DEEP LEARNING FRAMEWORKS

Contents

- Introduction to Deep Learning Frameworks
- Tensorflow & Keras
- Pytorch
- Other frameworks & Interoperability

Introduction

Deep Learning Frameworks

- Set of tools to train neural networks
- Optimized code
- Parallel processing (CPUs, GPUs, TPUs)
- Community support (new architectures, datasets, …)

Choosing a Framework

- Today, most of them have the same capabilities.
- Ease of use (understanding what you are doing)
- Ecosystem (research, development, production)
- Target application (web, server, IoT, …)
- Accessibility and flexibility for professionals from other fields.

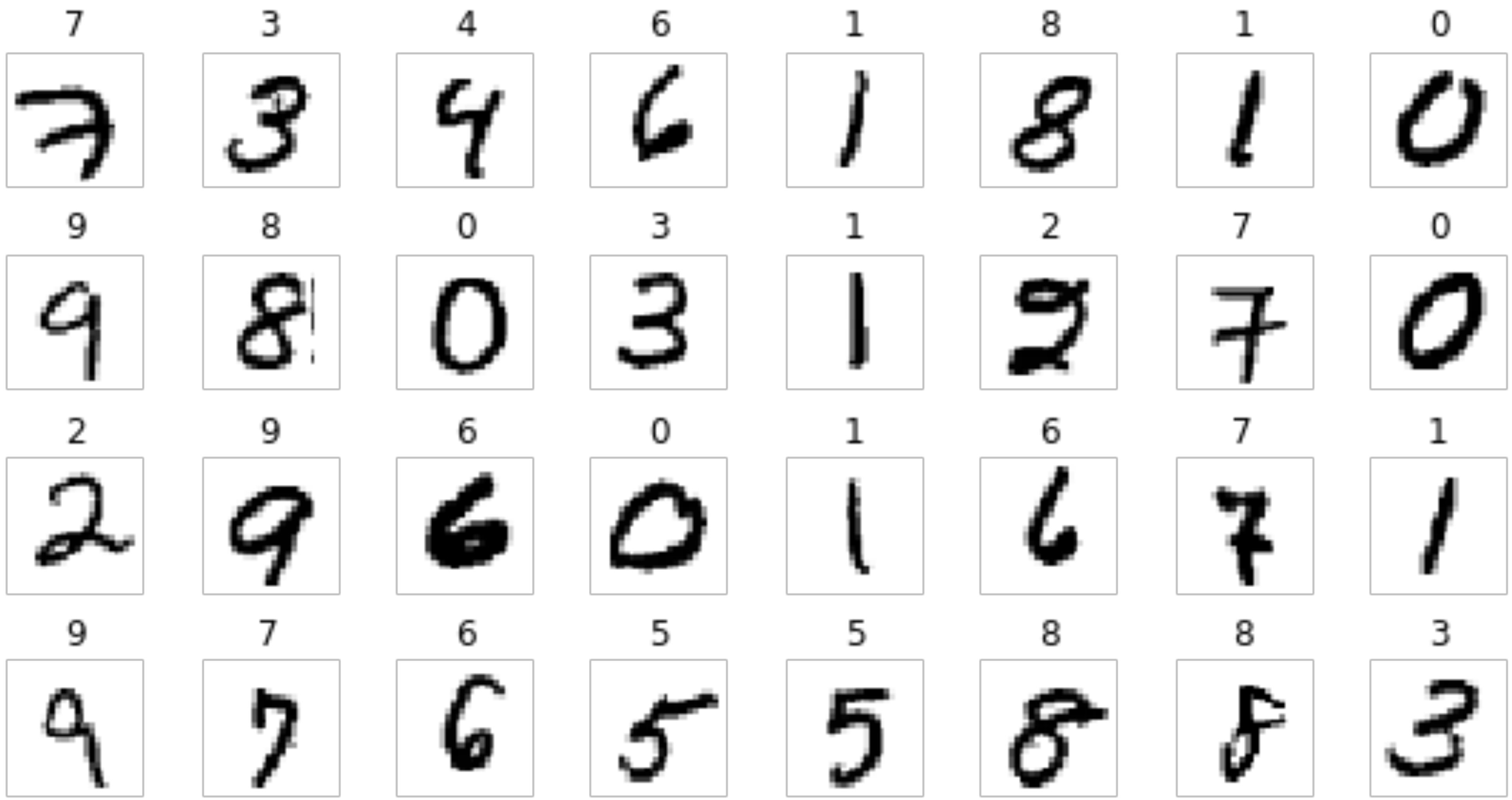
Tensorflow

TensorFlow

- Most used framework in the industry
- Developed by Google
- High-level API (Keras)
- Low-level API (TF2.0)
- Great production support (web, mobile, servers, IoT, …)

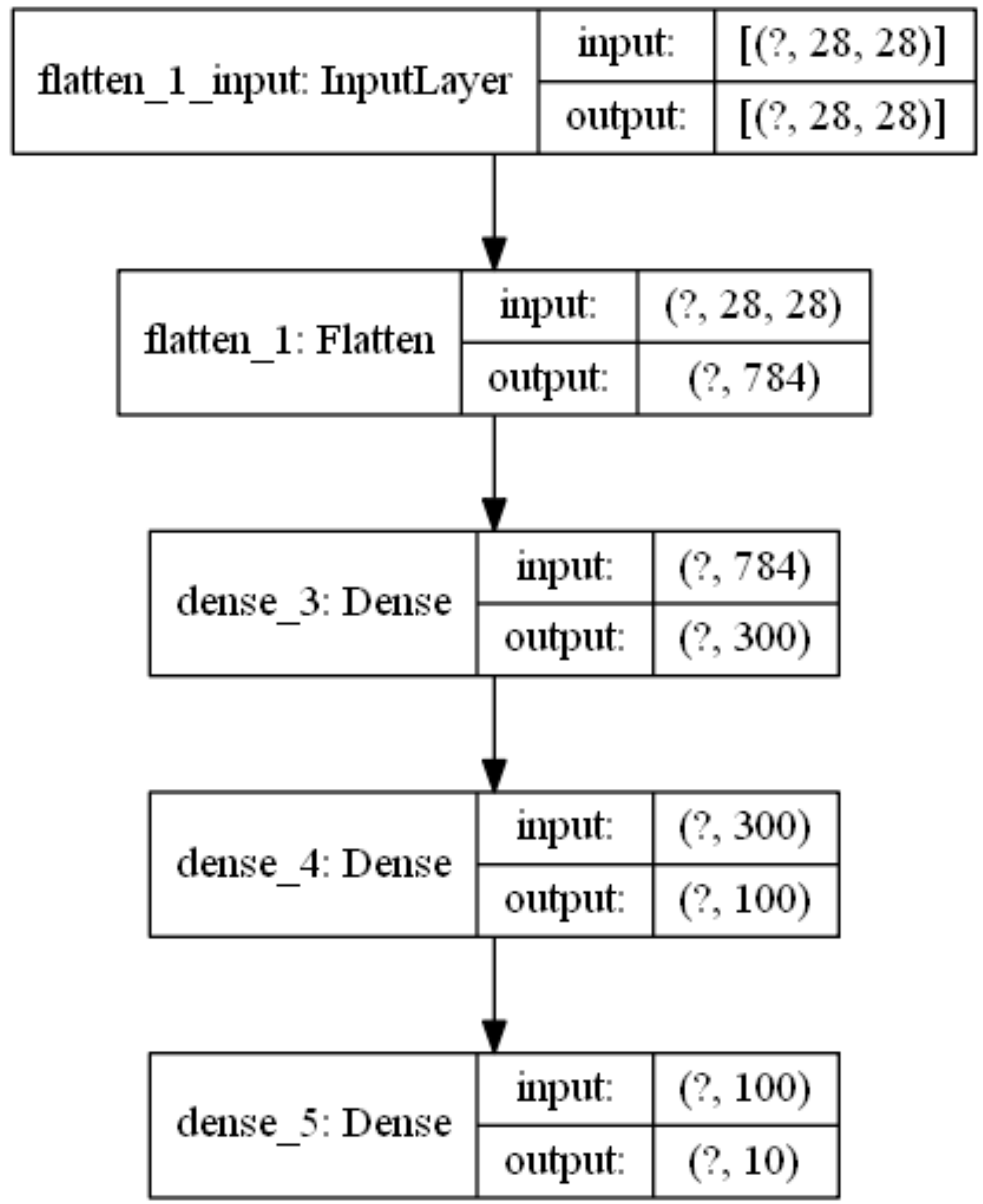
Keras

```
# import tensorflow and keras

import tensorflow as tf
from tensorflow import keras


# download a dataset

mnist = keras.datasets.mnist
(X_train_full, y_train_full), (X_test, y_test) = mnist.load_data()


# split and normalize


X_valid, X_train = X_train_full[:5000] / 255., X_train_full[5000:] / 255.
y_valid, y_train = y_train_full[:5000], y_train_full[5000:]
X_test = X_test / 255.
```

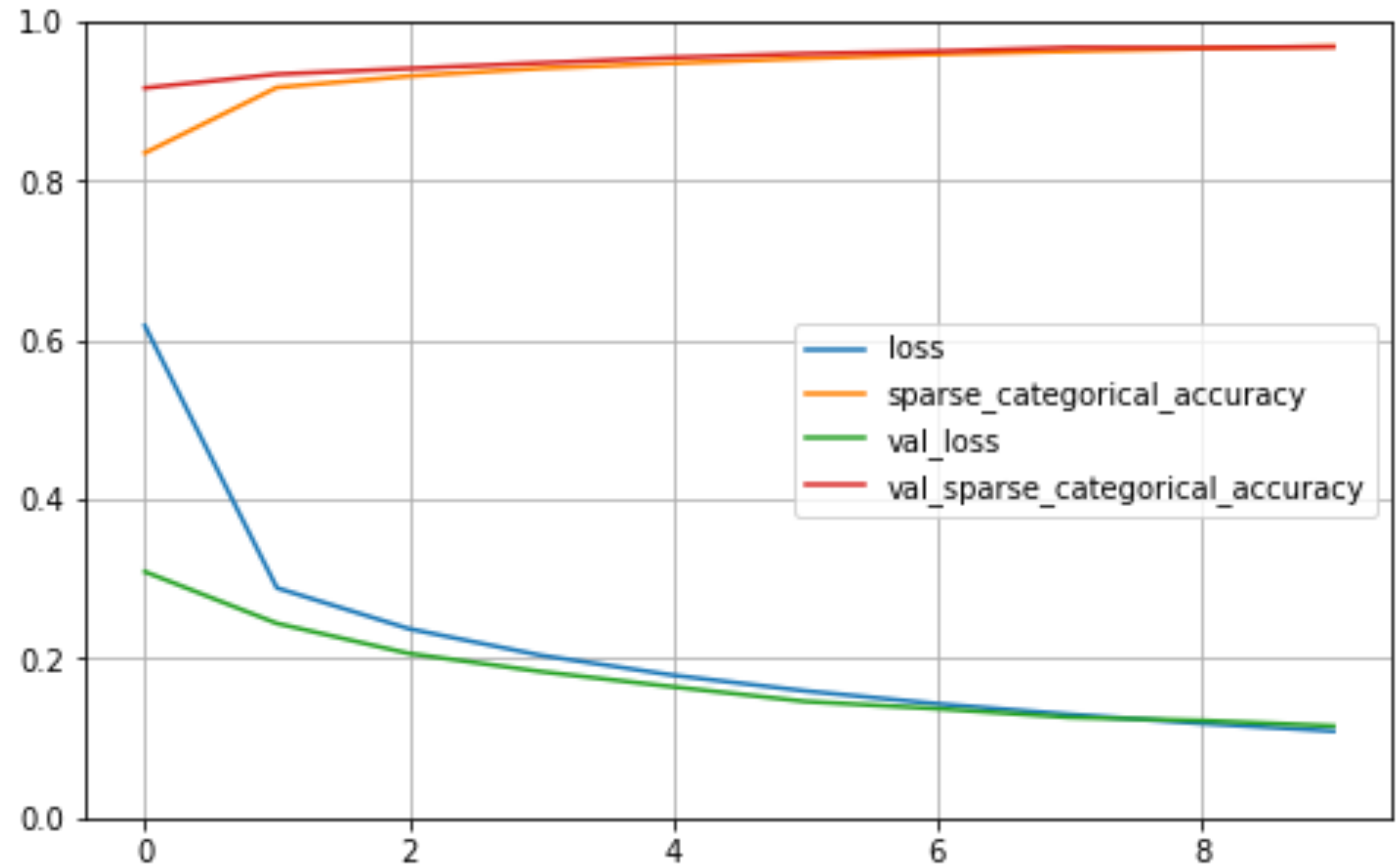

7 3 4 6 1 8 1 0
9 8 0 3 1 2 7 0
2 9 6 0 1 6 7 1
9 7 6 5 5 8 8 3

```python
# build a model

model = keras.models.Sequential([
    keras.layers.Flatten(input_shape=[28, 28]),
    keras.layers.Dense(300, activation="relu"),
    keras.layers.Dense(100, activation="relu"),
    keras.layers.Dense(10, activation="softmax")
])

model.summary()
```

```
Model: "sequential_1"
_________________________________________________________________
Layer (type)                 Output Shape              Param #
=================================================================
flatten_1 (Flatten)          (None, 784)               0
_________________________________________________________________
dense_3 (Dense)              (None, 300)               235500
_________________________________________________________________
dense_4 (Dense)              (None, 100)               30100
_________________________________________________________________
dense_5 (Dense)              (None, 10)                1010
=================================================================
Total params: 266,610
Trainable params: 266,610
Non-trainable params: 0
_________________________________________________________________
```


flatten_1_input: InputLayer
input:
[(?, 28, 28)]
output:
[(?, 28, 28)]
flatten_1: Flatten
input:
(?, 28, 28)
output:
(?, 784)
dense_3: Dense
input:
(?, 784)
output:
(?, 300)
dense_4: Dense
input:
(?, 300)
output:
(?, 100)
dense_5: Dense
input:
(?, 100)
output:
(?, 10)

```
# compile model

model.compile(loss="sparse_categorical_crossentropy",
              optimizer="sgd",
              metrics=[“accuracy”])

# train the model

history = model.fit(X_train, y_train, epochs=10, validation_data=(X_valid, y_valid))


# evaluate the model

model.evaluate(X_test, y_test)

# get some predictions

y_proba = model.predict(X_new)
```


1.0
0.8
0.6
0.4
0.2
0.0
0
2
4
6
8
loss
sparse_categorical_accuracy
val_loss
val_sparse_categorical_accuracy

Callbacks

- Callbacks allow us to modify the default behavior of the training loop.

```
# save best model during training

checkpoint_cb = keras.callbacks.ModelCheckpoint("my_keras_model.h5",
save_best_only=True)

history = model.fit(X_train, y_train, epochs=10,
                    validation_data=(X_valid, y_valid),
                    callbacks=[checkpoint_cb])

# rollback to best model

model = keras.models.load_model("my_keras_model.h5")
```

```python
# custom callbacks

class PrintValTrainRatioCallback(keras.callbacks.Callback):
    def on_epoch_end(self, epoch, logs):
        print("\nval/train: {:.2f}".format(logs["val_loss"] / logs["loss"]))


val_train_ratio_cb = PrintValTrainRatioCallback()

history = model.fit(X_train, y_train, epochs=3,
                    validation_data=(X_valid, y_valid),
                    callbacks=[val_train_ratio_cb])
```

```
Epoch 1/3
280/363 [======================>.......] - ETA: 0s - loss: 0.3830
val/train: 0.94
363/363 [==============================] - 0s 752us/step - loss: 0.3913 - val_loss: 0.3677
Epoch 2/3
356/363 [============================>.] - ETA: 0s - loss: 0.3927
val/train: 0.94
363/363 [==============================] - 0s 782us/step - loss: 0.3903 - val_loss: 0.3676
Epoch 3/3
274/363 [=====================>........] - ETA: 0s - loss: 0.3905
val/train: 0.96
363/363 [==============================] - 0s 749us/step - loss: 0.3890 - val_loss: 0.3744
```

Keras

- Great high level API with abstraction
- Attractive for less experienced practitioners
- Useful for people outside the CS field
- Tensorflow ecosystem (hardware, deployment, …)
- Some flexibility if necessary, but for non-standards problems we may need to go lower level -> Tensorflow !

Tensorflow

```
from tensorflow.keras.layers import Dense, Flatten, Conv2D
from tensorflow.keras import Model

# Build the dataset

train_ds = tf.data.Dataset.from_tensor_slices(
    (x_train, y_train)).shuffle(10000).batch(32)

test_ds = tf.data.Dataset.from_tensor_slices((x_test, y_test)).batch(32)

#  optimizer, loss and metrics

loss_object = tf.keras.losses.SparseCategoricalCrossentropy()

optimizer = tf.keras.optimizers.Adam()

train_loss = tf.keras.metrics.Mean(name='train_loss')
train_accuracy = tf.keras.metrics.SparseCategoricalAccuracy(name='train_accuracy')

test_loss = tf.keras.metrics.Mean(name='test_loss')
test_accuracy = tf.keras.metrics.SparseCategoricalAccuracy(name=‘test_accuracy')
```

```python
# Define a network

class MyModel(Model):
  def __init__(self):
    super(MyModel, self).__init__()
    self.conv1 = Conv2D(32, 3, activation='relu')
    self.flatten = Flatten()
    self.d1 = Dense(128, activation='relu')
    self.d2 = Dense(10, activation='softmax')

  def call(self, x):
    x = self.conv1(x)
    x = self.flatten(x)
    x = self.d1(x)
    return self.d2(x)

model = MyModel()
```

```python
@tf.function
def train_step(images, labels):
  with tf.GradientTape() as tape:
    predictions = model(images)
    loss = loss_object(labels, predictions)
  gradients = tape.gradient(loss, model.trainable_variables)
  optimizer.apply_gradients(zip(gradients, model.trainable_variables))

  train_loss(loss)
  train_accuracy(labels, predictions)

@tf.function
def test_step(images, labels):
  predictions = model(images)
  t_loss = loss_object(labels, predictions)

  test_loss(t_loss)
  test_accuracy(labels, predictions)
```

```python
# train

EPOCHS = 5

for epoch in range(EPOCHS):
  for images, labels in train_ds:
    train_step(images, labels)

  for test_images, test_labels in test_ds:
    test_step(test_images, test_labels)

  template = 'Epoch {}, loss: {}, acc: {}, test_loss: {}, test_acc: {}'
  print(template.format(epoch+1,
                        train_loss.result(),
                        train_accuracy.result()*100,
                        test_loss.result(),
                        test_accuracy.result()*100))

  # reset metrics for next epoch.
  train_loss.reset_states()
  train_accuracy.reset_states()
  test_loss.reset_states()
  test_accuracy.reset_states()
```

Tensorflow

- Low level API (ideal for research)
- Need to define your own dataset
- Need to define your own training loop
- More control and flexibility
- Great ecosystem and production ready tools.
- Still some details that make it confusing sometimes…

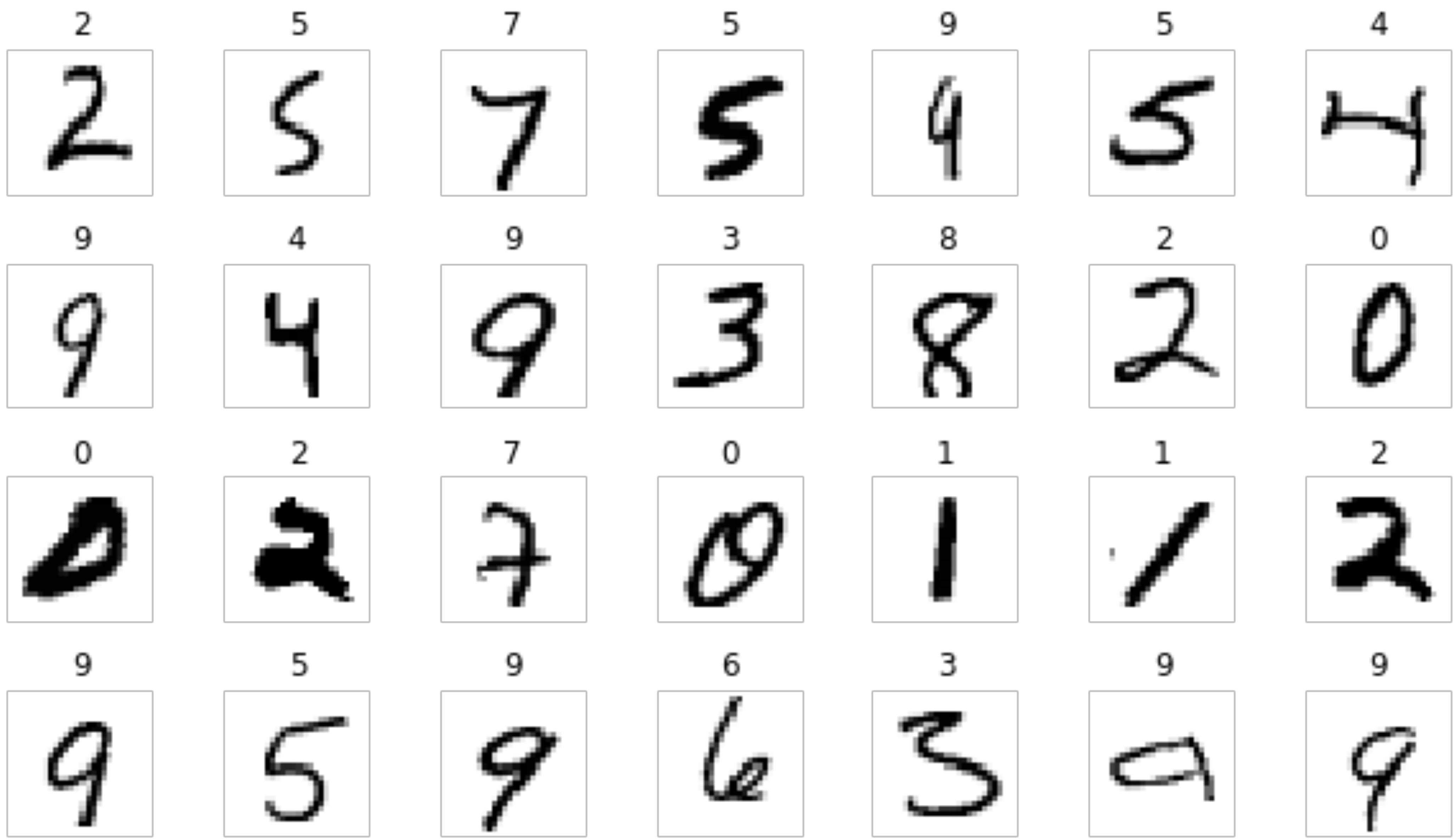

ME
TENSORFLOW
PYTORCH

PyTorch

- Most used framework in research and competitions
- Developed by Facebook
- No official High-level API (like Keras), but a good option is Fast.ai (plus really good education content).
- Low level API
- Catching up on the production side (torchscript, torch serve, …)

Pytorch

```
# import pytorch

import torch
import torchvision

# download data

import torchvision.transforms as transforms

train_dataset = torchvision.datasets.MNIST('./data', train=True, download=True,
                   transform=transforms.Compose([
                           transforms.ToTensor(),
                           transforms.Normalize((0.1307,), (0.3081,))
                       ]))

test_dataset = torchvision.datasets.MNIST('./data', train=False,
                   transform=transforms.Compose([
                        transforms.ToTensor(),
                        transforms.Normalize((0.1307,), (0.3081,))
                    ]))

train_loader = torch.utils.data.DataLoader(train_dataset, batch_size=32,
shuffle=True)
test_loader = torch.utils.data.DataLoader(test_dataset, batch_size=10000)
```



2 5 7 5 9 5 4
9 4 9 3 8 2 0
0 2 7 0 1 1 2
9 5 9 6 3 9 9

```python
# define the network

model = torch.nn.Sequential(
  torch.nn.Linear(28*28, 100),
  torch.nn.ReLU(),
  torch.nn.Linear(100, 10)
)



# optimizer and loss function

optimizer = torch.optim.SGD(model.parameters(), lr=0.1)
loss = torch.nn.CrossEntropyLoss()
```

```python
# train

def fit(model, train_loader, test_loader, optimizer, loss, epochs = 5):
  for epoch in range(1, epochs+1):
    # train
    model.train() # VERY IMPORTANT !
    train_loss, train_acc = [], 0
    for imgs, labels in tqdm(train_loader):
      optimizer.zero_grad() # VERY IMPORTANT !
      output = model(imgs.reshape(imgs.shape[0],28*28))
      l = loss(output, labels)
      l.backward()
      optimizer.step()
      train_acc += (torch.argmax(output, axis=1) == labels).sum()
      train_loss.append(l.item())
    # eval
    model.eval() # VERY IMPORTANT !
    test_loss, test_acc = [], 0
    with torch.no_grad():
      for imgs, labels in tqdm(test_loader):
        output = model(imgs.reshape(imgs.shape[0],28*28))
        l = loss(output, labels)
        test_acc += (torch.argmax(output, axis=1) == labels).sum()
        test_loss.append(l.item())
    print(f'Epoch: {epoch}/{epochs}\n Train loss: 
{np.mean(train_loss):.4f} Test loss: {np.mean(test_loss):.4f} Train Acc: 
{train_acc}/{len(train_dataset)} ({100*train_acc/len(train_dataset)} %) 
Test Acc: {test_acc}/{len(test_dataset)} ({100*test_acc/
len(test_dataset)} %)')
```

```python
# custom datasets

from torch.utils.data import Dataset

class MNISTDataset(Dataset):
  def __init__(self, images, labels):
    self.images = images
    self.labels = labels

  def __len__(self):
    return len(self.labels)

  def __getitem__(self, ix):
    img = torch.from_numpy(self.images[ix] / 255).float()
    return img, torch.tensor(self.labels[ix]).long()


train_dataset = MNISTDataset(X_train, y_train)
test_dataset = MNISTDataset(X_test, y_test)

# get first sample

img, label = train_dataset[0]
```

```python
# custom networks

import torch.nn.functional as F

class Net(torch.nn.Module):

  def __init__(self):
    super(Net, self).__init__()
    self.fc1 = torch.nn.Linear(784, 100)
    self.fc2 = torch.nn.Linear(100, 10)

  def forward(self, x):
    x = self.fc1(x)
    x = F.relu(x)
    x = self.fc2(x)
    return x

net = Net()
```

```python
# train on GPU

device = "cuda" if torch.cuda.is_available() else "cpu"

def fit_gpu(model, train_loader, test_loader, optimizer, loss, device,
epochs=10):
  model.to(device) # COPY MODEL TO GPU
  for epoch in range(1, epochs+1):
      # train
      model.train() # VERY IMPORTANT !
      train_loss, train_acc = [], 0
      for imgs, labels in tqdm(train_loader):
        imgs, labels = imgs.to(device), labels.to(device) # COPY DATA TO GPU
        optimizer.zero_grad() # VERY IMPORTANT !
        output = model(imgs.reshape(imgs.shape[0],28*28))
        l = loss(output, labels)
        l.backward()
        optimizer.step()
        train_acc += (torch.argmax(output, axis=1) == labels).sum()
        train_loss.append(l.item())
      # eval
      model.eval() # VERY IMPORTANT !
      test_loss, test_acc = [], 0
      with torch.no_grad():
        for imgs, labels in tqdm(test_loader):
          imgs, labels = imgs.to(device), labels.to(device) # COPY DATA TO GPU
          output = model(imgs.reshape(imgs.shape[0],28*28))
          l = loss(output, labels)
          test_acc += (torch.argmax(output, axis=1) == labels).sum()
          test_loss.append(l.item())
      print(f'Epoch: {epoch}/{epochs}\n Train loss: {np.mean(train_loss):.4f}
Test loss: {np.mean(test_loss):.4f} Train Acc: {train_acc}/{len(train_dataset)}
({100*train_acc/len(train_dataset)} %) Test Acc: {test_acc}/{len(test_dataset)}
({100*test_acc/len(test_dataset)} %)')
```

```
# save model

torch.save(net.state_dict(), “my_model_params.pth")

torch.save(net, “my_model.pth")

scripted_model = torch.jit.script(net)
scripte_model.save(“my_scripted_model.pth”)


# load model

net.load_state_dict(torch.load(“my_model_params.pth”))

net = torch.load(“my_model.pth")

net = torch.jit.load(“my_scripted_model.pth”)
```

Pytorch

- Low level API (ideal for research)
- Need to define your own datasets
- Need to define your own training loop
- More control and flexibility
- Great ecosystem.
- Still behind on production ready tools.

Other Frameworks & Interoperability

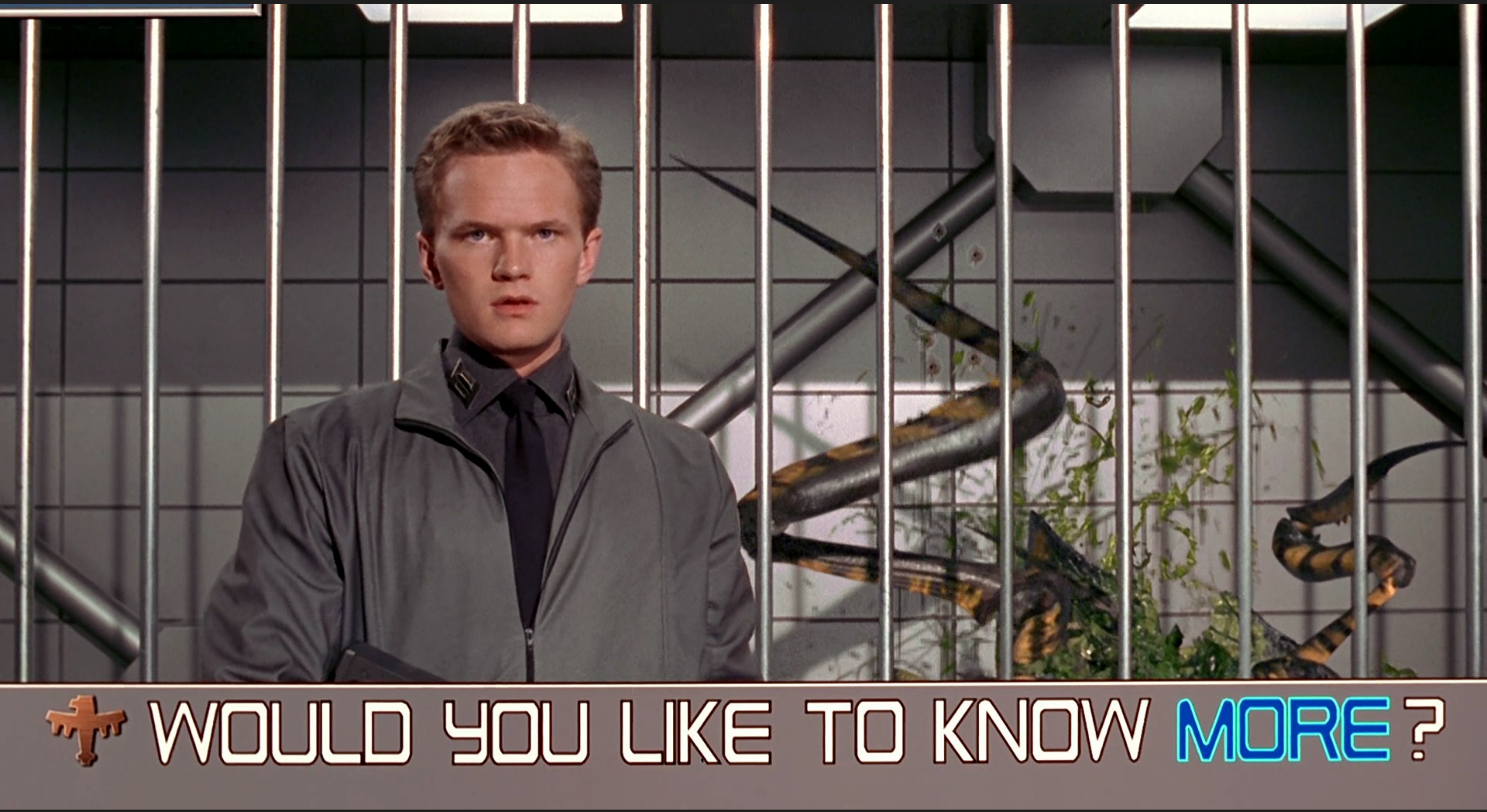
Other Frameworks

- MXNet (Amazon)
- CNTK (Microsoft)
- CoreML (Apple - iOS)
- Chainer
- …

Interoperability

- Standardize neural network representations
- Train in one framework, optimize in another, deploy in another.
- Opportunity for optimization (pruning, quantization, …)
- Language agnostic
- Example: ONNX (https://onnx.ai/)

https://colab.research.google.com/github/sensioai/dl/blob/master/frameworks/frameworks.ipynb



https://keras.io/ https://www.tensorflow.org/ https://www.fast.ai/ https://pytorch.org/

Practice with this exercise

https://colab.research.google.com/github/sensioai/dl/blob/master/frameworks/exercise.ipynb

DEEP LEARNING FRAMEWORKS