload = multer({ storage: multer.memoryStorage() })
const port = 3000

app.post('/classify-digit', upload.single('digit'), async function (req, res) {
    const image = Float32Array.from(await sharp(req.file.buffer).resize(28, 28).grayscale().raw().toBuffer())
    const session = await ort.InferenceSession.create('../models/mnist_cnn.onnx')
    const input = new ort.Tensor('float32', image, [1, 1, 28, 28])

    const output = await session.run({
        "input.1": input
    })
    const predictedNumber = argMax(output['18'].data)
    res.send({
        predictedNumber: predictedNumber
    })
})

app.listen(port, () => {
    console.log(`Staring inference server on ${port}`)
})

const argMax = (array) => {
    return [].reduce.call(array, (m, c, i, arr) => c > arr[m] ? i : m, 0)
}
```

Express.js inference server



```
~ via 🐍 v3.11.2
❯ curl --request POST \
  --url http://localhost:3000/classify-digit \
  --header 'Content-Type: multipart/form-data' \
  --header 'User-Agent: insomnia/8.3.0' \
  --form digit=@/examples/3.png
{"predictedNumber":3}
```

Q&A