Artificial Intelligence Image Classification using a proprietary indian snacks dataset

Developed by Dharak Vasavda

I will train a convolutional neural network to recognize three different types of indian sweets known as mithai.

Prerequisites

Import Statements

In [5]:
```
%reload_ext autoreload
%autoreload 2
%matplotlib inline
```

In [6]:
```
from fastai.vision import *
from fastai.metrics import error_rate
```

Global Batch Size. Reduce to ~16 this if using a small GPU.

In [7]:
```
batch_size=64
```

Gathering Training Data

Set the path for the data

In [8]:
```
path = Path('data/snacks')

dest_gulab_jamun = path/'gulab_jamun'
dest_kaju_katli = path/'kaju_katli'
dest_laddoo = path/'laddoo'

dest_gulab_jamun.mkdir(parents=True, exist_ok=True)
dest_kaju_katli.mkdir(parents=True, exist_ok=True)
dest_laddoo.mkdir(parents=True, exist_ok=True)
```

Specify classes for the images

In [9]:
```
classes = ['kaju_katli', 'gulab_jamun', 'laddoo']
```

Download all images using urls from their respective urls_*.txt All images will go into their respective data/snacks/... directory

In [11]:
```
doc(download_images)
```

In [13]:
```
download_images(path/'urls_gulab_jamun.txt', dest_gulab_jamun, max_pics=200)
download_images(path/'urls_kaju_katli.txt', dest_kaju_katli, max_pics=200)
download_images(path/'urls_laddoo.txt', dest_laddoo, max_pics=200)
```

Create our ImageDataBunch object. We can normalize our images to keep the same mean and standard deviation and improve performance. Images will be resized to 224x224 to maintain consistent GPU calculations.

In [16]:
```
data = ImageDataBunch.from_folder(path, train=".", valid_pct=0.2,
        ds_tfms=get_transforms(), size=224, num_workers=4).normalize(imagenet_stats)
```

Visualize some of our data to verify we are on the right track.

In [17]:
```
data.show_batch(rows=3, figsize=(7,8))
```

Training the Model

A convolutional neural network (CNN) will be used for training the model.

CNN Input <- Images of the snacks

CNN Output -> Probability predicition of each class category

Begin with resnet34 for initial training. Resnet34 will come with pre-trained weights. There is a chance this may not work with full accuracy as indian sweets such as mithai are uncommon in traditional models. However using resent34 will help train much faster.

ImageDataBunch will be used as a validation set to prevent overfitting.

In [19]:
```
learn = cnn_learner(data, models.resnet34, metrics=error_rate)
```

Cycle through the dataset. Train with 4 epochs as a start.

In [20]:
```
learn.fit_one_cycle(4)
```

epoch	train_loss	valid_loss	error_rate	time
0	1.390411	0.363758	0.141667	00:03
1	0.867003	0.283949	0.100000	00:02
2	0.659405	0.366264	0.108333	00:02
3	0.541137	0.359390	0.108333	00:02

Interpreting the Model

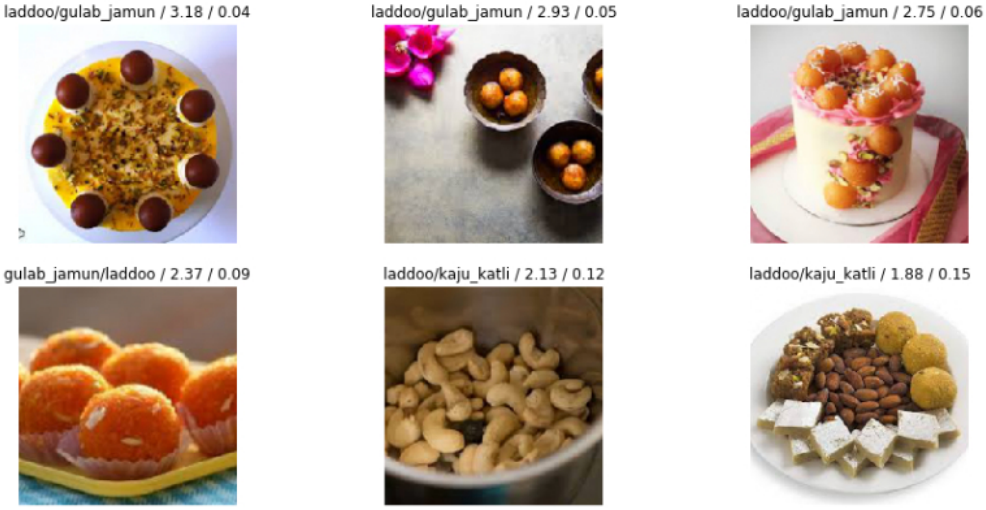
ClassificationInterpretation will help interpret the model.

plot_top_losses() helps identify where the model was most wrong. It can identify when the model knows it has a wrong answer.

In [22]:
```
interp = ClassificationInterpretation.from_learner(learn)

interp.plot_top_losses(9, figsize=(15,11))
```

Prediction/Actual/Loss/Probability



most_confused() shows which classes were most confusing to the model.

In [23]:
```
interp.most_confused(min_val=2)
```

Out[23]:
```
[('gulab_jamun', 'laddoo', 6),
 ('kaju_katli', 'laddoo', 4),
 ('laddoo', 'gulab_jamun', 3)]
```

```
Save the model.
```

In [24]:
```
learn.save('stage-1')
```

Fine-Tuning and Learning Rates

In [25]:
```
learn.unfreeze()
```

Learning rate and epochs can influence the accuracy of the model. Using a learning rate finder can help find the fastest rate at which the model can be trained before performance worsens.

In [29]:
```
learn.lr_find()
```

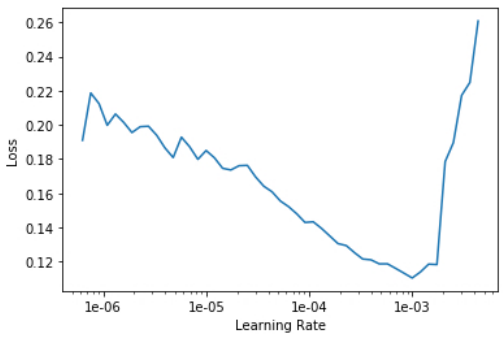
60.00% [9/15 00:21<00:14]

epoch	train_loss	valid_loss	error_rate	time
0	0.155827	#na#	00:02	
1	0.199747	#na#	00:02	
2	0.186531	#na#	00:02	
3	0.174604	#na#	00:02	
4	0.155481	#na#	00:02	
5	0.130533	#na#	00:02	
6	0.116024	#na#	00:02	
7	0.189643	#na#	00:02	
8	0.346819	#na#	00:02	

0.00% [0/7 00:00<00:00]

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

In [30]:
```
learn.recorder.plot()
```



Past 1e-03 is where loss of the model begins to ramp upwards. We can train again with the updated learning rate.

We can train the model with 2 more epochs. Using the slice() function specifies the learning rate for the first and last layers.

In [31]:
```
learn.fit_one_cycle(2, max_lr=slice(3e-5,3e-3))
```

epoch	train_loss	valid_loss	error_rate	time
0	0.160880	0.693759	0.100000	00:02
1	0.111636	0.656818	0.116667	00:02

Takeaways

The model prediction finished with a ~90% accuracy with only 6 total epochs of training and < 1,000 images. In order to take this further, we can try:

- Expanding the dataset
- Tuning the epochs
- Trying different learning rates
- Tuning batch size
- Using more layers with resnet50 instead of resnet34

In [ ]:
```
```