CIFAR 10

The CIFAR-10 dataset consists of 60000 32x32 color images in 10 classes, with 6000 images per class. There are 50000 training images and 10000 test images.
Small data sets better as we are more likely to be working with smaller data sets.
Medical imaging usually looks at specific areas that are usually 32 by 32

In [1]:
```
%matplotlib inline
%reload_ext autoreload
%autoreload 2
```

You can get the data via:

```
wget http://pjreddie.com/media/files/cifar.tgz
```

In [6]:
```
!wget 'http://pjreddie.com/media/files/cifar.tgz'
```

```
--2017-12-12 21:03:25--  http://pjreddie.com/media/files/cifar.tgz
Resolving pjreddie.com (pjreddie.com)... 128.208.3.39
Connecting to pjreddie.com (pjreddie.com)|128.208.3.39|:80... connected.
HTTP request sent, awaiting response... 301 Moved Permanently
Location: https://pjreddie.com/media/files/cifar.tgz [following]
--2017-12-12 21:03:26--  https://pjreddie.com/media/files/cifar.tgz
Connecting to pjreddie.com (pjreddie.com)|128.208.3.39|:443... connected.
HTTP request sent, awaiting response... 200 OK
Length: 168584360 (161M) [application/octet-stream]
Saving to: â€˜cifar.tgzâ€™

cifar.tgz           100%[===================>] 160.77M  54.2MB/s    in 3.0
s

2017-12-12 21:03:29 (54.2 MB/s) - â€˜cifar.tgzâ€™ saved [168584360/1685843
60]
```

In [9]:
```
!tar xzf cifar.tgz
```

In [14]:
```
!ls
```

```
cifar  cifar.tgz  command.sh  fastai  lesson7-CAM.ipynb  lesson7-cifar10.i
pynb
```

In [15]:
```
!ls cifar
```

```
labels.txt  test  train
```

please ignore the dependencies over next few lines

In [18]:
```
!pip install bcolz
```

```
Collecting bcolz
  Downloading bcolz-1.1.2.tar.gz (1.3MB)
    100% |â–ˆâ–ˆâ–ˆâ–ˆâ–ˆâ–ˆâ–ˆâ–ˆâ–ˆâ–ˆâ–ˆâ–ˆâ–ˆâ–ˆâ–ˆâ–ˆâ–ˆâ–ˆâ–ˆâ–ˆâ–ˆâ–ˆ
–ˆâ–ˆâ–ˆâ–ˆâ–ˆâ–ˆâ–ˆâ–ˆâ–ˆâ–ˆâ–ˆ| 1.3MB 1.1MB/s eta 0:00:01
Requirement already satisfied: numpy>=1.7 in /usr/local/lib/python3.6/site
-packages (from bcolz)
Building wheels for collected packages: bcolz
```

```
  Running setup.py bdist_wheel for bcolz ... done
  Stored in directory: /root/.cache/pip/wheels/e9/84/eb/f8f3caa627bb01ebc9
6034c3411f59870951246e5873b3f4c7
Successfully built bcolz
Installing collected packages: bcolz
Successfully installed bcolz-1.1.2
```

In [20]:
```
!pip install seaborn
```

```
Collecting seaborn
  Downloading seaborn-0.8.1.tar.gz (178kB)
    100% |â–ˆâ–ˆâ–ˆâ–ˆâ–ˆâ–ˆâ–ˆâ–ˆâ–ˆâ–ˆâ–ˆâ–ˆâ–ˆâ–ˆâ–ˆâ–ˆâ–ˆâ–ˆâ–ˆâ–ˆâ–ˆâ–ˆâ–ˆâ–ˆâ–ˆâ–ˆâ–ˆâ–ˆâ–ˆâ–ˆâ–ˆâ–ˆ| 184kB 4.1MB/s ta 0:00:01
Building wheels for collected packages: seaborn
  Running setup.py bdist_wheel for seaborn ... done
  Stored in directory: /root/.cache/pip/wheels/29/af/4b/ac6b04ec3e2da1a450
e74c6a0e86ade83807b4aaf40466ecda
Successfully built seaborn
Installing collected packages: seaborn
Successfully installed seaborn-0.8.1
```

In [22]:
```
!pip install graphviz
```

```
Collecting graphviz
  Downloading graphviz-0.8.1-py2.py3-none-any.whl
Installing collected packages: graphviz
Successfully installed graphviz-0.8.1
```

In [24]:
```
!pip install sklearn_pandas
```

```
Collecting sklearn_pandas
  Downloading sklearn_pandas-1.6.0-py2.py3-none-any.whl
Requirement already satisfied: scikit-learn>=0.15.0 in /usr/local/lib/pyth
on3.6/site-packages (from sklearn_pandas)
Requirement already satisfied: pandas>=0.11.0 in /usr/local/lib/python3.6/
site-packages (from sklearn_pandas)
Requirement already satisfied: scipy>=0.14 in /usr/local/lib/python3.6/sit
e-packages (from sklearn_pandas)
Requirement already satisfied: numpy>=1.6.1 in /usr/local/lib/python3.6/si
te-packages (from sklearn_pandas)
Requirement already satisfied: pytz>=2011k in /usr/local/lib/python3.6/sit
e-packages (from pandas>=0.11.0->sklearn_pandas)
Requirement already satisfied: python-dateutil>=2 in /usr/local/lib/python
3.6/site-packages (from pandas>=0.11.0->sklearn_pandas)
Requirement already satisfied: six>=1.5 in /usr/local/lib/python3.6/site-p
ackages (from python-dateutil>=2->pandas>=0.11.0->sklearn_pandas)
Installing collected packages: sklearn-pandas
Successfully installed sklearn-pandas-1.6.0
```

In [26]:
```
!pip install isoweek
```

```
Collecting isoweek
  Downloading isoweek-1.3.3-py2.py3-none-any.whl
Installing collected packages: isoweek
Successfully installed isoweek-1.3.3
```

In [28]:
```
!pip install pandas_summary
```

```
Collecting pandas_summary
  Downloading pandas-summary-0.0.41.tar.gz
Requirement already satisfied: numpy in /usr/local/lib/python3.6/site-pack
ages (from pandas_summary)
Requirement already satisfied: pandas in /usr/local/lib/python3.6/site-pac
kages (from pandas_summary)
Requirement already satisfied: pytz>=2011k in /usr/local/lib/python3.6/sit
e-packages (from pandas->pandas_summary)
Requirement already satisfied: python-dateutil>=2 in /usr/local/lib/python
3.6/site-packages (from pandas->pandas_summary)
Requirement already satisfied: six>=1.5 in /usr/local/lib/python3.6/site-p
ackages (from python-dateutil>=2->pandas->pandas_summary)
Building wheels for collected packages: pandas-summary
  Running setup.py bdist_wheel for pandas-summary ... done
  Stored in directory: /root/.cache/pip/wheels/20/29/c9/b3d9f2cbdb6f1eeeb9
8e263ae687d72e8138a26de91058bd0b
Successfully built pandas-summary
Installing collected packages: pandas-summary
Successfully installed pandas-summary-0.0.41
```

In [30]:
```
!pip install torchtext
```

```
Collecting torchtext
  Downloading torchtext-0.2.0-py3-none-any.whl (40kB)
    100% |â–ˆâ–ˆâ–ˆâ–ˆâ–ˆâ–ˆâ–ˆâ–ˆâ–ˆâ–ˆâ–ˆâ–ˆâ–ˆâ–ˆâ–ˆâ–ˆâ–ˆâ–ˆâ–ˆâ–ˆâ–ˆâ–ˆ
–ˆâ–ˆâ–ˆâ–ˆâ–ˆâ–ˆâ–ˆâ–ˆâ–ˆâ–ˆâ–ˆ| 40kB 2.4MB/s ta 0:00:011
Requirement already satisfied: requests in /usr/local/lib/python3.6/site-p
ackages (from torchtext)
Requirement already satisfied: tqdm in /usr/local/lib/python3.6/site-packa
ges (from torchtext)
Installing collected packages: torchtext
Successfully installed torchtext-0.2.0
```

In [31]:
```python
from fastai.conv_learner import *
PATH = "cifar/"
os.makedirs(PATH,exist_ok=True)
```

cifar has 10 classes

In [32]:
```python
classes = ('plane', 'car', 'bird', 'cat', 'deer', 'dog', 'frog', 'horse',
'ship', 'truck')
stats = (np.array([ 0.4914 ,  0.48216,  0.44653]), np.array([ 0.24703,  0.
24349,  0.26159]))
```

mean

standard deviation

since using model from scratch we have to provide the means and the standard deviation to normalize the data (calculated with numpy) http://forums.fast.ai/t/training-a-model-from-scratch-cifar-10/7897/57

In [ ]:
```python
def get_data(sz,bs):
    tfms = tfms_from_stats(stats, sz, aug_tfms=[RandomFlipXY()], pad=sz//8
)
    return ImageClassifierData.from_paths(PATH, val_name='test', tfms=tfms
, bs=bs)
```

we have been using tfms_from_model because we have been using pre-trained models using the means and standard deviations of the original model. Here we are going to use tfms_from_stats because we training the model from scratch

padding adds 4 pixels on each side of the sz, here sz is 32

```python
bs=256
```

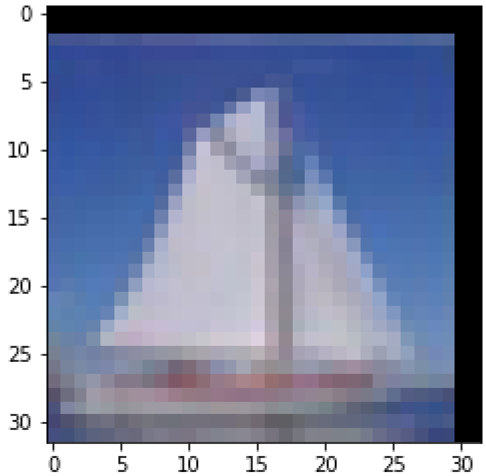
Look at data

```python
data = get_data(32,4)
```

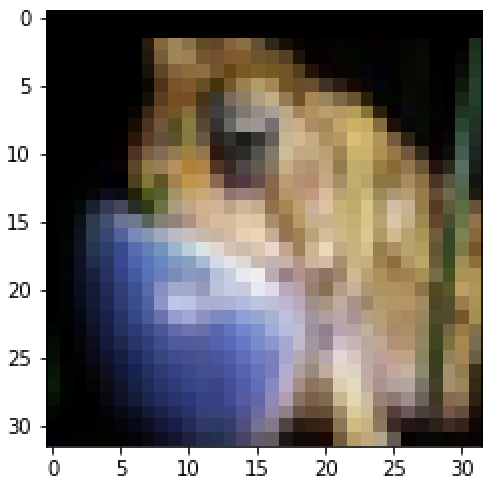
for cifar-10 data augmentation is usually flipping images randomly horizontally

```
In [7]: x,y=next(iter(data.trn_dl))
```

```
In [12]: plt.imshow(data.trn_ds.denorm(x)[0]);
```




```
In [13]: plt.imshow(data.trn_ds.denorm(x)[1]);
```



Fully connected model

FCM has a lot of parameters because each pixel has a corresponding weight - Accuracy is however low - 47% (see below)

FCM = dot product

```
In [6]: data = get_data(32,bs)
```

```
In [7]: lr=1e-2
```

From this notebook by our student Kerem Turgutlu:

class shows a list of fully connected layers

```
In [8]: class SimpleNet(nn.Module):
            def __init__(self, layers):
                super().__init__()
                self.layers = nn.ModuleList([
                    nn.Linear(layers[i], layers[i + 1]) for i in range(len(layers)
        - 1)])
```

```python
    def forward(self, x):
        x = x.view(x.size(0), -1)
        for l in self.layers:
            l_x = l(x)
            x = F.relu(l_x)
        return F.log_softmax(l_x, dim=-1)
```

flatten data because it is a fully connected layers

go through all the layers

linear

conduct relu

finally do a softmax

create a learn object from a custom model

```python
In [9]: learn = ConvLearner.from_model_data(SimpleNet([32*32*3, 40, 10]), data)
```

convolutional learner

```python
In [10]: learn, [o.numel() for o in learn.model.parameters()]
```

```
Out[10]: (SimpleNet(
            (layers): ModuleList(
              (0): Linear(in_features=3072, out_features=40)
              (1): Linear(in_features=40, out_features=10)
            )
          ), [122880, 40, 400, 10])
```

layer 0
in (3072*40) = 122880
out (40)

layer 1
in (40 by 10) = 400
out 10

```python
In [11]: learn.summary()
```

```
Out[11]: OrderedDict([('Linear-1',
                       OrderedDict([('input_shape', [-1, 3072]),
                                    ('output_shape', [-1, 40]),
                                    ('trainable', True),
                                    ('nb_params', 122920)])),
                      ('Linear-2',
                       OrderedDict([('input_shape', [-1, 40]),
                                    ('output_shape', [-1, 10]),
                                    ('trainable', True),
                                    ('nb_params', 410)]))])
```

32

32

3 classes

2

2

10 classes

batch size

2

2

10 classes

1

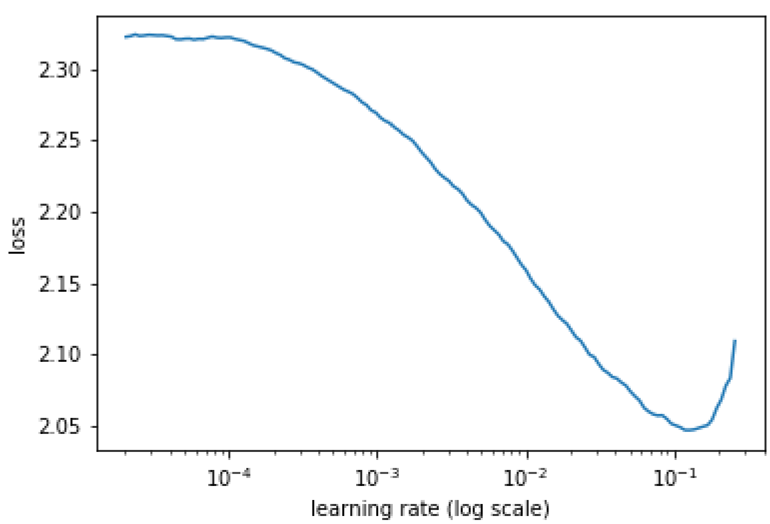
10 classes

input shape

output shape

```python
In [14]: learn.lr_find()
```

```python
In [30]: learn.sched.plot()
```



In [10]:
```
%time learn.fit(lr, 2)
```

```
[ 0.       1.7658   1.64148  0.42129]
[ 1.       1.68074  1.57897  0.44131]

CPU times: user 1min 11s, sys: 32.3 s, total: 1min 44s
Wall time: 55.1 s
```

In [11]:
```
%time learn.fit(lr, 2, cycle_len=1)
```

```
[ 0.       1.60857  1.51711  0.46631]
[ 1.       1.59361  1.50341  0.46924]

CPU times: user 1min 12s, sys: 31.8 s, total: 1min 44s
Wall time: 55.3 s
```

47% accuracy

CNN

convolutional model - uses a sum product of say 3 by 3 set of an image with a corresponding 3 by 3 filter and this is done with the whole image

kernel size is 3 by 3 pixels

In [12]:
```
class ConvNet(nn.Module):
    def __init__(self, layers, c):
        super().__init__()
        self.layers = nn.ModuleList([
            nn.Conv2d(layers[i], layers[i + 1], kernel_size=3, stride=2)
            for i in range(len(layers) - 1)])
        self.pool = nn.AdaptiveMaxPool2d(1)
        self.out = nn.Linear(layers[-1], c)

    def forward(self, x):
        for l in self.layers: x = F.relu(l(x))
        x = self.pool(x)
        x = x.view(x.size(0), -1)
        return F.log_softmax(self.out(x), dim=-1)
```

same code as FCM but replace nn.Linear with nn.Conv2d

stride convolution = move every 2 pixels - has similar effect as max pooling ie half-ing the resolution in each direction

adaptive maxpool is where you determine how big of a resolution to create instead of specifying how big of an area you want to pool Note: there are no weights within max pooling

number of classes I want to predict in the last layer

```
learn = ConvLearner.from_model_data(ConvNet([3, 20, 40, 80], 10), data)
```

3 channels - RGB

In [14]:
```
learn.summary()
```

Out[14]:
```
OrderedDict([('Conv2d-1',
              OrderedDict([('input_shape', [-1, 3, 32, 32]),
                           ('output_shape', [-1, 20, 15, 15]),
                           ('trainable', True),
                           ('nb_params', 560)])),
             ('Conv2d-2',
              OrderedDict([('input_shape', [-1, 20, 15, 15]),
                           ('output_shape', [-1, 40, 7, 7]),
                           ('trainable', True),
                           ('nb_params', 7240)])),
             ('Conv2d-3',
              OrderedDict([('input_shape', [-1, 40, 7, 7]),
                           ('output_shape', [-1, 80, 3, 3]),
                           ('trainable', True),
                           ('nb_params', 28880)])),
             ('AdaptiveMaxPool2d-4',
              OrderedDict([('input_shape', [-1, 80, 3, 3]),
```

32 32 3 classes

15 15 20 features

7 7 40 features

3 3 80 features

adaptive max pool = 1 by 1 tensor

```
                    ('output_shape', [-1, 80, 1, 1]),
                    ('nb_params', 0)])),
      ('Linear-5',
       OrderedDict([('input_shape', [-1, 80]),
                    ('output_shape', [-1, 10]),
                    ('trainable', True),
                    ('nb_params', 810)]))])
```

80 features

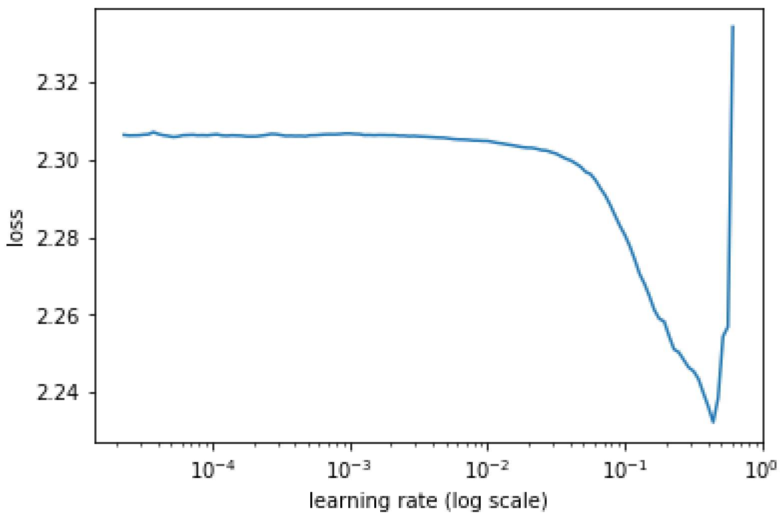
prediction

10 classes

In [20]:
```
learn.lr_find(end_lr=100)
```

```
 70%|â–ˆâ–ˆâ–ˆâ–ˆâ–ˆâ–ˆâ–ˆ   | 138/196 [00:16<00:09,  6.42it/s, loss=2.49]
```

In [21]:
```
learn.sched.plot()
```



In [15]:
```
%time learn.fit(1e-1, 2)
```

```
[ 0.       1.72594  1.63399  0.41338]
[ 1.       1.51599  1.49687  0.45723]

CPU times: user 1min 14s, sys: 32.3 s, total: 1min 46s
Wall time: 56.5 s
```

In [16]:
```
%time learn.fit(1e-1, 4, cycle_len=1)
```

```
[ 0.       1.36734  1.28901  0.53418]
[ 1.       1.28854  1.21991  0.56143]
[ 2.       1.22854  1.15514  0.58398]
[ 3.       1.17904  1.12523  0.59922]
```

accuracy now at 60%

```
CPU times: user 2min 21s, sys: 1min 3s, total: 3min 24s
Wall time: 1min 46s
```

Refactored

means of improving readability and reducing complexity

In [23]:
```
class ConvLayer(nn.Module):
    def __init__(self, ni, nf):
```

neural net

in pytorch a neural net is identical to a layer

Same code as CNN except padding is now added

```
        super().__init__()
        self.conv = nn.Conv2d(ni, nf, kernel_size=3, stride=2, padding=1)

    def forward(self, x): return F.relu(self.conv(x))
```

In [45]:
```
class ConvNet2(nn.Module):
    def __init__(self, layers, c):
        super().__init__()
        self.layers = nn.ModuleList([ConvLayer(layers[i], layers[i + 1])
            for i in range(len(layers) - 1)])
        self.out = nn.Linear(layers[-1], c)

    def forward(self, x):
        for l in self.layers: x = l(x)
        x = F.adaptive_max_pool2d(x, 1)
        x = x.view(x.size(0), -1)
        return F.log_softmax(self.out(x), dim=-1)
```

In [46]:
```
learn = ConvLearner.from_model_data(ConvNet2([3, 20, 40, 80], 10), data)
```

In [47]:
```
learn.summary()
```

Out[47]:
```
OrderedDict([('Conv2d-1',
              OrderedDict([('input_shape', [-1, 3, 32, 32]),
                           ('output_shape', [-1, 20, 16, 16]),
                           ('trainable', True),
                           ('nb_params', 560)])),
             ('ConvLayer-2',
              OrderedDict([('input_shape', [-1, 3, 32, 32]),
                           ('output_shape', [-1, 20, 16, 16]),
                           ('nb_params', 0)])),
             ('Conv2d-3',
              OrderedDict([('input_shape', [-1, 20, 16, 16]),
                           ('output_shape', [-1, 40, 8, 8]),
                           ('trainable', True),
                           ('nb_params', 7240)])),
             ('ConvLayer-4',
              OrderedDict([('input_shape', [-1, 20, 16, 16]),
                           ('output_shape', [-1, 40, 8, 8]),
                           ('nb_params', 0)])),
             ('Conv2d-5',
              OrderedDict([('input_shape', [-1, 40, 8, 8]),
                           ('output_shape', [-1, 80, 4, 4]),
                           ('trainable', True),
                           ('nb_params', 28880)])),
             ('ConvLayer-6',
              OrderedDict([('input_shape', [-1, 40, 8, 8]),
                           ('output_shape', [-1, 80, 4, 4]),
                           ('nb_params', 0)])),
             ('Linear-7',
              OrderedDict([('input_shape', [-1, 80]),
                           ('output_shape', [-1, 10]),
                           ('trainable', True),
                           ('nb_params', 810)]))])
```

sizes different in this re-factored example due to padding of 1

deeper network with more layers compared to CNN

In [48]:
```
%time learn.fit(1e-1, 2)
```

```
[ 0.         1.70151   1.64982   0.3832 ]
[ 1.         1.50838   1.53231   0.44795]

CPU times: user 1min 6s, sys: 28.5 s, total: 1min 35s
Wall time: 48.8 s
```

In [49]:
```
%time learn.fit(1e-1, 2, cycle_len=1)
```

```
[ 0.         1.51605   1.42927   0.4751 ]
[ 1.         1.40143   1.33511   0.51787]

CPU times: user 1min 6s, sys: 27.7 s, total: 1min 34s
Wall time: 48.7 s
```

not much change to accuracy

in a nutshell:
- increases resilance of the training
- increases the number of layers that can be trained
- increases the learning rate

BatchNorm

makes networks more resilient - make it easier to train deeper networks
Process of normalizing all the batches not just the inputs

In [17]:
```
class BnLayer(nn.Module):
    def __init__(self, ni, nf, stride=2, kernel_size=3):
        super().__init__()
        self.conv = nn.Conv2d(ni, nf, kernel_size=kernel_size, stride=stri
,
                              bias=False, padding=1)
        self.a = nn.Parameter(torch.zeros(nf,1,1))
        self.m = nn.Parameter(torch.ones(nf,1,1))

    def forward(self, x):
        x = F.relu(self.conv(x))
        x_chan = x.transpose(0,1).contiguous().view(x.size(1), -1)
        if self.training:
            self.means = x_chan.mean(1)[:,None,None]
            self.stds  = x_chan.std (1)[:,None,None]
        return (x-self.means) / self.stds *self.m + self.a
```

create a new added value for each channel in this case 3 zeros

create a new multiplier for each channel in this case 3 1s

this means the network does not have to scale every single value in the matrix - it can scale up the 3 trio of numbers instead

normalizing the batches

x | + 0

In [18]:
```
class ConvBnNet(nn.Module):
    def __init__(self, layers, c):
        super().__init__()
        self.conv1 = nn.Conv2d(3, 10, kernel_size=5, stride=1, padding=2)
        self.layers = nn.ModuleList([BnLayer(layers[i], layers[i + 1])
            for i in range(len(layers) - 1)])
        self.out = nn.Linear(layers[-1], c)

    def forward(self, x):
        x = self.conv1(x)
        for l in self.layers: x = l(x)
        x = F.adaptive_max_pool2d(x, 1)
        x = x.view(x.size(0), -1)
        return F.log_softmax(self.out(x), dim=-1)
```

added a single conv layer at the start with a bigger kernel size, new architectures use 5 by 5 or 7 by 7 and have a richer input, here 10 features

single initial conv layer

instead of 3 in previous examples

In [20]:
```
learn = ConvLearner.from_model_data(ConvBnNet([10, 20, 40, 80, 160], 10),
data)
```

In [21]:
```
learn.summary()
```

```
Out[21]: OrderedDict([('Conv2d-1',
                       OrderedDict([('input_shape', [-1, 3, 32, 32]),
                                    ('output_shape', [-1, 10, 32, 32]),
                                    ('trainable', True),
                                    ('nb_params', 760)])),
                      ('Conv2d-2',
                       OrderedDict([('input_shape', [-1, 10, 32, 32]),
                                    ('output_shape', [-1, 20, 16, 16]),
                                    ('trainable', True),
                                    ('nb_params', 1800)])),
                      ('BnLayer-3',
                       OrderedDict([('input_shape', [-1, 10, 32, 32]),
                                    ('output_shape', [-1, 20, 16, 16]),
                                    ('nb_params', 0)])),
                      ('Conv2d-4',
                       OrderedDict([('input_shape', [-1, 20, 16, 16]),
                                    ('output_shape', [-1, 40, 8, 8]),
                                    ('trainable', True),
                                    ('nb_params', 7200)])),
                      ('BnLayer-5',
                       OrderedDict([('input_shape', [-1, 20, 16, 16]),
                                    ('output_shape', [-1, 40, 8, 8]),
                                    ('nb_params', 0)])),
                      ('Conv2d-6',
                       OrderedDict([('input_shape', [-1, 40, 8, 8]),
                                    ('output_shape', [-1, 80, 4, 4]),
                                    ('trainable', True),
                                    ('nb_params', 28800)])),
                      ('BnLayer-7',
                       OrderedDict([('input_shape', [-1, 40, 8, 8]),
                                    ('output_shape', [-1, 80, 4, 4]),
                                    ('nb_params', 0)])),
                      ('Conv2d-8',
                       OrderedDict([('input_shape', [-1, 80, 4, 4]),
                                    ('output_shape', [-1, 160, 2, 2]),
                                    ('trainable', True),
                                    ('nb_params', 115200)])),
                      ('BnLayer-9',
                       OrderedDict([('input_shape', [-1, 80, 4, 4]),
                                    ('output_shape', [-1, 160, 2, 2]),
                                    ('nb_params', 0)])),
                      ('Linear-10',
                       OrderedDict([('input_shape', [-1, 160]),
                                    ('output_shape', [-1, 10]),
                                    ('trainable', True),
                                    ('nb_params', 1610)]))])
```

```
In [22]: %time learn.fit(3e-2, 2)
```

```
[ 0.       1.4966   1.39257  0.48965]
[ 1.       1.2975   1.20827  0.57148]

CPU times: user 1min 16s, sys: 32.5 s, total: 1min 49s
Wall time: 54.3 s
```

```
In [23]: %time learn.fit(1e-1, 4, cycle_len=1)
```

```
[ 0.       1.20966  1.07735  0.61504]
[ 1.       1.0771   0.97338  0.65215]
[ 2.       1.00103  0.91281  0.67402]
[ 3.       0.93574  0.89293  0.68135]
```

improvement on accuracy now 68%

```
CPU times: user 2min 34s, sys: 1min 4s, total: 3min 39s
Wall time: 1min 50s
```

Deep BatchNorm

same principle as batchnorm except now we will make the network deeper ie create more layers

In [47]:
```python
class ConvBnNet2(nn.Module):
    def __init__(self, layers, c):
        super().__init__()
        self.conv1 = nn.Conv2d(3, 10, kernel_size=5, stride=1, padding=2)
        self.layers = nn.ModuleList([BnLayer(layers[i], layers[i+1])
            for i in range(len(layers) - 1)])
        self.layers2 = nn.ModuleList([BnLayer(layers[i+1], layers[i + 1],
)
            for i in range(len(layers) - 1)])
        self.out = nn.Linear(layers[-1], c)

    def forward(self, x):
        x = self.conv1(x)
        for l,l2 in zip(self.layers, self.layers2):
            x = l(x)
            x = l2(x)
        x = F.adaptive_max_pool2d(x, 1)
        x = x.view(x.size(0), -1)
        return F.log_softmax(self.out(x), dim=-1)
```

original stride 2 layers

for each stride 2 layer also create a stride 1 layer

zip the stride 2 layers (layers) and stride 1 layers (layers2) together

first do the stride 2 layer

then do the stride 1 layer

now twice as deep

In [48]:
```python
learn = ConvLearner.from_model_data((ConvBnNet2([10, 20, 40, 80, 160], 10)
, data)
```

In [49]:
```python
%time learn.fit(1e-2, 2)
```

```
[ 0.       1.53499  1.43782  0.47588]
[ 1.       1.28867  1.22616  0.55537]

CPU times: user 1min 22s, sys: 34.5 s, total: 1min 56s
Wall time: 58.2 s
```

In [50]:
```python
%time learn.fit(1e-2, 2, cycle_len=1)
```

deep batchnorm does not help accuracy - this is because it is now 12 layers deep and does not help with accuracy

```
[ 0.       1.10933  1.06439  0.61582]
[ 1.       1.04663  0.98608  0.64609]

CPU times: user 1min 21s, sys: 32.9 s, total: 1min 54s
Wall time: 57.6 s
```

Resnet

For this reason we now use Resnet with the same code and make the network even more deeper

$y = x + f(x)$

y = x + f(x)
y = prediction
x = input
f(x) = function in this case a convolution

Resnet block

53]:
```
class ResnetLayer(BnLayer):
    def forward(self, x): return x + super().forward(x)
```

In [54]:
```
class Resnet(nn.Module):
    def __init__(self, layers, c):
        super().__init__()
        self.conv1 = nn.Conv2d(3, 10, kernel_size=5, stride=1, padding=2)
        self.layers = nn.ModuleList([BnLayer(layers[i], layers[i+1])
            for i in range(len(layers) - 1)])
        self.layers2 = nn.ModuleList([ResnetLayer(layers[i+1], layers[i +
1], 1)
            for i in range(len(layers) - 1)])
        self.layers3 = nn.ModuleList([ResnetLayer(layers[i+1], layers[i +
1], 1)
            for i in range(len(layers) - 1)])
        self.out = nn.Linear(layers[-1], c)

    def forward(self, x):
        x = self.conv1(x)
        for l,l2,l3 in zip(self.layers, self.layers2, self.layers3):
            x = l3(l2(l(x)))
        x = F.adaptive_max_pool2d(x, 1)
        x = x.view(x.size(0), -1)
        return F.log_softmax(self.out(x), dim=-1)
```

bottleneck block -more in part 2

prediction from previous layer

prediction

$f(x) = \underbrace{y - x}$

first single conv layer

3 stride layers

difference is the residual or error

adding the error each time helps gradually get closer to the answer = based on theory of boosting(calculating a model on the residual

In [55]:
```
learn = ConvLearner.from_model_data(Resnet([10, 20, 40, 80, 160], 10), dat
a)
```

In [56]:
```
wd=1e-5
```

In [57]:
```
%time learn.fit(1e-2, 2, wds=wd)
```

```
[ 0.       1.58191  1.40258  0.49131]
[ 1.       1.33134  1.21739  0.55625]

CPU times: user 1min 27s, sys: 34.3 s, total: 2min 1s
Wall time: 1min 3s
```

In [58]:
```
%time learn.fit(1e-2, 3, cycle_len=1, cycle_mult=2, wds=wd)
```

```
[ 0.       1.11534  1.05117  0.62549]
[ 1.       1.06272  0.97874  0.65185]
[ 2.       0.92913  0.90472  0.68154]
[ 3.       0.97932  0.94404  0.67227]
[ 4.       0.88057  0.84372  0.70654]
[ 5.       0.77817  0.77815  0.73018]
[ 6.       0.73235  0.76302  0.73633]

CPU times: user 5min 2s, sys: 1min 59s, total: 7min 1s
Wall time: 3min 39s
```

In [59]:
```
%time learn.fit(1e-2, 8, cycle_len=4, wds=wd)
```

```
[ 0.         0.8307    0.83635   0.7126 ]
[ 1.         0.74295   0.73682   0.74189]
[ 2.         0.66492   0.69554   0.75996]
[ 3.         0.62392   0.67166   0.7625 ]
[ 4.         0.73479   0.80425   0.72861]
[ 5.         0.65423   0.68876   0.76318]
[ 6.         0.58608   0.64105   0.77783]
[ 7.         0.55738   0.62641   0.78721]
[ 8.         0.66163   0.74154   0.7501 ]
[ 9.         0.59444   0.64253   0.78106]
[ 10.          0.53       0.61772    0.79385]
[ 11.          0.49747    0.65968    0.77832]
[ 12.          0.59463    0.67915    0.77422]
[ 13.          0.55023    0.65815    0.78106]
[ 14.          0.48959    0.59035    0.80273]
[ 15.          0.4459     0.61823    0.79336]
[ 16.          0.55848    0.64115    0.78018]
[ 17.          0.50268    0.61795    0.79541]
[ 18.          0.45084    0.57577    0.80654]
[ 19.          0.40726    0.5708     0.80947]
[ 20.          0.51177    0.66771    0.78232]
[ 21.          0.46516    0.6116     0.79932]
[ 22.          0.40966    0.56865    0.81172]
[ 23.          0.3852     0.58161    0.80967]
[ 24.          0.48268    0.59944    0.79551]
[ 25.          0.43282    0.56429    0.81182]
[ 26.          0.37634    0.54724    0.81797]
[ 27.          0.34953    0.54169    0.82129]
[ 28.          0.46053    0.58128    0.80342]
[ 29.          0.4041     0.55185    0.82295]
[ 30.          0.3599     0.53953    0.82861]
[ 31.          0.32937    0.55605    0.82227]

CPU times: user 22min 52s, sys: 8min 58s, total: 31min 51s
Wall time: 16min 38s
```

improved accuracy 82%

Resnet 2

In [63]:

```
class Resnet2(nn.Module):
    def __init__(self, layers, c, p=0.5):
        super().__init__()
        self.conv1 = BnLayer(3, 16, stride=1, kernel_size=7)
        self.layers = nn.ModuleList([BnLayer(layers[i], layers[i+1])
            for i in range(len(layers) - 1)])
        self.layers2 = nn.ModuleList([ResnetLayer(layers[i+1], layers[i + 
1], 1)
            for i in range(len(layers) - 1)])
        self.layers3 = nn.ModuleList([ResnetLayer(layers[i+1], layers[i + 
1], 1)
            for i in range(len(layers) - 1)])
        self.out = nn.Linear(layers[-1], c)
        self.drop = nn.Dropout(p)

    def forward(self, x):
        x = self.conv1(x)
```

added dropout

```python
        for l,l2,l3 in zip(self.layers, self.layers2, self.layers3):
            x = l3(l2(l(x)))
        x = F.adaptive_max_pool2d(x, 1)
        x = x.view(x.size(0), -1)
        x = self.drop(x)
        return F.log_softmax(self.out(x), dim=-1)
```

In [70]:
```python
learn = ConvLearner.from_model_data(Resnet2([16, 32, 64, 128, 256], 10, 0.2), data)
```

dropout

In [71]:
```python
wd=1e-6
```

In [72]:
```python
%time learn.fit(1e-2, 2, wds=wd)
```

```
[ 0.         1.7051    1.53364   0.46885]
[ 1.         1.47858   1.34297   0.52734]

CPU times: user 1min 29s, sys: 35.4 s, total: 2min 4s
Wall time: 1min 6s
```

In [73]:
```python
%time learn.fit(1e-2, 3, cycle_len=1, cycle_mult=2, wds=wd)
```

```
[ 0.         1.29414   1.26694   0.57041]
[ 1.         1.21206   1.06634   0.62373]
[ 2.         1.05583   1.0129    0.64258]
[ 3.         1.09763   1.11568   0.61318]
[ 4.         0.97597   0.93726   0.67266]
[ 5.         0.86295   0.82655   0.71426]
[ 6.         0.827     0.8655    0.70244]

CPU times: user 5min 11s, sys: 1min 58s, total: 7min 9s
Wall time: 3min 48s
```

In [74]:
```python
%time learn.fit(1e-2, 8, cycle_len=4, wds=wd)
```

```
[ 0.         0.92043   0.93876   0.67685]
[ 1.         0.8359    0.81156   0.72168]
[ 2.         0.73084   0.72091   0.74463]
[ 3.         0.68688   0.71326   0.74824]
[ 4.         0.81046   0.79485   0.72354]
[ 5.         0.72155   0.68833   0.76006]
[ 6.         0.63801   0.68419   0.76855]
[ 7.         0.59678   0.64972   0.77363]
[ 8.         0.71126   0.78098   0.73828]
[ 9.         0.63549   0.65685   0.7708 ]
[ 10.          0.56837    0.63656    0.78057]
[ 11.          0.52093    0.59159    0.79629]
[ 12.          0.66463    0.69927    0.76357]
[ 13.          0.58121    0.64529    0.77871]
[ 14.          0.52346    0.5751     0.80293]
[ 15.          0.47279    0.55094    0.80498]
[ 16.          0.59857    0.64519    0.77559]
[ 17.          0.54384    0.68057    0.77676]
[ 18.          0.48369    0.5821     0.80273]
```

```
[ 19.         0.43456   0.54708   0.81182]
[ 20.         0.54963   0.65753   0.78203]
[ 21.         0.49259   0.55957   0.80791]
[ 22.         0.43646   0.55221   0.81309]
[ 23.         0.39269   0.55158   0.81426]
[ 24.         0.51039   0.61335   0.7998 ]
[ 25.         0.4667    0.56516   0.80869]
[ 26.         0.39469   0.5823    0.81299]
[ 27.         0.36389   0.51266   0.82764]
[ 28.         0.48962   0.55353   0.81201]
[ 29.         0.4328    0.55394   0.81328]
[ 30.         0.37081   0.50348   0.83359]
[ 31.         0.34045   0.52052   0.82949]

CPU times: user 23min 30s, sys: 9min 1s, total: 32min 32s
Wall time: 17min 16s
```

accuracy keeps getting better

In [75]:
```python
learn.save('tmp3')
```

In [76]:
```python
log_preds,y = learn.TTA()
preds = np.mean(np.exp(log_preds),0)
```

In [77]:
```python
metrics.log_loss(y,preds), accuracy(preds,y)
```

Out[77]:
```
(0.44507397166057938, 0.84909999999999997)
```

accuracy after TTA 85%

End