1. Importing Required Libraries

In [2]:
```
# Import Required Python Packages :

import warnings
warnings.filterwarnings('ignore')

# Setting up our enviroment
# Data Viz & Regular Expression Libraries :
%reload_ext autoreload
%autoreload 2
%matplotlib inline

# Scientific and Data Manipulation Libraries :
import numpy as np # linear algebra
import pandas as pd # data processing, CSV file I/O (e.g. pd.read_csv)

# Import FastAI library
from fastai import *
from fastai.vision import *
from fastai.metrics import error_rate
import os
```

In [3]:
```
x  = '/kaggle/input/cat-and-dog/training_set/training_set'
path = Path(x)
path.ls()
```

Out[3]:
```
[PosixPath('/kaggle/input/cat-and-dog/training_set/training_set/dogs'),
 PosixPath('/kaggle/input/cat-and-dog/training_set/training_set/cats')]
```

In [4]:
```

np.random.seed(40)
data = ImageDataBunch.from_folder(path, train = '.', valid_pct=0.2,
                                  ds_tfms=get_transforms(), size=224,
                                  num_workers=4).normalize(imagenet_stats)
```

In [5]:
```
data.show_batch(rows=4, figsize=(7,6),recompute_scale_factor=True)
```

In [6]:
```
data
```

Out[6]:
```
ImageDataBunch;

Train: LabelList (6404 items)
x: ImageList
Image (3, 224, 224),Image (3, 224, 224),Image (3, 224, 224),Image (3, 224, 22
4),Image (3, 224, 224)
y: CategoryList
dogs,dogs,dogs,dogs,dogs
Path: /kaggle/input/cat-and-dog/training_set/training_set;

Valid: LabelList (1601 items)
x: ImageList
Image (3, 224, 224),Image (3, 224, 224),Image (3, 224, 224),Image (3, 224, 22
4),Image (3, 224, 224)
y: CategoryList
cats,dogs,cats,dogs,dogs
Path: /kaggle/input/cat-and-dog/training_set/training_set;

Test: None
```

In [7]:
```
print(data.classes)
len(data.classes)
data.c
```

```
['cats', 'dogs']
```

Out[7]:
```
2
```

Create Model

In [8]:
```
learn = cnn_learner(data, models.resnet34, metrics=[accuracy], model_dir = Pat
```

```
Downloading: "https://download.pytorch.org/models/resnet34-333f7ec4.pth" to /
root/.cache/torch/checkpoints/resnet34-333f7ec4.pth

HBox(children=(FloatProgress(value=0.0, max=87306240.0), HTML(value='')))
```

Training Neural Network

To find the perfect learning rates we can use the lr_find and recorder.plot methods which create a plot that relates the learning rate with the loss.

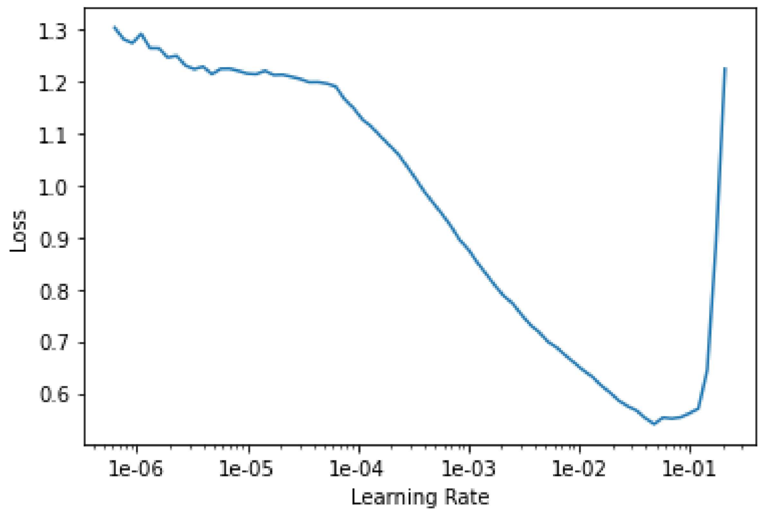
In [9]:
```
learn.lr_find()
learn.recorder.plot(suggestions=True)
```

0.00% [0/1 00:00<00:00]

epoch	train_loss	valid_loss	accuracy	time

84.00% [84/100 01:15<00:14 2.1211]

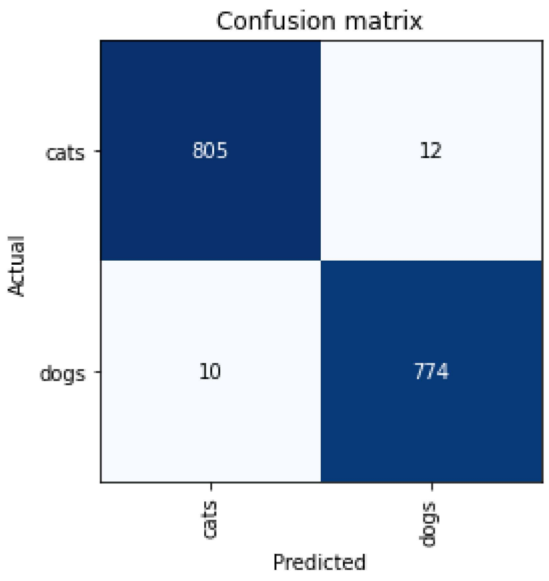
```
LR Finder is complete, type {learner_name}.recorder.plot() to see the graph.
```



In [10]:
```
lr1 = 1e-3
lr2 = 1e-1
learn.fit_one_cycle(4,slice(lr1,lr2))
```

epoch	train_loss	valid_loss	accuracy	time
0	0.472560	0.557231	0.952530	01:42
1	0.366406	0.074333	0.977514	01:40
2	0.166839	0.047577	0.988757	01:40
3	0.063848	0.041019	0.986259	01:41

In [12]:
```
interp = ClassificationInterpretation.from_learner(interp)
interp.plot_confusion_matrix()
```



5. Prediction using trained model

In [13]:
```
img = open_image('../input/cat-and-dog/test_set/test_set/dogs/dog.4001.jpg')
print(learn.predict(img)[0])
img
```

```
dogs
```

Out[13]:

In [19]:
```
from sklearn.metrics import accuracy_score, precision_score, recall_score, f1_

# Get true labels and predictions
true_labels = [str(data.classes[i]) for i in interp.y_true]
predicted_labels = [interp.data.classes[i] for i in interp.pred_class]

# Calculate accuracy
accuracy = accuracy_score(true_labels, predicted_labels)

# Calculate precision, recall, and F1-score
precision = precision_score(true_labels, predicted_labels, average='weighted')
recall = recall_score(true_labels, predicted_labels, average='weighted')
f1 = f1_score(true_labels, predicted_labels, average='weighted')

print(f"Accuracy: {accuracy}")
print(f"Precision: {precision}")
print(f"Recall: {recall}")
print(f"F1-Score: {f1}")
```

```
Accuracy: 0.9862585883822611
Precision: 0.986262332588285
Recall: 0.9862585883822611
F1-Score: 0.9862589208911284
```

In [ ]:
```
from flask import Flask, render_template, request, jsonify
from fastai.vision import open_image, load_learner

app = Flask(__name__)

# Load your trained model
model = load_learner("path/to/your/model")

@app.route("/", methods=["GET", "POST"])
def index():
    if request.method == "POST":
        image_file = request.files["file"]
        if image_file:
            # Save the uploaded image temporarily
            image_path = "temp.jpg"
            image_file.save(image_path)

            # Make a prediction
            img = open_image(image_path)
            prediction = model.predict(img)[0]

            return jsonify({"result": str(prediction)})

    return render_template("index.html")

if __name__ == "__main__":
    app.run(debug=True)
```