Deep Learning

A Programmer's Guide for Classification, Regression, and Clustering of Big Labelled Datasets

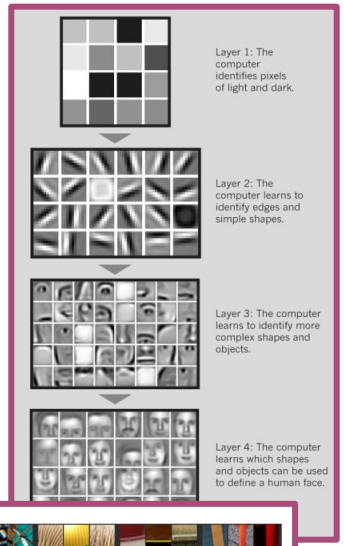
Dr Chris Willcocks

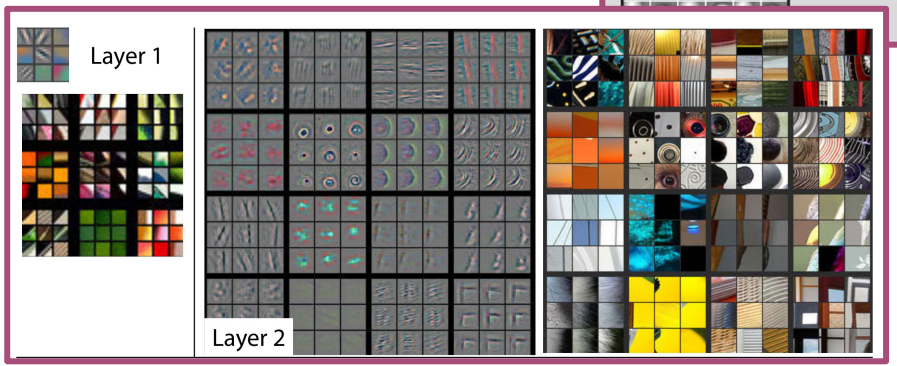
Durham University

intogral® LIMITED

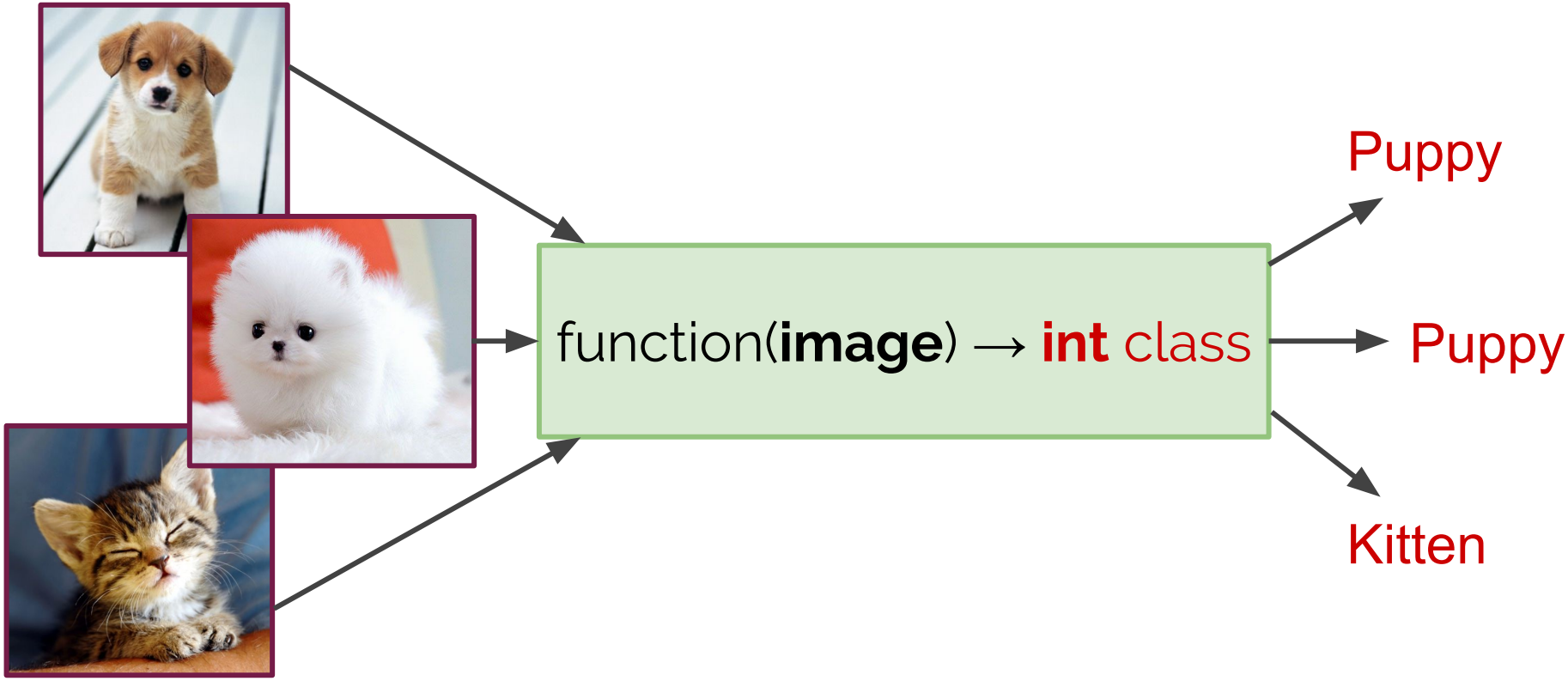
In this talk

1. Focus on the **intuition of concepts** and **programming** solutions
2. Basic concepts (classification, regression, clustering)
3. Programming directly on the server
4. Reshaping and thinking of **problems in tensors**
5. **Building & programming** the dataset
6. … the data loader
7. … loading models with transfer learning
8. … visdom
9. … training
10. … validation
11. **Deployment** of models
    - Programming client-server
12. Challenges

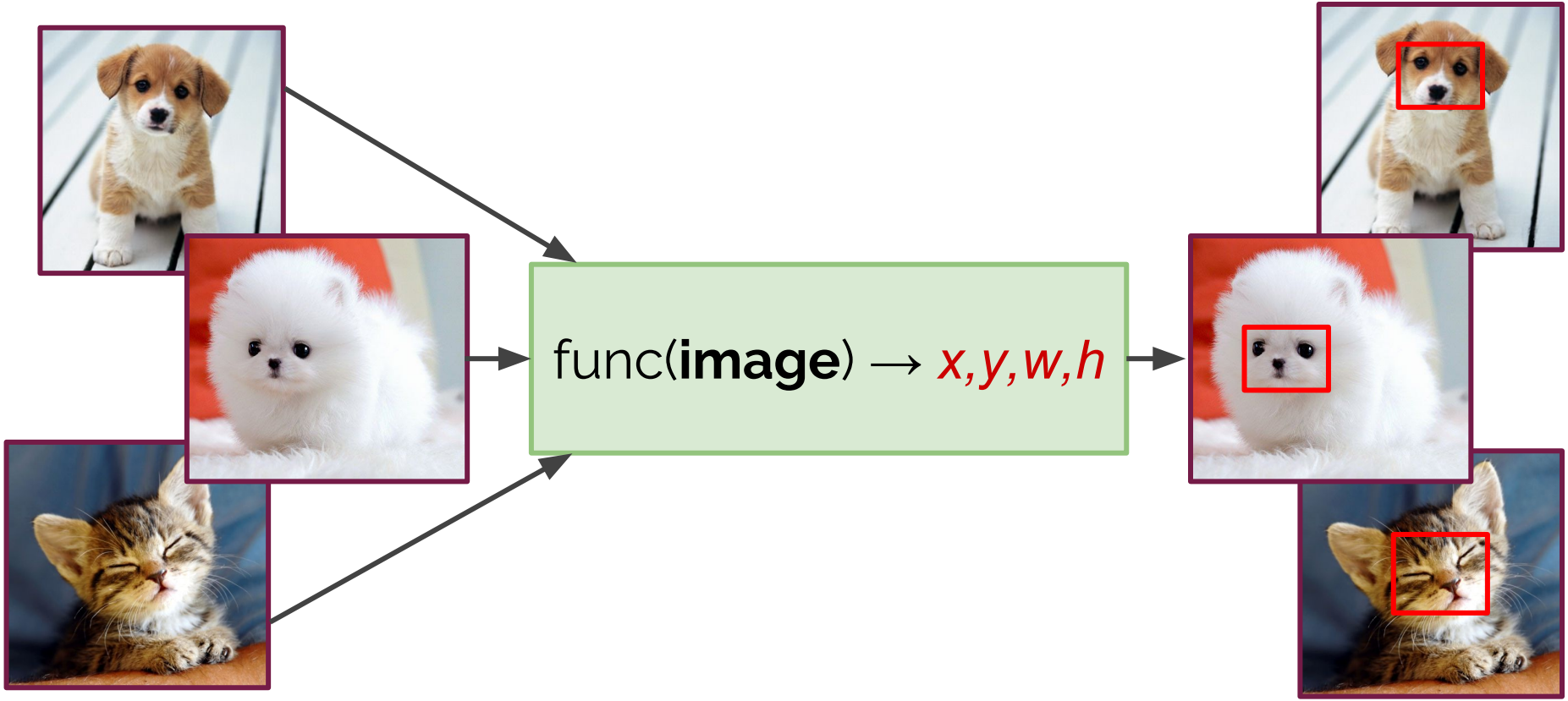




Classification Example

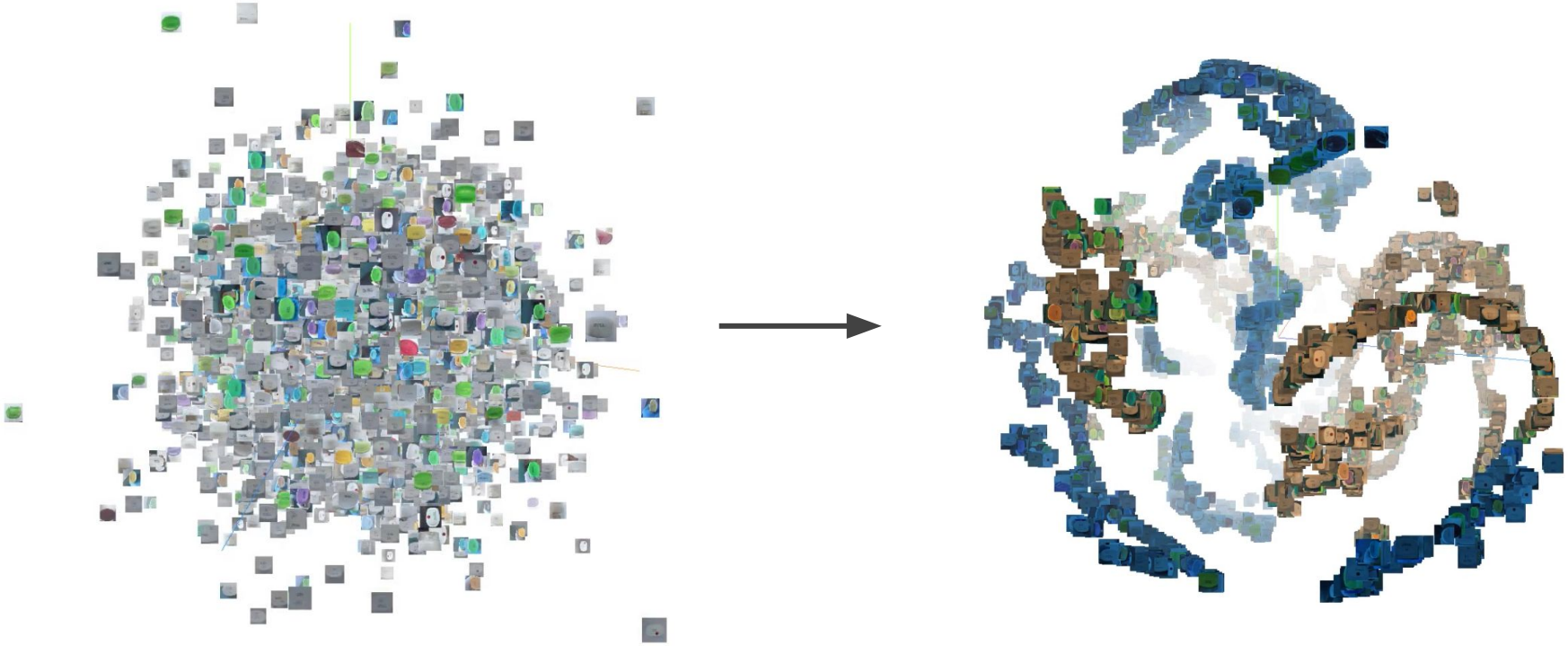


Regression Example

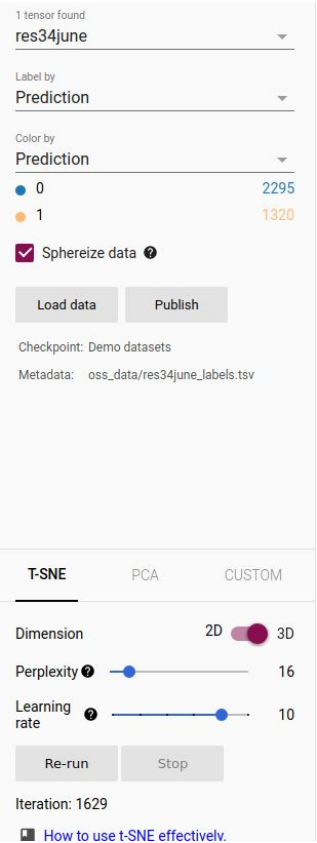


Clustering Example

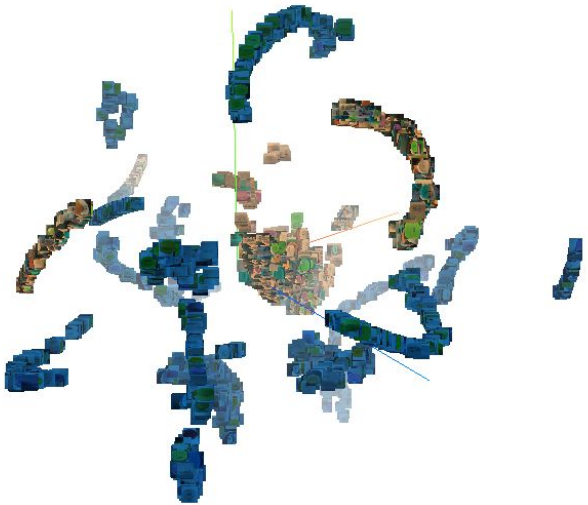
- Can we get some meaningful story and new insight from the dataset?



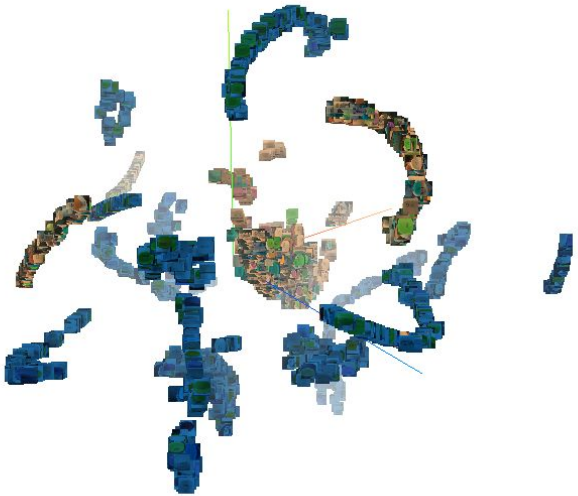
Clustering Example



Predictions



Groundtruth Labels



Clustering Example

Programming on the Server

Durham University

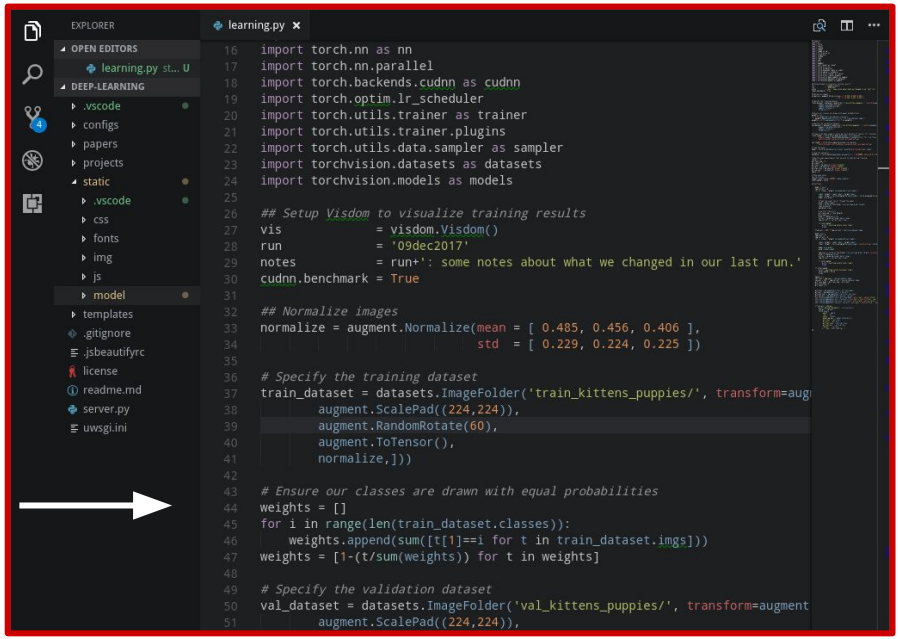
```
chris@chris-lab ❯ ~ ⚡ master ● ❯ ssh chris@50.232.78.63
```

```
master ● ❯ sshfs chris@50.232.78.63:/home/chris/deep-learning /tmp/gcp
```

Can be local on on server, use Github for code

```
chris@chris-lab ❯ ~/repos/miniconda3 ⚡ master ● ❯ source bin/activate
(root)  chris@chris-lab ❯ ~/repos/miniconda3 ⚡ master ● ❯ ../deep-learning
(root)  chris@chris-lab ❯ ~/repos/deep-learning ⚡ master ● ❯ ipython
Python 3.6.2 |Continuum Analytics, Inc.| (default, Jul 20 2017, 13:51:32)
Type 'copyright', 'credits' or 'license' for more information
IPython 6.1.0 -- An enhanced Interactive Python. Type '?' for help.

In [1]: 
```

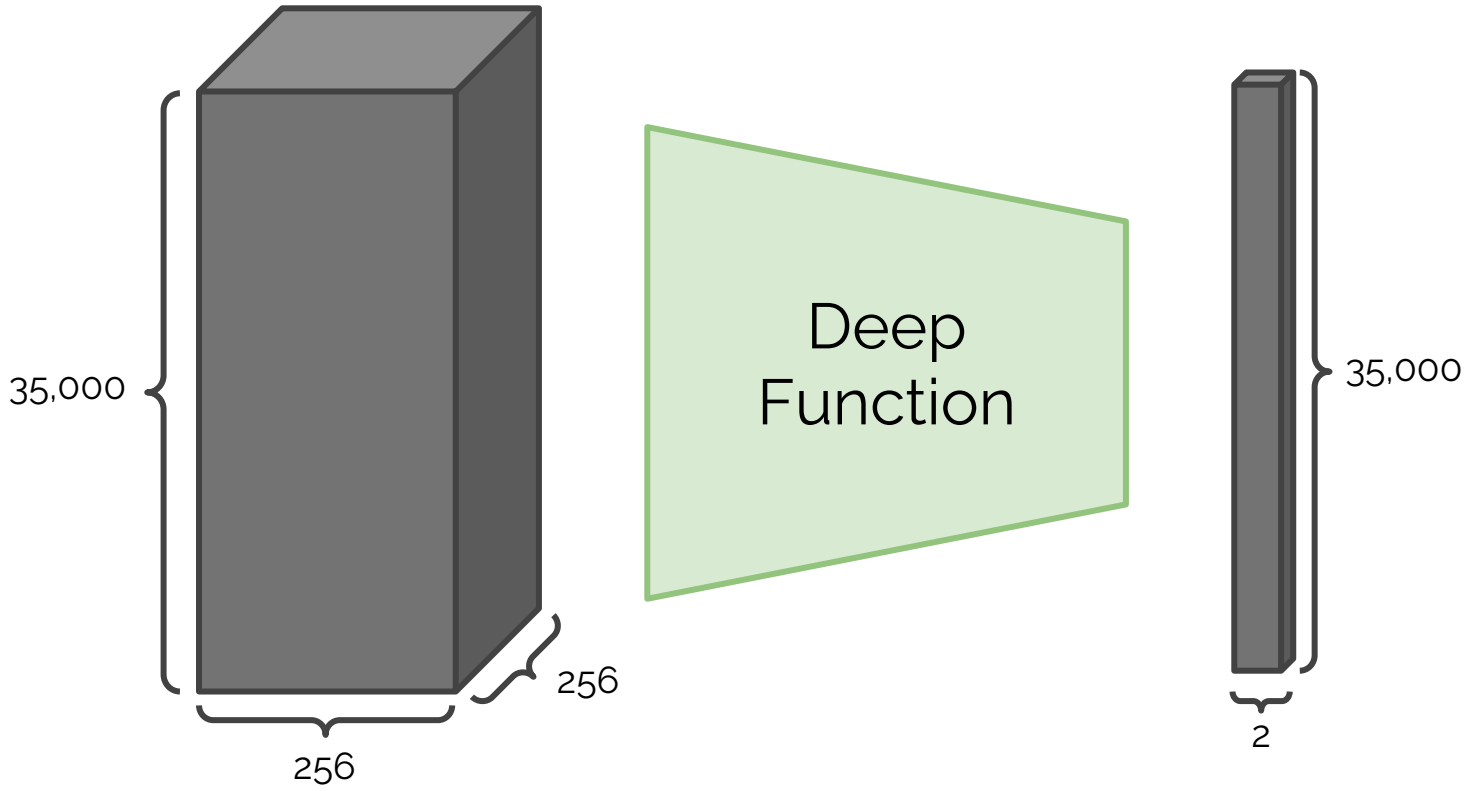
- Google Cloud Platform
- AWS
- Microsoft Azure
- Your home machine (if you have a good GPU)



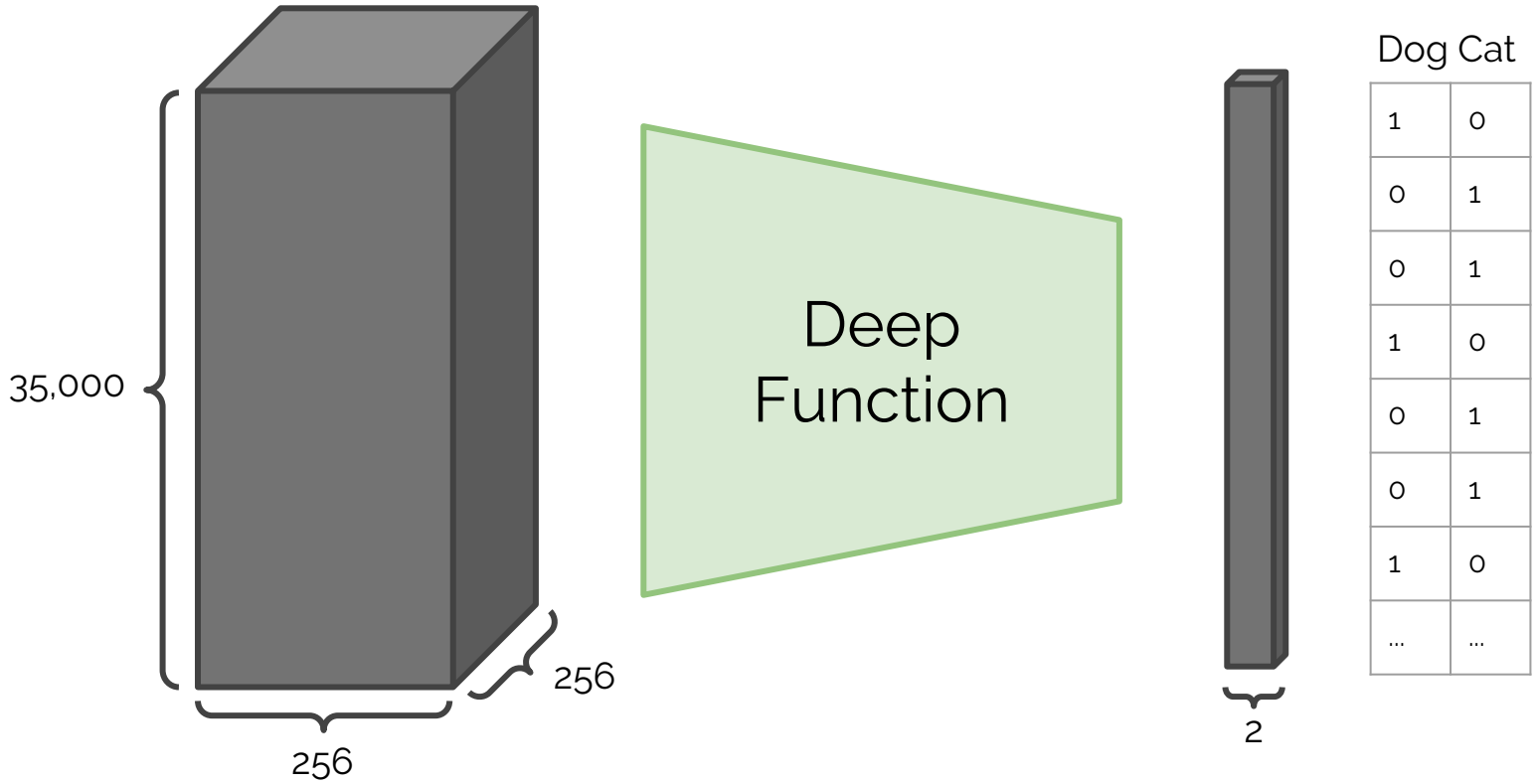
ssh, ipython

sshfs, visual studio code, pycharm, vim.

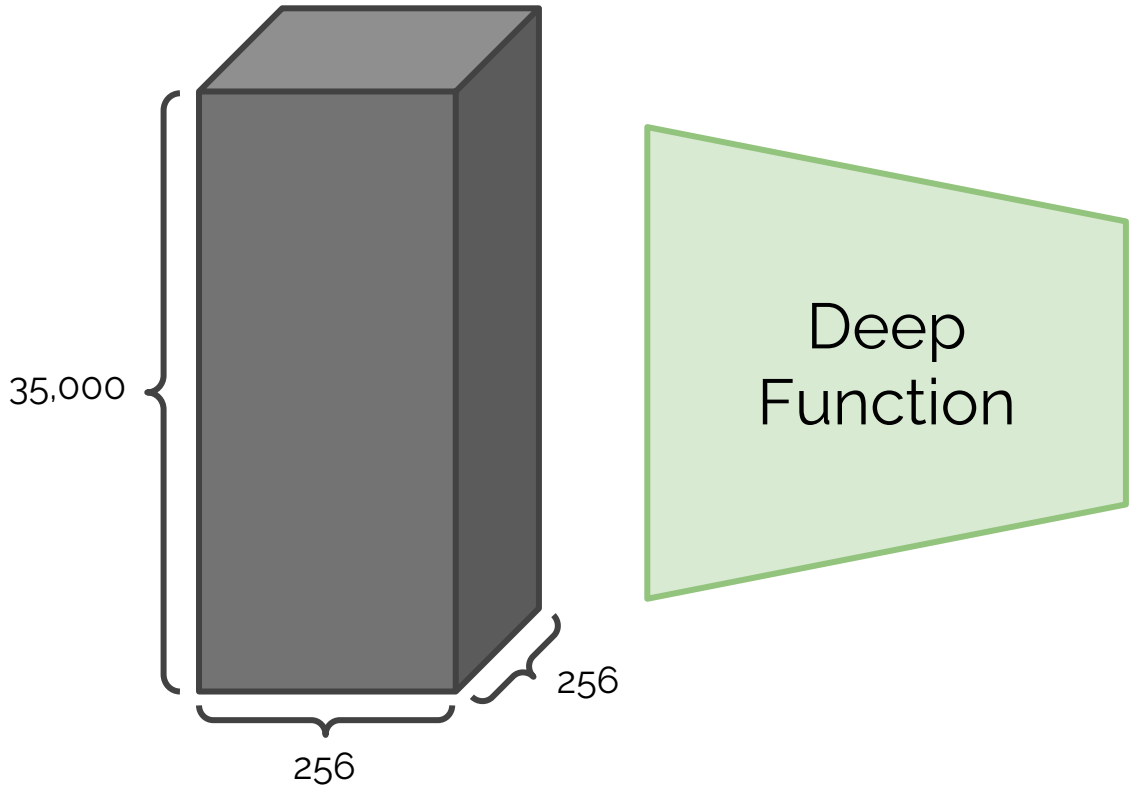
Tensors: Images → Classes



Tensors: Images → Classes

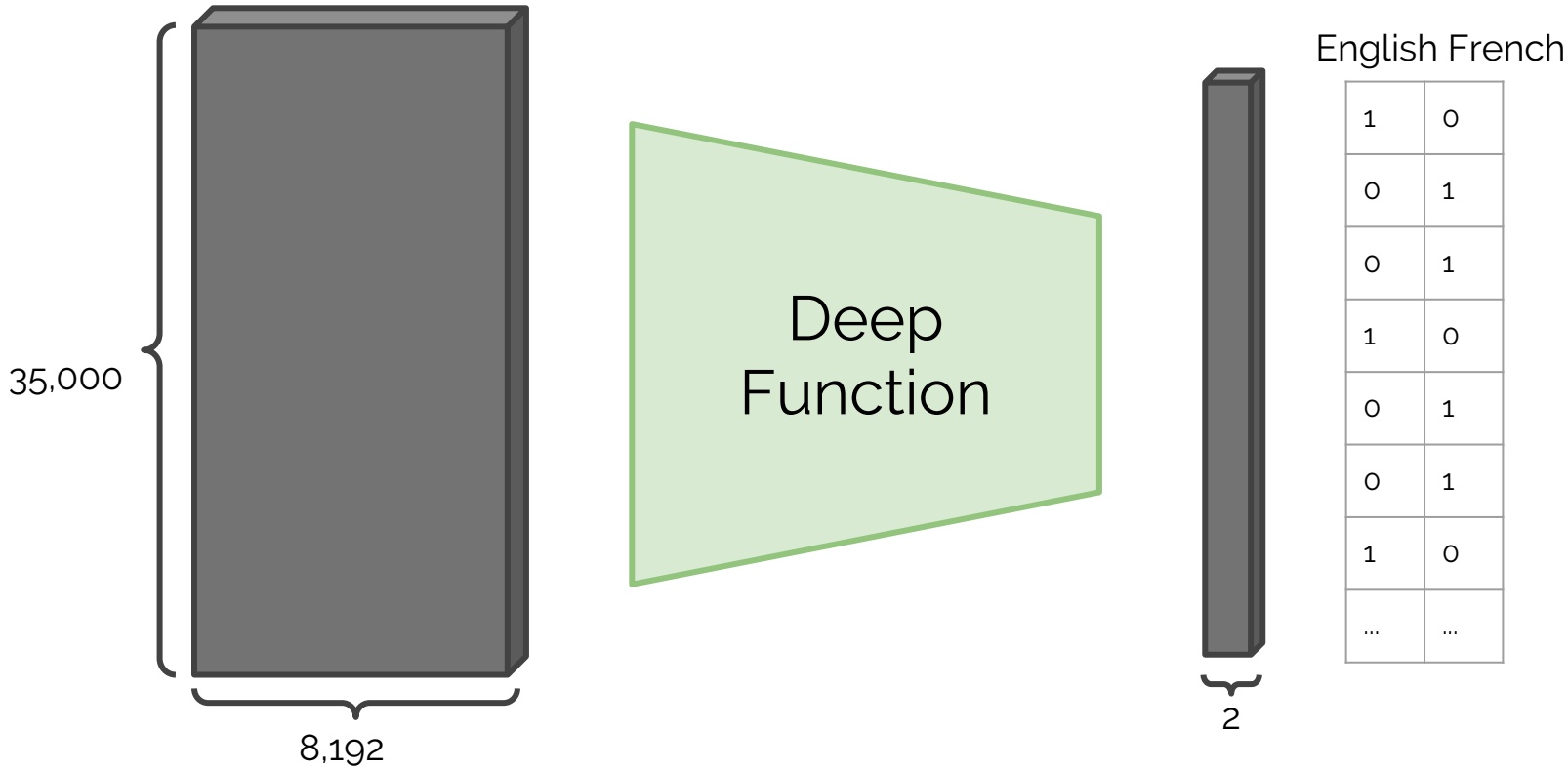


Tensors: Images → Classes

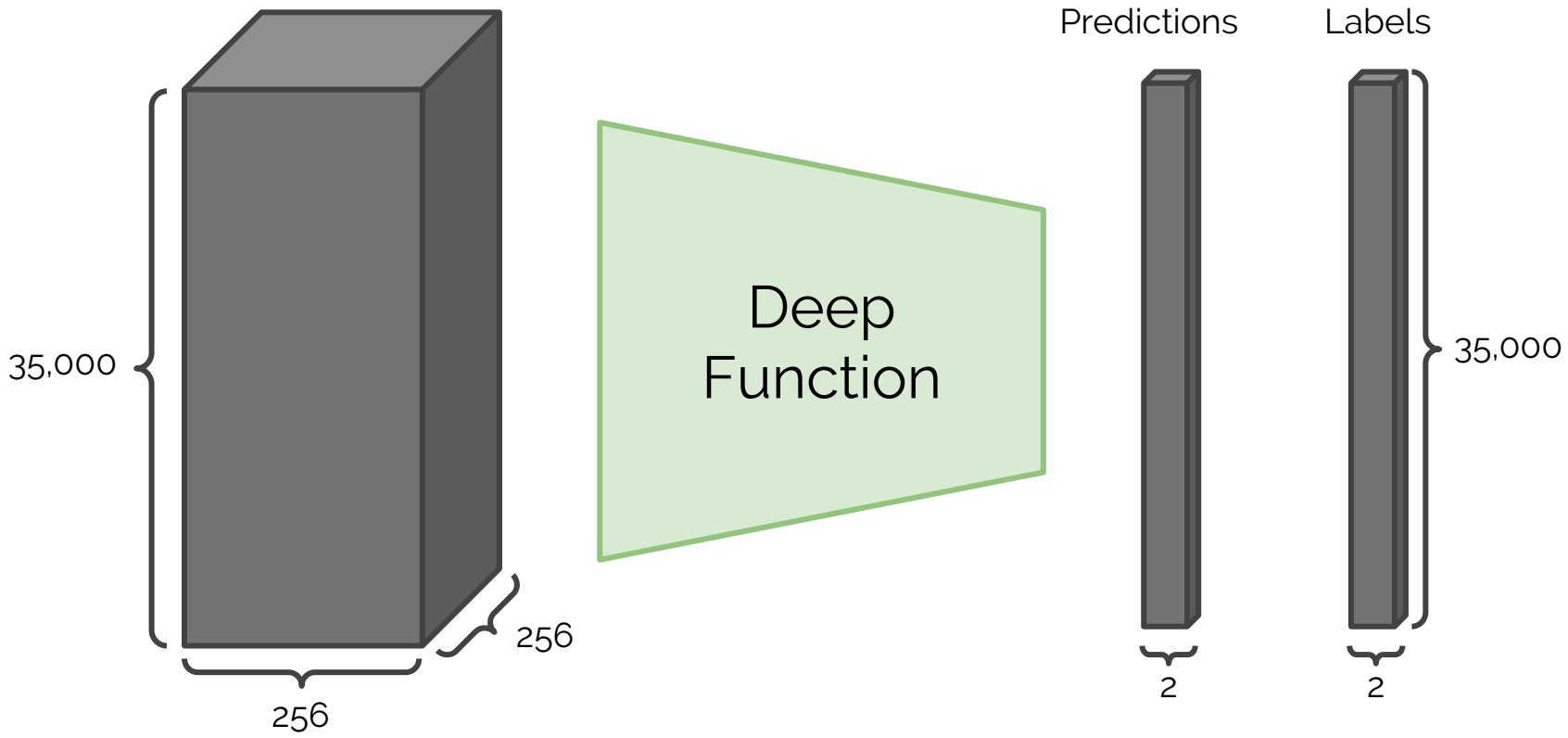


Dog	Cat	Fish
0	0	1
1	0	0
0	1	0
0	0	1
0	1	0
0	0	1
0	1	0
...	...	

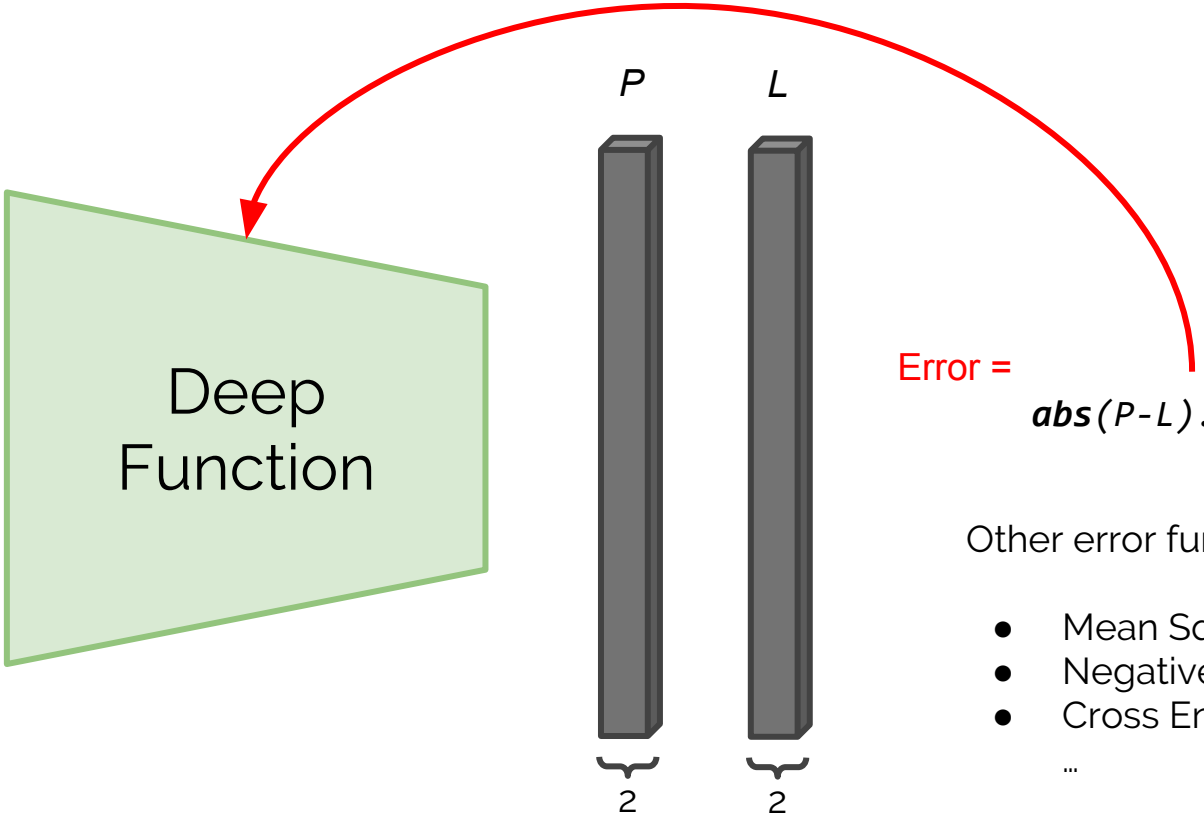
Tensors: Audio → Classes



Predictions and Labels



Error Functions



Error =

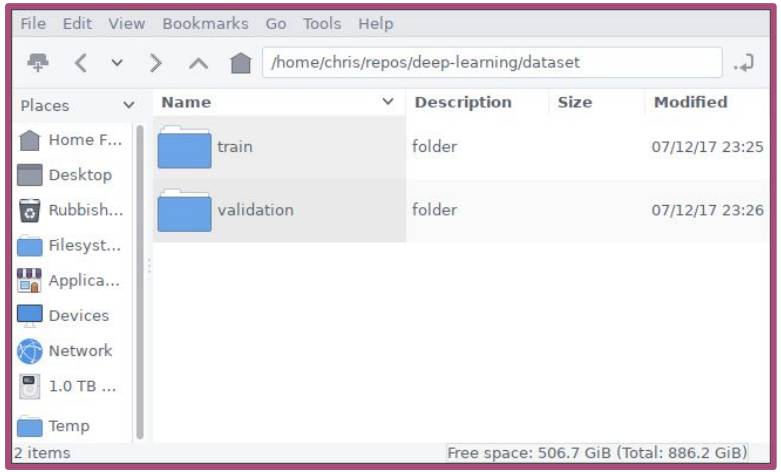
```
abs(P-L).mean()
```

Other error functions:

- Mean Squared Error (MSE)
- Negative Log Likelihood
- Cross Entropy Loss

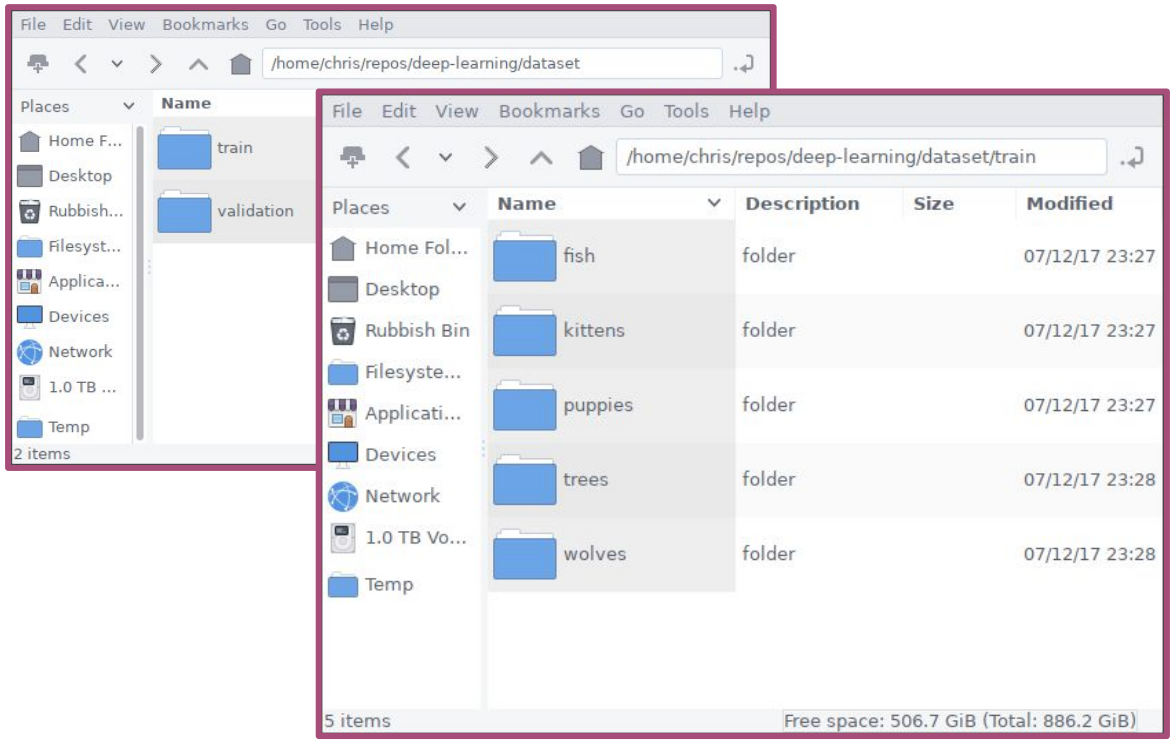
...

The Dataset



- Two folders
  - Train
  - Validation
- Validation contains about 10-20% of the train data
  - Don't randomly select, try to capture **harder** ones that represent different distributions, e.g:
    - Different weather conditions
    - Different cameras
    - Different geographical locations
    - Different poses

The Dataset



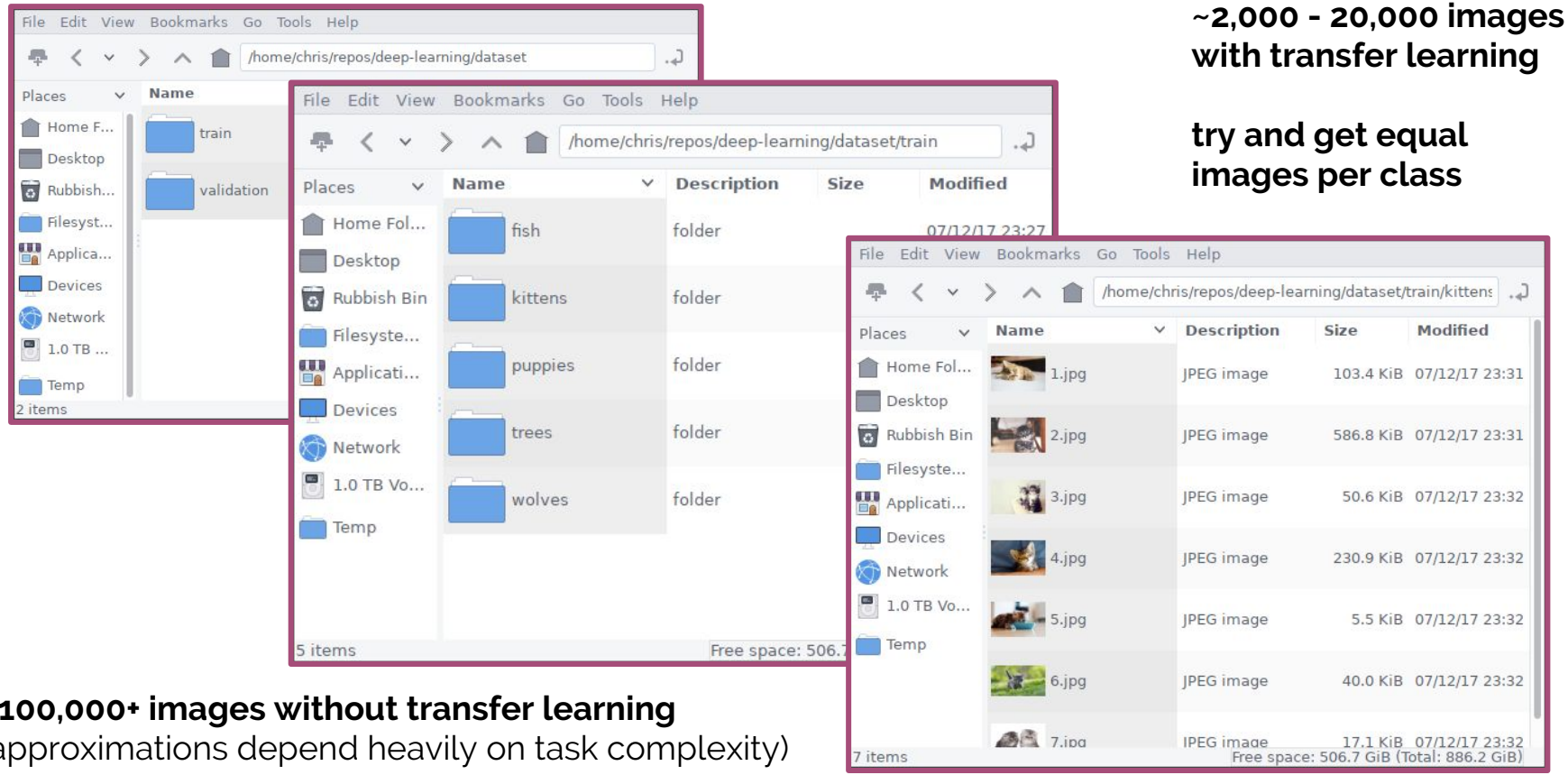
Classification:

- Put each class a separate folder inside train/validation
- Here we have 5-way classification

Regression:

Specify a CSV file with the target parameters, you will need to write your own code to parse this.

The Dataset



~2,000 - 20,000 images with transfer learning

try and get equal images per class

~100,000+ images without transfer learning
(approximations depend heavily on task complexity)

The Dataset

Durham University

Just a list of filenames and mappings to class indices (no images in memory)

```
...: import nets.resnet as resnet
...: import torch.nn as nn
...: import torch.nn.parallel
...: import torch.backends.cudnn as cudnn
...: import torch.optim.lr_scheduler
...: import torch.utils.trainer as trainer
...: import torch.utils.trainer.plugins
...: import torch.utils.data.sampler as sampler
...: import torchvision.datasets as datasets
...: import torchvision.models as models
...:

In [2]: vis             = visdom.Visdom()
...: run             = '09dec2017'
...: notes           = run+': some notes about what we changed in our last run.'
...: cudnn.benchmark = True
...:

In [3]: ## Normalize images
...: normalize = augment.Normalize(mean = [ 0.485, 0.456, 0.406 ],
...:                               std  = [ 0.229, 0.224, 0.225 ])
...:

In [4]: # Specify the training dataset
...: train_dataset = datasets.ImageFolder('../../dataset/train', transform=augment.Compose([
...:         augment.ScalePad((224,224)),
...:         augment.RandomRotate(60),
...:         augment.ToTensor(),
...:         normalize,]))
...:

In [5]: train_dataset.class_to_idx
        class_to_idx   loader()        transform
        classes        root
        imgs           target_transform
        instance
```

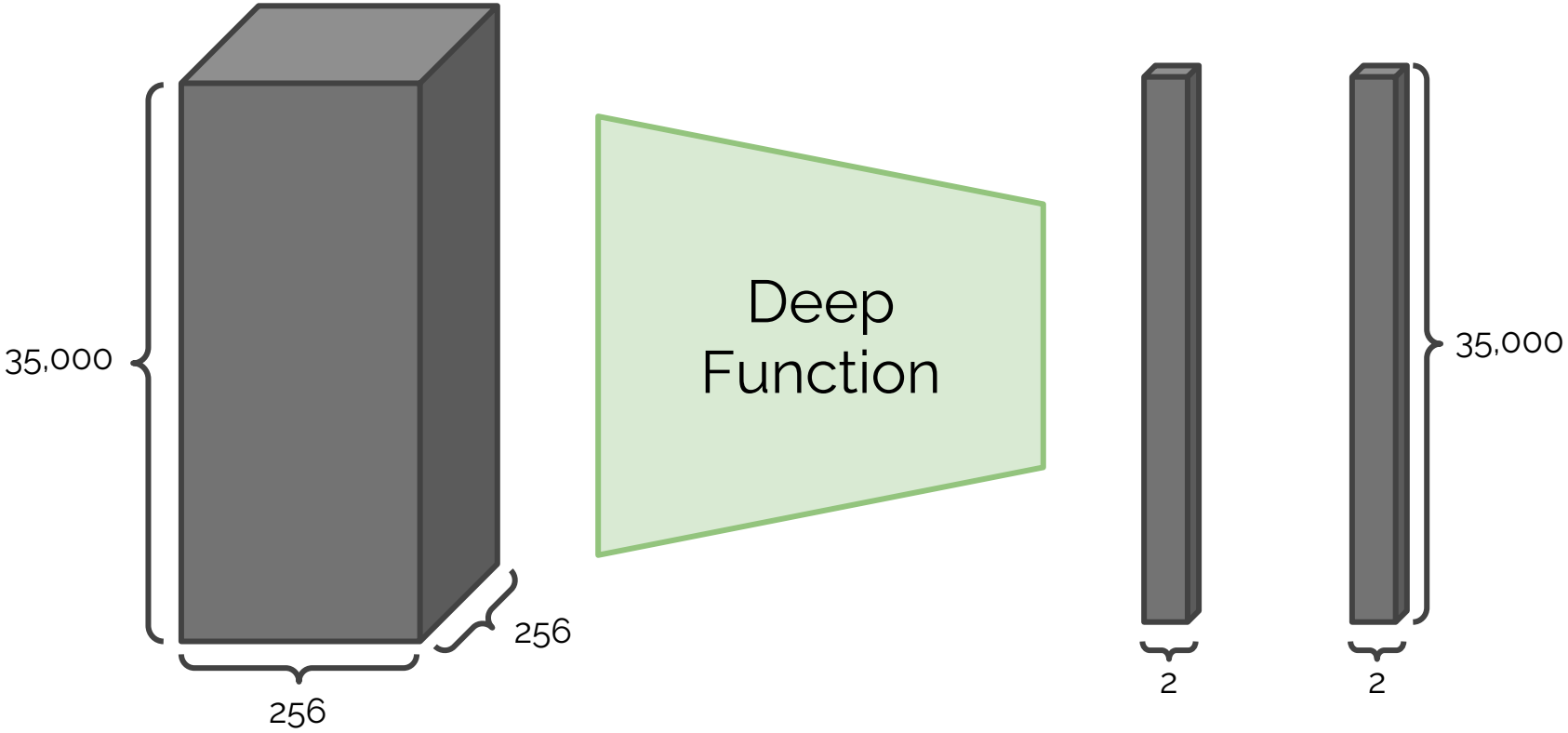
The Dataset

```
...: import nets.resnet as resnet
...: import torch.nn as nn
...: import torch.nn.parallel
...: import torch.backends.cudnn as cudnn
...: import torch.optim.lr_scheduler
...: import torch.utils.trainer as trainer
...: import torch.utils.trainer.plugins
...: import torch.utils.data.sampler as sampler
...: import torchvision.datasets as datasets
...: import torchvision.models as models
...:

In [2]: vis             = visdom.Visdom()
...: run             = '09dec2017'
...: notes           = run+': some notes about what we changed in our last run.'
...: cudnn.benchmark = True
...:

In [3]: ## Normalize images
...: normalize = augment.Normalize(mean = [ 0.485, 0.456, 0.406 ],
...:                               std  = [ 0.229, 0.224, 0.225 ])
...:

In [4]: # Specify the training dataset
...: train_dataset = datasets.ImageFolder('../../dataset/train', transform=augment.Compose([
...:         augment.ScalePad((224,224)),
...:         augment.RandomRotate(60),
...:         augment.ToTensor(),
...:         normalize,]))
...:

In [5]: train_dataset.class_to_idx
Out[5]: {'fish': 0, 'kittens': 1, 'puppies': 2, 'trees': 3, 'wolves': 4}

In [6]: []
```

Classes and their indices

We run out of memory...

35,000 * 3 * 256 * 256 * (32 bits) = **27.52 gigabytes** *not including the model!*



We run out of memory...

Durham University

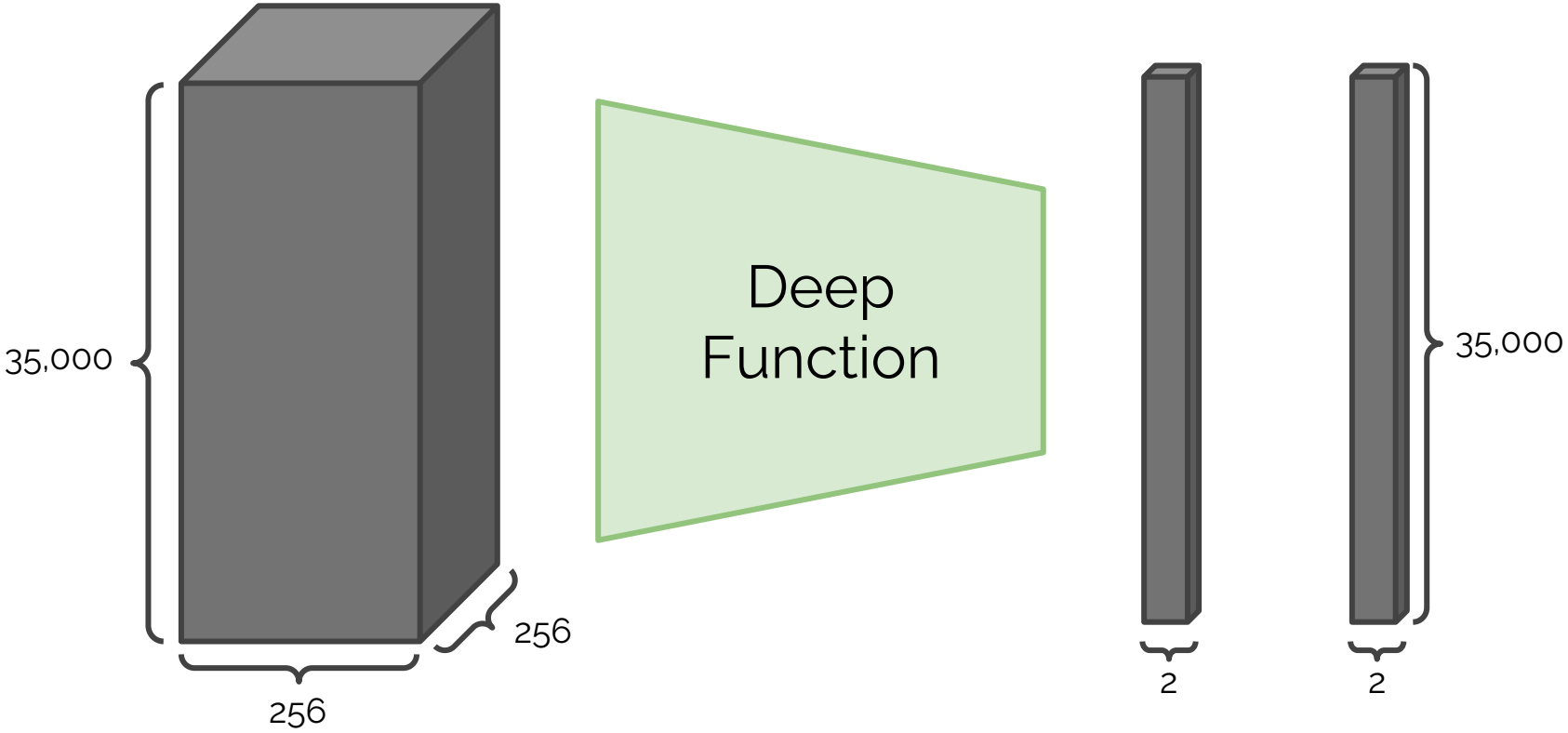
Try and create the tensor

```
(root) ✘ chris@chris-lab ~/repos/deep-learning master • ipython
Python 3.6.2 |Continuum Analytics, Inc.| (default, Jul 20 2017, 13:51:32)
Type 'copyright', 'credits' or 'license' for more information
IPython 6.1.0 -- An enhanced Interactive Python. Type '?' for help.

In [1]: import torch

In [2]: input = torch.zeros(35000,3,256,256)
---------------------------------------------------------------------------
RuntimeError                              Traceback (most recent call last)
<ipython-input-2-8ca165dcec32> in <module>()
----> 1 input = torch.zeros(35000,3,256,256)

RuntimeError: $ Torch: not enough memory: you tried to allocate 25GB. Buy new RAM!
opt/conda/conda-bld/pytorch_1502009910772/work/torch/lib/TH/THGeneral.c:270

In [3]:
```

Nope!

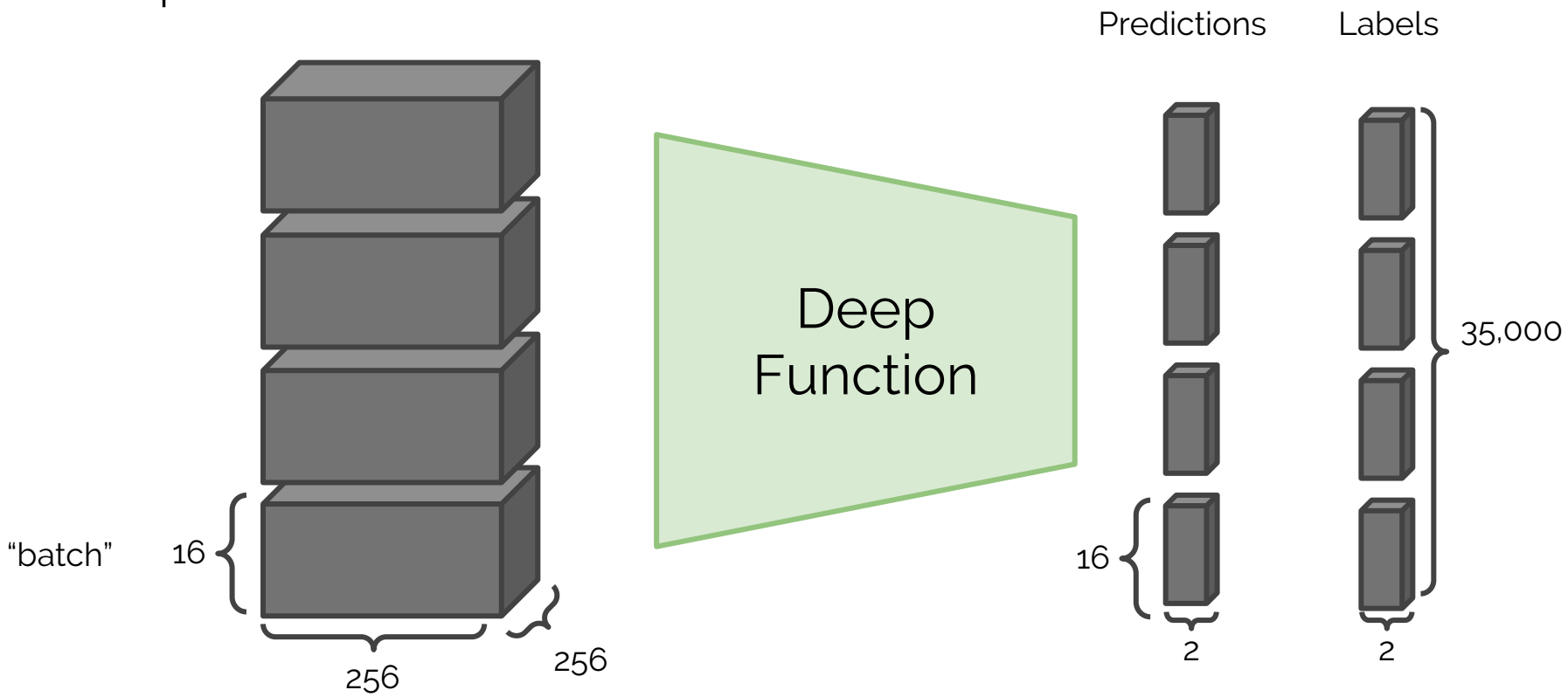
We run out of memory...

35,000 * 256 * 256 * 3 * (32 bits) = **27.52 gigabytes** *not including the model!*



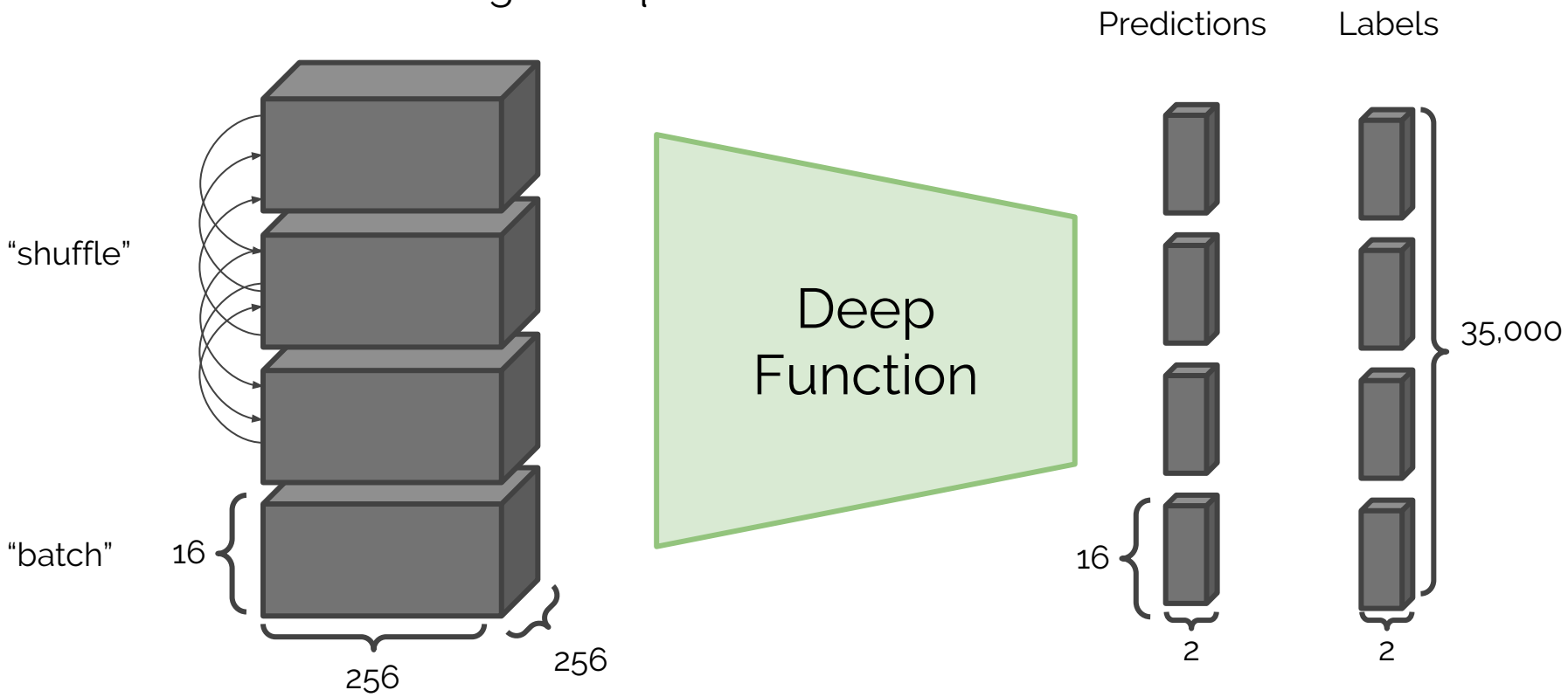
The Data Loader

- Split dataset into mini batches each iteration



The Data Loader

- Shuffle dataset to get unique batches



Mini batches in memory

```
(root) chris@chris-lab ~/repos/deep-learning master ● ipython
Python 3.6.2 |Continuum Analytics, Inc.| (default, Jul 20 2017, 13:51:32)
Type 'copyright', 'credits' or 'license' for more information
IPython 6.1.0 -- An enhanced Interactive Python. Type '?' for help.

In [1]: import torch

In [2]: images = torch.zeros(16,3,256,256)

In [3]: output = torch.zeros(16,2,1,1)

In [4]: images.size()
Out[4]: torch.Size([16, 3, 256, 256])

In [5]: output.size()
Out[5]: torch.Size([16, 2, 1, 1])

In [6]:
```

12.58 megabytes
much better!

The Data Loader

Durham University

Recap, the dataset is not "in memory" we just load a list of filenames and mappings to their class indices

```
...: import imp
...: import util
...: import augment
...: import nets.resnet as resnet
...: import torch.nn as nn
...: import torch.nn.parallel
...: import torch.backends.cudnn as cudnn
...: import torch.optim.lr_scheduler
...: import torch.utils.trainer as trainer
...: import torch.utils.trainer.plugins
...: import torch.utils.data.sampler as sampler
...: import torchvision.datasets as datasets
...: import torchvision.models as models
...:
...: ## Setup Visdom to visualize training results
...: vis             = visdom.Visdom()
...: run             = '09dec2017'
...: notes           = run+': some notes about what we changed in our last run.'
...: cudnn.benchmark = True
...:
...: ## Normalize images
...: normalize = augment.Normalize(mean = [ 0.485, 0.456, 0.406 ],
...:                               std  = [ 0.229, 0.224, 0.225 ])
...:
...: # Specify the training dataset
...: train_dataset = datasets.ImageFolder('../../dataset/train/', transform=augment.Compose([
...:         augment.ScalePad((224,224)),
...:         augment.RandomRotate(60),
...:         augment.ToTensor(),
...:         normalize,]))
...:
...: # Specify the validation dataset
...: val_dataset = datasets.ImageFolder('../../dataset/validation/', transform=augment.Compose([
...:         augment.ScalePad((224,224)),
...:         augment.ToTensor(),
...:         normalize,]))
...:

In [2]:
```

The Data Loader

Durham University

Create a data loader for the train dataset, with a **`batch_size`** of 16 and **`shuffle=True`**

```
...: import torch.nn.parallel
...: import torch.backends.cudnn as cudnn
...: import torch.optim.lr_scheduler
...: import torch.utils.trainer as trainer
...: import torch.utils.trainer.plugins
...: import torch.utils.data.sampler as sampler
...: import torchvision.datasets as datasets
...: import torchvision.models as models
...:
...: ## Setup Visdom to visualize training results
...: vis             = visdom.Visdom()
...: run             = '09dec2017'
...: notes           = run+': some notes about what we changed in our last run.'
...: cudnn.benchmark = True
...:
...: ## Normalize images
...: normalize = augment.Normalize(mean = [ 0.485, 0.456, 0.406 ],
...:                               std  = [ 0.229, 0.224, 0.225 ])
...:
...: # Specify the training dataset
...: train_dataset = datasets.ImageFolder('../../dataset/train/', transform=augment.Compose([
...:         augment.ScalePad((224,224)),
...:         augment.RandomRotate(60),
...:         augment.ToTensor(),
...:         normalize,]))
...:
...: # Specify the validation dataset
...: val_dataset = datasets.ImageFolder('../../dataset/validation/', transform=augment.Compose([
...:         augment.ScalePad((224,224)),
...:         augment.ToTensor(),
...:         normalize,]))
...:

In [2]: # Prepare the data loaders, which get mini-batches of tensors for training
...: train_loader = torch.utils.data.DataLoader(train_dataset,
...:         batch_size=16, shuffle=True,num_workers=12, pin_memory=True)
...:

In [3]:
```

The Data Loader

Durham University

```
...: import torch.utils.trainer.plugins
...: import torch.utils.data.sampler as sampler
...: import torchvision.datasets as datasets
...: import torchvision.models as models
...:
...: ## Setup Visdom to visualize training results
...: vis             = visdom.Visdom()
...: run             = '09dec2017'
...: notes           = run+': some notes about what we changed in our last run.'
...: cudnn.benchmark = True
...:
...: ## Normalize images
...: normalize = augment.Normalize(mean = [ 0.485, 0.456, 0.406 ],
...:                               std  = [ 0.229, 0.224, 0.225 ])
...:
...: # Specify the training dataset
...: train_dataset = datasets.ImageFolder('../../dataset/train/', transform=augment.Compose([
...:         augment.ScalePad((224,224)),
...:         augment.RandomRotate(60),
...:         augment.ToTensor(),
...:         normalize,]))
...:
...: # Specify the validation dataset
...: val_dataset = datasets.ImageFolder('../../dataset/validation/', transform=augment.Compose([
...:         augment.ScalePad((224,224)),
...:         augment.ToTensor(),
...:         normalize,]))
...:

In [2]: # Prepare the data loaders, which get mini-batches of tensors for training
...: train_loader = torch.utils.data.DataLoader(train_dataset,
...:         batch_size=16, shuffle=True,num_workers=12, pin_memory=True)
...:

In [3]: val_loader = torch.utils.data.DataLoader(val_dataset,
...:         batch_size=16, shuffle=False, num_workers=12, pin_memory=True)
...:

In [4]:
```

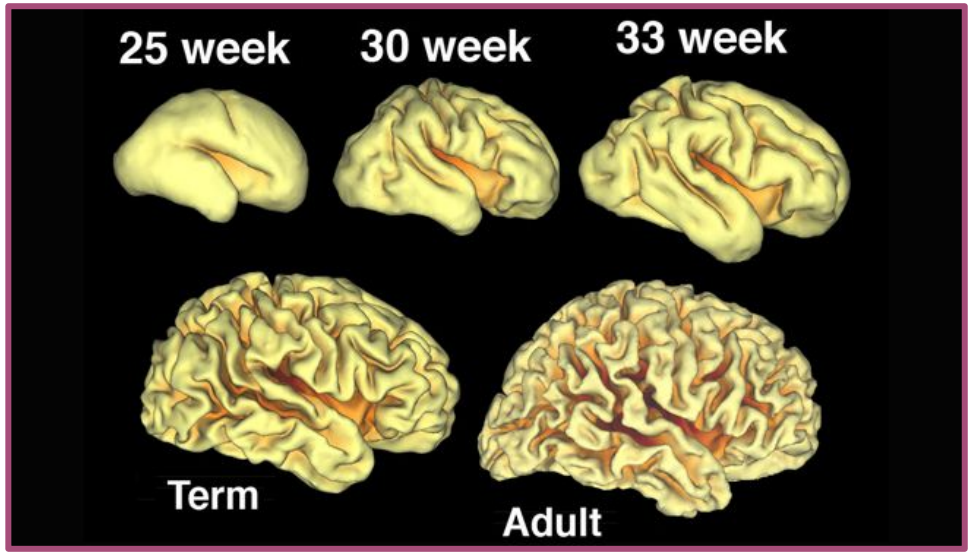
The validation data loader doesn't need to shuffle

Load Model with Transfer Learning

```
# Load the model
model = torch.nn.DataParallel(resnet.resnet34(pretrained=True)).cuda()
```

Nice!

- Is it easier to train a **baby** to detect cancer or an **adult** to detect cancer?
- We train networks on complex tasks with **<u>lots</u>** of public data (even until they outperform humans)
- We then change the data to our tasks, and continue training



Architectures Available

- **Official** architectures (networks) with pre-trained weights:
- Also lots of **unofficial** ones on Github

Branch: **master** ▾ **vision** / torchvision / **models** /

Create new file | Upload files | Find file | History

alykhantejani committed with **fmassa** add stride=1 for average pooling in models to have consistent printin… ··· 3 Latest commit `df7524f` on Oct 22

..

__init__.py	Fix models documentation	2 months ago
alexnet.py	move pre-trained models URL to download.pytorch.org	9 months ago
densenet.py	add stride=1 for average pooling in models to have consistent printin…	2 months ago
inception.py	fix unbroadcastable UserWarning in inception.py (#231)	4 months ago
resnet.py	add stride=1 for average pooling in models to have consistent printin…	2 months ago
squeezenet.py	add stride=1 for average pooling in models to have consistent printin…	2 months ago
vgg.py	Update vgg.py (#233)	3 months ago

Specify the Optimizer algorithm

- Currently, for now, use Adam (most of the time)

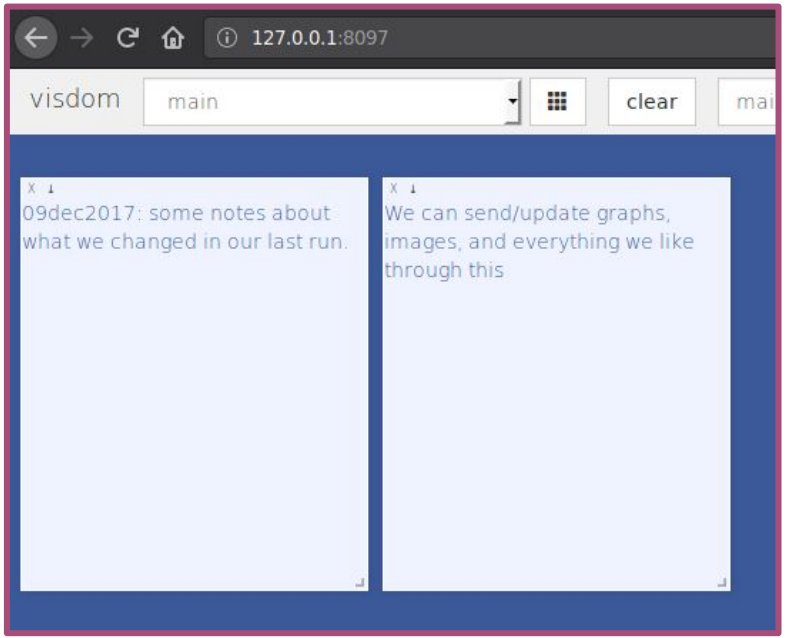
```
In [2]: # Prepare the data loaders, which get mini-batches of tensors for training
   ...: train_loader = torch.utils.data.DataLoader(train_dataset,
   ...:         batch_size=16, shuffle=True,num_workers=12, pin_memory=True)
   ...:

In [3]: val_loader = torch.utils.data.DataLoader(val_dataset,
   ...:         batch_size=16, shuffle=False, num_workers=12, pin_memory=True)
   ...:

In [4]: # Load the model
   ...: model = torch.nn.DataParallel(resnet.resnet34(pretrained=True)).cuda()

In [5]: # Load the optimizer
   ...: optimizer = torch.optim.Adam(model.parameters(), lr=0.00005, betas=(0.5, 0.999), weight_decay
   ...: =0.0002)

In [6]:
```

Set your **learning rate** carefully! (usually first thing you need to adjust)

Train and Collect Metrics

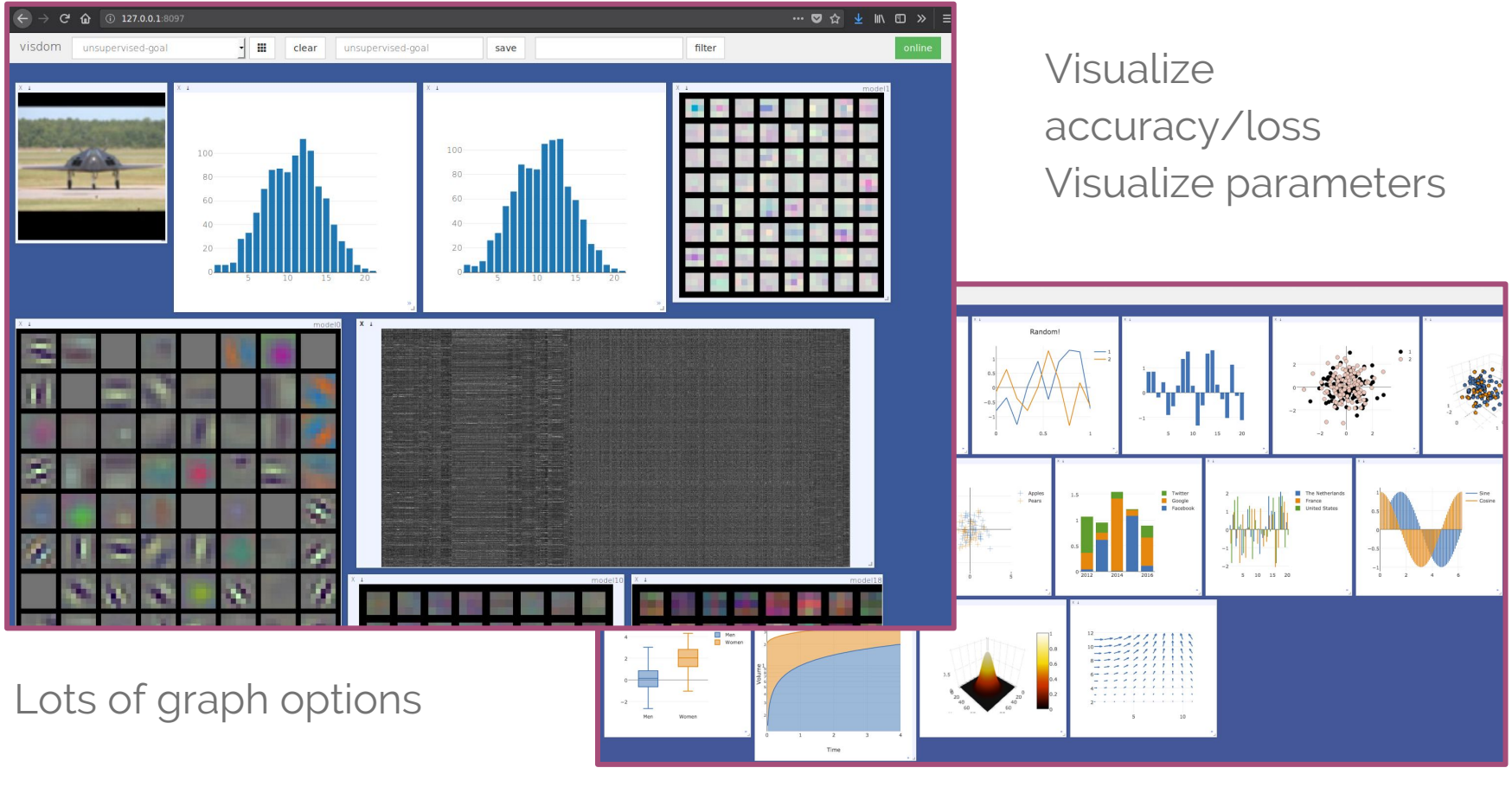
```
(root) ✘ chris@chris-lab ❯ ~/repos/deep-learning ⊬ master ● ❯ python -m visdom.server
It's Alive!
```

```
...:         batch_size=16, shuffle=False, num_workers=12, pin_memory=True)
...:
...: # Load the model
...: model = torch.nn.DataParallel(resnet.resnet34(pretrained=True)).cuda()
...:
...: # Load the optimizer
...: optimizer = torch.optim.Adam(model.parameters(), lr=0.00005, betas=(0.5, 0.999), weight_decay=0.000
...: 2)
...:
...: # Specify some measurements that we wish to take during training
...: chk_acc = 0
...: plt_loss_sum = 0
...: plt_count = 0
...: plt_loss = np.empty(0, dtype='float32')
...: plt_acc = np.empty(0, dtype='float32')
...: plt_val_loss = np.empty(0, dtype='float32')
...: plt_val_acc  = np.empty(0, dtype='float32')
...: epoch = 0
...:
...: # Function for cleanly exiting
...: def signal_handler(signal, frame):
...:     global interrupted
...:     interrupted = True
...:
...: # Show some notes
...: vis.text(notes)
...: signal.signal(signal.SIGINT, signal_handler)
...: interrupted = False
...:

In [2]: vis.text("We can send/update graphs, images, and everything we like through this")
Out[2]: 'pane_35be5596041be6'

In [3]: vis.
bar()        endpoint     image()      mesh()       quiver()     stem()       updateTrace()
boxplot()    env          images()     pie()        save()       surf()       video()
close()      heatmap()    ipv6         port         scatter()    svg()
contour()    histogram()  line()       proxy        server       text()
```



Visdom

Durham University



Visualize
accuracy/loss
Visualize parameters

Lots of graph options

Training

Durham University

```python
while(True):

    model.train()
    tmp_correct = 0
    for i, (input, target) in enumerate(train_loader):

        input, target = input.cuda(), target.cuda()
        input, target = torch.autograd.Variable(input), torch.autograd.Variable(target)
        model.zero_grad()

        # Feed the input tensor through the model
        pred = model(input)
        loss = torch.nn.functional.cross_entropy(pred, target)
        loss.backward()
        optimizer.step()

        # Collect statistics
        plt_loss_sum += loss.data[0]
        plt_count += 1
        pred = f.data.max(1)[1]
        correct = pred.eq(target.data).long()
        tmp_correct += correct.cpu().sum()

        if interrupted:
            print("\nexiting safely inner loop")
            break

    train_acc = 100. * tmp_correct / len(train_dataset.imgs)
```

Training

Durham University

```
while(True):

    model.train()
    tmp_correct = 0
    for i, (input, target) in enumerate(train_loader):

        input, target = input.cuda(), target.cuda()
        input, target = torch.autograd.Variable(input), torch.autograd.Variable(target)
        model.zero_grad()

        # Feed the input tensor through the model
        pred = model(input)
        loss = torch.nn.functional.cross_entropy(pred, target)
        loss.backward()
        optimizer.step()

        # Collect statistics
        plt_loss_sum += loss.data[0]
        plt_count += 1
        pred = f.data.max(1)[1]
        correct = pred.eq(target.data).long()
        tmp_correct += correct.cpu().sum()

        if interrupted:
            print("\nexiting safely inner loop")
            break

    train_acc = 100. * tmp_correct / len(train_dataset.imgs)
```

Repeat forever (until user terminates)

Training

Durham University

EXPLORER

learning.py ● main.py

OPEN EDITORS 1 UNSAVED
- learning.py st... U
- main.py stati... M

DEEP-LEARNING
- .vscode
- configs
- dataset
- papers
- projects
- static
  - .vscode
  - css
  - fonts
  - img
  - js
  - model
- templates
- .gitignore
- .jsbeautifyrc
- license
- readme.md
- server.py
- uwsgi.ini

Specify we can change model weights

```python
while(True):

    model.train()
    tmp_correct = 0
    for i, (input, target) in enumerate(train_loader):

        input, target = input.cuda(), target.cuda()
        input, target = torch.autograd.Variable(input), torch.autograd.Variable(target)
        model.zero_grad()

        # Feed the input tensor through the model
        pred = model(input)
        loss = torch.nn.functional.cross_entropy(pred, target)
        loss.backward()
        optimizer.step()

        # Collect statistics
        plt_loss_sum += loss.data[0]
        plt_count += 1
        pred = f.data.max(1)[1]
        correct = pred.eq(target.data).long()
        tmp_correct += correct.cpu().sum()

        if interrupted:
            print("\nexiting safely inner loop")
            break

    train_acc = 100. * tmp_correct / len(train_dataset.imgs)
```

ter* 0 0 103 python | learning.py Python 3.6.3 Ln 78, Col 10 Spaces: 4 UTF-8 LF Python

Training

Durham University

Iterate using data loader, returning mini-batches

```
while(True):

    model.train()
    tmp_correct = 0
    for i, (input, target) in enumerate(train_loader):

        input, target = input.cuda(), target.cuda()
        input, target = torch.autograd.Variable(input), torch.autograd.Variable(target)
        model.zero_grad()

        # Feed the input tensor through the model
        pred = model(input)
        loss = torch.nn.functional.cross_entropy(pred, target)
        loss.backward()
        optimizer.step()

        # Collect statistics
        plt_loss_sum += loss.data[0]
        plt_count += 1
        pred = f.data.max(1)[1]
        correct = pred.eq(target.data).long()
        tmp_correct += correct.cpu().sum()

        if interrupted:
            print("\nexiting safely inner loop")
            break

    train_acc = 100. * tmp_correct / len(train_dataset.imgs)
```

Training

Durham University

torch.Size([16,3,256,256])
[Batch Size, Channels, Height, Width]

```python
while(True):

    model.train()
    tmp_correct = 0
    for i, (input, target) in enumerate(train_loader):

        input, target = input.cuda(), target.cuda()
        input, target = torch.autograd.Variable(input), torch.autograd.Variable(target)
        model.zero_grad()

        # Feed the input tensor through the model
        pred = model(input)
        loss = torch.nn.functional.cross_entropy(pred, target)
        loss.backward()
        optimizer.step()

        # Collect statistics
        plt_loss_sum += loss.data[0]
        plt_count += 1
        pred = f.data.max(1)[1]
        correct = pred.eq(target.data).long()
        tmp_correct += correct.cpu().sum()

        if interrupted:
            print("\nexiting safely inner loop")
            break

    train_acc = 100. * tmp_correct / len(train_dataset.imgs)
```

Training

Durham University

```
while(True):

    model.train()
    tmp_correct = 0
    for i, (input, target) in enumerate(train_loader):

        input, target = input.cuda(), target.cuda()
        input, target = torch.autograd.Variable(input), torch.autograd.Variable(target)
        model.zero_grad()

        # Feed the input tensor through the model
        pred = model(input)
        loss = torch.nn.functional.cross_entropy(pred, target)
        loss.backward()
        optimizer.step()

        # Collect statistics
        plt_loss_sum += loss.data[0]
        plt_count += 1
        pred = f.data.max(1)[1]
        correct = pred.eq(target.data).long()
        tmp_correct += correct.cpu().sum()

        if interrupted:
            print("\nexiting safely inner loop")
            break

    train_acc = 100. * tmp_correct / len(train_dataset.imgs)
```

`torch.Size([16,5])`

[Batch Size, Classes]

Training

Durham University

EXPLORER

OPEN EDITORS 1 UNSAVED
- learning.py st... U
- main.py stati... M

DEEP-LEARNING
- .vscode
- configs
- dataset
- papers
- projects
- static
  - .vscode
  - css
  - fonts
  - img
  - js
  - model
- templates
- .gitignore
- .jsbeautifyrc
- license
- readme.md
- server.py
- uwsgi.ini

learning.py ● main.py

```python
while(True):

    model.train()
    tmp_correct = 0
    for i, (input, target) in enumerate(train_loader):

        input, target = input.cuda(), target.cuda()
        input, target = torch.autograd.Variable(input), torch.autograd.Variable(target)
        model.zero_grad()

        # Feed the input tensor through the model
        pred = model(input)
        loss = torch.nn.functional.cross_entropy(pred, target)
        loss.backward()
        optimizer.step()

        # Collect statistics
        plt_loss_sum += loss.data[0]
        plt_count += 1
        pred = f.data.max(1)[1]
        correct = pred.eq(target.data).long()
        tmp_correct += correct.cpu().sum()

        if interrupted:
            print("\nexiting safely inner loop")
            break

    train_acc = 100. * tmp_correct / len(train_dataset.imgs)
```

torch.Size([16,5])

CUDA float tensor containing:

```
[ 1.58, -4.15,  3.21, -5.13, -7.2,
  5.25, -5.11,  1.70, -7.27, -3.9,
 -2.12,  3.92, -1.23, -4.19, -3.2
 -1.01,  0.19,  3.22, -1.01, -2.0,
 -5.24,  1.95,  3.14, -1.95,  5.8,
  0.82,  1.78, -2.42,  3.10, -2.5,
  1.75,  4.21,  5.42, -2.99, -4.1,
                  ...
  3.21,  4.12, -3.92, -0.23, -3.19]
```

ter* 0 0 103 python | learning.py Python 3.6.3 Ln 78, Col 10 Spaces: 4 UTF-8 LF Python

Training

Durham University

EXPLORER

learning.py ● main.py

OPEN EDITORS 1 UNSAVED
- learning.py st... U
- main.py stati... M

DEEP-LEARNING
- .vscode
- configs
- dataset
- papers
- projects
- static
  - .vscode
  - css
  - fonts
  - img
  - js
  - model
- templates
- .gitignore
- .jsbeautifyrc
- license
- readme.md
- server.py
- uwsgi.ini

```python

while(True):

    model.train()
    tmp_correct = 0
    for i, (input, target) in enumerate(train_loader):

        input, target = input.cuda(), target.cuda()
        input, target = torch.autograd.Variable(input), torch.autograd.Variable(target)
        model.zero_grad()

        # Feed the input tensor through the model
        pred = model(input)
        loss = torch.nn.functional.cross_entropy(pred, target)
        loss.backward()
        optimizer.step()

        # Collect statistics
        plt_loss_sum += loss.data[0]
        plt_count += 1
        pred = f.data.max(1)[1]
        correct = pred.eq(target.data).long()
        tmp_correct += correct.cpu().sum()

        if interrupted:
            print("\nexiting safely inner loop")
            break

    train_acc = 100. * tmp_correct / len(train_dataset.imgs)
```

7.1766209602355

ter* 0 0 103 python | learning.py Python 3.6.3 Ln 78, Col 10 Spaces: 4 UTF-8 LF Python

Training

```python
while(True):

    model.train()
    tmp_correct = 0
    for i, (input, target) in enumerate(train_loader):

        input, target = input.cuda(), target.cuda()
        input, target = torch.autograd.Variable(input), torch.autograd.Variable(target)
        model.zero_grad()

        # Feed the input tensor through the model
        pred = model(input)
        loss = torch.nn.functional.cross_entropy(pred, target)
        loss.backward()
        optimizer.step()

        # Collect statistics
        plt_loss_sum += loss.data[0]
        plt_count += 1
        pred = f.data.max(1)[1]
        correct = pred.eq(target.data).long()
        tmp_correct += correct.cpu().sum()

        if interrupted:
            print("\nexiting safely inner loop")
            break

    train_acc = 100. * tmp_correct / len(train_dataset.imgs)
```

Backpropagate to recursively differentiate the loss with respect to each layer and parameter

...so we know whether we need to increase or decrease each parameter to decrease the error

Training

```python
while(True):

    model.train()
    tmp_correct = 0
    for i, (input, target) in enumerate(train_loader):

        input, target = input.cuda(), target.cuda()
        input, target = torch.autograd.Variable(input), torch.autograd.Variable(target)
        model.zero_grad()

        # Feed the input tensor through the model
        pred = model(input)
        loss = torch.nn.functional.cross_entropy(pred, target)
        loss.backward()
        optimizer.step()

        # Collect statistics
        plt_loss_sum += loss.data[0]
        plt_count += 1
        pred = f.data.max(1)[1]
        correct = pred.eq(target.data).long()
        tmp_correct += correct.cpu().sum()

        if interrupted:
            print("\nexiting safely inner loop")
            break

    train_acc = 100. * tmp_correct / len(train_dataset.imgs)
```

Tweak deep function parameters in direction of gradient by very small amount (based on learning rate) to decrease the error

Training

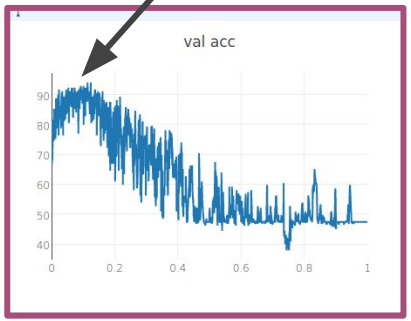
```
while(True):

    model.train()
    tmp_correct = 0
    for i, (input, target) in enumerate(train_loader):

        input, target = input.cuda(), target.cuda()
        input, target = torch.autograd.Variable(input), torch.autograd.Variable(target)
        model.zero_grad()

        # Feed the input tensor through the model
        pred = model(input)
        loss = torch.nn.functional.cross_entropy(pred, target)
        loss.backward()
        optimizer.step()

        # Collect statistics
        plt_loss_sum += loss.data[0]
        plt_count += 1
        pred = f.data.max(1)[1]
        correct = pred.eq(target.data).long()
        tmp_correct += correct.cpu().sum()

        if interrupted:
            print("\nexiting safely inner loop")
            break

    train_acc = 100. * tmp_correct / len(train_dataset.imgs)
```

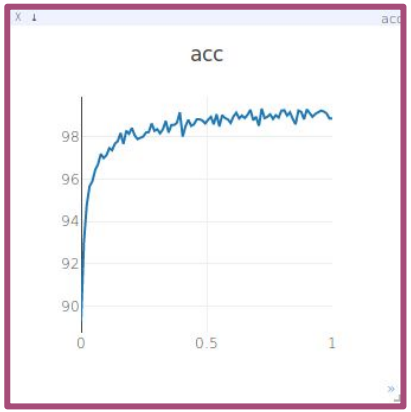
Predicted **class** is arg max of model predictions:

```
[ 2 → "puppy"
  0 → "fish"
  1 → "kitten"
  2 → "puppy"
  4 → "wolf"
  3 → "tree"
  2 → "puppy"
        ...
  1 → "kitten" ]
```

...eval

Checkpoint!





EXPLORER | main.py | learning.py

OPEN EDITORS 1 UNSAVED
- main.py stati... M
- learning.py st... U

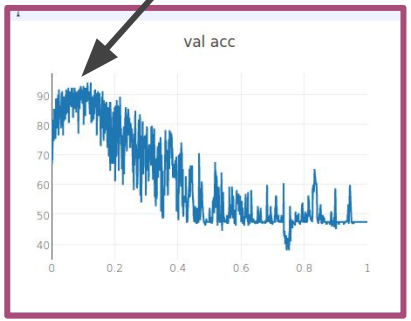
DEEP-LEARNING
- .vscode
- configs
- dataset
- papers
- projects
- static
  - .vscode
  - css
  - fonts
  - img
  - js
  - model
- templates
- .gitignore
- .jsbeautifyrc
- license
- readme.md
- server.py
- uwsgi.ini

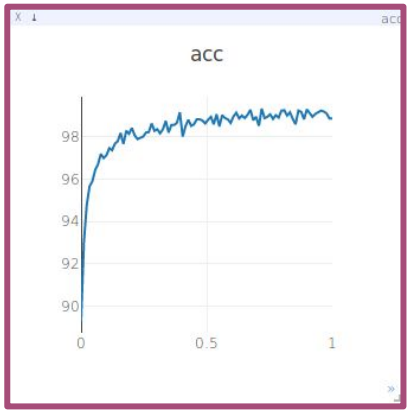
```
113     model.eval()
114     tmp_loss = 0
115     tmp_correct = 0
116     for i, (input, target) in enumerate(val_loader):
117
118         input, target = input.cuda(), target.cuda()
119         input, target = torch.autograd.Variable(input, volatile=True), torch.autograd.Variable(target)
120
121         model.zero_grad()
122         pred = model(input)
123
124         tmp_loss += torch.nn.functional.cross_entropy(pred, target, size_average=False).data[0]
125         pred = f.data.max(1)[1]
126         correct = pred.eq(target.data).long()
127         tmp_correct += correct.cpu().sum()
128
129         if interrupted:
130             print("\nexiting safely inner loop")
131             break
132
133     if interrupted:
134         print("\nexiting safely outermost loop")
135         interrupted = False
136         break
137
138     epoch += 1
139     val_loss = tmp_loss / len(val_dataset.imgs)
140     val_acc = 100. * tmp_correct / len(val_dataset.imgs)
141     plt_loss_mean = plt_loss_sum / plt_count
142     plt_loss_sum = 0
143     plt_count = 0
144
145     plt_loss = np.append(plt_loss, plt_loss_mean)
146     plt_acc = np.append(plt_acc, train_acc)
147     plt_val_loss = np.append(plt_val_loss, val_loss)
148     plt_val_acc = np.append(plt_val_acc, val_acc)
149     vis.line(np.append(plt_loss, plt_loss_mean), win='loss', opts={'title': 'loss'})
150     vis.line(np.append(plt_acc, train_acc), win='tacc', opts={'title': 'acc'})
151     vis.line(np.append(plt_val_loss, val_loss), win='vloss', opts={'title': 'val loss'})
152     vis.line(np.append(plt_val_acc, val_acc), win='vacc', opts={'title': 'val acc'})
153
154     if val_acc > chk_acc:
155         print('Saving Checkpoint: '+str(val_acc))
156         chk_acc = val_acc
157         torch.save({
158             'epoch' : epoch,
159             'run' : run,
160             'notes' : notes,
161             'model_params' : model.state_dict(),
162             'plt_loss' : plt_loss,
163             'plt_acc' : plt_acc,
```

Parameters are locked

...eval

Checkpoint!





EXPLORER | main.py | learning.py

OPEN EDITORS 1 UNSAVED
- main.py stati... M
- learning.py st... U

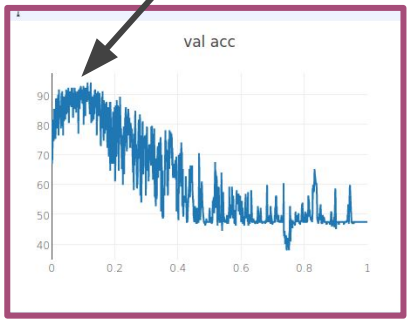
DEEP-LEARNING
- .vscode
- configs
- dataset
- papers
- projects
- static
  - .vscode
  - css
  - fonts
  - img
  - js
  - model
- templates
- .gitignore
- .jsbeautifyrc
- license
- readme.md
- server.py
- uwsgi.ini

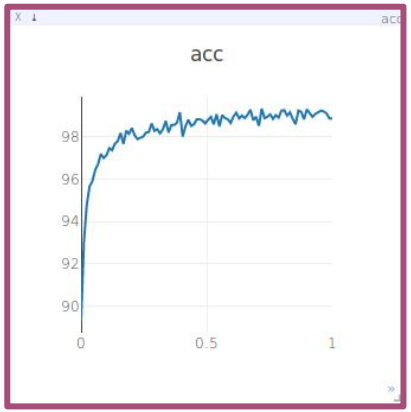
```python
    model.eval()
    tmp_loss = 0
    tmp_correct = 0
    for i, (input, target) in enumerate(val_loader):

        input, target = input.cuda(), target.cuda()
        input, target = torch.autograd.Variable(input, volatile=True), torch.autograd.Variable(target)

        model.zero_grad()
        pred = model(input)

        tmp_loss += torch.nn.functional.cross_entropy(pred, target, size_average=False).data[0]
        pred = f.data.max(1)[1]
        correct = pred.eq(target.data).long()
        tmp_correct += correct.cpu().sum()

        if interrupted:
            print("\nexiting safely inner loop")
            break

    if interrupted:
        print("\nexiting safely outermost loop")
        interrupted = False
        break

    epoch += 1
    val_loss = tmp_loss / len(val_dataset.imgs)
    val_acc = 100. * tmp_correct / len(val_dataset.imgs)
    plt_loss_mean = plt_loss_sum / plt_count
    plt_loss_sum = 0
    plt_count = 0

    plt_loss = np.append(plt_loss, plt_loss_mean)
    plt_acc = np.append(plt_acc, train_acc)
    plt_val_loss = np.append(plt_val_loss, val_loss)
    plt_val_acc = np.append(plt_val_acc, val_acc)
    vis.line(np.append(plt_loss, plt_loss_mean), win='loss', opts={'title': 'loss'})
    vis.line(np.append(plt_acc, train_acc), win='tacc', opts={'title': 'acc'})
    vis.line(np.append(plt_val_loss, val_loss), win='vloss', opts={'title': 'val loss'})
    vis.line(np.append(plt_val_acc, val_acc), win='vacc', opts={'title': 'val acc'})

    if val_acc > chk_acc:
        print('Saving Checkpoint: '+str(val_acc))
        chk_acc = val_acc
        torch.save({
            'epoch' : epoch,
            'run' : run,
            'notes' : notes,
            'model_params' : model.state_dict(),
            'plt_loss' : plt_loss,
            'plt_acc' : plt_acc,
```

Validation dataset

...eval

Checkpoint!





EXPLORER | main.py | learning.py

OPEN EDITORS 1 UNSAVED
- main.py stati... M
- learning.py st... U

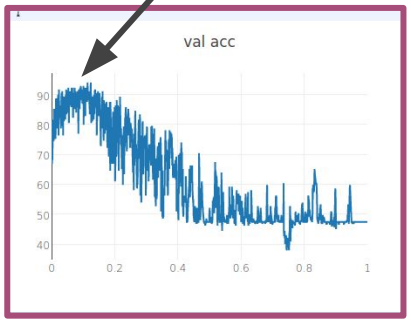
DEEP-LEARNING
- .vscode
- configs
- dataset
- papers
- projects
- static
  - .vscode
  - css
  - fonts
  - img
  - js
  - model
- templates
- .gitignore
- .jsbeautifyrc
- license
- readme.md
- server.py
- uwsgi.ini

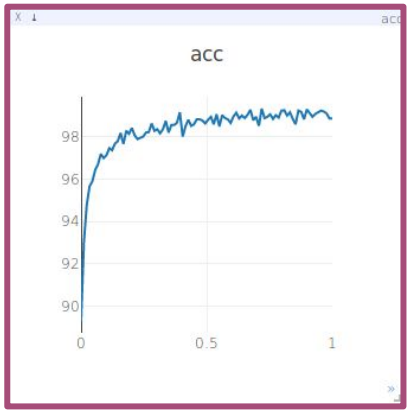
```python
    model.eval()
    tmp_loss = 0
    tmp_correct = 0
    for i, (input, target) in enumerate(val_loader):

        input, target = input.cuda(), target.cuda()
        input, target = torch.autograd.Variable(input, volatile=True), torch.autograd.Variable(target)

        model.zero_grad()
        pred = model(input)

        tmp_loss += torch.nn.functional.cross_entropy(pred, target, size_average=False).data[0]
        pred = f.data.max(1)[1]
        correct = pred.eq(target.data).long()
        tmp_correct += correct.cpu().sum()

        if interrupted:
            print("\nexiting safely inner loop")
            break

    if interrupted:
        print("\nexiting safely outermost loop")
        interrupted = False
        break

    epoch += 1
    val_loss = tmp_loss / len(val_dataset.imgs)
    val_acc = 100. * tmp_correct / len(val_dataset.imgs)
    plt_loss_mean = plt_loss_sum / plt_count
    plt_loss_sum = 0
    plt_count = 0

    plt_loss = np.append(plt_loss, plt_loss_mean)
    plt_acc = np.append(plt_acc, train_acc)
    plt_val_loss = np.append(plt_val_loss, val_loss)
    plt_val_acc = np.append(plt_val_acc, val_acc)
    vis.line(np.append(plt_loss, plt_loss_mean), win='loss', opts={'title': 'loss'})
    vis.line(np.append(plt_acc, train_acc), win='tacc', opts={'title': 'acc'})
    vis.line(np.append(plt_val_loss, val_loss), win='vloss', opts={'title': 'val loss'})
    vis.line(np.append(plt_val_acc, val_acc), win='vacc', opts={'title': 'val acc'})

    if val_acc > chk_acc:
        print('Saving Checkpoint: '+str(val_acc))
        chk_acc = val_acc
        torch.save({
            'epoch' : epoch,
            'run' : run,
            'notes' : notes,
            'model_params' : model.state_dict(),
            'plt_loss' : plt_loss,
            'plt_acc' : plt_acc,
```

Same again, but no backpropagation or optimization needed

...eval

Checkpoint!





EXPLORER | main.py | learning.py

OPEN EDITORS 1 UNSAVED
- main.py stati... M
- learning.py st... U

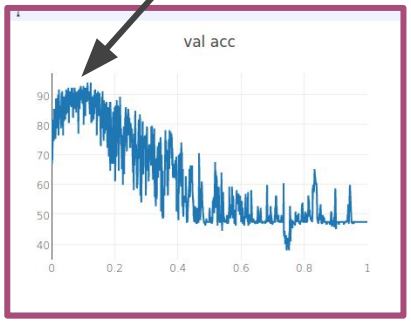
DEEP-LEARNING
- .vscode
- configs
- dataset
- papers
- projects
- static
  - .vscode
  - css
  - fonts
  - img
  - js
  - model
- templates
- .gitignore
- .jsbeautifyrc
- license
- readme.md
- server.py
- uwsgi.ini

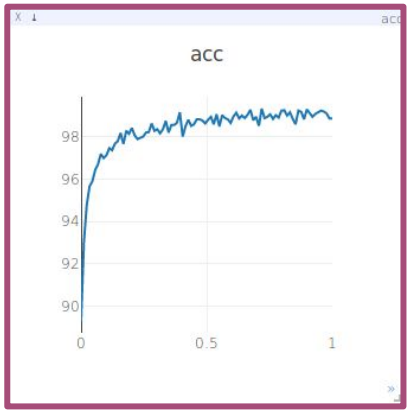
```python
    model.eval()
    tmp_loss = 0
    tmp_correct = 0
    for i, (input, target) in enumerate(val_loader):

        input, target = input.cuda(), target.cuda()
        input, target = torch.autograd.Variable(input, volatile=True), torch.autograd.Variable(target)

        model.zero_grad()
        pred = model(input)

        tmp_loss += torch.nn.functional.cross_entropy(pred, target, size_average=False).data[0]
        pred = f.data.max(1)[1]
        correct = pred.eq(target.data).long()
        tmp_correct += correct.cpu().sum()

        if interrupted:
            print("\nexiting safely inner loop")
            break

    if interrupted:
        print("\nexiting safely outermost loop")
        interrupted = False
        break

    epoch += 1
    val_loss = tmp_loss / len(val_dataset.imgs)
    val_acc = 100. * tmp_correct / len(val_dataset.imgs)
    plt_loss_mean = plt_loss_sum / plt_count
    plt_loss_sum = 0
    plt_count = 0

    plt_loss = np.append(plt_loss, plt_loss_mean)
    plt_acc = np.append(plt_acc, train_acc)
    plt_val_loss = np.append(plt_val_loss, val_loss)
    plt_val_acc = np.append(plt_val_acc, val_acc)
    vis.line(np.append(plt_loss, plt_loss_mean), win='loss', opts={'title': 'loss'})
    vis.line(np.append(plt_acc, train_acc), win='tacc', opts={'title': 'acc'})
    vis.line(np.append(plt_val_loss, val_loss), win='vloss', opts={'title': 'val loss'})
    vis.line(np.append(plt_val_acc, val_acc), win='vacc', opts={'title': 'val acc'})

    if val_acc > chk_acc:
        print('Saving Checkpoint: '+str(val_acc))
        chk_acc = val_acc
        torch.save({
            'epoch' : epoch,
            'run' : run,
            'notes' : notes,
            'model_params' : model.state_dict(),
            'plt_loss' : plt_loss,
            'plt_acc' : plt_acc,
```

Plot loss, accuracy and validation accuracy in visdom

...eval

Checkpoint!





EXPLORER | main.py | learning.py

OPEN EDITORS 1 UNSAVED
- main.py stati... M
- learning.py st... U

DEEP-LEARNING
- .vscode
- configs
- dataset
- papers
- projects
- static
  - .vscode
  - css
  - fonts
  - img
  - js
  - model
- templates
- .gitignore
- .jsbeautifyrc
- license
- readme.md
- server.py
- uwsgi.ini

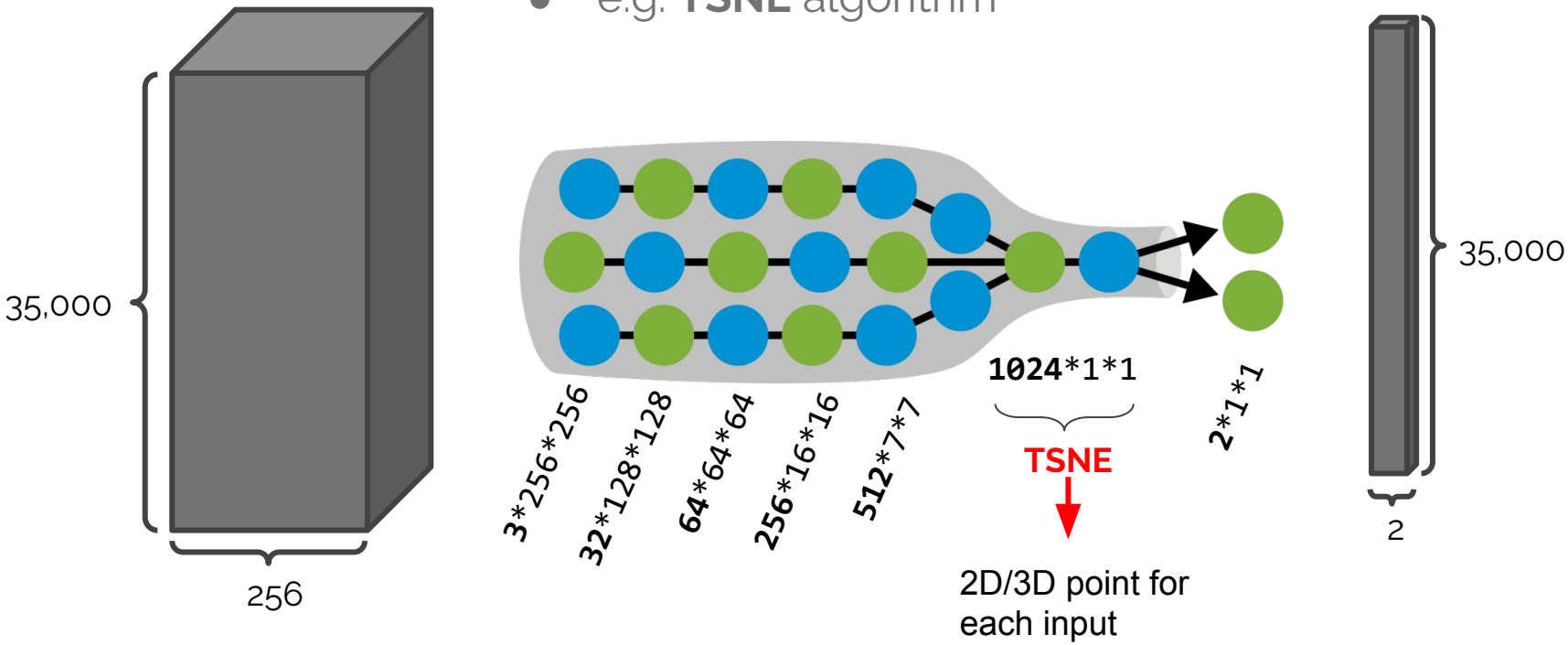
```python
    model.eval()
    tmp_loss = 0
    tmp_correct = 0
    for i, (input, target) in enumerate(val_loader):

        input, target = input.cuda(), target.cuda()
        input, target = torch.autograd.Variable(input, volatile=True), torch.autograd.Variable(target)

        model.zero_grad()
        pred = model(input)

        tmp_loss += torch.nn.functional.cross_entropy(pred, target, size_average=False).data[0]
        pred = f.data.max(1)[1]
        correct = pred.eq(target.data).long()
        tmp_correct += correct.cpu().sum()

        if interrupted:
            print("\nexiting safely inner loop")
            break

    if interrupted:
        print("\nexiting safely outermost loop")
        interrupted = False
        break

    epoch += 1
    val_loss = tmp_loss / len(val_dataset.imgs)
    val_acc = 100. * tmp_correct / len(val_dataset.imgs)
    plt_loss_mean = plt_loss_sum / plt_count
    plt_loss_sum = 0
    plt_count = 0

    plt_loss = np.append(plt_loss, plt_loss_mean)
    plt_acc = np.append(plt_acc, train_acc)
    plt_val_loss = np.append(plt_val_loss, val_loss)
    plt_val_acc = np.append(plt_val_acc, val_acc)
    vis.line(np.append(plt_loss, plt_loss_mean), win='loss', opts={'title': 'loss'})
    vis.line(np.append(plt_acc, train_acc), win='tacc', opts={'title': 'acc'})
    vis.line(np.append(plt_val_loss, val_loss), win='vloss', opts={'title': 'val loss'})
    vis.line(np.append(plt_val_acc, val_acc), win='vacc', opts={'title': 'val acc'})

    if val_acc > chk_acc:
        print('Saving Checkpoint: '+str(val_acc))
        chk_acc = val_acc
        torch.save({
            'epoch' : epoch,
            'run' : run,
            'notes' : notes,
            'model_params' : model.state_dict(),
            'plt_loss' : plt_loss,
            'plt_acc' : plt_acc,
```

Save checkpoint if you get a new best validation accuracy

Dimensionality Reduction & Clustering

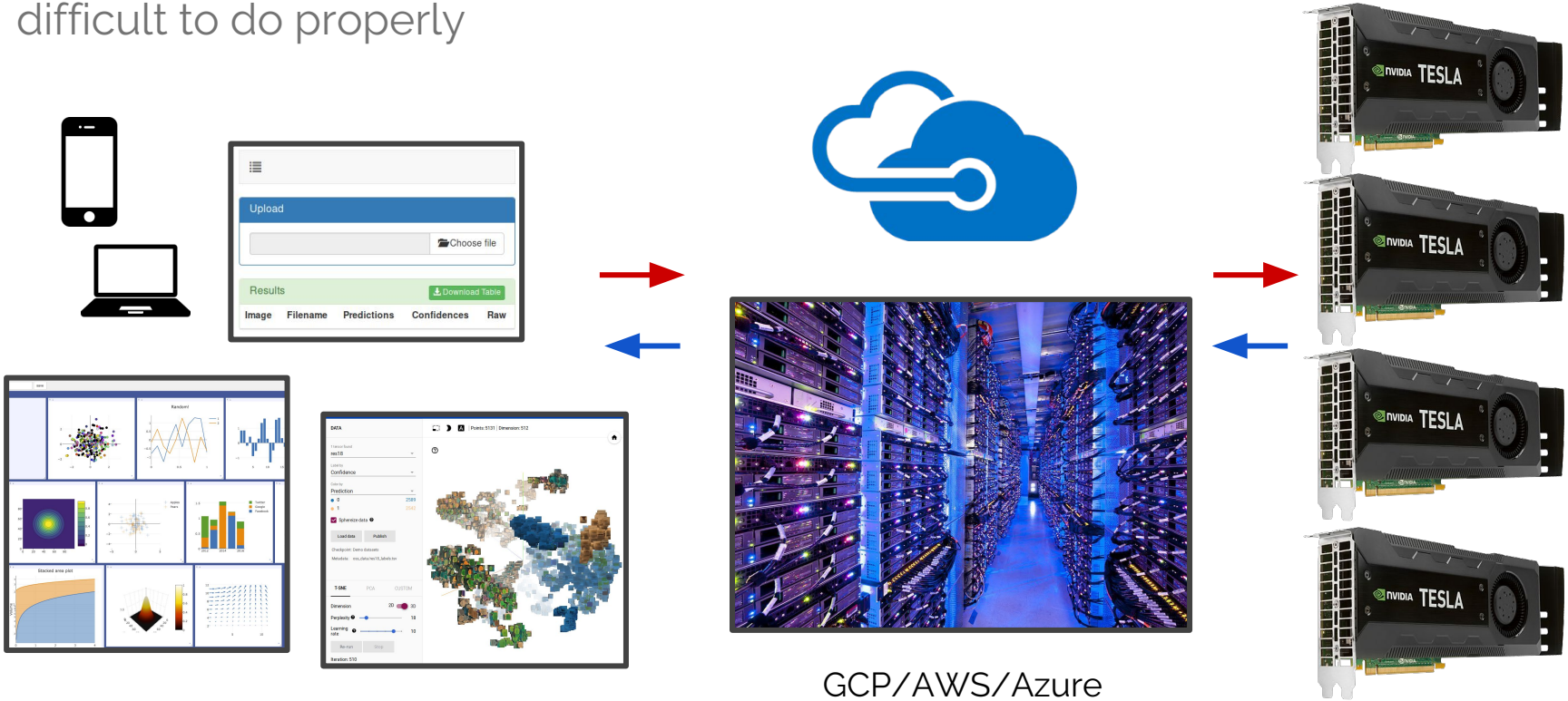
We reduce the dimensionality of the bottleneck to a 2D or 3D space

- e.g. **TSNE** algorithm



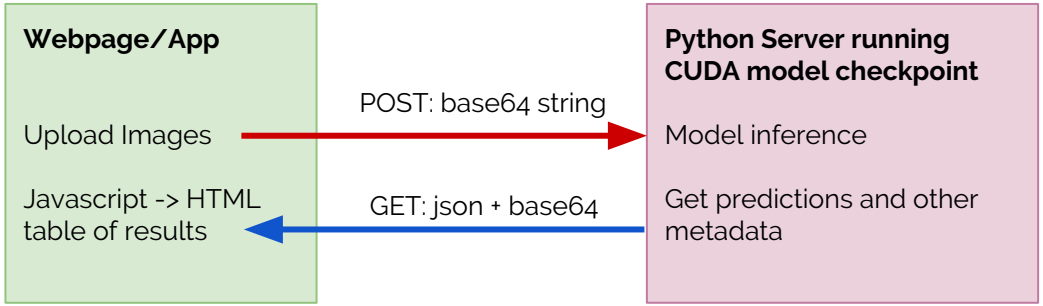
Deployment

Deployment of models is something easy to do badly and difficult to do properly



Proposed Stack

- Cross-platform
  - Runs on phone/browser
  - Upload entire folders at once
  - Download CSV results
- HTML5/CSS3
  - Javascript
    - AJAX
      - Flask
        - PyTorch
- Security
  - Password protected
  - No files needlessly stored on disk
  - Audit logs



- Simple architecture
  - **No PHP** server code, **no files stored on disk**, very simple codebase, all asynchronous responsive, no loops checking for signals etc
  - **very easy** to extend to handle new project requirements:
    - GPS, heatmap, d3 visualisations of model…

Server in 60 Lines

Load model at start

Queue access from different threads to GPU

Serve website

Client

```
$(document).ready(function() {

    function setupReader(file) {
        var name = file.name;
        var reader = new FileReader();
        reader.onload = function(e) {
            var fileName = file.name;
        $.post('/upload-image', { data: reader.result, name: fileName }, function(data) {
                if (data.success) {
                    var html = '<tr> <td><img src='+reader.result+' height="48" width="48"
                    $('#resultsTableBody tr').last().after(html);
                }
            });
        }
        reader.readAsDataURL(file);
    }
```

Upload

Convert downloaded model outputs to HTML

```
from flask import Flask, jsonify, render_template, request, g
from celery import Celery
from PIL import Image
from io import BytesIO
import traceback

app = Flask(__name__)
celery = Celery(__name__)
celery.config_from_object(app.config)
inference = None

@app.before_request
def modelLoad():
    global inference
    if inference is None:
        from static.model.inference import Inference
        inference = Inference()

@celery.task()
def runModel(im):
    with app.test_request_context() as request:
        return inference.run(im)

@app.route('/')
def home():
    return render_template('index.html')

@app.route('/upload-image', methods=['post'])
def uploadImage():
    try:
        dat = re.sub('^', '', request.form['data'
        im = Image.open(BytesIO(base64.b64decode(dat))).convert('RGB'
        outputs = runModel(im)
        return jsonify(success=True,outputs=outputs)
    except IOError as e:
        errno, strerror = e.args
        err = "I/O error({0}): {1}".format(errno, strerror)
        print(err)
        return jsonify(success=False,err=err)
    except ValueError:
        err = "Could not parse image data."
        print(err)
        return jsonify(success=False,err=err)
    except Exception as err:
        err = "Could not parse image data."
        print(err)
        # traceback.print_tb(err.__traceback__)
        return jsonify(success=False)
        raise

if __name__ == '__main__':
    app.run(
    threaded=True,
    debug=True,
```

Server

Challenges in Deep Learning

Durham University

1. Getting a good dataset to begin with
   - **This is the main thing and it's not straightforward**
   - Roll up your sleeves and get involved, don't rely on others as you will know best!
   - Look out for data bias and come up with solutions
2. **Wrangling tensors** to fit task at hand
   - Initially one of the hardest parts, later with experience this is one of the easiest
3. **Designing** architectures
   - Keep it simple, stupid! Trying to utilize transfer learning.
4. Dealing with **overfitting** and **underfitting**
   - Lots of tricks, the challenge is knowing which to try and not doing too many at once
   - Experience helps here
5. **Waiting** for your model to test your adjustments
6. Deployment
   - Easy to do badly
   - Very hard to do properly (multiple users, stable in production environments, keeping the human in the loop, updating with new data and changing project requirements)

Thank you

Durham University

- Questions? Feel free to find me afterwards, also in **E289**

Some popular ones:

Can I do X? How would you do Y? How long would it take to do Z?
Why PyTorch and not Tensorflow, Torch, Keras, Caffe, Neon, Theano, ...?
Why Python and not C++, CUDA, C#, Java, C, Haxe, Javascript, Blah ...
What Tutorials, best place to start? What books? What maths do I need?
Any Ethical or Security issues with this topic?

christopher.g.willcocks@durham.ac.uk
https://github.com/cwkx
https://twitter.com/cwkx

Live demo if time or catch me afterwards for live demo