評分標準

weighted average recall

$$\sum_{i=1}^{N} W(i) * Recall(i)$$

測試集：包含決賽、初賽資料 5:1

分類準則

Q： 兩項競賽「三類等級分類」的 ABC 級有什麼準則嗎？
A： 可參考圖片及以下敘述：

- A：完全沒有瑕疵，色澤均勻漂亮
- B：色澤不均，瑕疵範圍若可以一個拇指蓋住，還是可以賣的狀況，都是B。
- C：不能賣，只要有黑斑、炭疽病、爛，都算是C。

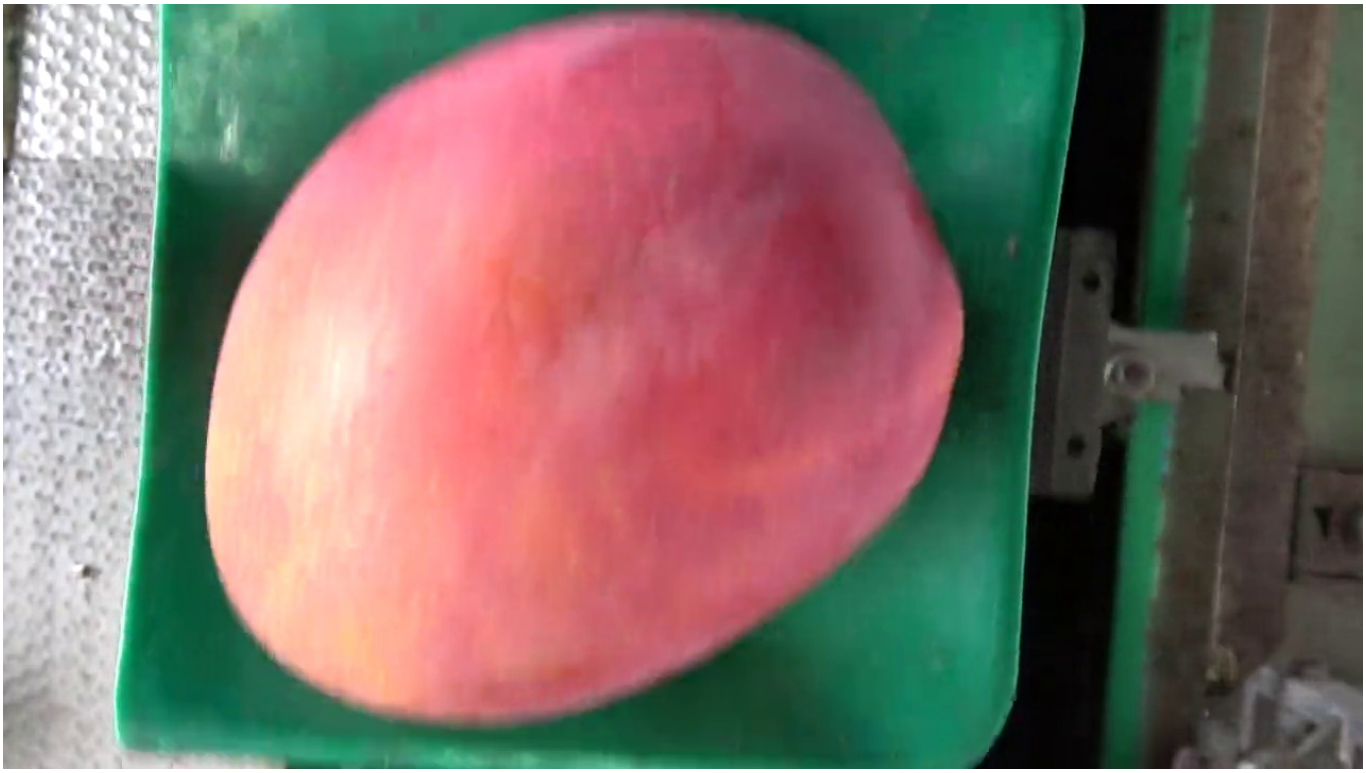
（瑕疵指的是圖中的所有病種）

EDA

EDA_cum_image_processing (https://www.kaggle.com/fireheart7/eda-cum-image-processing#Analysing-the-color-channel-distribution-in-each-Image)

去影像模糊、雜訊

Effects of Image Degradation and Degradation Removal to CNN-based Image Classification (https://drive.google.com/file/d/1YmMfXyK19hL0UAkc4l7mKYS2BmMLxDh8/view?usp=sharing)
Effects of Image Degradations to CNN-based Image Classification (https://arxiv.org/pdf/1810.05552.pdf)



不平衡數據

Grade	Train	Dev	Test
A	28585	5388	blinded
B	11140	1016	blinded
C	5275	596	blinded

Focal Loss for Class Imbalance Problem (https://arxiv.org/pdf/2001.03329.pdf)
不平衡數據處理 (https://blog.csdn.net/asialee_bird/article/details/83714612)
處理imbalanced dataset (https://medium.com/analytics-vidhya/handling-imbalanced-dataset-in-image-classification-dc6f1e13aeee)
How to deal with Unbalanced Image Datasets in less than 20 lines of code (https://medium.com/analytics-vidhya/how-to-apply-data-augmentation-to-deal-with-unbalanced-datasets-in-20-lines-of-code-ada8521320c9)
In classification, how do you handle an unbalanced training set? (https://www.quora.com/In-classification-how-do-you-handle-an-unbalanced-training-set/answers/1144228?srid=h3G6o)
scikit-learn API:imbalanced-learn (https://github.com/scikit-learn-contrib/imbalanced-learn)
Deep learning unbalanced training data?Solve it like this. (https://towardsdatascience.com/deep-learning-unbalanced-training-data-solve-it-like-this-6c528e9efea6)
A systematic study of the class imbalance problem in convolutional neural networks* (https://arxiv.org/pdf/1710.05381.pdf)

訓練技巧

label smoothing (https://zhuanlan.zhihu.com/p/101553787)
test time augmentation(TTA) (https://www.kaggle.com/andrewkh/test-time-augmentation-tta-worth-it)
Ranger 優化器 (https://zhuanlan.zhihu.com/p/100877314)
SGDR (https://zhuanlan.zhihu.com/p/52084949)
Mish 激活函數 (https://zhuanlan.zhihu.com/p/84418420)
flat cosine anneal (https://medium.com/@lessw/how-we-beat-the-fastai-leaderboard-score-by-19-77-a-cbb2338fab5c)
mixup (https://blog.csdn.net/u013841196/article/details/81049968)
label smoothing (https://arxiv.org/pdf/1906.02629.pdf)
transfer learning (https://blog.csdn.net/Emma_Love/article/details/88093975)
weight decay (https://www.zhihu.com/question/65626362/answer/960145051)
Snapshot Ensembles (https://zhuanlan.zhihu.com/p/93648558)
Pseudo Labeling (https://www.kaggle.com/cdeotte/pseudo-labeling-qda-0-969)
to_fp16 (https://zhuanlan.zhihu.com/p/84219777)
class weight (https://blog.csdn.net/xpy870663266/article/details/104600054)

調參

Bayesian optimization (https://medium.com/@hiepnguyen034/improving-neural-networks-performance-with-bayesian-optimization-efbaa801ad26)
調參
(https://medium.com/@jacky308082/%E8%87%AA%E5%8B%95%E5%8C%96%E8%AA%BF%E6%95%B4%E4%BD%BF%E7%94%A8python-40edb9f0b462)
貝葉斯優化 (https://zhuanlan.zhihu.com/p/53826787)

模型

ImageNet benchmark (https://paperswithcode.com/sota/image-classification-on-imagenet)
Fine-Grained Image Classification Benchmarks (https://paperswithcode.com/task/fine-grained-image-classification)
Pytorch 預訓練模型 (https://pytorch.org/hub/research-models)
ResNeXt (https://zhuanlan.zhihu.com/p/32913695)
Ghostnet (https://zhuanlan.zhihu.com/p/109325275)
xResNet (https://towardsdatascience.com/xresnet-from-scratch-in-pytorch-e64e309af722)
senet (https://arxiv.org/pdf/1709.01507.pdf)
densenet (https://zhuanlan.zhihu.com/p/37189203)
effientnet (https://arxiv.org/pdf/1905.11946.pdf)
resnest (https://zhuanlan.zhihu.com/p/132655457)
resnet (https://zhuanlan.zhihu.com/p/31852747)
Pnasnet5large (https://zhuanlan.zhihu.com/p/52798148)
SqueezeNet、MobileNet、ShuffleNet、Xception (https://zhuanlan.zhihu.com/p/32746221)
Inception系列 (https://zhuanlan.zhihu.com/p/37505777)

細粒度模型

S3N
(https://openaccess.thecvf.com/content_ICCV_2019/papers/Ding_Selective_Sparse_Sampling_for_Fine-Grained_Image_Recognition_ICCV_2019_paper.pdf)
S3N github (https://github.com/Yao-DD/S3N)
Multi-branch and Multi-scale Attention Learning for Fine-Grained Visual Categorization
(https://arxiv.org/pdf/2003.09150v3.pdf)
Weakly Supervised Fine-grained Image Classification via Guassian Mixture Model Oriented Discriminative Learning (https://openaccess.thecvf.com/content_CVPR_2020/papers/Wang_Weakly_Supervised_Fine-Grained_Image_Classification_via_Guassian_Mixture_Model_Oriented_CVPR_2020_paper.pdf)

切割圖片

Mask_RCNN (https://github.com/matterport/Mask_RCNN)
Instance Segmentation with Mask R-CNN (https://engineering.matterport.com/splash-of-color-instance-segmentation-with-mask-r-cnn-and-tensorflow-7c761e238b46)

模型融合

模型融合 (https://zhuanlan.zhihu.com/p/25836678)

比賽

kaggle 亞馬遜雨林衛星圖像比賽 1st (https://zhuanlan.zhihu.com/p/28084438)
Kaggle Top 2% APTOS 2019 (https://zhuanlan.zhihu.com/p/81695773)
Kaggle 識別座頭鯨 1st solution (https://zhuanlan.zhihu.com/p/58496385)
Kaggle 識別座頭鯨 code (https://github.com/earhian/Humpback-Whale-Identification-1st-)
Kaggle 分辨雜草和植物幼苗 5th solution (https://zhuanlan.zhihu.com/p/38359300)
1st place Solution for Intel Scene Classification Challenge (https://towardsdatascience.com/1st-place-solution-for-intel-scene-classification-challenge-c95cf941f8ed)
2019細粒度圖像分類挑戰賽冠軍 (https://www.zhihu.com/question/331320468/answer/727015760)
第二屆腫瘤切割挑戰賽-第二名 (https://medium.com/@aaronkao/%E8%A4%87%E7%9B%A4-%E7%AC%AC%E4%BA%8C%E5%B1%86%E5%8F%B0%E7%81%A3%E8%85%AB%E7%98%A4%E5%6b3742cdb5cb)

kaggle fgvc7

kaggle FGVC競賽 (https://www.kaggle.com/search?q=FGVC+in%3Acompetitions)

kaggle fgvc7

kaggle FGVC競賽 (https://www.kaggle.com/search?q=FGVC+in%3Acompetitions)

Semi-Supervised Recognition Challenge - FGVC7

3rd place solution (https://arxiv.org/pdf/2006.10702.pdf)
1st Place Solution (https://www.kaggle.com/c/semi-inat-2020/discussion/160724)

iWildCam 2020

3rd place solution (https://www.kaggle.com/c/iwildcam-2020-fgvc7/discussion/157932)

Herbarium 2020

1st place solution (https://www.kaggle.com/c/herbarium-2020-fgvc7/discussion/154351)
2nd Place Solution (https://www.kaggle.com/c/herbarium-2020-fgvc7/discussion/154186)

iMaterialist Challenge (Furniture) at FGVC5

4th place solution (https://www.kaggle.com/c/imaterialist-challenge-furniture-2018/discussion/57939)

iMet Collection 2019 - FGVC6

2nd place solution (https://www.kaggle.com/c/imet-2019-fgvc6/discussion/96149)
1st place solution (https://www.kaggle.com/c/imet-2019-fgvc6/discussion/94687)

Plant Pathology 2020

2020植物病理分類1st方案 (https://github.com/alipay/cvpr2020-plant-pathology)
BiLinear EfficientNet Focal Loss+ Label Smoothing (20th) (https://www.kaggle.com/jimitshah777/bilinear-efficientnet-focal-loss-label-smoothing)

2nd place Solution (https://www.kaggle.com/c/plant-pathology-2020-fgvc7/discussion/155929)
leaf_disease_main_notebook (https://www.kaggle.com/fireheart7/leaf-disease-main-notebook/notebook)
Plant Pathology 2020 in PyTorch - 0.974 score (https://www.kaggle.com/akasharidas/plant-pathology-2020-in-pytorch)

參考

多個模型 (http://home.ifi.uio.no/paalh/publications/files/ism2018-ovr.pdf)
Image Classification Baseline Model For 2020 (https://towardsdatascience.com/image-classification-baseline-model-for-2020-1d33f0986fc0)
How we beat the FastAI leaderboard score by +19.77% (https://medium.com/@lessw/how-we-beat-the-fastai-leaderboard-score-by-19-77-a-cbb2338fab5c)
Mango Classification kaggle範例 (https://www.kaggle.com/rkuo2000/mango-classification)
fastai 技巧 (https://zhuanlan.zhihu.com/p/41192499)
Facial age prediction with Fastai2 (https://medium.com/analytics-vidhya/facial-age-prediction-with-fastai2-d67fdb575539)
Senet詳解 (https://cloud.tencent.com/developer/article/1610426)
label smoothing理解 (https://zhuanlan.zhihu.com/p/72685158)
focal loss理解 (https://zhuanlan.zhihu.com/p/80594704)
cs230-deep learning (https://stanford.edu/~shervine/teaching/cs-230/cheatsheet-deep-learning-tips-and-tricks#good-practices)
fastai技巧 (https://zhuanlan.zhihu.com/p/41379279)
lr_find() (https://sgugger.github.io/how-do-you-find-a-good-learning-rate.html)
小白通过kaggle学习图像分类笔记之二 (https://zhuanlan.zhihu.com/p/105190491)
小白通過kaggle學習圖像分類筆記 (https://zhuanlan.zhihu.com/p/104694474)
fastai-callbacks (https://medium.com/@lessw/fastais-callbacks-for-better-cnn-training-meet-savemodelcallback-e55f254f1af5)

GAN

SinGAN (https://arxiv.org/pdf/1905.01164.pdf)
WGAN (http://ceur-ws.org/Vol-2563/aics_34.pdf)

In [ ]:

```
%reload_ext autoreload
%autoreload 2
%matplotlib inline
```

In [ ]:

```
import torch
torch.__version__
```

In [ ]:

```
torch.cuda.is_available()
```

In [ ]:

```
#!pip freeze > requirements.txt
```

In [ ]:

```
!pip list
```

In [ ]:

```
import fastai2
from efficientnet_pytorch import EfficientNet
from fastai.vision.models.cadene_models import *
from fastai2.callback.schedule import fit_flat_cos
```

In [ ]:

```
from fastai import *
from fastai.vision import *
from fastai.vision.learner import cnn_config
from fastai.callbacks import *
import matplotlib.pyplot as plt
from PIL import Image
import pandas as pd
import cv2
import shutil
import math
from math import floor
from fastai2.test_utils import *
import seaborn as sns
import numpy as np
import math
```

In [ ]:

```
import warnings
warnings.filterwarnings('ignore')
```

EDA

設置資料集

處理數據不平衡

In [ ]:

```
?
```

去影像模糊、雜訊

In [ ]:

```
?
```

切割圖片

In [ ]:

```
?
```

Histogram equalization

Data augmentation

Focal loss

Ranger optimizer

In [ ]:

```
from torch.optim.optimizer import Optimizer, required
class Ranger(Optimizer):

    def __init__(self, params, lr=1e-3,                       # lr
                 alpha=0.5, k=6, N_sma_threshhold=5,           # Ranger options
                 betas=(.95, 0.999), eps=1e-5, weight_decay=0,  # Adam options
                 # Gradient centralization on or off, applied to conv layers onl
y or conv + fc layers
                 use_gc=True, gc_conv_only=False
                 ):

        # parameter checks
        if not 0.0 <= alpha <= 1.0:
            raise ValueError(f'Invalid slow update rate: {alpha}')
        if not 1 <= k:
            raise ValueError(f'Invalid lookahead steps: {k}')
        if not lr > 0:
            raise ValueError(f'Invalid Learning Rate: {lr}')
        if not eps > 0:
            raise ValueError(f'Invalid eps: {eps}')

        # parameter comments:
        # beta1 (momentum) of .95 seems to work better than .90...
        # N_sma_threshold of 5 seems better in testing than 4.
        # In both cases, worth testing on your dataset (.90 vs .95, 4 vs 5) to m
ake sure which works best for you.

        # prep defaults and init torch.optim base
        defaults = dict(lr=lr, alpha=alpha, k=k, step_counter=0, betas=betas,
                        N_sma_threshhold=N_sma_threshhold, eps=eps, weight_decay
=weight_decay)
        super().__init__(params, defaults)

        # adjustable threshold
        self.N_sma_threshhold = N_sma_threshhold

        # look ahead params

        self.alpha = alpha
        self.k = k

        # radam buffer for state
        self.radam_buffer = [[None, None, None] for ind in range(10)]

        # gc on or off
        self.use_gc = use_gc

        # level of gradient centralization
        self.gc_gradient_threshold = 3 if gc_conv_only else 1

        print(
            f"Ranger optimizer loaded. \nGradient Centralization usage = {self.u
se_gc}")
        if (self.use_gc and self.gc_gradient_threshold == 1):
            print(f"GC applied to both conv and fc layers")
        elif (self.use_gc and self.gc_gradient_threshold == 3):
            print(f"GC applied to conv layers only")

    def __setstate__(self, state):
```

```
        print("set state called")
        super(Ranger, self).__setstate__(state)

    def step(self, closure=None):
        loss = None
        # note - below is commented out b/c I have other work that passes back t
he loss as a float, and thus not a callable closure.
        # Uncomment if you need to use the actual closure...

        # if closure is not None:
        #loss = closure()

        # Evaluate averages and grad, update param tensors
        for group in self.param_groups:

            for p in group['params']:
                if p.grad is None:
                    continue
                grad = p.grad.data.float()

                if grad.is_sparse:
                    raise RuntimeError(
                        'Ranger optimizer does not support sparse gradients')

                p_data_fp32 = p.data.float()

                state = self.state[p]  # get state dict for this param

                if len(state) == 0:  # if first time to run...init dictionary wi
th our desired entries
                    # if self.first_run_check==0:
                    # self.first_run_check=1
                    #print("Initializing slow buffer...should not see this at lo
ad from saved model!")
                    state['step'] = 0
                    state['exp_avg'] = torch.zeros_like(p_data_fp32)
                    state['exp_avg_sq'] = torch.zeros_like(p_data_fp32)

                    # look ahead weight storage now in state dict
                    state['slow_buffer'] = torch.empty_like(p.data)
                    state['slow_buffer'].copy_(p.data)

                else:
                    state['exp_avg'] = state['exp_avg'].type_as(p_data_fp32)
                    state['exp_avg_sq'] = state['exp_avg_sq'].type_as(
                        p_data_fp32)

                # begin computations
                exp_avg, exp_avg_sq = state['exp_avg'], state['exp_avg_sq']
                beta1, beta2 = group['betas']

                # GC operation for Conv layers and FC layers
                if grad.dim() > self.gc_gradient_threshold:
                    grad.add_(-grad.mean(dim=tuple(range(1, grad.dim())), keepdi
m=True))

                state['step'] += 1

                # compute variance mov avg
                exp_avg_sq.mul_(beta2).addcmul_(1 - beta2, grad, grad)
                # compute mean moving avg
```

```
                exp_avg.mul_(beta1).add_(1 - beta1, grad)

                buffered = self.radam_buffer[int(state['step'] % 10)]

                if state['step'] == buffered[0]:
                    N_sma, step_size = buffered[1], buffered[2]
                else:
                    buffered[0] = state['step']
                    beta2_t = beta2 ** state['step']
                    N_sma_max = 2 / (1 - beta2) - 1
                    N_sma = N_sma_max - 2 * \
                        state['step'] * beta2_t / (1 - beta2_t)
                    buffered[1] = N_sma
                    if N_sma > self.N_sma_threshhold:
                        step_size = math.sqrt((1 - beta2_t) * (N_sma - 4) / (N_s
ma_max - 4) * (
                            N_sma - 2) / N_sma * N_sma_max / (N_sma_max - 2)) /
(1 - beta1 ** state['step'])
                    else:
                        step_size = 1.0 / (1 - beta1 ** state['step'])
                    buffered[2] = step_size

                if group['weight_decay'] != 0:
                    p_data_fp32.add_(-group['weight_decay']
                                     * group['lr'], p_data_fp32)

                # apply lr
                if N_sma > self.N_sma_threshhold:
                    denom = exp_avg_sq.sqrt().add_(group['eps'])
                    p_data_fp32.addcdiv_(-step_size *
                                         group['lr'], exp_avg, denom)
                else:
                    p_data_fp32.add_(-step_size * group['lr'], exp_avg)

                p.data.copy_(p_data_fp32)

                # integrated look ahead...
                # we do it at the param level instead of group level
                if state['step'] % group['k'] == 0:
                    # get access to slow param tensor
                    slow_p = state['slow_buffer']
                    # (fast weights - slow weights) * alpha
                    slow_p.add_(self.alpha, p.data - slow_p)
                    # copy interpolated weights to RAdam param tensor
                    p.data.copy_(slow_p)
        return loss
```

Model

In [1]:

```
# AdamW  one-cycle-policy
# SGD    sgdr
# Ranger flat cosine annealing
# Mixup
# Labelsmoothing
# train 30 epochs
# fine tune 20 epochs (lr、wd 調整)
```

senet154

In [ ]:

```
ranger=partial(Ranger)
learn = cnn_learner(data,arch,metrics=[accuracy,Recall('weighted'),FBeta('macro'
)],opt_func=ranger,loss_func=LabelSmoothingCrossEntropy(),pretrained=True).to_fp
16()
```

In [ ]:

```
fit_fc(learn,30,1e-02,wd=1e-02)
```

In [ ]:

```
#Test time augmentation
preds,targs = learn.TTA()
accuracy(preds, targs).item()
```

se_resnet

se_resnext

efficientnet

In [ ]:

```
model=EfficientNet.from_pretrained('efficientnet-b0')
```

In [ ]:

```
feature = model._fc.in_features
model._fc = nn.Linear(in_features=feature,out_features=3,bias=True)
print(model)
```

In [ ]:

```
model._fc.weight
```

In [ ]:

```
nn.init.kaiming_uniform_(model._fc.weight)
```

In [ ]:

```
ct=0
for child in learn.model.children():
    ct+=1
    if ct<2:
        for param in child.parameters():
            param.requires_grad = False
```

pnasnet5large

inceptionresnetv2

dpn92

ResNext(WSL)

ResNeSt

In [10]:

```
torch.hub.list('zhanghang1989/ResNeSt', force_reload=True)
```

```
Downloading: "https://github.com/zhanghang1989/ResNeSt/archive/maste
r.zip" to /home/aistudent/.cache/torch/hub/master.zip
```

Out[10]:

```
['resnest101',
 'resnest200',
 'resnest269',
 'resnest50',
 'resnest50_fast_1s1x64d',
 'resnest50_fast_1s2x40d',
 'resnest50_fast_1s4x24d',
 'resnest50_fast_2s1x64d',
 'resnest50_fast_2s2x40d',
 'resnest50_fast_4s1x64d',
 'resnest50_fast_4s2x40d']
```

In [ ]:

```
model = torch.hub.load('zhanghang1989/ResNeSt', 'resnest101', pretrained=True)
```

In [ ]:

```
learn = cnn_learner(data,arch,metrics=[accuracy,Recall('weighted'),FBeta('macro'
)],opt_func=ranger,loss_func=LabelSmoothingCrossEntropy(),pretrained=True).to_fp
16()
```

In [ ]:

```
for layer in learn.layer_groups[0]:
    if type(layer)!=nn.Linear and type(layer)!=nn.BatchNorm2d:
        for param in layer.parameters():
                param.requires_grad = False
```

Resnet

Densenet

xception_cadene

nasnetamobile

inceptionv4

GhostNet

In [ ]:

```
model = torch.hub.load('huawei-noah/ghostnet', 'ghostnet_1x', pretrained=True)
```

vgg-nets

squeezenet

mobilenet

shufflenet

細粒度模型

In [ ]:

```
?
```

Mxresnet with ranger(非預訓練模型)

In [8]:

```
%run mxresnet.ipynb
```

```
Mish activation loaded...
```

In [ ]:

```
ranger=partial(Ranger)
```

In [ ]:

```
arch=mxresnet18
```

In [9]:

```
def convert_relu_to_mish(model):
    for child_name, child in model.named_children():
        if isinstance(child, nn.ReLU):
            setattr(model, child_name, Mish())
        else:
            convert_relu_to_mish(child)
```

In [ ]:

```
convert_relu_to_mish(arch())
```

In [ ]:

```
learn = Learner(data,arch,metrics=[accuracy,Recall('weighted'),FBeta('macro')],bn_wd=False,true_wd=True,wd=1e-02,opt_func=ranger,loss_func=LabelSmoothingCrossEntropy()).to_fp16()
```

雙模型

Stratified 5-fold cross-validation

Random search , Bayesian optimization(調參)

移除 cv>0.97 分類錯誤圖片

Pseudo labeling

Retrain

1. 5-fold cv

2. Random search or Bayesian optimization

Grad-CAM

模型融合

In [ ]:

```
?
```

預測test data、製作csv

fastai.vision learner.py

https://github.com/fastai/fastai/blob/master/fastai/vision/learner.py#L94 (https://github.com/fastai/fastai/blob/master/fastai/vision/learner.py#L94)

```
"`Learner` support for computer vision"
from ..torch_core import *
from ..basic_train import *
from ..basic_data import *
from .image import *
from . import models
from ..callback import *
from ..layers import *
from ..callbacks.hooks import *
from ..train import ClassificationInterpretation

__all__ = ['cnn_learner', 'create_cnn', 'create_cnn_model', 'create_body',
'create_head', 'unet_learner']
# By default split models between first and second layer
def _default_split(m:nn.Module): return (m[1],)
# Split a resnet style model
def _resnet_split(m:nn.Module): return (m[0][6],m[1])
# Split squeezenet model on maxpool layers
def _squeezenet_split(m:nn.Module): return (m[0][0][5], m[0][0][8], m[1])
def _densenet_split(m:nn.Module): return (m[0][0][7],m[1])
def _vgg_split(m:nn.Module): return (m[0][0][22],m[1])
def _alexnet_split(m:nn.Module): return (m[0][0][6],m[1])
def _mobilenetv2_split(m:nn.Module): return (m[0][0][10],m[1])

_default_meta     = {'cut':None, 'split':_default_split}
_resnet_meta      = {'cut':-2, 'split':_resnet_split }
_squeezenet_meta  = {'cut':-1, 'split': _squeezenet_split}
_densenet_meta    = {'cut':-1, 'split':_densenet_split}
_vgg_meta         = {'cut':-1, 'split':_vgg_split}
_alexnet_meta     = {'cut':-1, 'split':_alexnet_split}
_mobilenetv2_meta = {'cut':-1, 'split':_mobilenetv2_split}

model_meta = {
    models.resnet18 :{**_resnet_meta}, models.resnet34: {**_resnet_meta},
    models.resnet50 :{**_resnet_meta}, models.resnet101:{**_resnet_meta},
    models.resnet152:{**_resnet_meta},

    models.squeezenet1_0:{**_squeezenet_meta},
    models.squeezenet1_1:{**_squeezenet_meta},

    models.densenet121:{**_densenet_meta}, models.densenet169:{**_densenet
_meta},
    models.densenet201:{**_densenet_meta}, models.densenet161:{**_densenet
_meta},
    models.vgg11_bn:{**_vgg_meta}, models.vgg13_bn:{**_vgg_meta}, models.v
gg16_bn:{**_vgg_meta}, models.vgg19_bn:{**_vgg_meta},
    models.alexnet:{**_alexnet_meta},
    models.mobilenet_v2:{**_mobilenetv2_meta}}

def cnn_config(arch):
    "Get the metadata associated with `arch`."
    torch.backends.cudnn.benchmark = True
```

```
    return model_meta.get(arch, _default_meta)

def has_pool_type(m):
    if is_pool_type(m): return True
    for l in m.children():
        if has_pool_type(l): return True
    return False

def create_body(arch:Callable, pretrained:bool=True, cut:Optional[Union[in
t, Callable]]=None):
    "Cut off the body of a typically pretrained `model` at `cut` (int) or
 cut the model as specified by `cut(model)` (function)."
    model = arch(pretrained)
    cut = ifnone(cut, cnn_config(arch)['cut'])
    if cut is None:
        ll = list(enumerate(model.children()))
        cut = next(i for i,o in reversed(ll) if has_pool_type(o))
    if   isinstance(cut, int):      return nn.Sequential(*list(model.child
ren())[:cut])
    elif isinstance(cut, Callable): return cut(model)
    else:                           raise NamedError("cut must be either i
nteger or a function")


def create_head(nf:int, nc:int, lin_ftrs:Optional[Collection[int]]=None, p
s:Floats=0.5,
                concat_pool:bool=True, bn_final:bool=False):
    "Model head that takes `nf` features, runs through `lin_ftrs`, and abo
ut `nc` classes."
    lin_ftrs = [nf, 512, nc] if lin_ftrs is None else [nf] + lin_ftrs + [n
c]
    ps = listify(ps)
    if len(ps) == 1: ps = [ps[0]/2] * (len(lin_ftrs)-2) + ps
    actns = [nn.ReLU(inplace=True)] * (len(lin_ftrs)-2) + [None]
    pool = AdaptiveConcatPool2d() if concat_pool else nn.AdaptiveAvgPool2d
(1)
    layers = [pool, Flatten()]
    for ni,no,p,actn in zip(lin_ftrs[:-1], lin_ftrs[1:], ps, actns):
        layers += bn_drop_lin(ni, no, True, p, actn)
    if bn_final: layers.append(nn.BatchNorm1d(lin_ftrs[-1], momentum=0.01
))
    return nn.Sequential(*layers)

def create_cnn_model(base_arch:Callable, nc:int, cut:Union[int,Callable]=N
one, pretrained:bool=True,
                     lin_ftrs:Optional[Collection[int]]=None, ps:Floats=0.
5, custom_head:Optional[nn.Module]=None,
                     bn_final:bool=False, concat_pool:bool=True):
    "Create custom convnet architecture"
    body = create_body(base_arch, pretrained, cut)
    if custom_head is None:
        nf = num_features_model(nn.Sequential(*body.children())) * (2 if c
```

```
oncat_pool else 1)
        head = create_head(nf, nc, lin_ftrs, ps=ps, concat_pool=concat_poo
l, bn_final=bn_final)
    else: head = custom_head
    return nn.Sequential(body, head)

def cnn_learner(data:DataBunch, base_arch:Callable, cut:Union[int,Callable
]=None, pretrained:bool=True,
                lin_ftrs:Optional[Collection[int]]=None, ps:Floats=0.5, cu
stom_head:Optional[nn.Module]=None,
                split_on:Optional[SplitFuncOrIdxList]=None, bn_final:bool=
False, init=nn.init.kaiming_normal_,
                concat_pool:bool=True, **kwargs:Any)->Learner:
    "Build convnet style learner."
    meta = cnn_config(base_arch)
    model = create_cnn_model(base_arch, data.c, cut, pretrained, lin_ftrs,
ps=ps, custom_head=custom_head,
        bn_final=bn_final, concat_pool=concat_pool)
    learn = Learner(data, model, **kwargs)
    learn.split(split_on or meta['split'])
    if pretrained: learn.freeze()
    if init: apply_init(model[1], init)
    return learn

def create_cnn(data, base_arch, **kwargs):
    warn("`create_cnn` is deprecated and is now named `cnn_learner`.")
    return cnn_learner(data, base_arch, **kwargs)

def unet_learner(data:DataBunch, arch:Callable, pretrained:bool=True, blur
_final:bool=True,
                 norm_type:Optional[NormType]=None, split_on:Optional[Spli
tFuncOrIdxList]=None, blur:bool=False,
                 self_attention:bool=False, y_range:Optional[Tuple[float,f
loat]]=None, last_cross:bool=True,
                 bottle:bool=False, cut:Union[int,Callable]=None, **learn_
kwargs:Any)->Learner:
    "Build Unet learner from `data` and `arch`."
    meta = cnn_config(arch)
    body = create_body(arch, pretrained, cut)
    try:    size = data.train_ds[0][0].size
    except: size = next(iter(data.train_dl))[0].shape[-2:]
    model = to_device(models.unet.DynamicUnet(body, n_classes=data.c, img_
size=size, blur=blur, blur_final=blur_final,
          self_attention=self_attention, y_range=y_range, norm_type=norm_t
ype, last_cross=last_cross,
          bottle=bottle), data.device)
    learn = Learner(data, model, **learn_kwargs)
    learn.split(ifnone(split_on, meta['split']))
    if pretrained: learn.freeze()
    apply_init(model[2], nn.init.kaiming_normal_)
    return learn
```

```
@classmethod
def _cl_int_from_learner(cls, learn:Learner, ds_type:DatasetType=DatasetTy
pe.Valid, activ:nn.Module=None, tta=False):
    "Create an instance of `ClassificationInterpretation`. `tta` indicates
if we want to use Test Time Augmentation."
    preds = learn.TTA(ds_type=ds_type, with_loss=True) if tta else learn.g
et_preds(ds_type=ds_type, activ=activ, with_loss=True)

    return cls(learn, *preds, ds_type=ds_type)

def _test_cnn(m):
    if not isinstance(m, nn.Sequential) or not len(m) == 2: return False
    return isinstance(m[1][0], (AdaptiveConcatPool2d, nn.AdaptiveAvgPool2d
))

def _cl_int_gradcam(self, idx, ds_type:DatasetType=DatasetType.Valid, heat
map_thresh:int=16, image:bool=True):
    m = self.learn.model.eval()
    im,cl = self.learn.data.dl(ds_type).dataset[idx]
    cl = int(cl)
    xb,_ = self.data.one_item(im, detach=False, denorm=False) #put into a
 minibatch of batch size = 1
    with hook_output(m[0]) as hook_a:
        with hook_output(m[0], grad=True) as hook_g:
            preds = m(xb)
            preds[0,int(cl)].backward()
    acts  = hook_a.stored[0].cpu() #activation maps
    if (acts.shape[-1]*acts.shape[-2]) >= heatmap_thresh:
        grad = hook_g.stored[0][0].cpu()
        grad_chan = grad.mean(1).mean(1)
        mult = F.relu(((acts*grad_chan[...,None,None])).sum(0))
        if image:
            xb_im = Image(xb[0])
            _,ax = plt.subplots()
            sz = list(xb_im.shape[-2:])
            xb_im.show(ax,title=f"pred. class: {self.pred_class[idx]}, act
ual class: {self.learn.data.classes[cl]}")
            ax.imshow(mult, alpha=0.4, extent=(0,*sz[::-1],0),
              interpolation='bilinear', cmap='magma')
        return mult

ClassificationInterpretation.GradCAM =_cl_int_gradcam

def _cl_int_plot_top_losses(self, k, largest=True, figsize=(12,12), heatma
p:bool=False, heatmap_thresh:int=16,
                            alpha:float=0.6, cmap:str="magma", show_text:b
ool=True,
                            return_fig:bool=None)->Optional[plt.Figure]:
    "Show images in `top_losses` along with their prediction, actual, los
s, and probability of actual class."
    assert not heatmap or _test_cnn(self.learn.model), "`heatmap=True` req
uires a model like `cnn_learner` produces."
```

```
    if heatmap is None: heatmap = _test_cnn(self.learn.model)
    tl_val,tl_idx = self.top_losses(k, largest)
    classes = self.data.classes
    cols = math.ceil(math.sqrt(k))
    rows = math.ceil(k/cols)
    fig,axes = plt.subplots(rows, cols, figsize=figsize)
    if show_text: fig.suptitle('Prediction/Actual/Loss/Probability', weigh
t='bold', size=14)
    for i,idx in enumerate(tl_idx):
        im,cl = self.data.dl(self.ds_type).dataset[idx]
        cl = int(cl)
        title = f'{classes[self.pred_class[idx]]}/{classes[cl]} / {self.lo
sses[idx]:.2f} / {self.preds[idx][cl]:.2f}' if show_text else None
        im.show(ax=axes.flat[i], title=title)
        if heatmap:
            mult = self.GradCAM(idx,self.ds_type,heatmap_thresh,image=Fals
e)
            if mult is not None:
                sz = list(im.shape[-2:])
                axes.flat[i].imshow(mult, alpha=alpha, extent=(0,*sz[::-1
],0), interpolation='bilinear', cmap=cmap)
    if ifnone(return_fig, defaults.return_fig): return fig

def _cl_int_plot_multi_top_losses(self, samples:int=3, figsize:Tuple[int,i
nt]=(8,8), save_misclassified:bool=False):
    "Show images in `top_losses` along with their prediction, actual, los
s, and probability of predicted class in a multilabeled dataset."
    if samples >20:
        print("Max 20 samples")
        return
    losses, idxs = self.top_losses(self.data.c)
    l_dim = len(losses.size())
    if l_dim == 1: losses, idxs = self.top_losses()
    infolist, ordlosses_idxs, mismatches_idxs, mismatches, losses_mismatch
es, mismatchescontainer = [],[],[],[],[],[]
    truthlabels = np.asarray(self.y_true, dtype=int)
    classes_ids = [k for k in enumerate(self.data.classes)]
    predclass = np.asarray(self.pred_class)
    for i,pred in enumerate(predclass):
        where_truth = np.nonzero((truthlabels[i]>0))[0]
        mismatch = np.all(pred!=where_truth)
        if mismatch:
            mismatches_idxs.append(i)
            if l_dim > 1 : losses_mismatches.append((losses[i][pred], i))
            else: losses_mismatches.append((losses[i], i))
        if l_dim > 1: infotup = (i, pred, where_truth, losses[i][pred], np
.round(self.preds[i], decimals=3)[pred], mismatch)
        else: infotup = (i, pred, where_truth, losses[i], np.round(self.pr
eds[i], decimals=3)[pred], mismatch)
        infolist.append(infotup)
    ds = self.data.dl(self.ds_type).dataset
    mismatches = ds[mismatches_idxs]
```

github fastai2 xresnet.py

https://github.com/fastai/fastai2/blob/master/fastai2/vision/models/xresnet.py (https://github.com/fastai/fastai2/blob/master/fastai2/vision/models/xresnet.py)

```
__all__ = ['init_cnn', 'XResNet', 'xresnet18', 'xresnet34', 'xresnet50',
'xresnet101', 'xresnet152', 'xresnet18_deep',
           'xresnet34_deep', 'xresnet50_deep', 'xresnet18_deeper', 'xresne
t34_deeper', 'xresnet50_deeper', 'se_kwargs1',
           'se_kwargs2', 'se_kwargs3', 'g0', 'g1', 'g2', 'g3', 'xse_resnet
18', 'xse_resnext18', 'xresnext18',
           'xse_resnet34', 'xse_resnext34', 'xresnext34', 'xse_resnet50',
'xse_resnext50', 'xresnext50',
           'xse_resnet101', 'xse_resnext101', 'xresnext101', 'xse_resnet15
2', 'xsenet154', 'xse_resnext18_deep',
           'xse_resnext34_deep', 'xse_resnext50_deep', 'xse_resnext18_deep
er', 'xse_resnext34_deeper',
           'xse_resnext50_deeper']

# Cell
from ...torch_basics import *
from torchvision.models.utils import load_state_dict_from_url

# Cell
def init_cnn(m):
    if getattr(m, 'bias', None) is not None: nn.init.constant_(m.bias, 0)
    if isinstance(m, (nn.Conv2d,nn.Linear)): nn.init.kaiming_normal_(m.wei
ght)
    for l in m.children(): init_cnn(l)

# Cell
class XResNet(nn.Sequential):
    @delegates(ResBlock)
    def __init__(self, block, expansion, layers, p=0.0, c_in=3, n_out=1000
, stem_szs=(32,32,64),
                 widen=1.0, sa=False, act_cls=defaults.activation, **kwarg
s):
        store_attr(self, 'block,expansion,act_cls')
        stem_szs = [c_in, *stem_szs]
        stem = [ConvLayer(stem_szs[i], stem_szs[i+1], stride=2 if i==0 els
e 1, act_cls=act_cls)
                for i in range(3)]

        block_szs = [int(o*widen) for o in [64,128,256,512] +[256]*(len(la
yers)-4)]
        block_szs = [64//expansion] + block_szs
        blocks    = self._make_blocks(layers, block_szs, sa, **kwargs)

        super().__init__(
            *stem, nn.MaxPool2d(kernel_size=3, stride=2, padding=1),
            *blocks,
            nn.AdaptiveAvgPool2d(1), Flatten(), nn.Dropout(p),
            nn.Linear(block_szs[-1]*expansion, n_out),
        )
        init_cnn(self)

    def _make_blocks(self, layers, block_szs, sa, **kwargs):
```

```
        return [self._make_layer(ni=block_szs[i], nf=block_szs[i+1], block
s=l,
                                 stride=1 if i==0 else 2, sa=sa and i==len
(layers)-4, **kwargs)
                for i,l in enumerate(layers)]

    def _make_layer(self, ni, nf, blocks, stride, sa, **kwargs):
        return nn.Sequential(
            *[self.block(self.expansion, ni if i==0 else nf, nf, stride=st
ride if i==0 else 1,
                         sa=sa and i==(blocks-1), act_cls=self.act_cls, **kwa
rgs)
              for i in range(blocks)])

# Cell
def _xresnet(pretrained, expansion, layers, **kwargs):
    # TODO pretrain all sizes. Currently will fail with non-xrn50
    url = 'https://s3.amazonaws.com/fast-ai-modelzoo/xrn50_940.pth'
    res = XResNet(ResBlock, expansion, layers, **kwargs)
    if pretrained: res.load_state_dict(load_state_dict_from_url(url, map_l
ocation='cpu')['model'], strict=False)
    return res

def xresnet18 (pretrained=False, **kwargs): return _xresnet(pretrained, 1,
[2, 2,  2, 2], **kwargs)
def xresnet34 (pretrained=False, **kwargs): return _xresnet(pretrained, 1,
[3, 4,  6, 3], **kwargs)
def xresnet50 (pretrained=False, **kwargs): return _xresnet(pretrained, 4,
[3, 4,  6, 3], **kwargs)
def xresnet101(pretrained=False, **kwargs): return _xresnet(pretrained, 4,
[3, 4, 23, 3], **kwargs)
def xresnet152(pretrained=False, **kwargs): return _xresnet(pretrained, 4,
[3, 8, 36, 3], **kwargs)
def xresnet18_deep  (pretrained=False, **kwargs): return _xresnet(pretrain
ed, 1, [2,2,2,2,1,1], **kwargs)
def xresnet34_deep  (pretrained=False, **kwargs): return _xresnet(pretrain
ed, 1, [3,4,6,3,1,1], **kwargs)
def xresnet50_deep  (pretrained=False, **kwargs): return _xresnet(pretrain
ed, 4, [3,4,6,3,1,1], **kwargs)
def xresnet18_deeper(pretrained=False, **kwargs): return _xresnet(pretrain
ed, 1, [2,2,1,1,1,1,1,1], **kwargs)
def xresnet34_deeper(pretrained=False, **kwargs): return _xresnet(pretrain
ed, 1, [3,4,6,3,1,1,1,1], **kwargs)
def xresnet50_deeper(pretrained=False, **kwargs): return _xresnet(pretrain
ed, 4, [3,4,6,3,1,1,1,1], **kwargs)

# Cell
se_kwargs1 = dict(groups=1 , reduction=16)
se_kwargs2 = dict(groups=32, reduction=16)
se_kwargs3 = dict(groups=32, reduction=0)
g0 = [2,2,2,2]
g1 = [3,4,6,3]
```

```
g2 = [3,4,23,3]
g3 = [3,8,36,3]

# Cell
def xse_resnet18(n_out=1000, pretrained=False, **kwargs):   return XResNet
(SEBlock,  1, g0, n_out=n_out, **se_kwargs1, **kwargs)
def xse_resnext18(n_out=1000, pretrained=False, **kwargs):  return XResNet
(SEResNeXtBlock, 1, g0, n_out=n_out, **se_kwargs2, **kwargs)
def xresnext18(n_out=1000, pretrained=False, **kwargs):     return XResNet
(SEResNeXtBlock, 1, g0, n_out=n_out, **se_kwargs3, **kwargs)
def xse_resnet34(n_out=1000, pretrained=False, **kwargs):   return XResNet
(SEBlock,  1, g1, n_out=n_out, **se_kwargs1, **kwargs)
def xse_resnext34(n_out=1000, pretrained=False, **kwargs):  return XResNet
(SEResNeXtBlock, 1, g1, n_out=n_out, **se_kwargs2, **kwargs)
def xresnext34(n_out=1000, pretrained=False, **kwargs):     return XResNet
(SEResNeXtBlock, 1, g1, n_out=n_out, **se_kwargs3, **kwargs)
def xse_resnet50(n_out=1000, pretrained=False, **kwargs):   return XResNet
(SEBlock,  4, g1, n_out=n_out, **se_kwargs1, **kwargs)
def xse_resnext50(n_out=1000, pretrained=False, **kwargs):  return XResNet
(SEResNeXtBlock, 4, g1, n_out=n_out, **se_kwargs2, **kwargs)
def xresnext50(n_out=1000, pretrained=False, **kwargs):     return XResNet
(SEResNeXtBlock, 4, g1, n_out=n_out, **se_kwargs3, **kwargs)
def xse_resnet101(n_out=1000, pretrained=False, **kwargs):  return XResNet
(SEBlock,  4, g2, n_out=n_out, **se_kwargs1, **kwargs)
def xse_resnext101(n_out=1000, pretrained=False, **kwargs): return XResNet
(SEResNeXtBlock, 4, g2, n_out=n_out, **se_kwargs2, **kwargs)
def xresnext101(n_out=1000, pretrained=False, **kwargs):    return XResNet
(SEResNeXtBlock, 4, g2, n_out=n_out, **se_kwargs3, **kwargs)
def xse_resnet152(n_out=1000, pretrained=False, **kwargs):  return XResNet
(SEBlock,  4, g3, n_out=n_out, **se_kwargs1, **kwargs)
def xsenet154(n_out=1000, pretrained=False, **kwargs):
    return XResNet(SEBlock, g3, groups=64, reduction=16, p=0.2, n_out=n_ou
t)
def xse_resnext18_deep  (n_out=1000, pretrained=False, **kwargs):  return
XResNet(SEResNeXtBlock, 1, g0+[1,1], n_out=n_out, **se_kwargs2, **kwargs)
def xse_resnext34_deep  (n_out=1000, pretrained=False, **kwargs):  return
XResNet(SEResNeXtBlock, 1, g1+[1,1], n_out=n_out, **se_kwargs2, **kwargs)
def xse_resnext50_deep  (n_out=1000, pretrained=False, **kwargs):  return
XResNet(SEResNeXtBlock, 4, g1+[1,1], n_out=n_out, **se_kwargs2, **kwargs)
def xse_resnext18_deeper(n_out=1000, pretrained=False, **kwargs):  return
XResNet(SEResNeXtBlock, 1, [2,2,1,1,1,1,1,1], n_out=n_out, **se_kwargs2, *
*kwargs)
def xse_resnext34_deeper(n_out=1000, pretrained=False, **kwargs):  return
XResNet(SEResNeXtBlock, 1, [3,4,4,2,2,1,1,1], n_out=n_out, **se_kwargs2, *
*kwargs)
def xse_resnext50_deeper(n_out=1000, pretrained=False, **kwargs):  return
XResNet(SEResNeXtBlock, 4, [3,4,4,2,2,1,1,1], n_out=n_out, **se_kwargs2, *
*kwargs)
```

fastai pretrained cadene_models

https://github.com/fastai/fastai/blob/master/fastai/vision/models/cadene_mode
(https://github.com/fastai/fastai/blob/master/fastai/vision/models/cadene_mode

```
#These models are dowloaded via the repo https://github.com/Cadene/pretrai
ned-models.pytorch
#See licence here: https://github.com/Cadene/pretrained-models.pytorch/blo
b/master/LICENSE.txt
from torch import nn
from ..learner import model_meta
from ...core import *

pretrainedmodels = try_import('pretrainedmodels')
if not pretrainedmodels:
    raise Exception('Error: `pretrainedmodels` is needed. `pip install pre
trainedmodels`')

__all__ = ['inceptionv4', 'inceptionresnetv2', 'nasnetamobile', 'dpn92',
'xception_cadene', 'se_resnet50',
           'se_resnet101', 'se_resnext50_32x4d', 'senet154', 'pnasnet5larg
e']

def get_model(model_name:str, pretrained:bool, seq:bool=False, pname:str=
'imagenet', **kwargs):
    pretrained = pname if pretrained else None
    model = getattr(pretrainedmodels, model_name)(pretrained=pretrained, *
*kwargs)
    return nn.Sequential(*model.children()) if seq else model

def inceptionv4(pretrained:bool=False):
    model = get_model('inceptionv4', pretrained)
    all_layers = list(model.children())
    return nn.Sequential(*all_layers[0], *all_layers[1:])
model_meta[inceptionv4] = {'cut': -2, 'split': lambda m: (m[0][11], m[1])}

def nasnetamobile(pretrained:bool=False):
    model = get_model('nasnetamobile', pretrained, num_classes=1000)
    model.logits = noop
    return nn.Sequential(model)
model_meta[nasnetamobile] = {'cut': noop, 'split': lambda m: (list(m[0][0]
.children())[8], m[1])}

def pnasnet5large(pretrained:bool=False):
    model = get_model('pnasnet5large', pretrained, num_classes=1000)
    model.logits = noop
    return nn.Sequential(model)
model_meta[pnasnet5large] = {'cut': noop, 'split': lambda m: (list(m[0][0]
.children())[8], m[1])}

def inceptionresnetv2(pretrained:bool=False):   return get_model('inceptio
nresnetv2', pretrained, seq=True)
def dpn92(pretrained:bool=False):               return get_model('dpn92',
pretrained, pname='imagenet+5k', seq=True)
def xception_cadene(pretrained=False):          return get_model('xceptio
n', pretrained, seq=True)
def se_resnet50(pretrained:bool=False):         return get_model('se_resne
```

```
t50', pretrained)
def se_resnet101(pretrained:bool=False):        return get_model('se_resne
t101', pretrained)
def se_resnext50_32x4d(pretrained:bool=False):  return get_model('se_resne
xt50_32x4d', pretrained)
def se_resnext101_32x4d(pretrained:bool=False): return get_model('se_resne
xt101_32x4d', pretrained)
def senet154(pretrained:bool=False):            return get_model('senet15
4', pretrained)

model_meta[inceptionresnetv2] = {'cut': -2, 'split': lambda m: (m[0][9],
m[1])}
model_meta[dpn92]             = {'cut': -1, 'split': lambda m: (m[0][0][16
], m[1])}
model_meta[xception_cadene]   = {'cut': -1, 'split': lambda m: (m[0][11],
m[1])}
model_meta[senet154]          = {'cut': -3, 'split': lambda m: (m[0][3],
m[1])}
_se_resnet_meta               = {'cut': -2, 'split': lambda m: (m[0][3],
m[1])}
model_meta[se_resnet50]         = _se_resnet_meta
model_meta[se_resnet101]        = _se_resnet_meta
model_meta[se_resnext50_32x4d]  = _se_resnet_meta
model_meta[se_resnext101_32x4d] = _se_resnet_meta

# TODO: add "resnext101_32x4d" "resnext101_64x4d" after serialization issu
e is fixed:
# https://github.com/Cadene/pretrained-models.pytorch/pull/128
```