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 *
```

In [15]:
```
torch.cuda.is_available()
```

Out[15]:
```
True
```

In [16]:
```
torch.backends.cudnn.enabled
```

Out[16]:
```
True
```

Gather Data

In [4]:
```
PATH = '/home/danieldiamond/data/dogbreed/'
```

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

Out[5]:
```
['labels.csv', 'models', 'tmp', 'train', 'sample_submission.csv', 'test']
```

In [6]:
```
labels=pd.read_csv(PATH+'labels.csv')
```

In [7]:
```
labels_train = labels.sample(frac=0.8)
labels_valid = labels[~labels.index.isin(labels_train.index)]
```

In [8]:
```
labels_train.shape,labels_valid.shape
```

Out[8]:
```
((8178, 2), (2044, 2))
```

In [9]:
```
labels.head()
```

Out[9]:

	id	breed
0	000bec180eb18c7604dcecc8fe0dba07	boston_bull
1	001513dfcb2ffafc82cccf4d8bbaba97	dingo
2	001cdf01b096e06d78e9e5112d419397	pekinese
3	00214f311d5d2247d5dfe4fe24b2303d	bluetick
4	0021f9ceb3235effd7fcde7f7538ed62	golden_retriever

In [10]:
```
labels.pivot_table(index='breed',aggfunc=len).sort_values('id',ascending=Fa
```

Out[10]:

	id
breed	
scottish_deerhound	126
maltese_dog	117
afghan_hound	116
entlebucher	115
bernese_mountain_dog	114

In [11]:
```
files = os.listdir(f'{PATH}test')[:5]
files
```

Out[11]:
```
['41bb88e8da45c400490febc6d8d13689.jpg',
 'e06fdea86b416e992137ad52bb5da5a8.jpg',
 '4b6ba16df7da3185a747cacbf37d7072.jpg',
 '555aab3f3ce67bf7ffd6313717abcd77.jpg',
 '5bba97f88dbe095b004c5e25bd9e5a90.jpg']
```

In [12]:
```
img = plt.imread(f'{PATH}test/{files[0]}')
plt.imshow(img);
```

In [13]:
```
img.shape
```
Out[13]:
```
(332, 500, 3)
```

Image Size and Batch Size

In [14]:
```
# set the size for the transformer to resize all images
sz = 224

# set the batch size (smaller allows computing to occur faster)
bs = 58
```

Architecture

Deep Convolutional Neural Networks have been groundbreaking in terms of image classification. Thus, there has been a trending to increase /improve the classification/recognition accuracy. However, the deeper you go, the more difficult training of neural network becomes and similarly the accuracy starts to saturate and degrade. This is where Residual Learning comes in.

In [17]:
```
# arch=resnext50
arch=resnext101_64
```

What is Residual Learning?

In general, in a deep convolutional neural network, several layers are stacked and are trained to the task at hand. The network learns several low/mid/high level features at the end of its layers. In residual learning, instead of trying to learn some features, we try to learn some residual. Residual can be simply understood as subtraction of feature learned from input of that layer. ResNet does this using shortcut connections (directly connecting input of nth layer to some (n+x)th layer. It has proved that training this form of networks is easier than training simple deep convolutional neural networks and also the problem of degrading accuracy is resolved.

This is the fundamental concept of ResNet.
[https://arxiv.org/pdf/1512.03385.pdf (https://arxiv.org/pdf/1512.03385.pdf)](https://arxiv.org/pdf/1512.03385.pdf)

ResNet50 is a 50 layer Residual Network.
[http://ethereon.github.io/netscope/#/gist/db945b393d40bfa26006 (http://ethereon.github.io/netscope/#/gist/db945b393d40bfa26006)](http://ethereon.github.io/netscope/#/gist/db945b393d40bfa26006)

There are other variants like ResNet101 and ResNet152 also.

Deep neural networks are susceptible to vanishing gradients. Resnet used skip connection to propagate information over layers allowing Data scientist to build deeper networks. Skip connection helps the network to understand global features. In CNN due to maxpooling information can get lost. It enhances the detection of smaller objects in the image.

Transformer

In [18]:
```
# images require a basic transformation for ConvLearner to work
# i.e. images must be consistent size e.g. sz x sz x 3
tfms=tfms_from_model(arch,sz,aug_tfms=transforms_side_on,max_zoom=1.1)
```

Note On Data Augmentation

- If you start to train on more epochs you'll start to overfit i.e. recognizing specific images from training set instead of generalizing s.t. can also get good results from validation set.
- More data is better and an effective way to create more data is data augmentation i.e. through horizontal flipping, zooming, and rotating.
- Add aug_tfms (augmentation transforms) to the tfms function with a list of functions
- NOTE: for normal photos you want to flip horizontally (not vertically) and maybe zoom a little. For strange/particular photos like satellite or medical scans (which the pre-trained model hasn't been trained on) you may feel comfortable to flip vertically.

In [19]:
```
# create data object for the ConvLearner
data = ImageClassifierData.from_csv(PATH,'train',PATH+'labels.csv',test_nam
                                     val_idxs=labels_valid.index,
                                     suffix='.jpg',tfms=tfms,bs=bs)
# NOTE: test_name = 'test' is for if you want to submit to kaggle later
```

In [20]:
```
fn = PATH+data.trn_ds.fnames[0]; fn
```

Out[20]:
```
'/home/danieldiamond/data/dogbreed/train/001cdf01b096e06d78e9e5112d41939
7.jpg'
```

In [21]:
```
img = PIL.Image.open(fn); img
```

Out[21]:

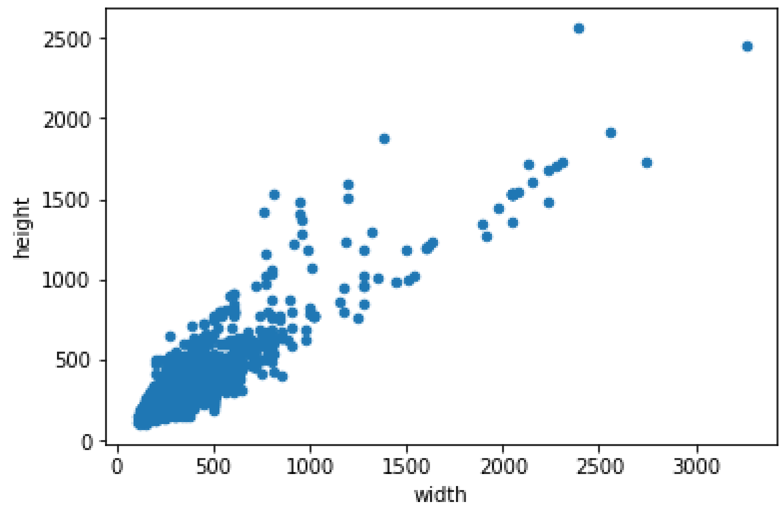
In [22]:
```
print ('data :',list(data.__dict__.keys()))
print ('training dataset :',list(data.trn_ds.__dict__.keys()))
```

```
data : ['path', 'bs', 'num_workers', 'classes', 'trn_dl', 'val_dl', 'fix_
dl', 'aug_dl', 'test_dl', 'test_aug_dl']
training dataset : ['y', 'path', 'fnames', 'transform', 'n', 'c', 'sz']
```

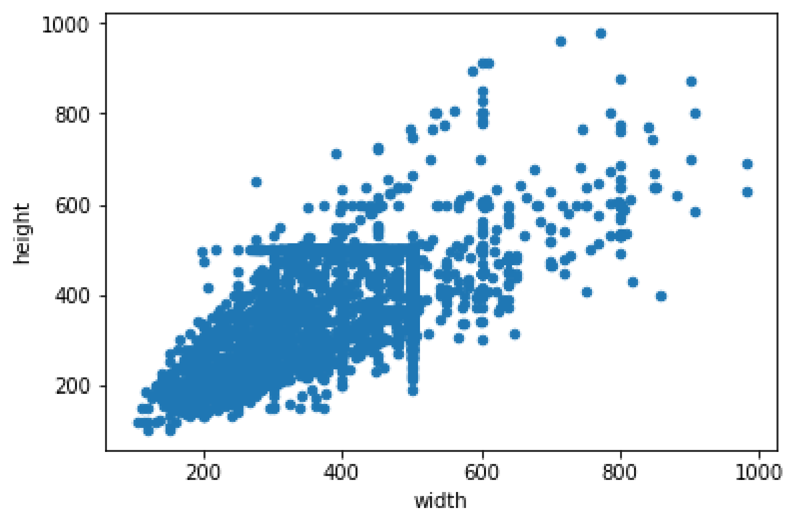
Investigate Image Sizes

In [23]:
```
size_df=pd.DataFrame([(i,)+PIL.Image.open(PATH+i).size for i in data.trn_ds
                columns=['fn','width','height'])
```

In [24]:
```
size_df.plot(kind='scatter',x='width',y='height');
```



In [25]:
```
size_df[(size_df.width<1000)&
        (size_df.height<1000)].plot(kind='scatter',x='width',y='height');
```



In [26]:
```
len(data.trn_ds),len(data.test_ds)
```

Out[26]:
```
(8178, 10357)
```

In [27]:
```
len(data.classes)
```
Out[27]:
```
120
```

First Quick Model

- arch = resnext101_64
- tfms = tfms_from_model(arch,sz,aug_tfms=transforms_side_on,max_zoom=1.1)
- data = ImageClassifierData.from_csv(PATH, tfms=tfms, bs=bs)
- learn = ConvLearner.pretrained(arch, data, precompute=True)
- learn.fit(0.01, 2)

Initial Model

In [33]:
```
def get_data(sz,bs):
    tfms = tfms_from_model(arch,sz,aug_tfms=transforms_side_on,max_zoom=1.1
    data = ImageClassifierData.from_csv(PATH,'train',PATH+'labels.csv',test
                                         val_idxs=labels_valid.index,
                                         suffix='.jpg',tfms=tfms,bs=bs)

    # Quick check to resize large images before further analysis
    return data if sz>300 else data.resize(340,'tmp')
```

Precompute

In [*]:
```
# data=get_data(sz,bs)
# data=get_data(350,bs=20)
data=get_data(200,bs=20)
```

In [76]:
```
learn=ConvLearner.pretrained(arch,data,precompute=True)
```

Hyperparameters

Hyperparameters cannot be directly learned from the regular training process. These parameters represent “higher-level” properties of the model such as its complexity or how fast it should learn. e.g. learning rate and the number of epochs.

In [77]:
```
learn.fit(1e-2,3)
```

Epoch 100% 3/3 [00:08<00:00, 2.82s/it]

```
epoch      trn_loss   val_loss   accuracy
    0      0.936773   0.375145   0.927104
    1      0.3937     0.278635   0.931018
    2      0.245153   0.244518   0.934442
```

Out[77]:
```
[array([0.24452]), 0.9344422747244341]
```

- Accuracy = ratio of correct predictions / total predictions
- Loss function or cost function is representing the price paid for inaccuracy of predictions.
  - loss associated with one example in binary classification is given by: -(y * log(p) + (1-y) * log (1-p)) where y is the true label of x and p is the probability predicted by our model that the label is 1.

In [78]:
```
# This is the label for a val data
data.val_y
```

Out[78]:
```
array([19, 37, 15, ..., 98, 73,  3])
```

In [46]:
```
# Classes i.e. affenpinscher is label 0, afghan_hound is label 1 etc.
data.classes[:5]
```

Out[46]:
```
['affenpinscher',
 'afghan_hound',
 'african_hunting_dog',
 'airedale',
 'american_staffordshire_terrier']
```

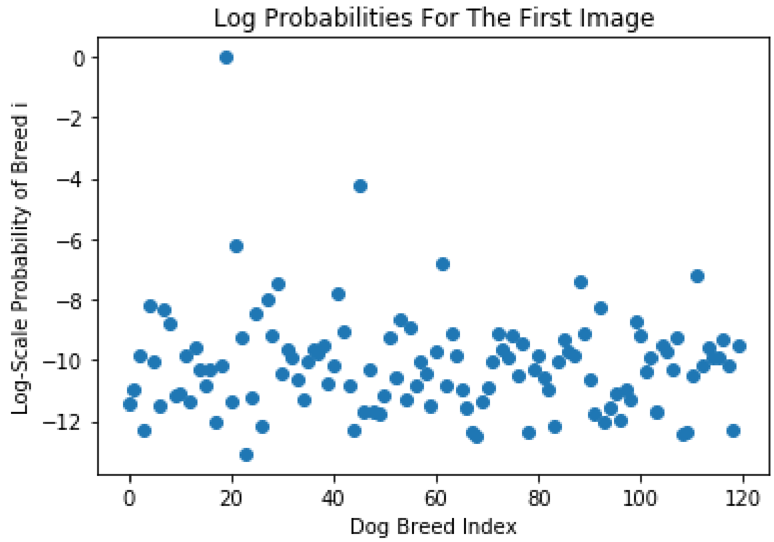
In [47]:
```
log_preds=learn.predict()
```

In [48]:
```
log_preds.shape
```

Out[48]:
```
(2044, 120)
```

i.e. For each image in the validation set, there is a probability that the image pertains to that one (of 120) specific breeds.

In [49]:
```
plt.scatter(range(len(log_preds[0])),log_preds[0])
plt.title('Log Probabilities For The First Image')
plt.xlabel('Dog Breed Index')
plt.ylabel('Log-Scale Probability of Breed i');
```



In [50]:
```
# np.argmax returns the index associated with the highest probability.
# i.e. the most likely breed for each image
preds = np.argmax(log_preds, axis=1)

# learn.predict() returns log-scale probabilities,
# therefore convert these into probabilitys from 0 to 1.
# i.e. probability of breed associated with it's index.
probs = np.exp(log_preds[:,1])
```

In [51]:
```
def print_img_pred(i):
    print ('img:',labels_valid.iloc[i].id,'\n'
           'breed:',labels_valid.iloc[i].breed,'\n'
           'predicted breed:',data.classes[preds[i]],'\n'
           'probability:',probs[i],'\n')
```

In [52]:
```
print_img_pred(1)
print_img_pred(15)
```

```
img: 001513dfcb2ffafc82cccf4d8bbaba97
breed: dingo
predicted breed: dingo
probability: 3.8751173e-06

img: 015b363b062f602e7ec04ce28e640d05
breed: walker_hound
predicted breed: walker_hound
probability: 6.56673e-06
```

In [53]:
```
valid_df=labels_valid.copy()
```

In [54]:
```
valid_df['pred_breed']=[data.classes[i] for i in preds]
valid_df['probability']=[i for i in probs]
```

In [55]:
```
valid_df.head()
```

Out[55]:

	id	breed	pred_breed	probability
0	000bec180eb18c7604dcecc8fe0dba07	boston_bull	boston_bull	0.000017
1	001513dfcb2ffafc82cccf4d8bbaba97	dingo	dingo	0.000004
3	00214f311d5d2247d5dfe4fe24b2303d	bluetick	bluetick	0.000171
8	003df8b8a8b05244b1d920bb6cf451f9	basenji	basenji	0.000015
10	004396df1acd0f1247b740ca2b14616e	shetland_sheepdog	shetland_sheepdog	0.000025

In [56]:
```
print (valid_df.loc[valid_df.breed==valid_df.pred_breed].shape[0],
       'correct predictions out of',valid_df.shape[0])
```

```
1906 correct predictions out of 2044
```

Analyzing Predictions

In [57]:
```
def rand_by_mask(mask): return np.random.choice(np.where(mask)[0], min(len(p
def rand_by_correct(is_correct): return rand_by_mask((preds == data.val_y)==

def plots(ims, figsize=(12,6), rows=1, titles=None):
    f = plt.figure(figsize=figsize)
    for i in range(len(ims)):
        sp = f.add_subplot(rows, len(ims)//rows, i+1)
        sp.axis('Off')
        if titles is not None: sp.set_title(titles[i], fontsize=16)
        plt.imshow(ims[i])

def load_img_id(ds, idx): return np.array(PIL.Image.open(PATH+ds.fnames[idx]

def plot_val_with_title(idxs, title):
    imgs = [load_img_id(data.val_ds,x) for x in idxs]
    title_probs = [probs[x] for x in idxs]
    print(title)
    return plots(
        imgs, rows=1, titles=title_probs, figsize=(16,8)
            ) if len(imgs)>0 else print('Not Found.')

def most_by_mask(mask, mult):
    idxs = np.where(mask)[0]
    return idxs[np.argsort(mult * probs[idxs])[:4]]

def most_by_correct(y, is_correct):
    mult = -1 if (y==1)==is_correct else 1
    return most_by_mask((
        (preds == data.val_y)==is_correct)&
        (data.val_y == y), mult)
```

In [58]:
```
# 1. A few correct labels at random
plot_val_with_title(rand_by_correct(True), "Correctly classified")
```

Correctly classified

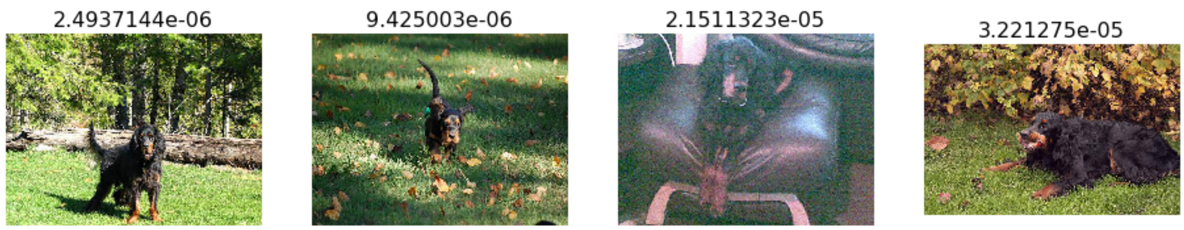
In [59]:
```
plot_val_with_title(rand_by_correct(False), "Incorrectly classified")
```

Incorrectly classified

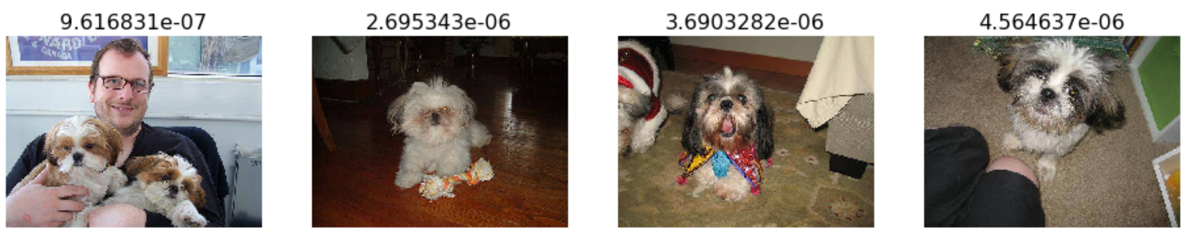
In [60]:
```python
for i in [2,50,100]:
    plot_val_with_title(most_by_correct(i, True), "Most correct "+data.clas
    plt.show()
```

```
Most correct african_hunting_dog
```


```
Most correct gordon_setter
```




```
Most correct shih-tzu
```



In [61]:
```
for i in [2,50,100]:
    plot_val_with_title(most_by_correct(i, False), "Most Incorrect "+data.c
    plt.show()
```

```
Most Incorrect african_hunting_dog
Not Found.
Most Incorrect gordon_setter
```

0.0012645988

```
Most Incorrect shih-tzu
```

0.00091145857

0.00039997554

0.000253731

0.00018092537

In [62]:
```python
most_uncertain = np.argsort(np.abs(probs -0.5))[:4]
plot_val_with_title(most_uncertain, "Most uncertain predictions")
```

```
Most uncertain predictions
```

Improving Predictions

Learning Rate Finder

function based on Leslie Smith's research on Cyclical Learning Rates for Training NNs. [https://arxiv.org/pdf/1506.01186.pdf (https://arxiv.org/pdf/1506.01186.pdf)](https://arxiv.org/pdf/1506.01186.pdf)

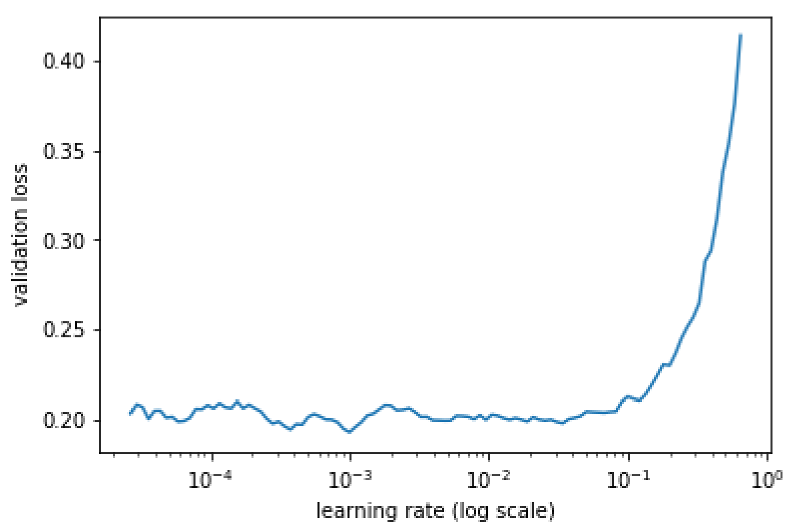
In [ ]:
```python
# learn=ConvLearner.pretrained(arch,data,precompute=True)
```

In [79]:
```python
lrf=learn.lr_find()
```

Epoch 0% 0/1 [00:00<?, ?it/s]

```
84%|█████████   | 118/141 [00:02<00:00, 49.45it/s, loss=0.609]
```

In [80]:
```python
learn.sched.plot()
```



A strange looking Loss vs Learning Rate plot, however the loss is still improving somewhat at lr=1e-2 (0.01), which is often a good LR starting point.

In [ ]:

```

```

In [81]:

```
learn.precompute=False
```

In [82]:
```
learn.fit(1e-2, 3, cycle_len=1)
```

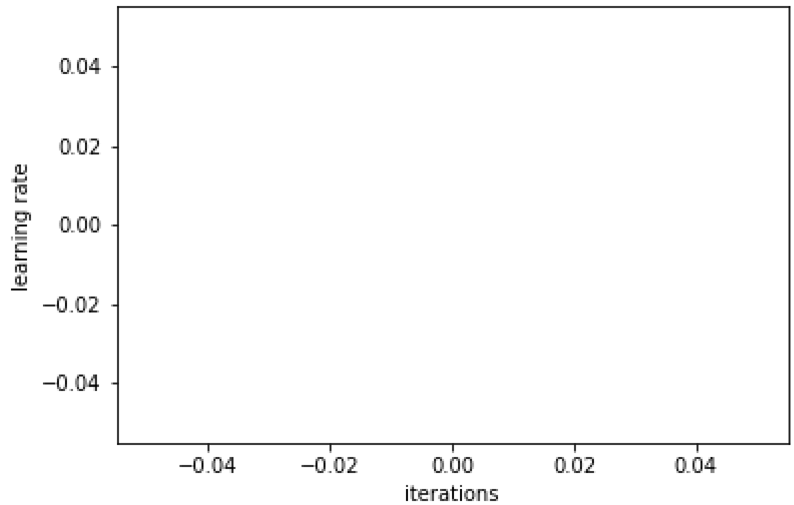
Epoch 0% 0/3 [00:00<?, ?it/s]

```
  0%|          | 0/141 [00:00<?, ?it/s]

---------------------------------------------------------------------------
RuntimeError                              Traceback (most recent call last)
<ipython-input-82-6a6bf8b06eed> in <module>
----> 1 learn.fit(1e-2, 3, cycle_len=1)

~/fastai/courses/dl1/fastai/learner.py in fit(self, lrs, n_cycle, wds, **kwargs)
    300         self.sched = None
    301         layer_opt = self.get_layer_opt(lrs, wds)
--> 302         return self.fit_gen(self.model, self.data, layer_opt, n_cycle, **kwargs)
    303 
    304     def warm_up(self, lr, wds=None):

~/fastai/courses/dl1/fastai/learner.py in fit_gen(self, model, data, layer_opt, n_cycle, cycle_len, cycle_mult, cycle_save_name, best_save_name, use_clr, use_clr_beta, metrics, callbacks, use_wd_sched, norm_wds, wds_sched_mult, use_swa, swa_start, swa_eval_freq, **kwargs)
    247             metrics=metrics, callbacks=callbacks, reg_fn=self.reg_fn, clip=self.clip, fp16=self.fp16,
    248             swa_model=self.swa_model if use_swa else None, swa_start=swa_start,
--> 249             swa_eval_freq=swa_eval_freq, **kwargs)
    250 
    251     def get_layer_groups(self): return self.models.get_layer_groups()

~/fastai/courses/dl1/fastai/model.py in fit(model, data, n_epochs, opt, crit, metrics, callbacks, stepper, swa_model, swa_start, swa_eval_freq, visualize, **kwargs)
    139             batch_num += 1
    140             for cb in callbacks: cb.on_batch_begin()
--> 141             loss = model_stepper.step(V(x),V(y), epoch)
    142             avg_loss = avg_loss * avg_mom + loss * (1-avg_mom)
    143             debias_loss = avg_loss / (1 - avg_mom**batch_num)

~/fastai/courses/dl1/fastai/model.py in step(self, xs, y, epoch)
     48     def step(self, xs, y, epoch):
     49         xtra = []
---> 50         output = self.m(*xs)
     51         if isinstance(output,tuple): output,*xtra = output
     52         if self.fp16: self.m.zero_grad()

~/anaconda3/envs/fastai/lib/python3.6/site-packages/torch/nn/modules/module.py in __call__(self, *input, **kwargs)
    355             result = self._slow_forward(*input, **kwargs)
    356         else:
--> 357             result = self.forward(*input, **kwargs)
```

```
    358         for hook in self._forward_hooks.values():
    359             hook_result = hook(self, input, result)

~/anaconda3/envs/fastai/lib/python3.6/site-packages/torch/nn/modules/cont
ainer.py in forward(self, input)
     65     def forward(self, input):
     66         for module in self._modules.values():
---> 67             input = module(input)
     68         return input
     69 

~/anaconda3/envs/fastai/lib/python3.6/site-packages/torch/nn/modules/modu
le.py in __call__(self, *input, **kwargs)
    355             result = self._slow_forward(*input, **kwargs)
    356         else:
--> 357             result = self.forward(*input, **kwargs)
    358         for hook in self._forward_hooks.values():
    359             hook_result = hook(self, input, result)

~/anaconda3/envs/fastai/lib/python3.6/site-packages/torch/nn/modules/con
v.py in forward(self, input)
    280     def forward(self, input):
    281         return F.conv2d(input, self.weight, self.bias, self.strid
e,
--> 282                         self.padding, self.dilation, self.groups)
    283 
    284 

~/anaconda3/envs/fastai/lib/python3.6/site-packages/torch/nn/functional.p
y in conv2d(input, weight, bias, stride, padding, dilation, groups)
     88                 _pair(0), groups, torch.backends.cudnn.benchmark,
     89                 torch.backends.cudnn.deterministic, torch.backend
s.cudnn.enabled)
---> 90     return f(input, weight, bias)
     91 
     92 

RuntimeError: cuda runtime error (2) : out of memory at /opt/conda/conda-
bld/pytorch_1518244421288/work/torch/lib/THC/generic/THCStorage.cu:58
```

cycle_len = 1: a technique called stochastic gradient descent with restarts (SGDR) a variant of learning rate annealing i.e. change the LR as you move along the GD. Instead of linearly dropping the LR, we will use a half COSINE and repeat the process in order to find a zone in the GD that is accurate and stable. From time to time the LR will restart and jump to a different area.

In [83]:
```
learn.sched.plot_lr()
```



Refining Model Utilizing Learning Rate Annealing

In [72]:
```
learn.unfreeze()
```

Earlier layers from the selected pre-trained architecture have already been trained to recognize imagenet photos, in contrast to our final layers. Thus, be cautious of extremely modifying the fine tuned pretrained early layers once the model is unfrozen.

To reiterate, the earlier layers are more general-purpose features and therefore should probably require less fine-tuning for new datasets. Thus different learning rates are recommended for different layers: the first few layers will be at 1e-4, the middle layers at 1e-3, and our FC layers we'll leave at 1e-2 as before. We refer to this as *differential learning rates*, although there's no standard name for this techique in the literature that we're aware of.

In [69]:
```
lr=np.array([1e-4,1e-3,1e-2])
```

In [70]:
```python
learn.fit(lr, 3, cycle_len=1, cycle_mult=2)
```

```
~/fastai/courses/dl1/fastai/models/resnext_101_64x4d.py in forward(self,
 input)
     22 class LambdaMap(LambdaBase):
     23     def forward(self, input):
---> 24         return list(map(self.lambda_func,self.forward_prepare(inp
ut)))
     25 
     26 class LambdaReduce(LambdaBase):

~/fastai/courses/dl1/fastai/models/resnext_101_64x4d.py in forward_prepar
e(self, input)
     13         output = []
     14         for module in self._modules.values():
---> 15             output.append(module(input))
     16         return output if output else input
     17 

~/anaconda3/envs/fastai/lib/python3.6/site-packages/torch/nn/modules/modu
```

Additional Use of Data Augmentation

At *inference* (test) time, make predictions on validation set as well as augmented versions of the validation set i.e. original image + 4 randomly augmented version. Then take the average predictions of these images.
This is called Test Time Augmentation (TTA)

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

In [ ]:
```python
accuracy_np(probs, y)
```

In [ ]:
```python

```

In [ ]:
```python
preds = np.argmax(probs, axis=1)
probs = probs[:,1]
```

In [ ]:
```python
from sklearn.metrics import confusion_matrix
cm = confusion_matrix(y, preds)
```

In [ ]:
```python
plot_confusion_matrix(cm, data.classes)
```

In [ ]:
```python

```

Additional notes

data parameters:

- Data.rns_ds.fnames list of the file names
- data. `__dict__` .keys()
- data.trn_ds. `__dict__` .keys()

learn parameters:

- learn.fit(…….
  - LR
  - number of epochs
  - cycle_len = 1
  - cycle_mult = 2 e.g. doubling the # of iterations in the epoch
- learn.predict() predications for validation set in log-scale
- learn.lr_find()
- learn.sched.plot_lr() and .plot() to plot learning rate finder results
- learn.precompute = False
  - Precompute utilized to determine what the network does prior to training
  - freeze/unfreeze utilized to determine what the network does during training.
- Precompute = True precomputes the activations for all but the last layers. But those activations can change during the training depending on whether layers are frozen or unfrozen. In the case, precompute = True, layer.unfreeze(), Initial weights in the network are from precomputed activations. During the training, those weights changes in all the layers as now you have kepts weights in every layer as trainable parameter.
- learn.unfreeze() indicates whether the parameters in the network are trainable or not during training of the network.
  - Irrespective of precompute is True/False, weights in the network will change during the training it the layers are unfrozen.
- Precompute = False, learn.freeze() means only the last layer can learn now and all the others layers have random initialization of weights which are not optimised for any dataset. Hence, they are far away from local minima most of the times and setting every layer except last layer severely limits the learning ability of network.
- pre-compute refers to activations before the last layer.
- freeze/un-freeze refers to weights of the layers before the last layer
- learn.save('model_name') save model before unfreezing and retraining earlier layers.
- learn.load('model_name')
- learn.TTA() test time augmentation, only use at inference (or test) time (on validation set)
  - TTA makes simple predictions on images in the validation set AND also a number of randomly augmented version of them too
  - By default it uses original images and 4 randomly augmented versions and then takes the average predictions from these images
  - log_preds,y = learn.TTA()
  - probs = np.mean(np.exp(log_preds),0)
  - accuracy_np(probs, y)