jupyter road-classifiation-trainvalidtest Last Checkpoint: 07/06/2018 (unsaved changes)

File Edit View Insert Cell Kernel Help | Python 3

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

In [2]:
```
from fastai.imports import *
```

In [3]:
```
from fastai.transforms import *
from fastai.conv_learner import *
from fastai.model import *
from fastai.dataset import *
from fastai.sgdr import *
from fastai.plots import *
```

We need a path that points to the dataset. In this path we will also store temporary data and final results. ImageClassifierData.from_paths reads data from a provided path and creates a dataset ready for training.

In [4]:
```
PATH = "data/road-damage/"
sz=250
```

In [5]:
```
os.listdir(PATH)
```

Out[5]:
```
['train', 'tmp', 'test', 'models', 'valid']
```

tfms stands for transformations. tfms_from_model takes care of resizing, image cropping, initial normalization (creating data with (mean,stdev) of (0,1)), and more.

ConvLearner.pretrained builds learner that contains a pre-trained model. The last layer of the model needs to be replaced with the layer of the right dimensions. The pretained model was trained for 1000 classes therfore the final layer predicts a vector of 1000 probabilities.

Parameters are learned by fitting a model to the data. Hyperparameters are another kind of parameter, that cannot be directly learned from the regular training process. These parameters express "higher-level" properties of the model such as its complexity or how fast it should learn. Two examples of hyperparameters are the learning rate and the number of epochs.

During iterative training of a neural network, a batch or mini-batch is a subset of training samples used in one iteration of Stochastic Gradient Descent (SGD). An epoch is a single pass through the entire training set which consists of multiple iterations of SGD.

We can now fit the model, that is, use gradient descent to find the best parameters for the fully connected layer we added, that can generate set pictures from

jupyter road-classifiation-trainvalidtest Last Checkpoint: 07/06/2018 (unsaved changes)

File Edit View Insert Cell Kernel Help | Python 3

```
data = ImageClassifierData.from_paths(PATH, tfms=tfms_from_model(arch, sz), test_name='test')
```

In [7]:
```
data.classes
```

Out[7]:
```
['Adachi', 'Chiba', 'Ichihara', 'Muroran', 'Nagakute', 'Numazu', 'Sumida']
```

In [8]:
```
data.val_y
```

Out[8]:
```
array([0, 0, 0, ..., 6, 6, 6])
```

Accuracy is the ratio of correct prediction to the total number of predictions.

In machine learning the loss function or cost function is representing the price paid for inaccuracy of predictions.

The loss associated finction used here is binary classification

In [9]:
```
learn = ConvLearner.pretrained(arch, data, precompute=True)
learn.fit(0.01, 3)
```

```
100%|██████████| 26/26 [00:24<00:00,  1.05it/s]

HBox(children=(IntProgress(value=0, description='Epoch', max=3), HTML(value='')))

epoch      trn_loss   val_loss   accuracy
    0      0.923621   0.553276   0.811029
    1      0.672276   0.47213    0.841388
    2      0.532727   0.421664   0.854399
```

Out[9]:
```
[array([0.42166]), 0.8543990086741016]
```

In [10]:
```
lrf=learn.lr_find()
```

```
HBox(children=(IntProgress(value=0, description='Epoch', max=1), HTML(value='')))

 89%|▉▉▉▉▉▉▉▉▉ | 81/91 [00:01<00:00, 52.80it/s, loss=1.66]
```

The learning rate determines how quickly or how slowly you want to update the weights (or parameters) The method learn.lr_find() helps you find an optimal learning rate. It uses the technique developed in the 2015 paper Cyclical Learning Rates for Training Neural Networks, where we simply keep increasing the learning rate from a very small value, until the loss stops decreasing. We can plot the learning rate across batches to see what this looks like.

jupyter road-classifiation-trainvalidtest Last Checkpoint: 07/06/2018 (unsaved changes)

File Edit View Insert Cell Kernel Help | Python 3

In [12]:
```
learn.sched.plot()
```

Data augmentation

We try training for more epochs, overfitting arises, which means that our model is learning to recognize the specific images in the training set, rather than generalizing such that we also get good results on the validation set. One way to fix this is to effectively create more data, through data augmentation. This refers to randomly changing the images in ways that shouldn't impact their interpretation, such as horizontal flipping, zooming, and rotating.

We can do this by passing aug_tfms (augmentation transforms) to tfms_from_model, with a list of functions to apply that randomly change the image however we wish. For photos that are largely taken from the side (e.g. most photos of dogs and cats, as opposed to photos taken from the top down, such as satellite imagery) we can use the pre-defined list of functions transforms_side_on. We can also specify random zooming of images up to specified scale by adding the max_zoom parameter.

In [13]:
```
tfms = tfms_from_model(resnet34, sz, aug_tfms=transforms_side_on, max_zoom=1.2)
```

In [15]:
```
learn.precompute=False
```

What is that cycle_len parameter? What we've done here is used a technique called stochastic gradient descent with restarts (SGDR), a variant of learning rate annealing, which gradually decreases the learning rate as training progresses. This is helpful because as we get closer to the optimal weights, we want to take smaller steps.

In [16]:
```
learn.fit(0.1, 3, cycle_len=1)
```

```
HBox(children=(IntProgress(value=0, description='Epoch', max=3), HTML(value='')))

epoch      trn_loss   val_loss   accuracy
    0      0.744636   0.457857   0.839529
    1      0.597133   0.383499   0.872367
    2      0.504122   0.350871   0.884758
```

Out[16]:
```
[array([0.35087]), 0.8847583643122676]
```

In [17]:
```
learn.unfreeze()
```

In [18]:
```
lr_rate = 0.1
```

In [19]:
```
lr=np.array([lr_rate/6,lr_rate/3,lr_rate])
```

jupyter road-classifiation-trainvalidtest Last Checkpoint: 07/06/2018 (unsaved changes)

File Edit View Insert Cell Kernel Help | Python 3

```
    2      0.045088   0.039023   0.987608
    3      0.044473   0.068469   0.979554
    4      0.026264   0.034442   0.989467
    5      0.013161   0.023796   0.992565
    6      0.005519   0.023487   0.992565
```

Out[20]:
```
[array([0.02349]), 0.9925650557620818]
```

In [30]:
```
log_preds,y = learn.TTA(is_test=True)
probs = np.mean(np.exp(log_preds),0)
```

In [31]:
```
probs.shape
```

Out[31]:
```
(1654, 7)
```

In [28]:
```
log_predict = learn.predict(is_test=True)
probs_predict = np.exp(log_predict)
```

In [32]:
```
probs_predict
```

Out[32]:
```
array([[0., 0., 0., ..., 0., 1., 0.],
       [0., 0., 0., ..., 1., 0., 0.],
       [0., 0., 0., ..., 1., 0., 0.],
       ...,
       [1., 0., 0., ..., 0., 0., 0.],
       [0., 0., 0., ..., 0., 0., 1.],
       [0., 0., 0., ..., 0., 1., 0.]], dtype=float32)
```

In [33]:
```
probs
```

Out[33]:
```
array([[0., 0., 0., ..., 0., 1., 0.],
       [0., 0., 0., ..., 1., 0., 0.],
       [0., 0., 0., ..., 1., 0., 0.],
       ...,
       [1., 0., 0., ..., 0., 0., 0.],
       [0., 0., 0., ..., 0., 0., 1.],
       [0., 0., 0., ..., 0., 1., 0.]], dtype=float32)
```

In [34]:
```
df = pd.DataFrame(probs)
df.columns = data.classes
```

jupyter road-classifiation-trainvalidtest Last Checkpoint: 07/06/2018 (unsaved changes)

File Edit View Insert Cell Kernel Help | Python 3

```
'Numazu.1249.jpg',
'Nagakute.728.jpg',
'Numazu.513.jpg',
'Numazu.696.jpg',
'Nagakute.914.jpg',
'Adachi.1457.jpg',
'Nagakute.54.jpg',
'Adachi.1189.jpg',
'Muroran.79.jpg',
'Nagakute.570.jpg',
'Muroran.1638.jpg',
'Numazu.1604.jpg',
'Nagakute.928.jpg',
'Adachi.1445.jpg',
'Nagakute.462.jpg',
'Numazu.1299.jpg',
```

In [ ]:
```
df.insert(0,'Image Name',[o for o in os.listdir(PATH+'/test')])
```

or df.insert(0,'Image Name',[o[5:-4] for o in data.test_ds.fnames])

In [41]:
```
df
```

Out[41]:

	Image Name	Adachi	Chiba	Ichihara	Muroran	Nagakute	Numazu	Sumida
0	Numazu.422.jpg	2.069496e-06	1.943013e-06	4.307226e-09	5.718895e-07	4.170158e-07	9.999952e-01	2.165057e-07
1	Nagakute.1231.jpg	2.872752e-07	4.146168e-10	2.255610e-07	9.524808e-09	1.000000e+00	4.734822e-09	3.218959e-11
2	Nagakute.414.jpg	9.119393e-08	2.696280e-11	6.072780e-10	4.408620e-09	1.000000e+00	2.271534e-08	2.421692e-11
3	Numazu.1249.jpg	7.814291e-07	4.625776e-07	7.324715e-05	1.774327e-06	1.460993e-05	9.999084e-01	6.229383e-07
4	Nagakute.728.jpg	1.335089e-05	6.960449e-09	7.497317e-08	1.780544e-07	9.999676e-01	1.920045e-05	1.007155e-08
5	Numazu.513.jpg	8.354662e-08	6.442651e-07	1.002262e-09	1.134596e-07	1.796447e-07	9.999990e-01	4.891082e-08
6	Numazu.696.jpg	2.683161e-06	2.413837e-06	3.740679e-08	1.042310e-07	4.198528e-08	9.999971e-01	1.504321e-07
7	Nagakute.914.jpg	5.698593e-07	1.404815e-10	3.422988e-09	2.842053e-08	1.000000e+00	1.178035e-07	9.431713e-11
8	Adachi.1457.jpg	9.999466e-01	5.012008e-07	9.295572e-07	3.339109e-05	1.160926e-05	5.658203e-06	1.295053e-06
9	Nagakute.54.jpg	1.491753e-08	7.229739e-11	6.281005e-08	1.174419e-08	1.000000e+00	1.580607e-09	7.625716e-12

In [ ]: