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

In [2]:
```
cd C:\python related\fastai
```

```
C:\python related\fastai
```

In [3]:
```
from fastai.conv_learner import *
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 [4]:
```
arch=resnext101_64
```

In [5]:
```
PATH ="C:/Users/admin/Desktop/45_top_roof/"
```

In [6]:
```
label_csv= f'{PATH}labels.csv'
n=len(list(open(label_csv)))-1
val_idxs = get_cv_idxs(n)
```

In [7]:
```
def get_data(sz,bs):
    tfms=tfms_from_model(arch, sz, aug_tfms=transforms_top_down,max_zoom=1.1)
    data=ImageClassifierData.from_csv(PATH,'train',f'{PATH}labels.csv',test_na
me='test',num_workers=4,
                                    val_idxs=val_idxs,suffix='.jpg',tfms=tfms,bs
=bs)
    return data if sz>300 else data.resize(340,'tmp')
```

In [8]:
```
learn = ConvLearner.pretrained(arch,get_data(128,10),precompute=True)
```

In [9]:
```
learn.fit(0.0025,1)
```

```
epoch      trn_loss   val_loss   accuracy

    0      1.094905   0.735745   0.710526
```

Out[9]:
```
[0.73574525, 0.7105263126523871]
```

In [9]:
```
learn.precompute=False
```

In [10]:
```
learn.fit(0.0025,3,cycle_len=2,cycle_mult=2)
```

```
epoch      trn_loss   val_loss   accuracy
    0      1.047337   0.790557   0.694737
    1      0.901853   0.740711   0.718421
    2      0.944017   0.710935   0.7
    3      0.839106   0.677871   0.723684
    4      0.78139    0.687236   0.728947
    5      0.829066   0.685289   0.721053
    6      0.798702   0.713715   0.723684
    7      0.835192   0.694386   0.736842
    8      0.781453   0.660521   0.75
    9      0.762389   0.675494   0.731579
    10     0.71981    0.651968   0.765789
    11     0.668231   0.625537   0.763158
    12     0.671672   0.621763   0.763158
    13     0.710721   0.62504    0.771053
```

Out[10]: [0.62504, 0.7710526271870262]

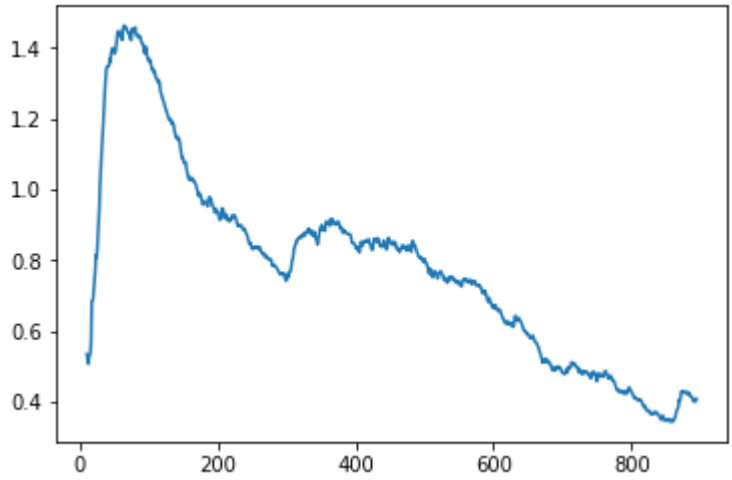
In [11]:
```
lr=0.0025
lrs= np.array([lr/9,lr/3,lr])
learn.unfreeze()
learn.bn_freeze(True)
```

In [12]:
```
learn.fit(lrs,2,cycle_len=2,cycle_mult=2)
```

```
epoch      trn_loss   val_loss   accuracy

    0      1.087849   0.839694   0.673684
    1      0.747818   0.657045   0.731579

    2      0.857808   0.800711   0.707895

    3      0.681105   0.548005   0.778947

    4      0.482625   0.521697   0.815789

    5      0.408998   0.45001    0.834211
```

Out[12]: [0.45000973, 0.8342105216885868]

In [13]:
```
learn.sched.plot_loss()
```



In [14]:
```
learn.save('45_type_res101')
```

In [12]:
```
learn.load('45_type_res101')
```

In [28]:
```
learn.set_data(get_data(256,5))
```

In [29]:
```
learn.freeze()
learn.fit(0.002,3,cycle_len=1,cycle_mult=2)
```

```
epoch      trn_loss   val_loss   accuracy

    0      0.376022   0.456758   0.850667
    1      0.444866   0.459038   0.842667

    2      0.392767   0.464406   0.837333

    3      0.458624   0.50721    0.829333

    4      0.418079   0.477186   0.832

    5      0.318718   0.462265   0.848

    6      0.353512   0.44151    0.856
```

Out[29]:
```
[0.4415097, 0.8560000060002009]
```

In [31]:
```
learn.save('45_type_res101')
```

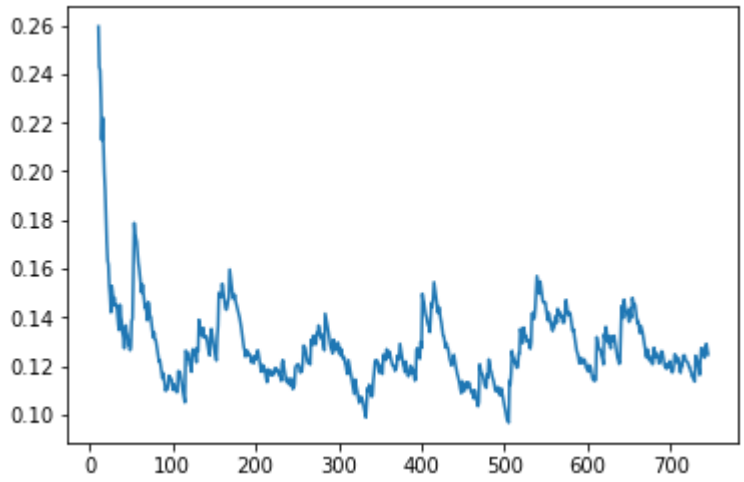
In [14]:
```
learn.load('45_type_res101')
```

In [27]:
```
learn.freeze()
learn.fit(0.002,5)
```

```
epoch      trn_loss   val_loss   accuracy

    0      0.129977   0.330079   0.902632
    1      0.126438   0.342032   0.886842

    2      0.110216   0.342365   0.892105

    3      0.122587   0.335631   0.894737

    4      0.124645   0.343857   0.897368
```

Out[27]:
```
[0.34385678, 0.8973684185429623]
```

In [28]:
```
learn.sched.plot_loss()
```



In [10]:
```
learn.set_data(get_data(500,5))
```

In [15]:
```
learn.fit(0.002,3)
```

epoch	trn_loss	val_loss	accuracy
0	0.861149	0.469517	0.829333
1	0.674875	0.483616	0.818667
2	0.7417	0.471676	0.824

Out[15]: [0.47167593, 0.8240000069141388]

In [16]:
```
lr=0.002
lrs= np.array([lr/9,lr/3,lr])
learn.fit(lrs,3,cycle_len=1,cycle_mult=2)
```

epoch	trn_loss	val_loss	accuracy
0	0.652587	0.478341	0.802667
1	0.588298	0.443993	0.842667
2	0.572753	0.413714	0.850667
3	0.664576	0.479893	0.832
4	0.56774	0.419581	0.858667
5	0.614655	0.429554	0.832
6	0.54791	0.407386	0.850667

Out[16]: [0.407386, 0.8506666727860769]

In [17]:
```
learn.save('45_top_roof_res101')
```

In [15]:
```
learn.load('45_top_roof_res101')
```

In [15]:
```
learn.fit(0.002,3)
```

```
epoch      trn_loss   val_loss   accuracy

    0      0.579885   0.458729   0.84
    1      0.641549   0.42089    0.829333

    2      0.565723   0.430086   0.861333
```

Out[15]:
```
[0.43008617, 0.861333339413007]
```

In [16]:
```
log_preds,y=learn.TTA()
probs = np.mean(np.exp(log_preds), axis=0)
probs.shape
```

Out[16]:
```
(371, 4)
```

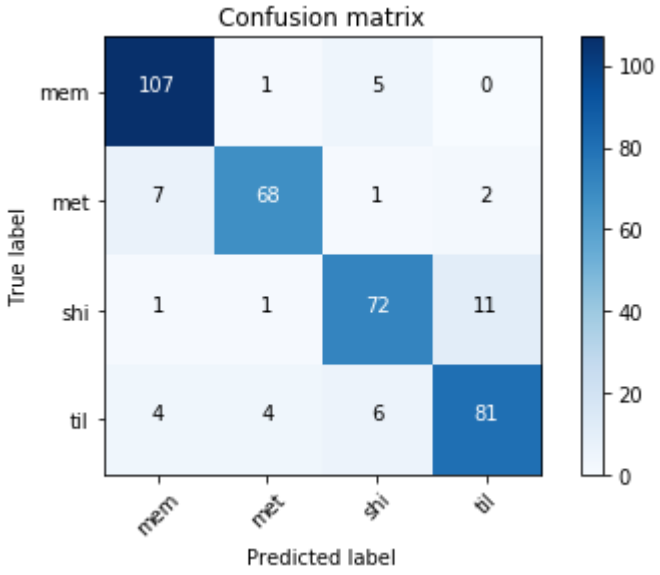
In [17]:
```
a=accuracy_np(probs, y)
a
```

Out[17]:
```
0.8840970350404312
```

In [18]:
```
preds = np.argmax(probs, axis=1)
probs = probs[:,1]

from sklearn.metrics import confusion_matrix
cm = confusion_matrix(y, preds)
```

In [20]:
```
data = get_data(500,5)
plot_confusion_matrix(cm, data.classes)
```

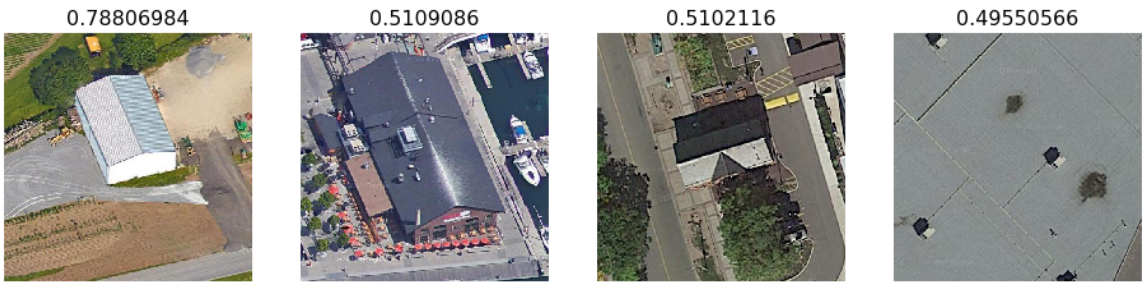
```
[[107   1   5   0]
 [  7  68   1   2]
 [  1   1  72  11]
 [  4   4   6  81]]
```



In [21]:
```
def rand_by_mask(mask): return np.random.choice(np.where(mask)[0], 4, replace=
False)
def rand_by_correct(is_correct): return rand_by_mask((preds == data.val_y)==is
_correct)

def plot_val_with_title(idxs, title):
    imgs = np.stack([data.val_ds[x][0] for x in idxs])
    title_probs = [probs[x] for x in idxs]
    print(title)
    return plots(data.val_ds.denorm(imgs), rows=1, titles=title_probs)

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))

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)
```

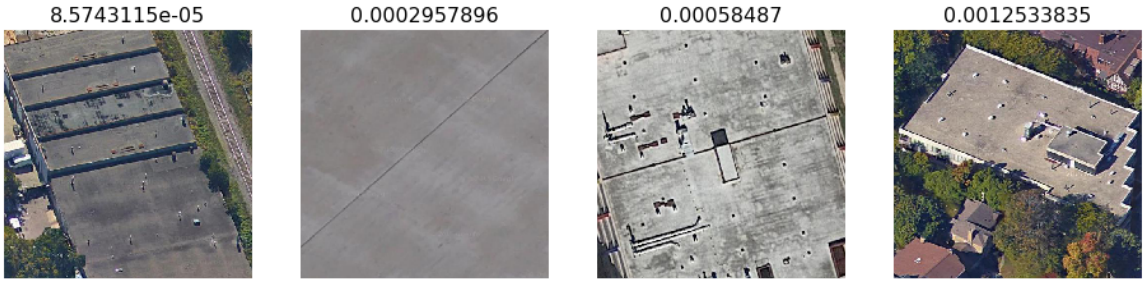
In [31]:
```
plot_val_with_title(most_by_correct(0, False), "Most incorrect membrane")
```

```
Most incorrect membrane
```



In [42]:
```
plot_val_with_title(most_by_correct(0, True), "Most correct membrane")
```

```
Most correct membrane
```



In [34]:
```
log_preds,y=learn.TTA(is_test=True)
probs = np.mean(np.exp(log_preds), axis=0)
probs.shape
```

Out[34]:
```
(588, 4)
```

In [36]:
```
data = get_data(500,5)
ds = pd.DataFrame(probs)
ds.columns= data.classes
```

In [37]:
```
ds
```

Out[37]:

	mem	met	shi	til
0	0.979027	0.000997	0.017565	0.002411
1	0.224203	0.015699	0.731048	0.029049
2	0.841481	0.033236	0.084216	0.041067
3	0.994122	0.001446	0.003053	0.001379
4	0.942838	0.031418	0.007250	0.018494
5	0.480283	0.013204	0.221317	0.285195
6	0.980631	0.007840	0.008228	0.003300
7	0.724848	0.221680	0.016982	0.036490
8	0.895583	0.091411	0.002330	0.010675
9	0.637288	0.175646	0.143259	0.043808
10	0.901318	0.048721	0.005941	0.044021
11	0.603855	0.023223	0.265893	0.107029
12	0.914495	0.040077	0.017256	0.028172
13	0.514339	0.233660	0.106381	0.145620
14	0.951504	0.006738	0.040622	0.001136
15	0.876411	0.017241	0.094162	0.012186
16	0.986994	0.010716	0.001447	0.000842
17	0.801728	0.019571	0.119277	0.059424
18	0.971952	0.014623	0.005092	0.008333
19	0.867314	0.099371	0.027980	0.005335
20	0.634928	0.348428	0.012252	0.004392
21	0.985076	0.009228	0.002119	0.003577
22	0.498466	0.349525	0.086133	0.065876
23	0.994219	0.003703	0.001374	0.000704
24	0.531976	0.189448	0.162013	0.116563
25	0.815779	0.086014	0.064236	0.033972
26	0.723444	0.094407	0.084987	0.097162
27	0.522458	0.309744	0.149423	0.018375
28	0.015804	0.824364	0.048574	0.111258
29	0.024919	0.909435	0.034657	0.030989
...	...	...	...	...
558	0.000246	0.001208	0.004583	0.993964

	mem	met	shi	til
559	0.001695	0.090774	0.016589	0.890942
560	0.010231	0.016477	0.077158	0.896133
561	0.004449	0.010217	0.007849	0.977485
562	0.000275	0.008235	0.025477	0.966012
563	0.000125	0.001502	0.009247	0.989126
564	0.000382	0.009282	0.102551	0.887785
565	0.000796	0.007876	0.485307	0.506021
566	0.000763	0.004225	0.757780	0.237232
567	0.000887	0.002046	0.100177	0.896890
568	0.000342	0.023387	0.641378	0.334893
569	0.001433	0.000896	0.117029	0.880642
570	0.002609	0.028197	0.552689	0.416505
571	0.000270	0.001674	0.451076	0.546981
572	0.008124	0.002069	0.020295	0.969511
573	0.015762	0.003720	0.049765	0.930753
574	0.000060	0.000817	0.009838	0.989285
575	0.042411	0.006103	0.089949	0.861537
576	0.006196	0.005819	0.067400	0.920585
577	0.006475	0.010576	0.114602	0.868347
578	0.000440	0.002151	0.008162	0.989247
579	0.000685	0.001031	0.014309	0.983974
580	0.004209	0.019120	0.147297	0.829374
581	0.000177	0.000645	0.046264	0.952914
582	0.008128	0.058761	0.009481	0.923630
583	0.034455	0.028686	0.014016	0.922843
584	0.054405	0.001137	0.050556	0.893901
585	0.081122	0.001914	0.113688	0.803275
586	0.000132	0.001451	0.007158	0.991258
587	0.002271	0.000670	0.090543	0.906516

588 rows × 4 columns

In [38]:
```
ds.insert(0,'id',[o[5:-4] for o in data.test_ds.fnames])
```

In [39]:
```
ds.to_csv("test_roof_prediction.csv", encoding='utf-8',index=False)
```