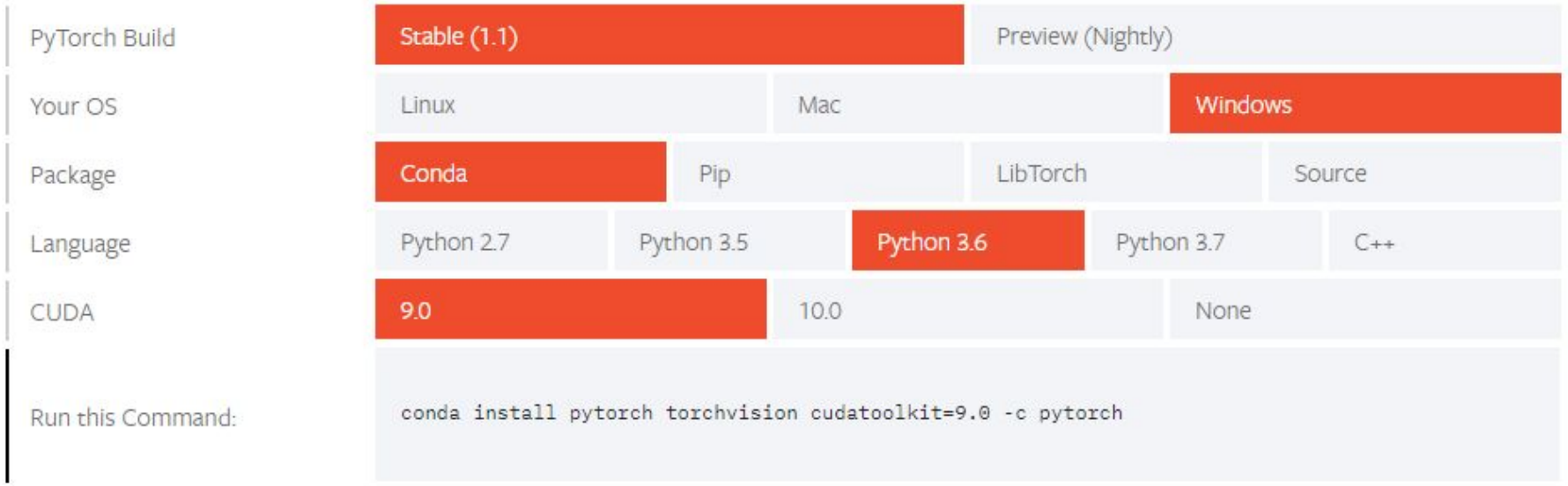
PYTORCH

Tomasz Supłat
Tomasz Pecyna

Pytorch

- Następca biblioteki Torch
  - initial release: październik 2016
  - preview release: październik 2018
  - stable release: luty 2019
- Obecnie rozwijana i wspierana przez facebook`a
- Początkowo przeznaczona głównie do NLP
- Wsparcie dla CPU, GPU, TPU (alpha)
- Wsparcie dla obliczeń rozproszonych
- Integracja z Numpy

Pip install pytorch



Tensors in pytorch

Just like well-known and liked Numpy

```
>>> import torch
>>> x = torch.arange(0, 10)
>>> x[3:6]
tensor([3, 4, 5])
```

Tensors in pytorch

Możemy konwertować z numpy do pytorcha

```
>>> import numpy as np
>>> y = np.ones(10, dtype='int64')
>>> x + torch.from_numpy(y)
tensor([ 1,  2,  3,  4,  5,  6,  7,  8,  9, 10])
>>>
```

I na odwrót

```
>>> torch.rand(10).numpy() + y
array([1.65107757, 1.34298605, 1.35336411, 1.08194017, 1.61983663,
       1.29232937, 1.48570156, 1.46275377, 1.89105892, 1.84551764])
```

Tensors in pytorch

Layout, czyli optymalizacja dostępów do pamięci.

```
>>> x = x.reshape((2, 5))
>>> x
tensor([[0, 1, 2, 3, 4],
        [5, 6, 7, 8, 9]])
>>> x[0, 0]
tensor(0)
>>> x[1, 0]
tensor(5)
>>> x[0, 1]
tensor(1)
>>> x.stride()
(5, 1)
```

Tensors in pytorch

Możemy korzystać też z funkcji matematycznych.

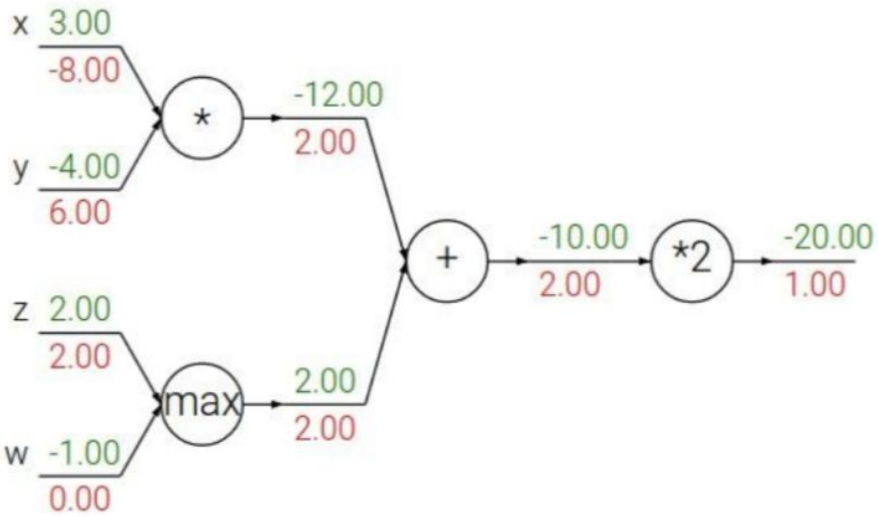
```
>>> x.mean()
Traceback (most recent call last):
  File "<stdin>", line 1, in <module>
RuntimeError: Can only calculate the mean of floating types. Got Long instead.
>>> x = x.type(torch.float32)
>>> x.mean()
tensor(4.5000)
```

GPU Tensors

```python
# let us run this cell only if CUDA is available
# We will use ``torch.device`` objects to move tensors in and out of GPU
if torch.cuda.is_available():
    device = torch.device("cuda")          # a CUDA device object
    y = torch.ones_like(x, device=device)  # directly create a tensor on GPU
    x = x.to(device)                       # or just use strings ``.to("cuda")``
    z = x + y
    print(z)
    print(z.to("cpu", torch.double))       # ``.to`` can also change dtype together!
```

```
>>> x = x.to("cuda")
>>> x
tensor([0, 1, 2, 3, 4, 5, 6, 7, 8, 9], device='cuda:0')
>>> x = x ** 2
>>> x = x.to("cpu")
>>> x
tensor([ 0,  1,  4,  9, 16, 25, 36, 49, 64, 81])
```

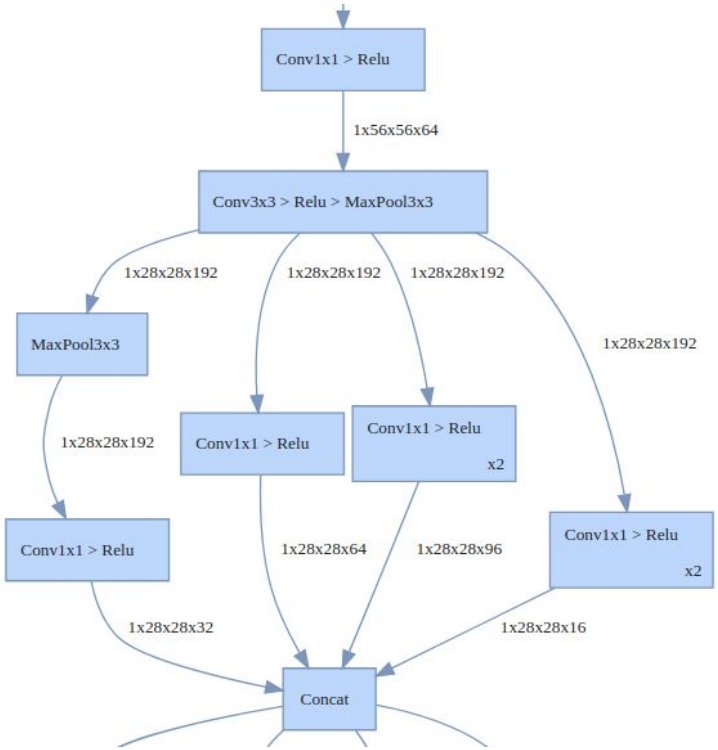
Autograd

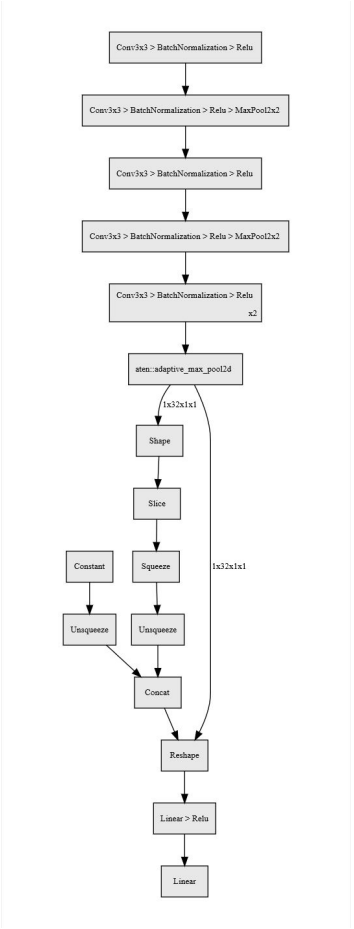



```
>>> x = torch.tensor(3., requires_grad=True)
>>> y = torch.tensor(-4., requires_grad=True)
>>> z = torch.tensor(2., requires_grad=True)
>>> w = torch.tensor(-1., requires_grad=True)
>>> out = (x*y + max(z, w)) * 2
>>> out
tensor(-20., grad_fn=<MulBackward0>)
>>> out.backward()
>>> x.grad
tensor(-8.)
>>> y.grad
tensor(6.)
>>> z.grad
tensor(2.)
>>> w.grad
>>>
```

Source: http://www.cs.put.poznan.pl/kkrawiec/wiki/uploads/Zajecia/EIO.pdf

Graf obliczeń





Obliczenia równoległe

Obliczenia na wielu GPU:

```
model = nn.DataParallel(model)
```

Przykład:

```
model = Model(input_size, output_size)
if torch.cuda.device_count() > 1:
  print("Let's use", torch.cuda.device_count(), "GPUs!")
  # dim = 0 [30, xxx] -> [10, ...], [10, ...], [10, ...] on 3 GPUs
  model = nn.DataParallel(model)

model.to(device)
```

Obliczenia równoległe

```
Let's use 3 GPUs!
    In Model: input size torch.Size([10, 5]) output size torch.Size([10, 2])
    In Model: input size torch.Size([10, 5]) output size torch.Size([10, 2])
    In Model: input size torch.Size([10, 5]) output size torch.Size([10, 2])
Outside: input size torch.Size([30, 5]) output_size torch.Size([30, 2])
    In Model: input size torch.Size([10, 5]) output size torch.Size([10, 2])
    In Model: input size torch.Size([10, 5]) output size torch.Size([10, 2])
    In Model: input size torch.Size([10, 5]) output size torch.Size([10, 2])
Outside: input size torch.Size([30, 5]) output_size torch.Size([30, 2])
    In Model: input size torch.Size([10, 5]) output size torch.Size([10, 2])
    In Model: input size torch.Size([10, 5]) output size torch.Size([10, 2])
    In Model: input size torch.Size([10, 5]) output size torch.Size([10, 2])
Outside: input size torch.Size([30, 5]) output_size torch.Size([30, 2])
    In Model: input size torch.Size([4, 5]) output size torch.Size([4, 2])
    In Model: input size torch.Size([4, 5]) output size torch.Size([4, 2])
    In Model: input size torch.Size([2, 5]) output size torch.Size([2, 2])
Outside: input size torch.Size([10, 5]) output_size torch.Size([10, 2])
```

Obliczenia rozproszone

```python
from torch.nn.parallel import DistributedDataParallel as DDP


def setup(rank, world_size):
    os.environ['MASTER_ADDR'] = 'localhost'
    os.environ['MASTER_PORT'] = '12355'

    # initialize the process group
    dist.init_process_group("gloo", rank=rank, world_size=world_size)

    # Explicitly setting seed to make sure that models created in two processes
    # start from same random weights and biases.
    torch.manual_seed(42)
```

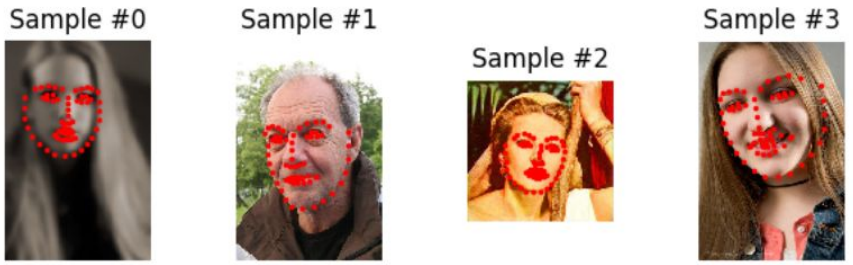
```
>>> torch.distributed.init_process_group(backend='nccl', world_size=4,
init_method='...')
>>> model = DistributedDataParallel(model, device_ids=[i], output_device=i)
```

- Wsparcie dla wielu modeli komunikacji (TCP, GLOO, MPI, itd)
- Wsparcie dla podstawowych operacji (map, reduce, scatter, gather)
- Możliwość wysyłania wiadomości do innych procesów

Wczytywanie danych

```python
class FaceLandmarksDataset(Dataset):
    """Face Landmarks dataset."""

    def __init__(self, csv_file, root_dir, transform=None):
        """
        Args:
            csv_file (string): Path to the csv file with annotations.
            root_dir (string): Directory with all the images.
            transform (callable, optional): Optional transform to be applied
                on a sample.
        """
        self.landmarks_frame = pd.read_csv(csv_file)
        self.root_dir = root_dir
        self.transform = transform

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

    def __getitem__(self, idx):
        img_name = os.path.join(self.root_dir,
                                self.landmarks_frame.iloc[idx, 0])
        image = io.imread(img_name)
        landmarks = self.landmarks_frame.iloc[idx, 1:].as_matrix()
        landmarks = landmarks.astype('float').reshape(-1, 2)
        sample = {'image': image, 'landmarks': landmarks}

        if self.transform:
            sample = self.transform(sample)

        return sample
```

```python
face_dataset = FaceLandmarksDataset(csv_file='data/faces/face_landmarks.csv',
                                    root_dir='data/faces/')

fig = plt.figure()

for i in range(len(face_dataset)):
    sample = face_dataset[i]

    print(i, sample['image'].shape, sample['landmarks'].shape)
```



Wczytywanie danych

```
import torch
from torchvision import transforms, datasets

data_transform = transforms.Compose([
        transforms.RandomSizedCrop(224),
        transforms.RandomHorizontalFlip(),
        transforms.ToTensor(),
        transforms.Normalize(mean=[0.485, 0.456, 0.406],
                             std=[0.229, 0.224, 0.225])
    ])
hymenoptera_dataset = datasets.ImageFolder(root='hymenoptera_data/train',
                                           transform=data_transform)
dataset_loader = torch.utils.data.DataLoader(hymenoptera_dataset,
                                             batch_size=4, shuffle=True,
                                             num_workers=4)
```

Model

```
import torch


class TwoLayerNet(torch.nn.Module):
    def __init__(self, D_in, H, D_out):
        """
        In the constructor we instantiate two nn.Linear modules and assign them as
        member variables.
        """
        super(TwoLayerNet, self).__init__()
        self.linear1 = torch.nn.Linear(D_in, H)
        self.linear2 = torch.nn.Linear(H, D_out)

    def forward(self, x):
        """
        In the forward function we accept a Tensor of input data and we must return
        a Tensor of output data. We can use Modules defined in the constructor as
        well as arbitrary operators on Tensors.
        """
        h_relu = self.linear1(x).clamp(min=0)
        y_pred = self.linear2(h_relu)
        return y_pred
```

Wczytywanie i zapisywanie modeli

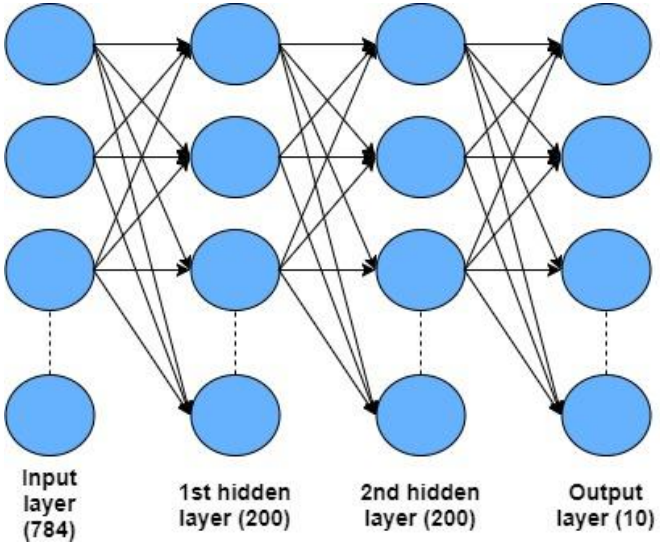
Save:

```
torch.save(model.state_dict(), PATH)
```

Load:

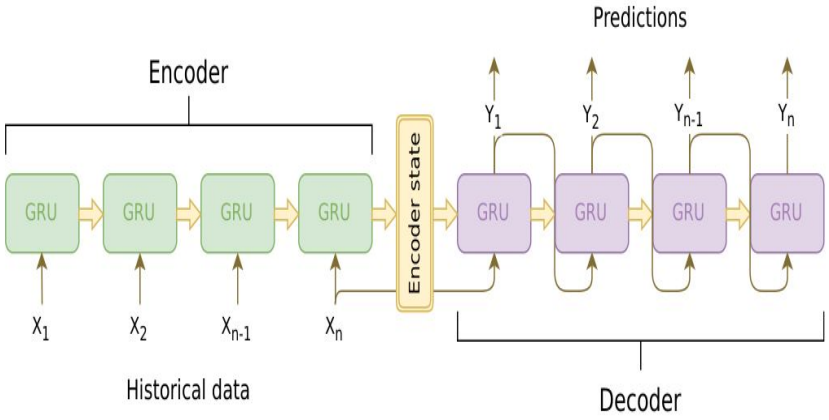
```
model = TheModelClass(*args, **kwargs)
model.load_state_dict(torch.load(PATH))
model.eval()
```

Sieci neuronowe




```python
import torch.nn as nn
import torch.nn.functional as F

class Net(nn.Module):
    def __init__(self):
        super(Net, self).__init__()
        self.fc1 = nn.Linear(28 * 28, 200)
        self.fc2 = nn.Linear(200, 200)
        self.fc3 = nn.Linear(200, 10)
```

```python
    def forward(self, x):
        x = F.relu(self.fc1(x))
        x = F.relu(self.fc2(x))
        x = self.fc3(x)
        return F.log_softmax(x)
```




```python
class EncoderRNN(nn.Module):
    def __init__(self, hidden_size, embedding, n_layers=1, dropout=0):
        super(EncoderRNN, self).__init__()
        self.n_layers = n_layers
        self.hidden_size = hidden_size
        self.embedding = embedding
        self.gru = nn.GRU(hidden_size, hidden_size, n_layers,
                          dropout=(0 if n_layers == 1 else dropout), bidirectional=True)

    def forward(self, input_seq, input_lengths, hidden=None):
        embedded = self.embedding(input_seq)
        packed = torch.nn.utils.rnn.pack_padded_sequence(embedded, input_lengths)
        outputs, hidden = self.gru(packed, hidden)
        outputs, _ = torch.nn.utils.rnn.pad_packed_sequence(outputs)
        outputs = outputs[:, :, :self.hidden_size] + outputs[:, : ,self.hidden_size:]
        return outputs, hidden
```

Debugowanie

```
    def init_hidden(self):
        # Before we've done anything, we dont have any hidden state.
        # Refer to the Pytorch documentation to see exactly
        # why they have this dimensionality.
        # The axes semantics are (num_layers, minibatch_size, hidden_dim)
        return (torch.zeros(self.num_layers, 1, self.hidden_dim).to(self.device),
                torch.zeros(self.num_layers, 1, self.hidden_dim).to(self.device))

    def forward(self, sentence):  self: LstmModule(\n  (lstm): LSTM(1, 32, num_layers=2)\n  (hidden2tag): Linear(in_features=32, o
        lstm_out, self.hidden = self.lstm(  lstm_out: tensor([[[ 0.0493,  0.0514, -0.0051,  ..., -0.0190,  0.0181,  0.0563]],\n\n
            sentence.view(len(sentence), 1, -1), self.hidden)
        tag_space = self.hidden2tag(lstm_out.view(len(sentence), -1))
        #tag_scores = F.log_softmax(tag_space, dim=1)
        return tag_space
        #return tag_scores
```

LstmModule › forward()

Variables | Watches

lstm_out = {Tensor} tensor([[[0.0493, 0.0514, -0.0051, ..., -0.0190, 0.0181, 0.0563]],\n\n [[0.0805, 0.0735, -0.0181, ..., -0.0059, 0.0369, 0.1017]],\n\n [[0.0890, 0.0901, -0.

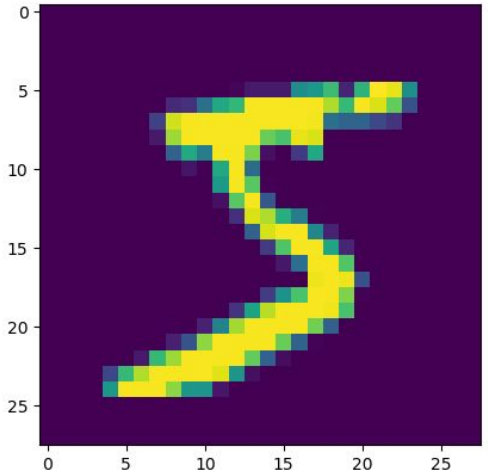
self = {LstmModule} LstmModule(\n (lstm): LSTM(1, 32, num_layers=2)\n (hidden2tag): Linear(in_features=32, out_features=1, bias=True)\n)

Torchvision - proste, jasne, oczywiste.

- Wiele dostępnych zbiorów danych: mnist, cifar, flickr, ...
- Wcześniej nauczone sieci: alexnet, resnet, googlenet, NASAnet, UAMnet, polibudaNET, dotNet ...
- Transformacje: padding, scaling, adjusting colour, ...

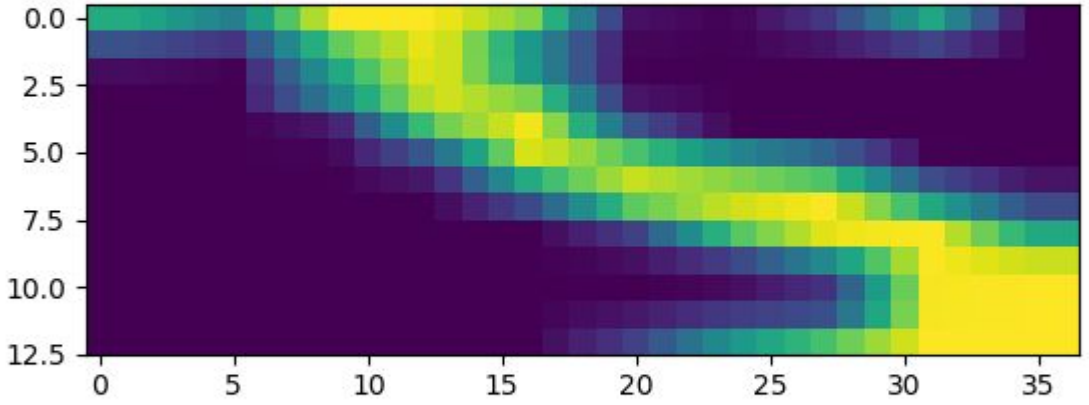
Torchvision - from mnist import mnist?

```
>>> from torchvision import datasets
>>> from torchvision import datasets, transforms
>>> import matplotlib.pyplot as plt
>>>
>>> train_set = datasets.MNIST('./data', train=True, download=True, transform=transforms.ToTensor())
>>> train_loader = DataLoader(train_set)
>>> image, digit = next(iter(train_loader))
>>> plt.imshow(image[0, 0])
<matplotlib.image.AxesImage object at 0x000001DC5F6939E8>
>>> plt.show()
>>> digit
tensor([5])
```



Torchvision - a co jeśli chcemy żeby obrazki wyglądały inaczej?

```
>>> some_composition = transforms.Compose([
...     transforms.CenterCrop(10),
...     transforms.ColorJitter(),
...     transforms.Resize((13, 37)),
...     transforms.ToTensor()])
>>> train_set = datasets.MNIST('./data', train=True, download=True, transform=some_composition)
```



Torchvision - from resnet import resnet?

Mamo, co zrobić z tymi 20 teslami co się kurzą w szafie?

Fine-tuning the convnet

```
model_ft = models.resnet18(pretrained=True)
num_ftrs = model_ft.fc.in_features
model_ft.fc = nn.Linear(num_ftrs, 2)
```

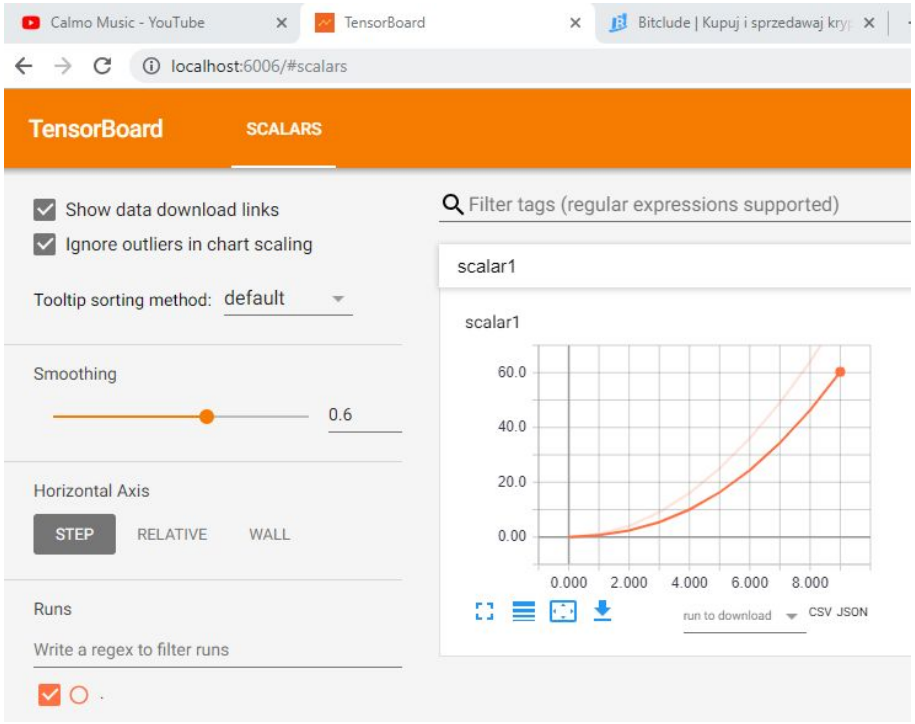
Convnet as fixed feature extractor

```
model_conv = torchvision.models.resnet18(pretrained=True)
for param in model_conv.parameters():
    param.requires_grad = False

# Parameters of newly constructed modules have requires_grad=True by default
num_ftrs = model_conv.fc.in_features
model_conv.fc = nn.Linear(num_ftrs, 2)
```

TensorboardX

- Wymaga TF i tensorboard’a
- pip install tensorboardX
- pip remove pytorch

```
>>> import torch
>>> from tensorboardX import SummaryWriter
>>> x = torch.arange(10)
>>> x = x**2
>>> writer = SummaryWriter()
>>> for i in range(10):
...     writer.add_scalar('scalar1', x[i], i)
...
>>> writer.close()
```



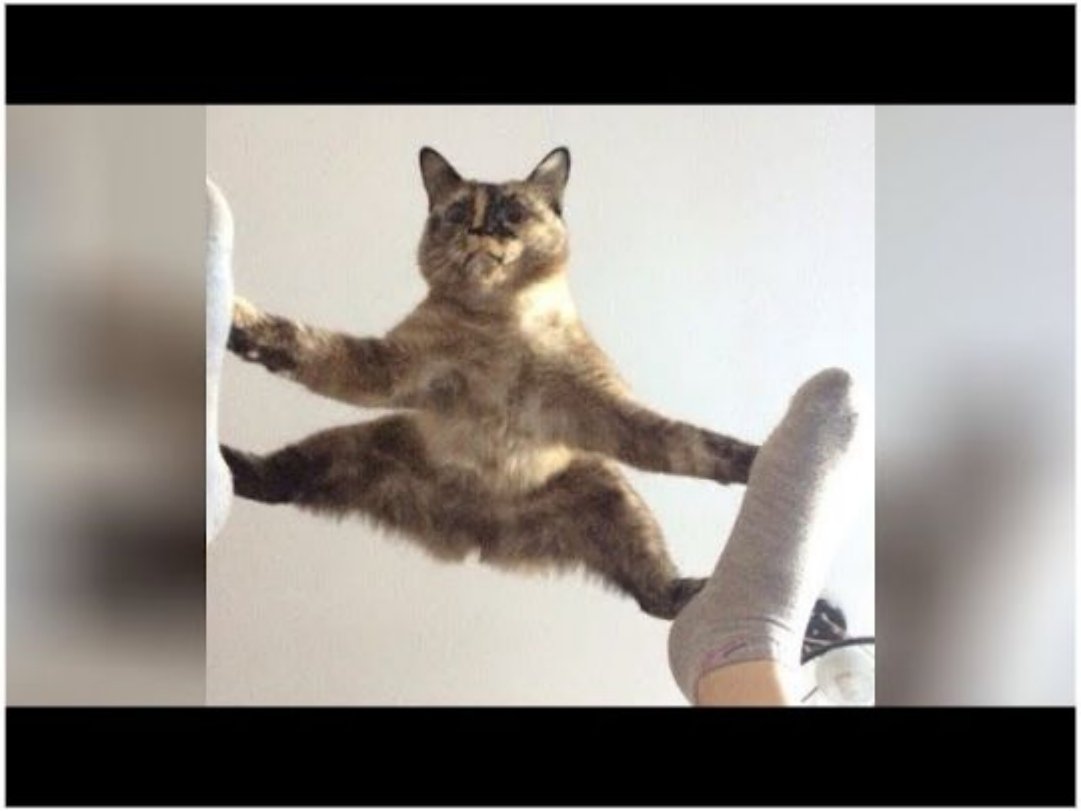

```
C:\Users\pecyna\Desktop\ZIELONE\runs>tensorboard --logdir Apr24_20-59-53_PL1WXL-108526
TensorBoard 1.12.0 at http://PL1WXL-108526:6006 (Press CTRL+C to quit)
```

What about cloud?

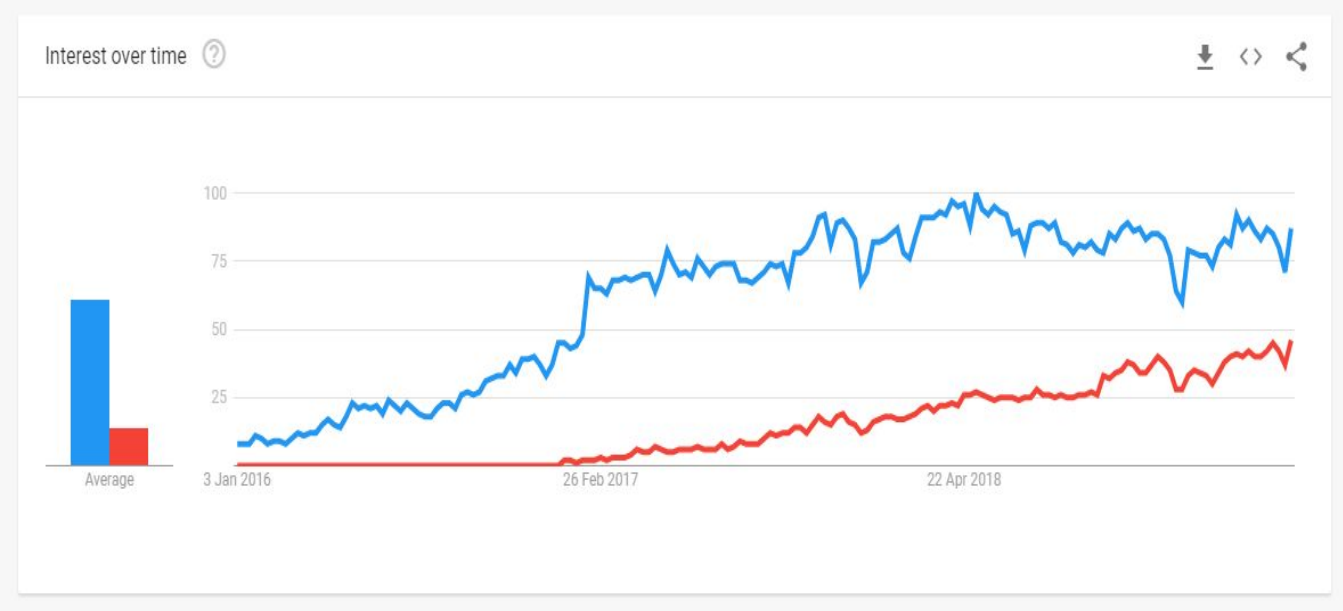
QUICK START WITH CLOUD PARTNERS

Get up and running with PyTorch quickly through popular cloud platforms and machine learning services.

- AWS >
- Google Cloud Platform >
- Microsoft Azure >



TF vs Pytorch



Pytania?