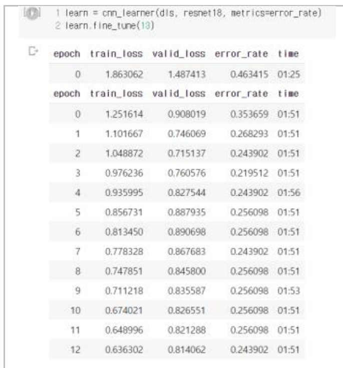
2021 People space Internship

Image Classification Assignment

Helen Eunseo Ko

0. Train my model.

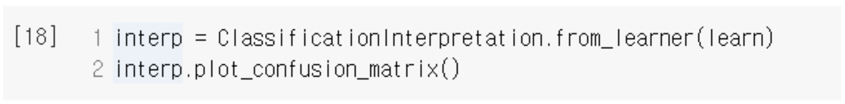


```
learn = cnn_learner(dls, resnet18, metrics=error_rate)
learn.fine_tune(13)
```

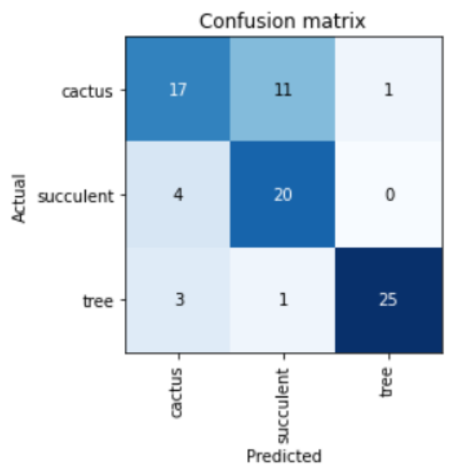
```
epoch  train_loss  valid_loss  error_rate  time
    0    1.863062    1.487413    0.463415  01:25
epoch  train_loss  valid_loss  error_rate  time
    0    1.251614    0.908019    0.353659  01:51
    1    1.101667    0.746069    0.268293  01:51
    2    1.048872    0.715137    0.243902  01:51
    3    0.976236    0.760576    0.219512  01:51
    4    0.935995    0.827544    0.243902  01:56
    5    0.856731    0.887935    0.256098  01:51
    6    0.813450    0.890698    0.256098  01:51
    7    0.778328    0.867683    0.243902  01:51
    8    0.747851    0.845800    0.256098  01:51
    9    0.711218    0.835587    0.256098  01:53
   10    0.674021    0.826551    0.256098  01:51
   11    0.648996    0.821288    0.256098  01:51
   12    0.636302    0.814062    0.243902  01:51
```

As mentioned in the lecture, I chose the CNN method and trained the data using the resnet18 model. Although we wanted to increase the number of fine_tuning to reduce the error_rate further, we could see where the value_loss went up at some point. So, it was no longer fine_tuning, but matched with fine_train (16).

1. The confusion matrix for your dataset

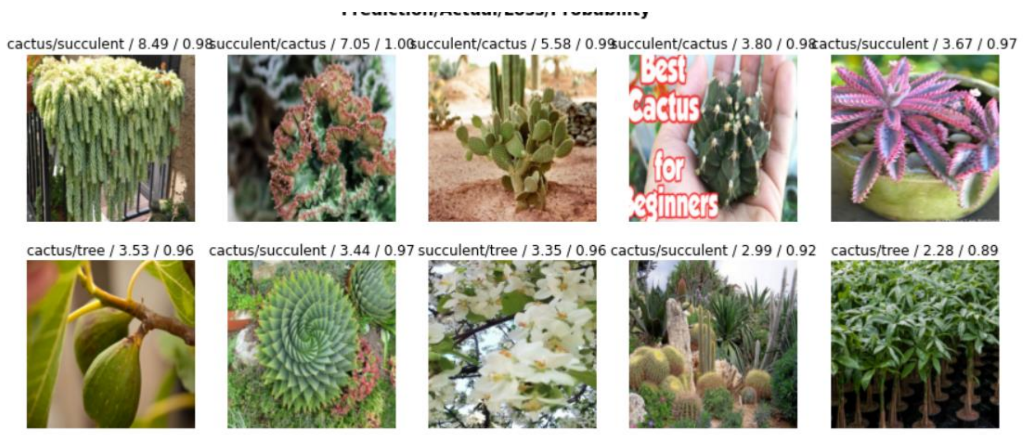


```
[18]  interp = ClassificationInterpretation.from_learner(learn)
      interp.plot_confusion_matrix()
```



The evaluation method used mainly in classification is the confusion matrix. The confusion matrix is the relationship matrix between the Active Values (GT) value and the Predicted Values of the deep learning model as shown below. Through this, we can calculate the accuracy and precision. In my case, the actual picture is a cactus, but it is often predicted as a succulent plant, so I can see that the accuracy and precision are poor in this part

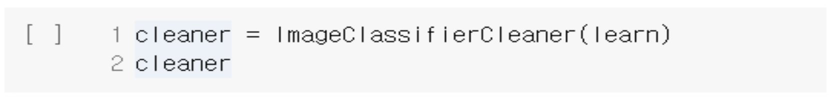
2. A screenshot of the output printed of the 10 images with the highest loss in your dataset


cactus/succulent / 8.49 / 0.98
succulent/cactus / 7.05 / 1.00
succulent/cactus / 5.58 / 0.99
succulent/cactus / 3.80 / 0.98
Best Cactus for
cactus/succulent / 3.67 / 0.97
cactus/tree / 3.53 / 0.96
cactus/succulent / 3.44 / 0.97
succulent/tree / 3.35 / 0.96
cactus/succulent / 2.99 / 0.92
cactus/tree / 2.28 / 0.89

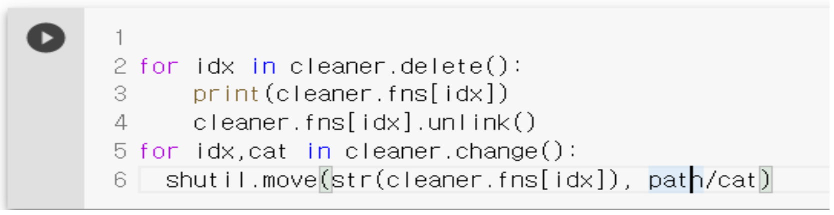
: As mentioned above, when we put together the 10 most costly datasets, we can see that they are actually cactus, but most of them are pictures of what are predicted to be succulent plants.

3. For each of the 10 images, give an explanation of why this image was included in the top loss plot, and the resulting action you took. Did you remove the image, why? And if not, Why did you include it?

- **Cleaner means below function.**

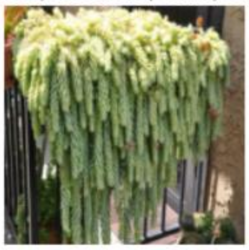
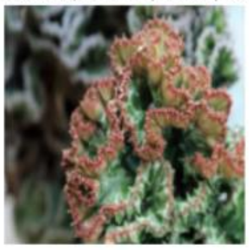
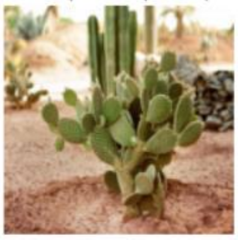
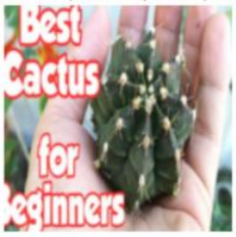
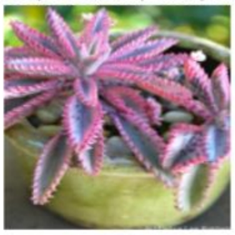


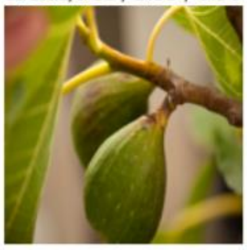
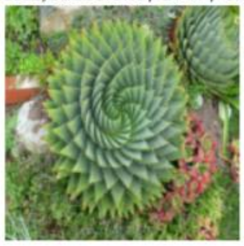
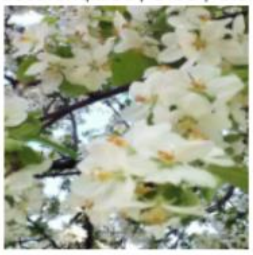
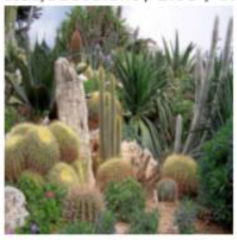
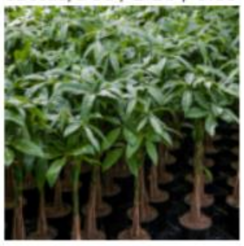
```
cleaner = ImageClassifierCleaner(learn)
cleaner
```



```

for idx in cleaner.delete():
    print(cleaner.fns[idx])
    cleaner.fns[idx].unlink()
for idx,cat in cleaner.change():
  shutil.move(str(cleaner.fns[idx]), path/cat)
```

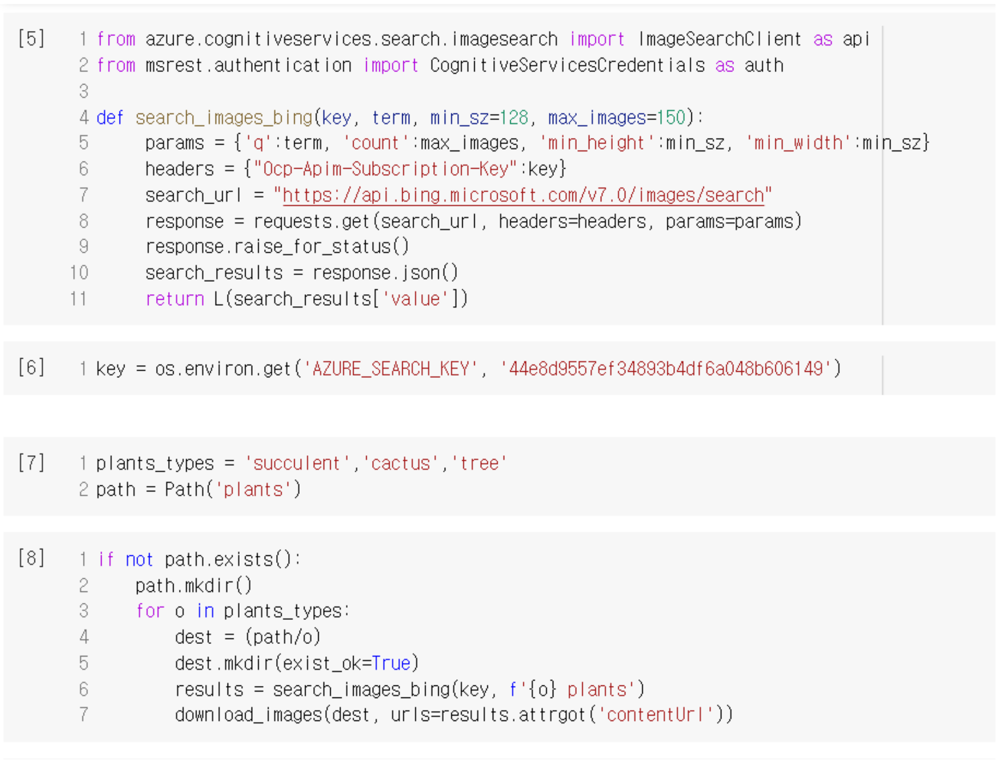
cactus/succulent / 8.49 / 0.98 	This is the most lost data. I didn't change it because I thought it was a cactus because it looked like a thorn, but it was actually a succulent plant.
succulent/cactus / 7.05 / 1.0 	Although it is actually a cactus, it seems to have been predicted to be a succulent plant due to its splendid shape. I kept the same thought in the cleaner part.
succulent/cactus / 5.58 / 0.99 	It was actually a cactus, but it was predicted to be a succulent plant. This was checked in the cleaner section, moved directly to the cactus category, and then reloaded the data.
succulent/cactus / 3.80 / 0.98 	It was thought to be a succulent plant grown at home because it was a small-sized plant, but it was actually a cactus. I kept it because I felt the same way.
:actus/succulent / 3.67 / 0.97 	A thorn appeared on the edge of the leaf, so it was thought to be a cactus, but it was actually a succulent plant. I kept it because I felt the same way.

cactus/tree / 3.53 / 0.96 	The tree is clear from the presence of fruit and leaf flies. Therefore, we checked the corresponding pictures in the cleaner section, moved them directly to the tree category, and reloaded the data.
cactus/succulent / 3.44 / 0.97 	The tip of the leaf was expected to be a cactus, but it was actually a succulent plant. I kept it because I felt the same way
succulent/tree / 3.35 / 0.96 	. This is a tree because it is a picture of flowers in full bloom. There are also several colored succulent plants, which are expected to be succulent plants. After moving the tree category directly from the cleaner part, the data was reloaded.
cactus/succulent / 2.99 / 0.92 	It is estimated that there are several cactus plants and succulent plants. This was deleted from the cleaner
cactus/tree / 2.28 / 0.89 	It is a tree based on the tree's roots and abundant leaves, but each leaf is presumed to be pointed, so it is predicted to be a cactus. After moving directly from the cleaner section to the tree category,

	the data was reloaded.

4. A summary of my approach to acquiring the data

I imported the image using the bing API mentioned in the lecture. The azure site issued related API key values, and to import more than 150 images, I referred to the article posted on reddit, but the existing method was used because all types of folders were not saved. If I have time, I want to bring many images and train them again.



```
[5]
from azure.cognitiveservices.search.imagesearch import ImageSearchClient as api
from msrest.authentication import CognitiveServicesCredentials as auth

def search_images_bing(key, term, min_sz=128, max_images=150):
     params = {'q':term, 'count':max_images, 'min_height':min_sz, 'min_width':min_sz}
     headers = {"Ocp-Apim-Subscription-Key":key}
     search_url = "https://api.bing.microsoft.com/v7.0/images/search"
     response = requests.get(search_url, headers=headers, params=params)
     response.raise_for_status()
     search_results = response.json()
     return L(search_results['value'])
```

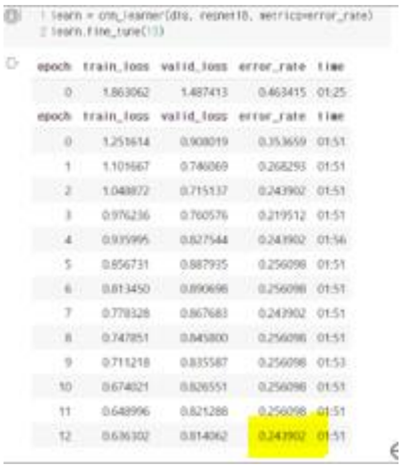
```
[6]
key = os.environ.get('AZURE_SEARCH_KEY', '44e8d9557ef34893b4df6a048b606149')
```

```
[7]
plants_types = 'succulent','cactus','tree'
path = Path('plants')
```

```
[8]
if not path.exists():
    path.mkdir()
    for o in plants_types:
        dest = (path/o)
        dest.mkdir(exist_ok=True)
        results = search_images_bing(key, f'{o} plants')
        download_images(dest, urls=results.attrgot('contentUrl'))
```

5. Why error rate did not decrease from 0.2 ? (just my opinion!)

First, there were not many basic images. Considering dividing into test sets and valid sets, we trained with less data than 150 sheets. Furthermore, it was difficult to carefully clean the data (difficult to accurately distinguish cactus from succulent plants) and finally, it was difficult to figure out which type of model best matched the dataset. It simply tried to increase the fine_tuning, but as I mentioned at first, the valid_loss rose and couldn't. For this complex reason, error_rate did not decrease from 0.2.



```
learn = cnn_learner(dls, resnet18, metrics=error_rate)
learn.fine_tune(13)
```

```
epoch  train_loss  valid_loss  error_rate  time
    0    1.863062    1.487413    0.463415  01:25
epoch  train_loss  valid_loss  error_rate  time
    0    1.251614    0.908019    0.353659  01:51
    1    1.101667    0.746069    0.268293  01:51
    2    1.048872    0.715137    0.243902  01:51
    3    0.976236    0.760576    0.219512  01:51
    4    0.935995    0.827544    0.243902  01:56
    5    0.856731    0.887935    0.256098  01:51
    6    0.813450    0.890696    0.256098  01:51
    7    0.778328    0.867683    0.243902  01:51
    8    0.747851    0.845800    0.256098  01:51
    9    0.711218    0.835587    0.256098  01:53
   10    0.674021    0.826551    0.256098  01:51
   11    0.648996    0.821288    0.256098  01:51
   12    0.636302    0.814062    0.243902  01:51
```

Thank you for reading 😊