part2

March 20, 2024

1 EECS 442 Homework 4 - PyTorch ConvNets

In this notebook we will explore how to use a pre-trained PyTorch convolution neural network (ConvNet).

Before we start, please put your name and UMID in following format Firstname LASTNAME, #00000000 // e.g.) David FOUHEY, #12345678

Your Answer:
Wensong HU #24908654

1.1 Setup

[1]:
```
import os
import json
import torch
import torchvision
import torchvision.transforms as T
import random
import numpy as np
from scipy.ndimage.filters import gaussian_filter1d
import matplotlib.pyplot as plt
SQUEEZENET_MEAN = torch.tensor([0.485, 0.456, 0.406], dtype=torch.float)
SQUEEZENET_STD = torch.tensor([0.229, 0.224, 0.225], dtype=torch.float)
from PIL import Image

%matplotlib inline
plt.rcParams['figure.figsize'] = (10.0, 8.0) # set default size of plots
plt.rcParams['image.interpolation'] = 'nearest'
plt.rcParams['image.cmap'] = 'gray'
```

```
/tmp/ipykernel_1781330/770024508.py:8: DeprecationWarning: Please import
`gaussian_filter1d` from the `scipy.ndimage` namespace; the
`scipy.ndimage.filters` namespace is deprecated and will be removed in SciPy
2.0.0.
  from scipy.ndimage.filters import gaussian_filter1d
```

[2]:
```
if torch.cuda.is_available():
    print("Using the GPU. You are good to go!")
```

```
    device = 'cuda'
else:
    print("Using the CPU. Overall speed may be slowed down")
    device = 'cpu'
```

```
Using the GPU. You are good to go!
```

For all of our experiments, we will start with a convolutional neural network which was pretrained to perform image classification on ImageNet [1]. We can use any model here, but for the purposes of this assignment we will use SqueezeNet [2], which achieves accuracies comparable to AlexNet but with a significantly reduced parameter count and computational complexity.

Using SqueezeNet rather than AlexNet or VGG or ResNet means that we can easily perform all experiments without heavy computation. Run the following cell to download and initialize your model.

[1] Olga Russakovsky*, Jia Deng*, Hao Su, Jonathan Krause, Sanjeev Satheesh, Sean Ma, Zhiheng Huang, Andrej Karpathy, Aditya Khosla, Michael Bernstein, Alexander C. Berg and Li Fei-Fei. ImageNet Large Scale Visual Recognition Challenge. IJCV, 2015

[2] Iandola et al, "SqueezeNet: AlexNet-level accuracy with 50x fewer parameters and < 0.5MB model size", arXiv 2016

[3]:
```
print('Download and load the pretrained SqueezeNet model.')
model = torchvision.models.squeezenet1_1(pretrained=True).to(device)

# We don't want to train the model, so tell PyTorch not to compute gradients
# with respect to model parameters.
for param in model.parameters():
    param.requires_grad = False

# Make sure the model is in "eval" mode
model.eval()

# you may see warning regarding initialization deprecated, that's fine,
# please continue to next steps
```

```
Download and load the pretrained SqueezeNet model.
```

```
/home/umhws/anaconda3/envs/eecs442/lib/python3.10/site-
packages/torchvision/models/_utils.py:208: UserWarning: The parameter
'pretrained' is deprecated since 0.13 and may be removed in the future, please
use 'weights' instead.
  warnings.warn(
/home/umhws/anaconda3/envs/eecs442/lib/python3.10/site-
packages/torchvision/models/_utils.py:223: UserWarning: Arguments other than a
weight enum or `None` for 'weights' are deprecated since 0.13 and may be removed
in the future. The current behavior is equivalent to passing
`weights=SqueezeNet1_1_Weights.IMAGENET1K_V1`. You can also use
`weights=SqueezeNet1_1_Weights.DEFAULT` to get the most up-to-date weights.
  warnings.warn(msg)
```

```
[3]: SqueezeNet(
       (features): Sequential(
         (0): Conv2d(3, 64, kernel_size=(3, 3), stride=(2, 2))
         (1): ReLU(inplace=True)
         (2): MaxPool2d(kernel_size=3, stride=2, padding=0, dilation=1,
     ceil_mode=True)
         (3): Fire(
           (squeeze): Conv2d(64, 16, kernel_size=(1, 1), stride=(1, 1))
           (squeeze_activation): ReLU(inplace=True)
           (expand1x1): Conv2d(16, 64, kernel_size=(1, 1), stride=(1, 1))
           (expand1x1_activation): ReLU(inplace=True)
           (expand3x3): Conv2d(16, 64, kernel_size=(3, 3), stride=(1, 1), padding=(1,
     1))
           (expand3x3_activation): ReLU(inplace=True)
         )
         (4): Fire(
           (squeeze): Conv2d(128, 16, kernel_size=(1, 1), stride=(1, 1))
           (squeeze_activation): ReLU(inplace=True)
           (expand1x1): Conv2d(16, 64, kernel_size=(1, 1), stride=(1, 1))
           (expand1x1_activation): ReLU(inplace=True)
           (expand3x3): Conv2d(16, 64, kernel_size=(3, 3), stride=(1, 1), padding=(1,
     1))
           (expand3x3_activation): ReLU(inplace=True)
         )
         (5): MaxPool2d(kernel_size=3, stride=2, padding=0, dilation=1,
     ceil_mode=True)
         (6): Fire(
           (squeeze): Conv2d(128, 32, kernel_size=(1, 1), stride=(1, 1))
           (squeeze_activation): ReLU(inplace=True)
           (expand1x1): Conv2d(32, 128, kernel_size=(1, 1), stride=(1, 1))
           (expand1x1_activation): ReLU(inplace=True)
           (expand3x3): Conv2d(32, 128, kernel_size=(3, 3), stride=(1, 1),
     padding=(1, 1))
           (expand3x3_activation): ReLU(inplace=True)
         )
         (7): Fire(
           (squeeze): Conv2d(256, 32, kernel_size=(1, 1), stride=(1, 1))
           (squeeze_activation): ReLU(inplace=True)
           (expand1x1): Conv2d(32, 128, kernel_size=(1, 1), stride=(1, 1))
           (expand1x1_activation): ReLU(inplace=True)
           (expand3x3): Conv2d(32, 128, kernel_size=(3, 3), stride=(1, 1),
     padding=(1, 1))
           (expand3x3_activation): ReLU(inplace=True)
         )
         (8): MaxPool2d(kernel_size=3, stride=2, padding=0, dilation=1,
     ceil_mode=True)
         (9): Fire(
```

```
      (squeeze): Conv2d(256, 48, kernel_size=(1, 1), stride=(1, 1))
      (squeeze_activation): ReLU(inplace=True)
      (expand1x1): Conv2d(48, 192, kernel_size=(1, 1), stride=(1, 1))
      (expand1x1_activation): ReLU(inplace=True)
      (expand3x3): Conv2d(48, 192, kernel_size=(3, 3), stride=(1, 1),
padding=(1, 1))
      (expand3x3_activation): ReLU(inplace=True)
    )
    (10): Fire(
      (squeeze): Conv2d(384, 48, kernel_size=(1, 1), stride=(1, 1))
      (squeeze_activation): ReLU(inplace=True)
      (expand1x1): Conv2d(48, 192, kernel_size=(1, 1), stride=(1, 1))
      (expand1x1_activation): ReLU(inplace=True)
      (expand3x3): Conv2d(48, 192, kernel_size=(3, 3), stride=(1, 1),
padding=(1, 1))
      (expand3x3_activation): ReLU(inplace=True)
    )
    (11): Fire(
      (squeeze): Conv2d(384, 64, kernel_size=(1, 1), stride=(1, 1))
      (squeeze_activation): ReLU(inplace=True)
      (expand1x1): Conv2d(64, 256, kernel_size=(1, 1), stride=(1, 1))
      (expand1x1_activation): ReLU(inplace=True)
      (expand3x3): Conv2d(64, 256, kernel_size=(3, 3), stride=(1, 1),
padding=(1, 1))
      (expand3x3_activation): ReLU(inplace=True)
    )
    (12): Fire(
      (squeeze): Conv2d(512, 64, kernel_size=(1, 1), stride=(1, 1))
      (squeeze_activation): ReLU(inplace=True)
      (expand1x1): Conv2d(64, 256, kernel_size=(1, 1), stride=(1, 1))
      (expand1x1_activation): ReLU(inplace=True)
      (expand3x3): Conv2d(64, 256, kernel_size=(3, 3), stride=(1, 1),
padding=(1, 1))
      (expand3x3_activation): ReLU(inplace=True)
    )
  )
  (classifier): Sequential(
    (0): Dropout(p=0.5, inplace=False)
    (1): Conv2d(512, 1000, kernel_size=(1, 1), stride=(1, 1))
    (2): ReLU(inplace=True)
    (3): AdaptiveAvgPool2d(output_size=(1, 1))
  )
)
```

Next, we will download the imagenet labels which we are going to use in the notebook. ImageNet labels are stored in the `idx2label` dictionary of `{index(int): label(str)}`.

[4]:
```
# Loading the imagenet class labels
# If this cell failed due to wget probelm, you can put the link below into your␣
 ↪browser to download the file directly
# Put the downloaded file under the same directory as this jupyter notebook
!wget https://s3.amazonaws.com/deep-learning-models/image-models/
 ↪imagenet_class_index.json

class_idx = json.load(open("imagenet_class_index.json"))
idx2label = {k:class_idx[str(k)][1] for k in range(len(class_idx))}
idx2label
```

```
--2024-03-20 13:42:07--  https://s3.amazonaws.com/deep-learning-models/image-
models/imagenet_class_index.json
Resolving s3.amazonaws.com (s3.amazonaws.com)… 52.217.200.40, 54.231.172.152,
52.216.113.205, …
Connecting to s3.amazonaws.com (s3.amazonaws.com)|52.217.200.40|:443…
connected.
HTTP request sent, awaiting response… 200 OK
Length: 35363 (35K) [application/octet-stream]
Saving to: ‘imagenet_class_index.json.1’

imagenet_class_inde 100%[===================>]  34.53K  --.-KB/s    in 0.03s

2024-03-20 13:42:07 (1.29 MB/s) - ‘imagenet_class_index.json.1’ saved
[35363/35363]
```

[4]:
```
{0: 'tench',
 1: 'goldfish',
 2: 'great_white_shark',
 3: 'tiger_shark',
 4: 'hammerhead',
 5: 'electric_ray',
 6: 'stingray',
 7: 'cock',
 8: 'hen',
 9: 'ostrich',
 10: 'brambling',
 11: 'goldfinch',
 12: 'house_finch',
 13: 'junco',
 14: 'indigo_bunting',
 15: 'robin',
 16: 'bulbul',
 17: 'jay',
 18: 'magpie',
 19: 'chickadee',
```

```
20: 'water_ouzel',
21: 'kite',
22: 'bald_eagle',
23: 'vulture',
24: 'great_grey_owl',
25: 'European_fire_salamander',
26: 'common_newt',
27: 'eft',
28: 'spotted_salamander',
29: 'axolotl',
30: 'bullfrog',
31: 'tree_frog',
32: 'tailed_frog',
33: 'loggerhead',
34: 'leatherback_turtle',
35: 'mud_turtle',
36: 'terrapin',
37: 'box_turtle',
38: 'banded_gecko',
39: 'common_iguana',
40: 'American_chameleon',
41: 'whiptail',
42: 'agama',
43: 'frilled_lizard',
44: 'alligator_lizard',
45: 'Gila_monster',
46: 'green_lizard',
47: 'African_chameleon',
48: 'Komodo_dragon',
49: 'African_crocodile',
50: 'American_alligator',
51: 'triceratops',
52: 'thunder_snake',
53: 'ringneck_snake',
54: 'hognose_snake',
55: 'green_snake',
56: 'king_snake',
57: 'garter_snake',
58: 'water_snake',
59: 'vine_snake',
60: 'night_snake',
61: 'boa_constrictor',
62: 'rock_python',
63: 'Indian_cobra',
64: 'green_mamba',
65: 'sea_snake',
66: 'horned_viper',
```

```
67: 'diamondback',
68: 'sidewinder',
69: 'trilobite',
70: 'harvestman',
71: 'scorpion',
72: 'black_and_gold_garden_spider',
73: 'barn_spider',
74: 'garden_spider',
75: 'black_widow',
76: 'tarantula',
77: 'wolf_spider',
78: 'tick',
79: 'centipede',
80: 'black_grouse',
81: 'ptarmigan',
82: 'ruffed_grouse',
83: 'prairie_chicken',
84: 'peacock',
85: 'quail',
86: 'partridge',
87: 'African_grey',
88: 'macaw',
89: 'sulphur-crested_cockatoo',
90: 'lorikeet',
91: 'coucal',
92: 'bee_eater',
93: 'hornbill',
94: 'hummingbird',
95: 'jacamar',
96: 'toucan',
97: 'drake',
98: 'red-breasted_merganser',
99: 'goose',
100: 'black_swan',
101: 'tusker',
102: 'echidna',
103: 'platypus',
104: 'wallaby',
105: 'koala',
106: 'wombat',
107: 'jellyfish',
108: 'sea_anemone',
109: 'brain_coral',
110: 'flatworm',
111: 'nematode',
112: 'conch',
113: 'snail',
```

```
114: 'slug',
115: 'sea_slug',
116: 'chiton',
117: 'chambered_nautilus',
118: 'Dungeness_crab',
119: 'rock_crab',
120: 'fiddler_crab',
121: 'king_crab',
122: 'American_lobster',
123: 'spiny_lobster',
124: 'crayfish',
125: 'hermit_crab',
126: 'isopod',
127: 'white_stork',
128: 'black_stork',
129: 'spoonbill',
130: 'flamingo',
131: 'little_blue_heron',
132: 'American_egret',
133: 'bittern',
134: 'crane',
135: 'limpkin',
136: 'European_gallinule',
137: 'American_coot',
138: 'bustard',
139: 'ruddy_turnstone',
140: 'red-backed_sandpiper',
141: 'redshank',
142: 'dowitcher',
143: 'oystercatcher',
144: 'pelican',
145: 'king_penguin',
146: 'albatross',
147: 'grey_whale',
148: 'killer_whale',
149: 'dugong',
150: 'sea_lion',
151: 'Chihuahua',
152: 'Japanese_spaniel',
153: 'Maltese_dog',
154: 'Pekinese',
155: 'Shih-Tzu',
156: 'Blenheim_spaniel',
157: 'papillon',
158: 'toy_terrier',
159: 'Rhodesian_ridgeback',
160: 'Afghan_hound',
```

```
161: 'basset',
162: 'beagle',
163: 'bloodhound',
164: 'bluetick',
165: 'black-and-tan_coonhound',
166: 'Walker_hound',
167: 'English_foxhound',
168: 'redbone',
169: 'borzoi',
170: 'Irish_wolfhound',
171: 'Italian_greyhound',
172: 'whippet',
173: 'Ibizan_hound',
174: 'Norwegian_elkhound',
175: 'otterhound',
176: 'Saluki',
177: 'Scottish_deerhound',
178: 'Weimaraner',
179: 'Staffordshire_bullterrier',
180: 'American_Staffordshire_terrier',
181: 'Bedlington_terrier',
182: 'Border_terrier',
183: 'Kerry_blue_terrier',
184: 'Irish_terrier',
185: 'Norfolk_terrier',
186: 'Norwich_terrier',
187: 'Yorkshire_terrier',
188: 'wire-haired_fox_terrier',
189: 'Lakeland_terrier',
190: 'Sealyham_terrier',
191: 'Airedale',
192: 'cairn',
193: 'Australian_terrier',
194: 'Dandie_Dinmont',
195: 'Boston_bull',
196: 'miniature_schnauzer',
197: 'giant_schnauzer',
198: 'standard_schnauzer',
199: 'Scotch_terrier',
200: 'Tibetan_terrier',
201: 'silky_terrier',
202: 'soft-coated_wheaten_terrier',
203: 'West_Highland_white_terrier',
204: 'Lhasa',
205: 'flat-coated_retriever',
206: 'curly-coated_retriever',
207: 'golden_retriever',
```

```
208: 'Labrador_retriever',
209: 'Chesapeake_Bay_retriever',
210: 'German_short-haired_pointer',
211: 'vizsla',
212: 'English_setter',
213: 'Irish_setter',
214: 'Gordon_setter',
215: 'Brittany_spaniel',
216: 'clumber',
217: 'English_springer',
218: 'Welsh_springer_spaniel',
219: 'cocker_spaniel',
220: 'Sussex_spaniel',
221: 'Irish_water_spaniel',
222: 'kuvasz',
223: 'schipperke',
224: 'groenendael',
225: 'malinois',
226: 'briard',
227: 'kelpie',
228: 'komondor',
229: 'Old_English_sheepdog',
230: 'Shetland_sheepdog',
231: 'collie',
232: 'Border_collie',
233: 'Bouvier_des_Flandres',
234: 'Rottweiler',
235: 'German_shepherd',
236: 'Doberman',
237: 'miniature_pinscher',
238: 'Greater_Swiss_Mountain_dog',
239: 'Bernese_mountain_dog',
240: 'Appenzeller',
241: 'EntleBucher',
242: 'boxer',
243: 'bull_mastiff',
244: 'Tibetan_mastiff',
245: 'French_bulldog',
246: 'Great_Dane',
247: 'Saint_Bernard',
248: 'Eskimo_dog',
249: 'malamute',
250: 'Siberian_husky',
251: 'dalmatian',
252: 'affenpinscher',
253: 'basenji',
254: 'pug',
```

```
255: 'Leonberg',
256: 'Newfoundland',
257: 'Great_Pyrenees',
258: 'Samoyed',
259: 'Pomeranian',
260: 'chow',
261: 'keeshond',
262: 'Brabancon_griffon',
263: 'Pembroke',
264: 'Cardigan',
265: 'toy_poodle',
266: 'miniature_poodle',
267: 'standard_poodle',
268: 'Mexican_hairless',
269: 'timber_wolf',
270: 'white_wolf',
271: 'red_wolf',
272: 'coyote',
273: 'dingo',
274: 'dhole',
275: 'African_hunting_dog',
276: 'hyena',
277: 'red_fox',
278: 'kit_fox',
279: 'Arctic_fox',
280: 'grey_fox',
281: 'tabby',
282: 'tiger_cat',
283: 'Persian_cat',
284: 'Siamese_cat',
285: 'Egyptian_cat',
286: 'cougar',
287: 'lynx',
288: 'leopard',
289: 'snow_leopard',
290: 'jaguar',
291: 'lion',
292: 'tiger',
293: 'cheetah',
294: 'brown_bear',
295: 'American_black_bear',
296: 'ice_bear',
297: 'sloth_bear',
298: 'mongoose',
299: 'meerkat',
300: 'tiger_beetle',
301: 'ladybug',
```

```
302: 'ground_beetle',
303: 'long-horned_beetle',
304: 'leaf_beetle',
305: 'dung_beetle',
306: 'rhinoceros_beetle',
307: 'weevil',
308: 'fly',
309: 'bee',
310: 'ant',
311: 'grasshopper',
312: 'cricket',
313: 'walking_stick',
314: 'cockroach',
315: 'mantis',
316: 'cicada',
317: 'leafhopper',
318: 'lacewing',
319: 'dragonfly',
320: 'damselfly',
321: 'admiral',
322: 'ringlet',
323: 'monarch',
324: 'cabbage_butterfly',
325: 'sulphur_butterfly',
326: 'lycaenid',
327: 'starfish',
328: 'sea_urchin',
329: 'sea_cucumber',
330: 'wood_rabbit',
331: 'hare',
332: 'Angora',
333: 'hamster',
334: 'porcupine',
335: 'fox_squirrel',
336: 'marmot',
337: 'beaver',
338: 'guinea_pig',
339: 'sorrel',
340: 'zebra',
341: 'hog',
342: 'wild_boar',
343: 'warthog',
344: 'hippopotamus',
345: 'ox',
346: 'water_buffalo',
347: 'bison',
348: 'ram',
```

```
349: 'bighorn',
350: 'ibex',
351: 'hartebeest',
352: 'impala',
353: 'gazelle',
354: 'Arabian_camel',
355: 'llama',
356: 'weasel',
357: 'mink',
358: 'polecat',
359: 'black-footed_ferret',
360: 'otter',
361: 'skunk',
362: 'badger',
363: 'armadillo',
364: 'three-toed_sloth',
365: 'orangutan',
366: 'gorilla',
367: 'chimpanzee',
368: 'gibbon',
369: 'siamang',
370: 'guenon',
371: 'patas',
372: 'baboon',
373: 'macaque',
374: 'langur',
375: 'colobus',
376: 'proboscis_monkey',
377: 'marmoset',
378: 'capuchin',
379: 'howler_monkey',
380: 'titi',
381: 'spider_monkey',
382: 'squirrel_monkey',
383: 'Madagascar_cat',
384: 'indri',
385: 'Indian_elephant',
386: 'African_elephant',
387: 'lesser_panda',
388: 'giant_panda',
389: 'barracouta',
390: 'eel',
391: 'coho',
392: 'rock_beauty',
393: 'anemone_fish',
394: 'sturgeon',
395: 'gar',
```

```
396: 'lionfish',
397: 'puffer',
398: 'abacus',
399: 'abaya',
400: 'academic_gown',
401: 'accordion',
402: 'acoustic_guitar',
403: 'aircraft_carrier',
404: 'airliner',
405: 'airship',
406: 'altar',
407: 'ambulance',
408: 'amphibian',
409: 'analog_clock',
410: 'apiary',
411: 'apron',
412: 'ashcan',
413: 'assault_rifle',
414: 'backpack',
415: 'bakery',
416: 'balance_beam',
417: 'balloon',
418: 'ballpoint',
419: 'Band_Aid',
420: 'banjo',
421: 'bannister',
422: 'barbell',
423: 'barber_chair',
424: 'barbershop',
425: 'barn',
426: 'barometer',
427: 'barrel',
428: 'barrow',
429: 'baseball',
430: 'basketball',
431: 'bassinet',
432: 'bassoon',
433: 'bathing_cap',
434: 'bath_towel',
435: 'bathtub',
436: 'beach_wagon',
437: 'beacon',
438: 'beaker',
439: 'bearskin',
440: 'beer_bottle',
441: 'beer_glass',
442: 'bell_cote',
```

```
443: 'bib',
444: 'bicycle-built-for-two',
445: 'bikini',
446: 'binder',
447: 'binoculars',
448: 'birdhouse',
449: 'boathouse',
450: 'bobsled',
451: 'bolo_tie',
452: 'bonnet',
453: 'bookcase',
454: 'bookshop',
455: 'bottlecap',
456: 'bow',
457: 'bow_tie',
458: 'brass',
459: 'brassiere',
460: 'breakwater',
461: 'breastplate',
462: 'broom',
463: 'bucket',
464: 'buckle',
465: 'bulletproof_vest',
466: 'bullet_train',
467: 'butcher_shop',
468: 'cab',
469: 'caldron',
470: 'candle',
471: 'cannon',
472: 'canoe',
473: 'can_opener',
474: 'cardigan',
475: 'car_mirror',
476: 'carousel',
477: "carpenter's_kit",
478: 'carton',
479: 'car_wheel',
480: 'cash_machine',
481: 'cassette',
482: 'cassette_player',
483: 'castle',
484: 'catamaran',
485: 'CD_player',
486: 'cello',
487: 'cellular_telephone',
488: 'chain',
489: 'chainlink_fence',
```

```
490: 'chain_mail',
491: 'chain_saw',
492: 'chest',
493: 'chiffonier',
494: 'chime',
495: 'china_cabinet',
496: 'Christmas_stocking',
497: 'church',
498: 'cinema',
499: 'cleaver',
500: 'cliff_dwelling',
501: 'cloak',
502: 'clog',
503: 'cocktail_shaker',
504: 'coffee_mug',
505: 'coffeepot',
506: 'coil',
507: 'combination_lock',
508: 'computer_keyboard',
509: 'confectionery',
510: 'container_ship',
511: 'convertible',
512: 'corkscrew',
513: 'cornet',
514: 'cowboy_boot',
515: 'cowboy_hat',
516: 'cradle',
517: 'crane',
518: 'crash_helmet',
519: 'crate',
520: 'crib',
521: 'Crock_Pot',
522: 'croquet_ball',
523: 'crutch',
524: 'cuirass',
525: 'dam',
526: 'desk',
527: 'desktop_computer',
528: 'dial_telephone',
529: 'diaper',
530: 'digital_clock',
531: 'digital_watch',
532: 'dining_table',
533: 'dishrag',
534: 'dishwasher',
535: 'disk_brake',
536: 'dock',
```

```
537: 'dogsled',
538: 'dome',
539: 'doormat',
540: 'drilling_platform',
541: 'drum',
542: 'drumstick',
543: 'dumbbell',
544: 'Dutch_oven',
545: 'electric_fan',
546: 'electric_guitar',
547: 'electric_locomotive',
548: 'entertainment_center',
549: 'envelope',
550: 'espresso_maker',
551: 'face_powder',
552: 'feather_boa',
553: 'file',
554: 'fireboat',
555: 'fire_engine',
556: 'fire_screen',
557: 'flagpole',
558: 'flute',
559: 'folding_chair',
560: 'football_helmet',
561: 'forklift',
562: 'fountain',
563: 'fountain_pen',
564: 'four-poster',
565: 'freight_car',
566: 'French_horn',
567: 'frying_pan',
568: 'fur_coat',
569: 'garbage_truck',
570: 'gasmask',
571: 'gas_pump',
572: 'goblet',
573: 'go-kart',
574: 'golf_ball',
575: 'golfcart',
576: 'gondola',
577: 'gong',
578: 'gown',
579: 'grand_piano',
580: 'greenhouse',
581: 'grille',
582: 'grocery_store',
583: 'guillotine',
```

```
584: 'hair_slide',
585: 'hair_spray',
586: 'half_track',
587: 'hammer',
588: 'hamper',
589: 'hand_blower',
590: 'hand-held_computer',
591: 'handkerchief',
592: 'hard_disc',
593: 'harmonica',
594: 'harp',
595: 'harvester',
596: 'hatchet',
597: 'holster',
598: 'home_theater',
599: 'honeycomb',
600: 'hook',
601: 'hoopskirt',
602: 'horizontal_bar',
603: 'horse_cart',
604: 'hourglass',
605: 'iPod',
606: 'iron',
607: "jack-o'-lantern",
608: 'jean',
609: 'jeep',
610: 'jersey',
611: 'jigsaw_puzzle',
612: 'jinrikisha',
613: 'joystick',
614: 'kimono',
615: 'knee_pad',
616: 'knot',
617: 'lab_coat',
618: 'ladle',
619: 'lampshade',
620: 'laptop',
621: 'lawn_mower',
622: 'lens_cap',
623: 'letter_opener',
624: 'library',
625: 'lifeboat',
626: 'lighter',
627: 'limousine',
628: 'liner',
629: 'lipstick',
630: 'Loafer',
```

```
631: 'lotion',
632: 'loudspeaker',
633: 'loupe',
634: 'lumbermill',
635: 'magnetic_compass',
636: 'mailbag',
637: 'mailbox',
638: 'maillot',
639: 'maillot',
640: 'manhole_cover',
641: 'maraca',
642: 'marimba',
643: 'mask',
644: 'matchstick',
645: 'maypole',
646: 'maze',
647: 'measuring_cup',
648: 'medicine_chest',
649: 'megalith',
650: 'microphone',
651: 'microwave',
652: 'military_uniform',
653: 'milk_can',
654: 'minibus',
655: 'miniskirt',
656: 'minivan',
657: 'missile',
658: 'mitten',
659: 'mixing_bowl',
660: 'mobile_home',
661: 'Model_T',
662: 'modem',
663: 'monastery',
664: 'monitor',
665: 'moped',
666: 'mortar',
667: 'mortarboard',
668: 'mosque',
669: 'mosquito_net',
670: 'motor_scooter',
671: 'mountain_bike',
672: 'mountain_tent',
673: 'mouse',
674: 'mousetrap',
675: 'moving_van',
676: 'muzzle',
677: 'nail',
```

```
678: 'neck_brace',
679: 'necklace',
680: 'nipple',
681: 'notebook',
682: 'obelisk',
683: 'oboe',
684: 'ocarina',
685: 'odometer',
686: 'oil_filter',
687: 'organ',
688: 'oscilloscope',
689: 'overskirt',
690: 'oxcart',
691: 'oxygen_mask',
692: 'packet',
693: 'paddle',
694: 'paddlewheel',
695: 'padlock',
696: 'paintbrush',
697: 'pajama',
698: 'palace',
699: 'panpipe',
700: 'paper_towel',
701: 'parachute',
702: 'parallel_bars',
703: 'park_bench',
704: 'parking_meter',
705: 'passenger_car',
706: 'patio',
707: 'pay-phone',
708: 'pedestal',
709: 'pencil_box',
710: 'pencil_sharpener',
711: 'perfume',
712: 'Petri_dish',
713: 'photocopier',
714: 'pick',
715: 'pickelhaube',
716: 'picket_fence',
717: 'pickup',
718: 'pier',
719: 'piggy_bank',
720: 'pill_bottle',
721: 'pillow',
722: 'ping-pong_ball',
723: 'pinwheel',
724: 'pirate',
```

```
725: 'pitcher',
726: 'plane',
727: 'planetarium',
728: 'plastic_bag',
729: 'plate_rack',
730: 'plow',
731: 'plunger',
732: 'Polaroid_camera',
733: 'pole',
734: 'police_van',
735: 'poncho',
736: 'pool_table',
737: 'pop_bottle',
738: 'pot',
739: "potter's_wheel",
740: 'power_drill',
741: 'prayer_rug',
742: 'printer',
743: 'prison',
744: 'projectile',
745: 'projector',
746: 'puck',
747: 'punching_bag',
748: 'purse',
749: 'quill',
750: 'quilt',
751: 'racer',
752: 'racket',
753: 'radiator',
754: 'radio',
755: 'radio_telescope',
756: 'rain_barrel',
757: 'recreational_vehicle',
758: 'reel',
759: 'reflex_camera',
760: 'refrigerator',
761: 'remote_control',
762: 'restaurant',
763: 'revolver',
764: 'rifle',
765: 'rocking_chair',
766: 'rotisserie',
767: 'rubber_eraser',
768: 'rugby_ball',
769: 'rule',
770: 'running_shoe',
771: 'safe',
```

```
772: 'safety_pin',
773: 'saltshaker',
774: 'sandal',
775: 'sarong',
776: 'sax',
777: 'scabbard',
778: 'scale',
779: 'school_bus',
780: 'schooner',
781: 'scoreboard',
782: 'screen',
783: 'screw',
784: 'screwdriver',
785: 'seat_belt',
786: 'sewing_machine',
787: 'shield',
788: 'shoe_shop',
789: 'shoji',
790: 'shopping_basket',
791: 'shopping_cart',
792: 'shovel',
793: 'shower_cap',
794: 'shower_curtain',
795: 'ski',
796: 'ski_mask',
797: 'sleeping_bag',
798: 'slide_rule',
799: 'sliding_door',
800: 'slot',
801: 'snorkel',
802: 'snowmobile',
803: 'snowplow',
804: 'soap_dispenser',
805: 'soccer_ball',
806: 'sock',
807: 'solar_dish',
808: 'sombrero',
809: 'soup_bowl',
810: 'space_bar',
811: 'space_heater',
812: 'space_shuttle',
813: 'spatula',
814: 'speedboat',
815: 'spider_web',
816: 'spindle',
817: 'sports_car',
818: 'spotlight',
```

```
819: 'stage',
820: 'steam_locomotive',
821: 'steel_arch_bridge',
822: 'steel_drum',
823: 'stethoscope',
824: 'stole',
825: 'stone_wall',
826: 'stopwatch',
827: 'stove',
828: 'strainer',
829: 'streetcar',
830: 'stretcher',
831: 'studio_couch',
832: 'stupa',
833: 'submarine',
834: 'suit',
835: 'sundial',
836: 'sunglass',
837: 'sunglasses',
838: 'sunscreen',
839: 'suspension_bridge',
840: 'swab',
841: 'sweatshirt',
842: 'swimming_trunks',
843: 'swing',
844: 'switch',
845: 'syringe',
846: 'table_lamp',
847: 'tank',
848: 'tape_player',
849: 'teapot',
850: 'teddy',
851: 'television',
852: 'tennis_ball',
853: 'thatch',
854: 'theater_curtain',
855: 'thimble',
856: 'thresher',
857: 'throne',
858: 'tile_roof',
859: 'toaster',
860: 'tobacco_shop',
861: 'toilet_seat',
862: 'torch',
863: 'totem_pole',
864: 'tow_truck',
865: 'toyshop',
```

```
866: 'tractor',
867: 'trailer_truck',
868: 'tray',
869: 'trench_coat',
870: 'tricycle',
871: 'trimaran',
872: 'tripod',
873: 'triumphal_arch',
874: 'trolleybus',
875: 'trombone',
876: 'tub',
877: 'turnstile',
878: 'typewriter_keyboard',
879: 'umbrella',
880: 'unicycle',
881: 'upright',
882: 'vacuum',
883: 'vase',
884: 'vault',
885: 'velvet',
886: 'vending_machine',
887: 'vestment',
888: 'viaduct',
889: 'violin',
890: 'volleyball',
891: 'waffle_iron',
892: 'wall_clock',
893: 'wallet',
894: 'wardrobe',
895: 'warplane',
896: 'washbasin',
897: 'washer',
898: 'water_bottle',
899: 'water_jug',
900: 'water_tower',
901: 'whiskey_jug',
902: 'whistle',
903: 'wig',
904: 'window_screen',
905: 'window_shade',
906: 'Windsor_tie',
907: 'wine_bottle',
908: 'wing',
909: 'wok',
910: 'wooden_spoon',
911: 'wool',
912: 'worm_fence',
```

```
913: 'wreck',
914: 'yawl',
915: 'yurt',
916: 'web_site',
917: 'comic_book',
918: 'crossword_puzzle',
919: 'street_sign',
920: 'traffic_light',
921: 'book_jacket',
922: 'menu',
923: 'plate',
924: 'guacamole',
925: 'consomme',
926: 'hot_pot',
927: 'trifle',
928: 'ice_cream',
929: 'ice_lolly',
930: 'French_loaf',
931: 'bagel',
932: 'pretzel',
933: 'cheeseburger',
934: 'hotdog',
935: 'mashed_potato',
936: 'head_cabbage',
937: 'broccoli',
938: 'cauliflower',
939: 'zucchini',
940: 'spaghetti_squash',
941: 'acorn_squash',
942: 'butternut_squash',
943: 'cucumber',
944: 'artichoke',
945: 'bell_pepper',
946: 'cardoon',
947: 'mushroom',
948: 'Granny_Smith',
949: 'strawberry',
950: 'orange',
951: 'lemon',
952: 'fig',
953: 'pineapple',
954: 'banana',
955: 'jackfruit',
956: 'custard_apple',
957: 'pomegranate',
958: 'hay',
959: 'carbonara',
```

```
960: 'chocolate_sauce',
961: 'dough',
962: 'meat_loaf',
963: 'pizza',
964: 'potpie',
965: 'burrito',
966: 'red_wine',
967: 'espresso',
968: 'cup',
969: 'eggnog',
970: 'alp',
971: 'bubble',
972: 'cliff',
973: 'coral_reef',
974: 'geyser',
975: 'lakeside',
976: 'promontory',
977: 'sandbar',
978: 'seashore',
979: 'valley',
980: 'volcano',
981: 'ballplayer',
982: 'groom',
983: 'scuba_diver',
984: 'rapeseed',
985: 'daisy',
986: "yellow_lady's_slipper",
987: 'corn',
988: 'acorn',
989: 'hip',
990: 'buckeye',
991: 'coral_fungus',
992: 'agaric',
993: 'gyromitra',
994: 'stinkhorn',
995: 'earthstar',
996: 'hen-of-the-woods',
997: 'bolete',
998: 'ear',
999: 'toilet_tissue'}
```

1.1.1 Helper Functions

Our pretrained model was trained on images that had been preprocessed by subtracting the per-color mean and dividing by the per-color standard deviation. We define a few helper functions for performing and undoing this preprocessing. You don't need to do anything in this cell.

[5]:
```
def preprocess(img, size=224):
    transform = T.Compose([
        T.Resize((size, size)),
        T.ToTensor(),
        T.Normalize(mean=SQUEEZENET_MEAN.tolist(),
              std=SQUEEZENET_STD.tolist()),
        T.Lambda(lambda x: x[None]),
    ])
    return transform(img)

def deprocess(img, should_rescale=True):
    transform = T.Compose([
        T.Lambda(lambda x: x[0]),
        T.Normalize(mean=[0, 0, 0], std=(1.0 / SQUEEZENET_STD).tolist()),
        T.Normalize(mean=(-SQUEEZENET_MEAN).tolist(), std=[1, 1, 1]),
        T.Lambda(rescale) if should_rescale else T.Lambda(lambda x: x),
        T.ToPILImage(),
    ])
    return transform(img)

def rescale(x):
    low, high = x.min(), x.max()
    x_rescaled = (x - low) / (high - low)
    return x_rescaled

def blur_image(X, sigma=1):
    X_np = X.cpu().clone().numpy()
    X_np = gaussian_filter1d(X_np, sigma, axis=2)
    X_np = gaussian_filter1d(X_np, sigma, axis=3)
    X.copy_(torch.Tensor(X_np).type_as(X))
    return X
```

1.2 Task 6 - Pre-trained Convolution Network

In order to get a better sense of the classification decisions made by convolutional networks, your job is now to experiment by running whatever images you want through a model pretrained on ImageNet. These can be images from your own photo collection, from the internet, or somewhere else but they **should belong to one of the ImageNet classes**. Look at the `idx2label` dictionary for all the ImageNetclasses.

You need to find: 1. One image (`img1`) where the SqueezeNet model gives reasonable predictions, and produces a category label that seems to correctly describe the content of the image 2. One image (`img2`) where the SqueezeNet model gives unreasonable predictions, and produces a category label that does not correctly describe the content of the image.

You can upload images in Colab by using the upload button on the top left. For more details about using Colab, please see our Colab tutorial.

[17]:
```
################################################################################
# TODO: Upload your image and run the forward pass to get the ImageNet class. #
# This code will crash when you run it, since the maxresdefault.jpg image is  #
# not found. You should upload your own images to the Colab notebook and edit #
# these lines to load your own image.                                          #
################################################################################
img1 = Image.open('cup1.png').convert('RGB')
img2 = Image.open('cup2.png').convert('RGB')
names = ['cup1', 'cup2']
################################################################################
#                END OF YOUR CODE                                              #
################################################################################
for i, img in enumerate([img1, img2]):
    X = preprocess(img).to(device)
    pred_class = torch.argmax(model(X)).item()
    plt.figure(figsize=(6,8))
    plt.imshow(img)
    plt.title('Predicted Class: %s' % idx2label[pred_class])
    plt.axis('off')
    plt.savefig(f'{names[i]}_pred.jpg')
    plt.show()
```

Predicted Class: measuring_cup