In [3]:
```
!pip install -Uqq fastbook
import fastbook
fastbook.setup_book()
from fastbook import *
```

Training a State-of-the-Art Model

Image Classification에 사용되는 SOTA 테크닉을 배워볼 수 있음

- normalization, data augmentation, progressive resizing, time augmentation
  - 전이학습이 아닌 전체 모델을 학습함
  - ImageNet의 일부분인 Imagenette을 사용
    - 전체 데이터의 일부분인 10개를 분류

Imagenette

기본적인 fastai 에서 컴퓨터 비전 모델을 만들 때 사용한 데이터

- ImageNet: 130만개의 ~500 pixel 이미지, 1000 개의 카테고리 (학습시간이 며칠걸림)
- MNIST: 50,000 개의 28X28 pixel 그레이 스케일의 숫자 데이터
- CIFAR10 60,000 32X32 pixel 의 컬러 이미지, 10 개의 카테고리

  작은 데이터는 큰 사이즈의 ImageNet으로 generalize가 안됨 (ImageNet을 잘 분류하려면 ImageNet 데이터로 학습해야 함)
- 큰 데이터/컴퓨팅 파워의 인프라를 사용해야 좋은 image 분류 알고리즘을 만들 수 있음
- 새로운 데이터를 만드는 작업이 필요함: 작고 빠른 실험을 통해 큰 데이터에도 generalize 될 수 있는 알고리즘을 만들 수 있는 데이터
  - Imagenette: 전체 ImageNet에서 distinct 하게 다른 10 class를 추출
  - Imagenette 에서 최적화된 모델이 ImageNet에서도 잘 perform 함

주어진 데이터를 이용한 학습이 오래 걸리면, 데이터를 어떻게 추출해서 빠르게 학습해서 실험해 볼 수 있을지 고민해봐야 함

In [4]:
```
# Imagenette 데이터를 불러옴
from fastai.vision.all import *
path = untar_data(URLs.IMAGENETTE)
```

In [5]:
```
# 불러온 path를 이용하여 datablock 및 dataloaders 에 넣음
dblock = DataBlock(blocks=(ImageBlock(), CategoryBlock()),
                   get_items=get_image_files,
                   get_y=parent_label,
                   item_tfms=Resize(460),
                   batch_tfms=aug_transforms(size=224, min_scale=0.75))
dls = dblock.dataloaders(path, bs=64)
```

In [6]:
```
# 모델을 학습시킴 (pretrained 모델을 사용하지 않고 전체 모델을 처음부터 학습)
model = xresnet50()
learn = Learner(dls, model, loss_func=CrossEntropyLossFlat(), metrics=accuracy
)
learn.fit_one_cycle(3, 3e-3)
```

epoch	train_loss	valid_loss	accuracy	time
0	1.596185	2.164125	0.439507	02:13
1	1.140900	1.011172	0.679238	02:14
2	0.839472	0.758799	0.759149	02:13

Normalization

- 모델을 학습하는데는 normalization 이 도움이 됨
  - 데이터를 mean을 0, stdev 를 1로 만듬
  - 일반적으로 이미지는 0 ~ 255 의 픽셀값을 가짐

In [7]:
```
# 앞의 dataloaders에서 batch를 확인
# x 의 mean이 0, stdev가 1이 아님을 알 수 있음
x,y = dls.one_batch()
x.mean(dim=[0,2,3]), x.std(dim=[0,2,3])
```

Out[7]:
```
(TensorImage([0.4898, 0.4743, 0.4745], device='cuda:0'),
 TensorImage([0.2877, 0.2870, 0.3101], device='cuda:0'))
```

In [8]:
```
# Normalize transform 을 이용해서 batch 마다 normalize가 가능함 (batch_tfms 의 a
rgument로 작용)
# Normalize에 필요한 mean, stdev를 지정해주지 않으면 fastai가 첫 batch 에서 계산
된 값을 사용
# ImageNet은 벌써 mean, stdev가 계산되어있는 데이터임
def get_dls(bs, size):
    dblock = DataBlock(blocks=(ImageBlock, CategoryBlock),
                       get_items=get_image_files,
                       get_y=parent_label,
                       item_tfms=Resize(460),
                       batch_tfms=[*aug_transforms(size=size, min_scale=0.75),
                                   Normalize.from_stats(*imagenet_stats)])
    return dblock.dataloaders(path, bs=bs)
```

In [9]:
```
# mean 과 stdev 가 normalize 된 걸 확인할 수 있음
dls = get_dls(64, 224)
x,y = dls.one_batch()
x.mean(dim=[0,2,3]),x.std(dim=[0,2,3])
```

Out[9]:
```
(TensorImage([-0.1891, -0.0950, -0.0159], device='cuda:0'),
 TensorImage([1.2402, 1.2382, 1.3171], device='cuda:0'))
```

기본적으로 normalization은 pretrained model 을 사용할 때 도움됨

- normaliztion을 한 모델이 raw data input을 받으면 값이 달라짐
  - 학습한 모델을 deploy할 때 관련 된 stat 도 같이 배포해야 함
  - fastai 의 cnn_learner는 자동적으로 normalize/관련 stat을 적용함

Progressive Resizing

2018년에 DAWNBench 대회에서 fastai 가 우승할 수 있었던 이유는 progressive resizing 테크닉을 사용했기 때문

- 초반의 epoch 은 작은 사이즈의 이미지로 학습, 학습 후반은 사이즈가 큰 이미지로 학습
- 학습의 시간이 많이 단축됨

Conv layer의 feature들은 이미지의 사이즈와 무관함

- 초반 레이어는 edge와 gradient같은 feature을 후반 레이어는 nose와 sunset같은 특정 feature를 학습함
- 학습 구간에 따라 이미지 사이즈를 변경하는 부분은 transfer learning 을 접목시켰기 때문
  - 이미지를 resize하고 fine_tune 기능을 사용함

Progressive resizing 은 data augmentation의 일부분이기 때문에 학습된 모델이 generalize가 잘 됨

In [10]:
```
# 이미지의 사이즈와 batch 사이즈를 받는 함수 get_dls 를 이용하여 dataloaders 를 만듬
dls = get_dls(128, 128)
learn = Learner(dls, xresnet50(), loss_func=CrossEntropyLossFlat(),
                metrics=accuracy)
learn.fit_one_cycle(4, 3e-3)
```

epoch	train_loss	valid_loss	accuracy	time
0	1.821151	2.989995	0.354369	01:58
1	1.332325	1.074323	0.662808	01:57
2	0.994421	0.820618	0.744959	01:57
3	0.765756	0.669199	0.798730	01:58

In [11]:
```
# 학습된 learn 모델의 dataloaders를 사이즈를 변경한 이미지로 replace 한 후 fine-tune 한다
# 학습효과가 더 좋아진것을 볼 수 있음
learn.dls = get_dls(64, 224)
learn.fine_tune(5, 1e-3)
```

epoch	train_loss	valid_loss	accuracy	time
0	0.823573	1.163354	0.610530	02:14

epoch	train_loss	valid_loss	accuracy	time
0	0.682341	0.679250	0.786408	02:13
1	0.673007	0.622537	0.806946	02:14
2	0.596636	0.679535	0.783047	02:14
3	0.513168	0.518942	0.835698	02:13
4	0.422147	0.458502	0.859597	02:13

초반에 사이즈가 작은 이미지를 학습할 때 학습시간이 더 작음 사이즈가 큰 이미지를 추가해서 epoch을 돌리면 모델 perform이 올라감

- pretrained 모델에 사용하면 오히려 모델 perform이 떨어질 수 있음
  - 기존의 모델이 학습한 이미지와 비슷한 이미지를 사용할 경우
  - 학습되지 않은 다른 사이즈와 형태의 이미지를 학습하는 경우는 progressive resizing이 도움될 수 있음

Test Time Augmentation

일반적인 data augmentation에 지금까지는 random cropping 을 사용함

- generalization과 training time에 도움됨
- fastai 는 validation set 에는 center cropping 을 적용함 (이미지의 중심을 cropping 하는데 사용될 수 있는 가장 큰 정사각형)
  - multi-label 데이터에서 작은 object가 이미지 구석에 있으면 문제가 될 수 있음
  - 애완동물을 분류하는데 필요한 critical feature이 이미지의 중심에 없을 수도 있음

해결방법

- Rectangle 이미지를 stretch/squeeze 를 통해 정사각형으로 만드는 data augmentation
  - 이미지가 변형되기 때문에 모델이 학습하는데 더 어려워지는 문제가 있음
- Validation 중 이미지의 중심이 아닌 여러 부분에서 cropping 한 데이터를 모델을 통과시키고 결과 prediction 값의 max 또는 average를 이용해서 결과값 예측 (Test Time Augmentation)
  - fastai 는 TTA 동안 data augmentation을 하지 않은 이미지의 center crop + 4개의 randomly cropped 이미지를 사용함

In [12]:
```
# Dataloader 를 사용하여 바로 tta 적용
# TTA 가 모델의 perform을 올리긴 하지만 (추가 학습이 필요없음) inferencing 시간이 길어짐 (5배)
preds,targs = learn.tta()
accuracy(preds, targs).item()
```

Out[12]: 0.863704264163971

Mixup

2017에 논문으로 발표된 Data augmentation 방법

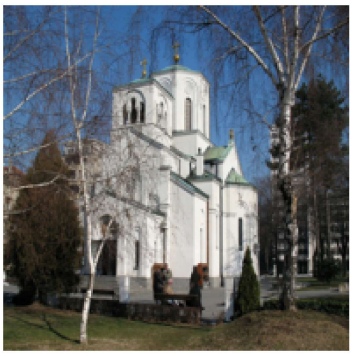
- 데이터가 얼마 없고 테스트하는 데이터와 비슷한 데이터로 학습된 pretrained 모델이 없을 때 사용
  - data augmentation 은 모델의 perform 을 올리긴 하지만, dataset dependent 하고 expert knowledge 가 필요함

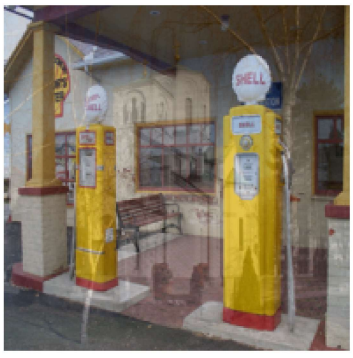
Mixup

1. 데이터에서 random 하게 다른 이미지를 선택
2. random 한 weight을 선택
3. 2번의 weight을 이용하여 선택된 이미지와 현재 예측하는 이미지와 weighted average를 함 (indep variable)
4. 선택된 이미지의 label과 예측하는 이미지의 label을 weighted average 함 (dep variable)

In [13]:
```
# Mixup 의 예, 교회와 (30%) 주유소의 (70%) weighted average 된 모습
church = PILImage.create(get_image_files_sorted(path/'train'/'n03028079')[0])
gas = PILImage.create(get_image_files_sorted(path/'train'/'n03425413')[0])
church = church.resize((256,256))
gas = gas.resize((256,256))
tchurch = tensor(church).float() / 255.
tgas = tensor(gas).float() / 255.

_,axs = plt.subplots(1, 3, figsize=(12,4))
show_image(tchurch, ax=axs[0]);
show_image(tgas, ax=axs[1]);
show_image((0.3*tchurch + 0.7*tgas), ax=axs[2]);
```





In [ ]:
```
# Mixup의 결과값은 one hot encoded 되어있기 때문에 각 class의 확률값으로 예측됨
#fastai 에서는 Callback을 이용해서 training loop에 커스텀한 function/behavior를
 넣을 수 있음
```

Mixup의 결과값은 one hot encoded 되어있기 때문에 각 class의 확률값으로 예측됨

- [0, 0, 0.3, 0, 0, 0, 0, 0.7, 0, 0]

fastai 에서는 Callback을 이용해서 training loop에 커스텀한 function/behavior를 넣을 수 있음 Mixup은 더 많은 epoch이 학습시 필요함 Mixup 은 사진이 아닌 다른 application에도 사용 될 수 있음 (NLP) 데이터의 label은 0 또는 1이지만 sigmoid/softmax 는 0 과 1 은 예측 못함 (근사치만 가능)

- epoch 수가 높아질수록 점점 더 activationd은 0 과 1 값에 가까워지려 커질 수 있음
- Mixup을 사용하면 0 과 1이 가능할 때가 있음 (randomly 두개의 사진이 같은 class일 경우)

Label Smoothing

모델 학습의 label이 0 또는 1이지만 근사치만 예측 가능하기 때문에 (0.999) 학습시에 모델은 더 confident 한 activation을 배우기 위해 gradient 를 계속 업데이트 함

- 모델이 overfit 할 수 있음
- 데이터가 mislabel 되어있을 수도 있음

모델의 값을 0보다 약간 높게, 1보다 약간 낮게 바꿈 (label smoothing)

- 모델이 less confident 하게 함으로써 좀 더 generalize하게 할 수 있음

one hot encode 되어있는 label 의 0을 $\frac{\epsilon}{N}$ 로 치환 (N 은 데이터의 class 갯수, ϵ 는 학습되는 파라미터)

- 일반적으로 0.1 -> 10%정도의 데이터 label이 정확하지 않을 수 있다

반대로 label의 1은 합이 1이될 수 있게 $1 - \epsilon + \frac{\epsilon}{N}$ 로 치환

예를들어 index 3번을 예측할 때의 결과값

```
[0.01, 0.01, 0.01, 0.91, 0.01, 0.01, 0.01, 0.01, 0.01, 0.01]
```

In [ ]:
```
# Learner 의 loss function을 label smoothing 함수로 지정할 수 있음
model = xresnet50()
learn = Learner(dls, model, loss_func=LabelSmoothingCrossEntropy(),
                metrics=accuracy)
learn.fit_one_cycle(5, 3e-3)
```