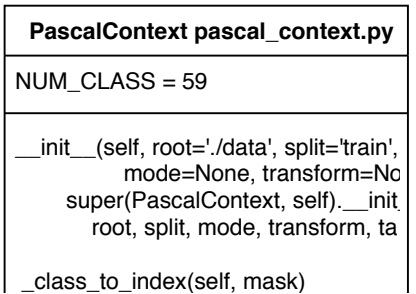
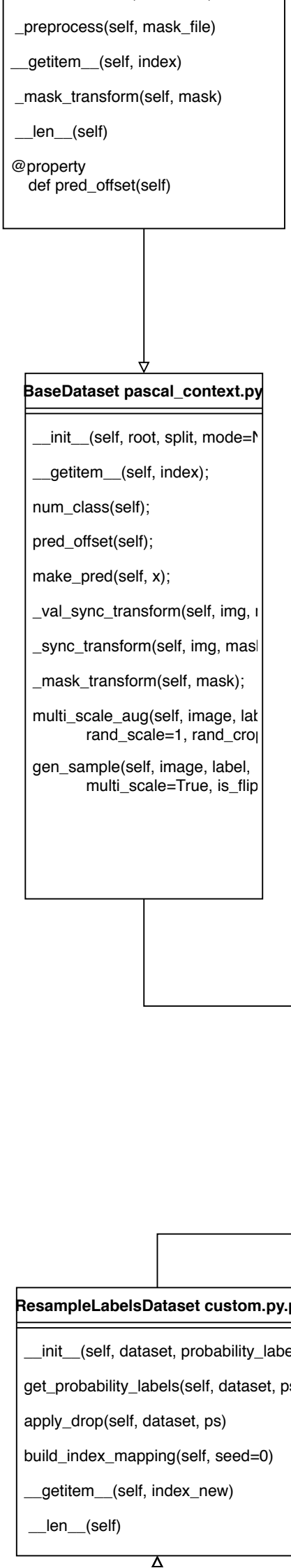

PascalContext pascal_context.py
NUM_CLASS = 59
__init__(self, root='./data', split='train',
mode=None, transform=No
super(PascalContext, self).__init
root, split, mode, transform, ta
_class_to_index(self, mask)

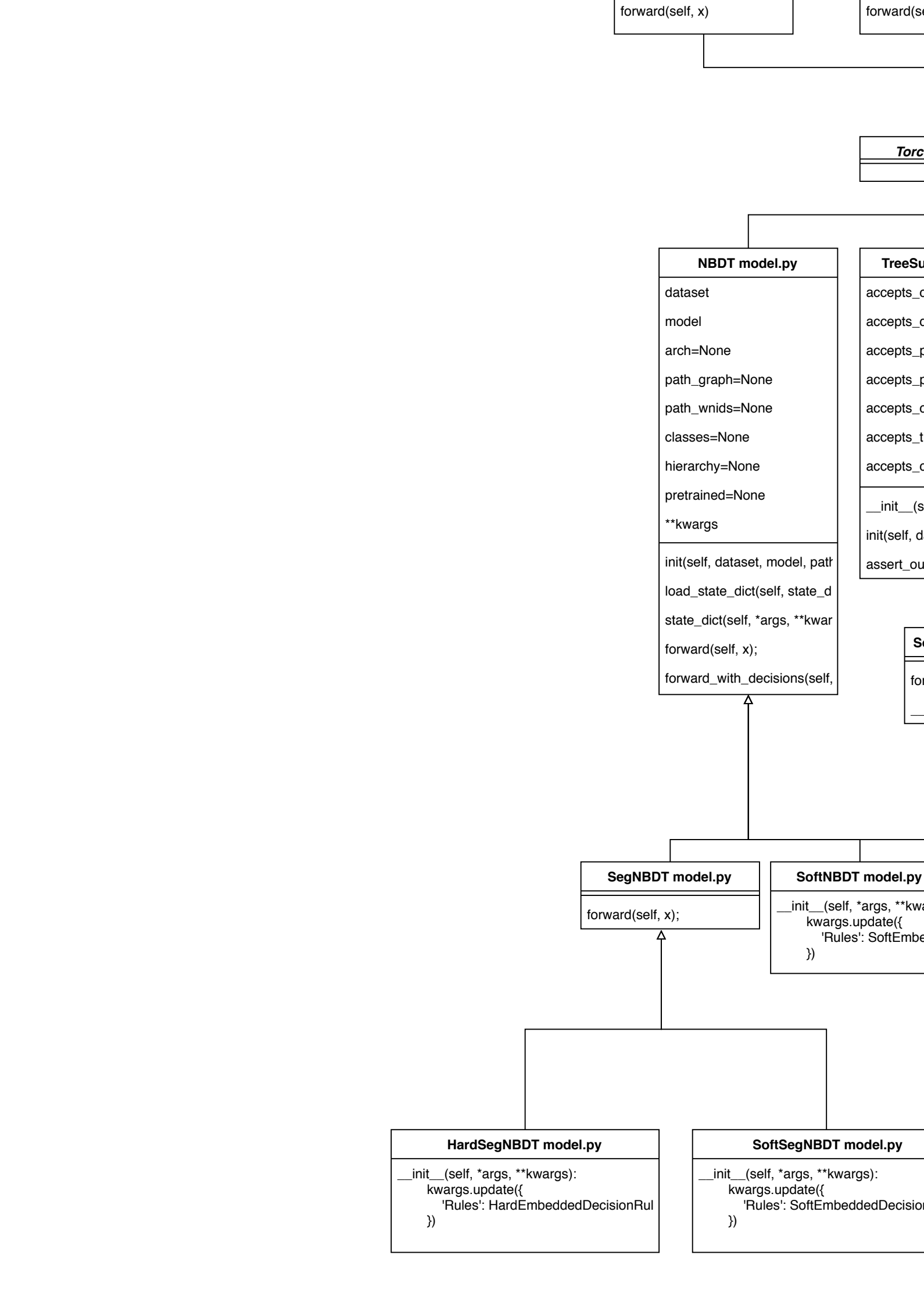
BasicBlock resnet.py
expansion = 1
__init__(self, in_planes, plan

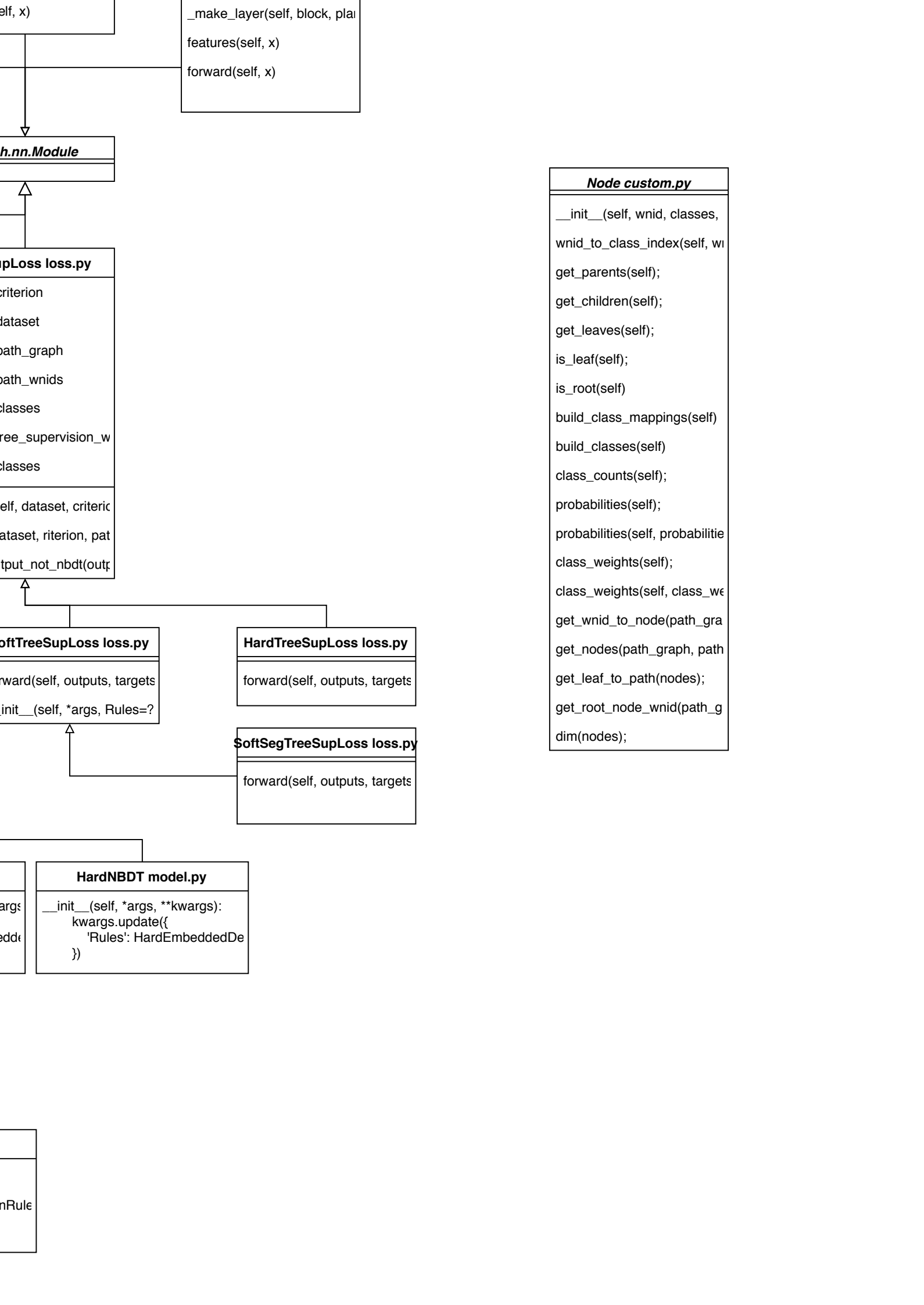
Bottlen
expansion
__init__(s

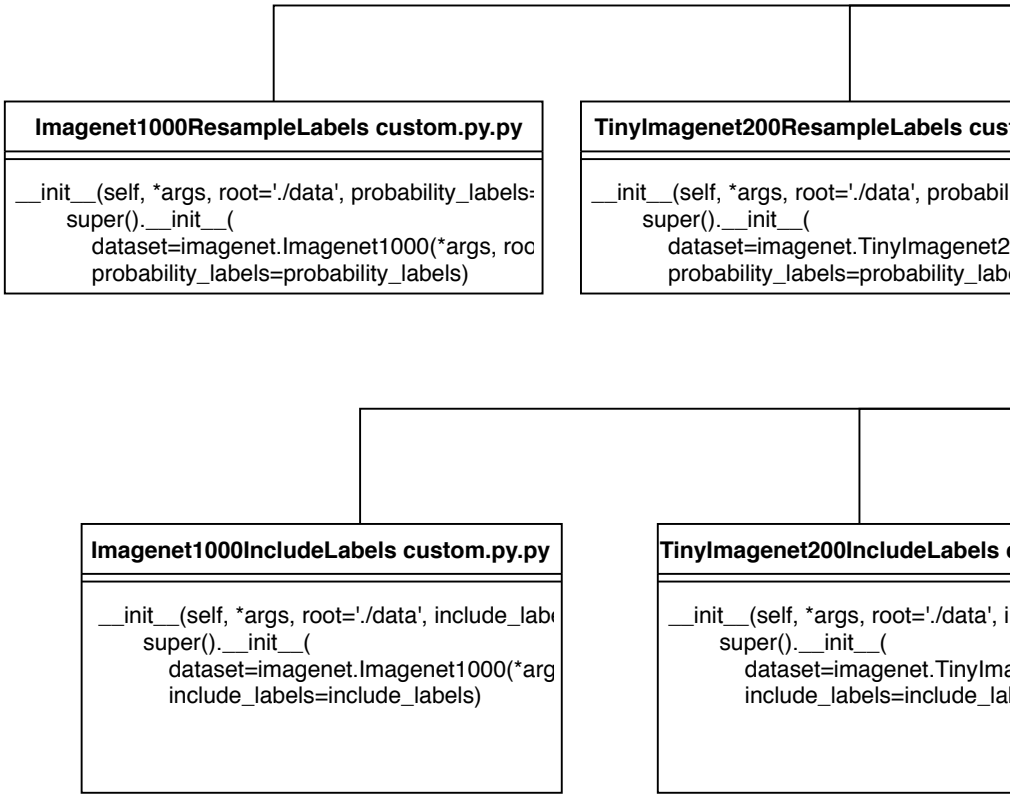

neck resnet.py
n = 4
elf, in_planes, plan
ResNet resnet.py
__init__(self, block, num_bl
super(ResNet, self).__i
self.in_planes = 64

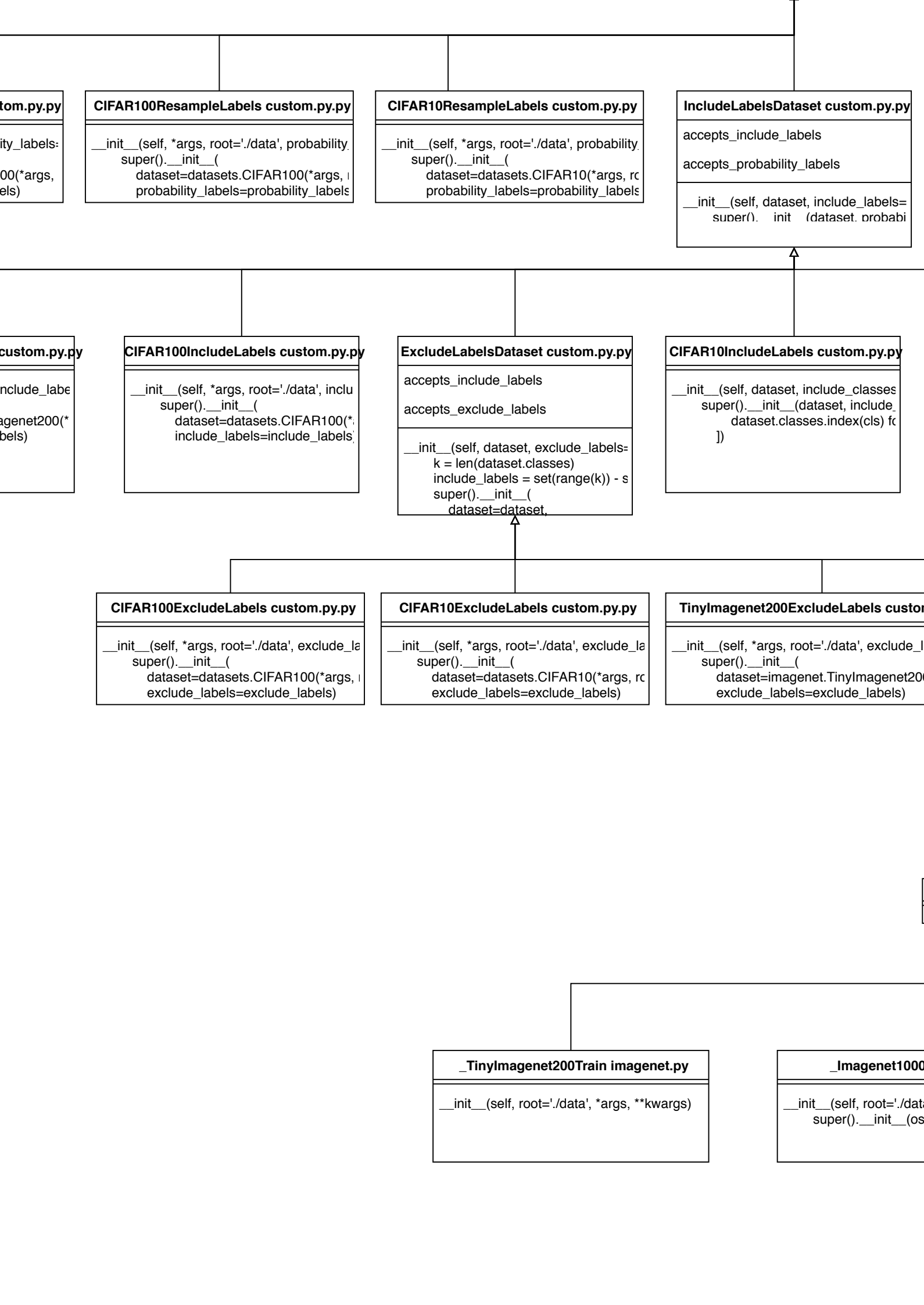

_preprocess(self, mask_file)
__getitem__(self, index)
_mask_transform(self, mask)
__len__(self)
@property
def pred_offset(self)
BaseDataset pascal_context.py
__init__(self, root, split, mode=N
__getitem__(self, index);
num_class(self);
pred_offset(self);
make_pred(self, x);
_val_sync_transform(self, img, r
_sync_transform(self, img, mas
_mask_transform(self, mask);
multi_scale_aug(self, image, lat
rand_scale=1, rand_cro
gen_sample(self, image, label,
multi_scale=True, is_flip
ResampleLabelsDataset custom.py.py
__init__(self, dataset, probability_labe
get_probability_labels(self, dataset, ps
apply_drop(self, dataset, ps)
build_index_mapping(self, seed=0)
__getitem__(self, index_new)
__len__(self)

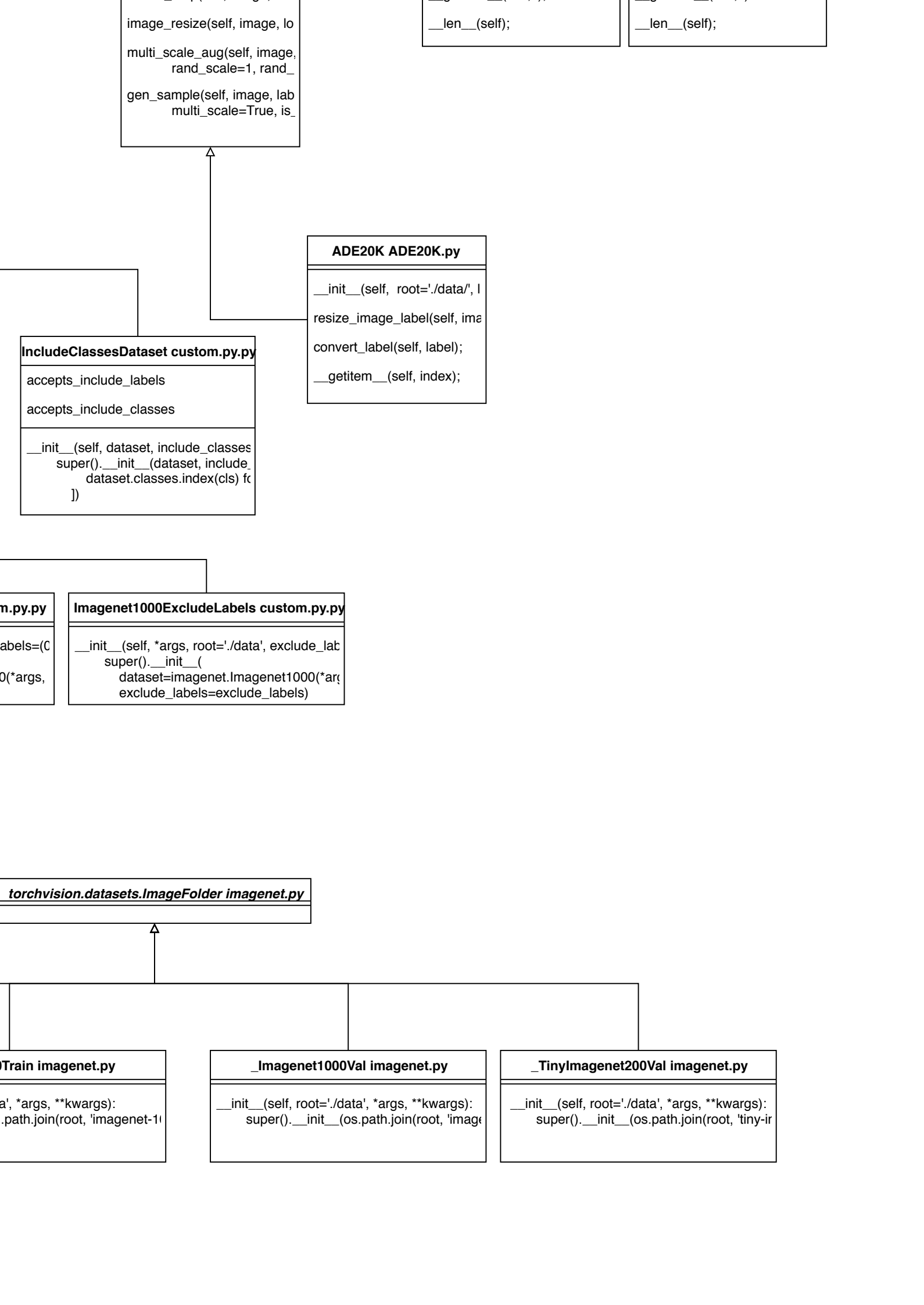

LookIntoPerson lip.py
__init__(self, root='./data/', list_
read_files(self);
resize_image(self, image, label,
__getitem__(self, index);
BaseDataset lip.py
__init__(self,
ignore_label=-1
__len__(self)
input_transform(self, image)
label_transform(self, label)
pad_image(self, image, h, w, siz
rand_crop(self, image, label)
center_crop(self, image, label)
image_resize(self, image, long_
multi_scale_aug(self, image, lab
rand_scale=1, rand_cro
gen_sample(self, image, label,
multi_scale=True, is_flip
torch.utils.data.Dataset
BaseDataset ADE20K.py
__init__(self,
ignore_label=-1,
__len__(self)
input_transform(self, image)
label_transform(self, label)
pad_image(self, image, h, w
rand_crop(self, image, label)
center_crop(self, image, labe
TinyImagenet200 imagenet.py
url = 'http://cs231n.stanford.edu
dataset = None
__init__(self, root='./data', *args
transform_train(input_size=64);
transform_val(input_size=-1);
download(self, root='./');
__getitem__(self, i);
Imagenet1000 imagenet.py
__init__(self, root='./data',
*args, train=True, downl
download(self, root='./')
@staticmethod
def transform_train(input_siz
@staticmethod
def transform_val(input_size=
__getitem__(self, i)

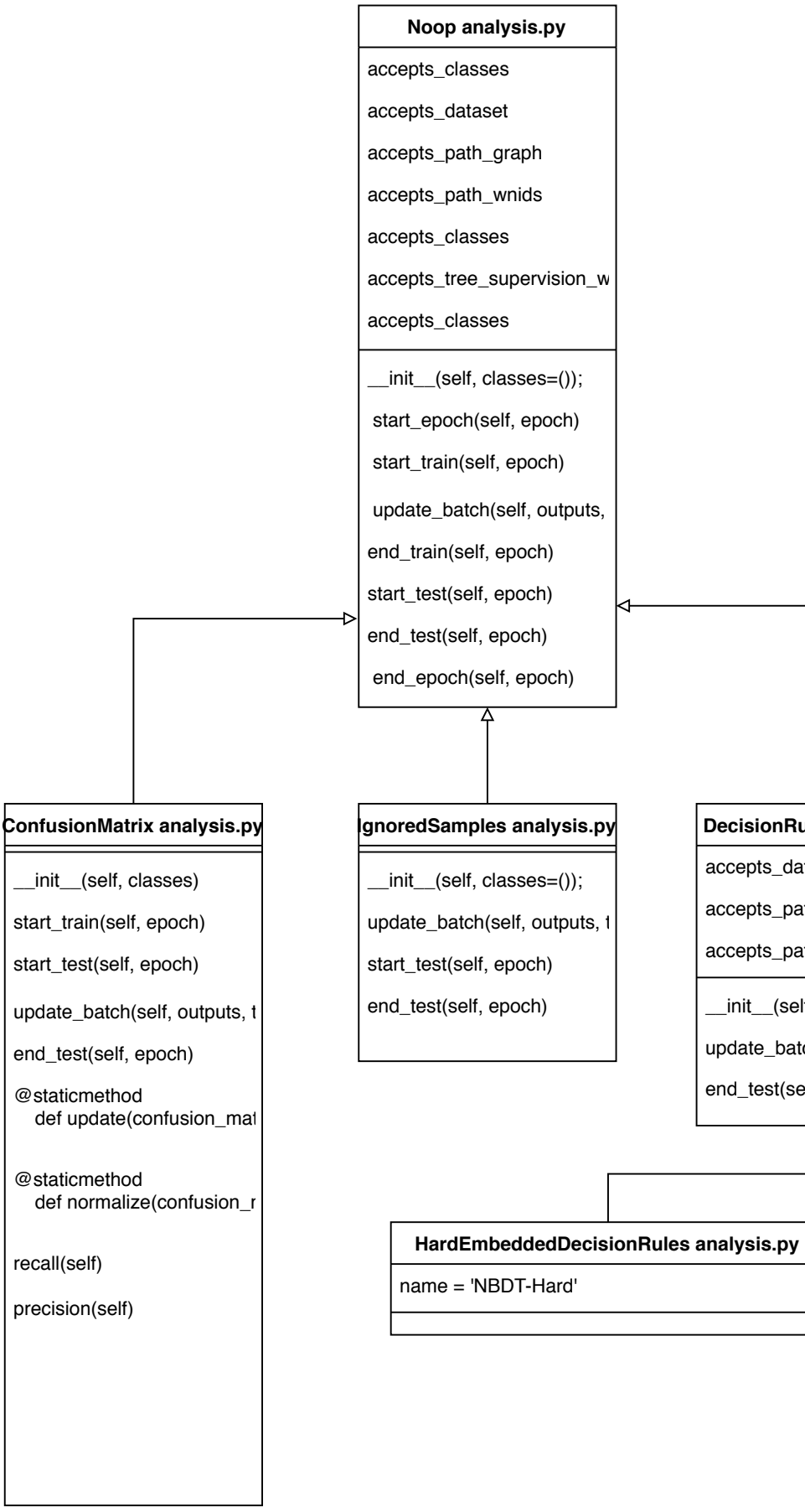

forward(self, x)
forward(se
Torc
NBDT model.py
dataset
model
arch=None
path_graph=None
path_wnids=None
classes=None
hierarchy=None
pretrained=None
**kwargs
init(self, dataset, model, patl
load_state_dict(self, state_d
state_dict(self, *args, **kwar
forward(self, x);
forward_with_decisions(self,
TreeSu
accepts_c
accepts_c
accepts_p
accepts_p
accepts_c
accepts_t
accepts_c
__init__(s
init(self, d
assert_ou
Sc
for
SegNBDT model.py
forward(self, x);
SoftNBDT model.py
__init__(self, *args, **kwa
kwargs.update({
'Rules': SoftEmbe
})
HardSegNBDT model.py
__init__(self, *args, **kwargs):
kwargs.update({
'Rules': HardEmbeddedDecisionRul
})
SoftSegNBDT model.py
__init__(self, *args, **kwargs):
kwargs.update({
'Rules': SoftEmbeddedDecisio
})

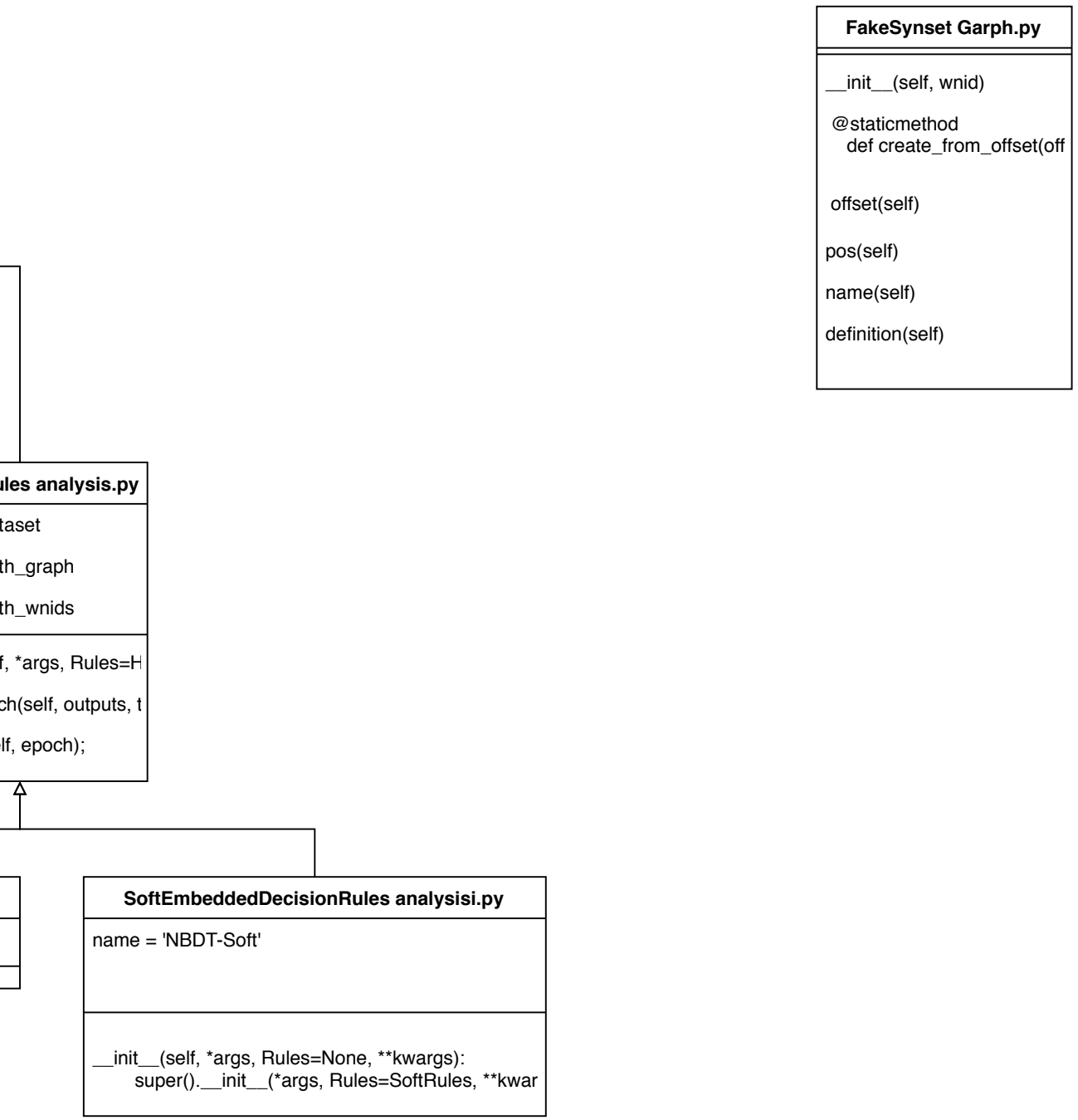

elf, x)
_make_layer(self, block, pla
features(self, x)
forward(self, x)
h.nn.Module
Node custom.py
__init__(self, wnid, classes,
wnid_to_class_index(self, wi
get_parents(self);
get_children(self);
get_leaves(self);
is_leaf(self);
is_root(self)
build_class_mappings(self)
build_classes(self)
class_counts(self);
probabilities(self);
probabilities(self, probabilitie
class_weights(self);
class_weights(self, class_we
get_wnid_to_node(path_gra
get_nodes(path_graph, path
get_leaf_to_path(nodes);
get_root_node_wnid(path_g
dim(nodes);
pLoss loss.py
criterion
dataset
path_graph
path_wnids
classes
ree_supervision_w
classes
elf, dataset, criteric
ataset, riterion, pat
tput_not_nbdt(outp
oftTreeSupLoss loss.py
rward(self, outputs, targets
init__(self, *args, Rules=?
HardTreeSupLoss loss.py
forward(self, outputs, targets
SoftSegTreeSupLoss loss.py
forward(self, outputs, targets
HardNBDT model.py
__init__(self, *args, **kwargs):
kwargs.update({
'Rules': HardEmbeddedDe
})
nRule


Imagenet1000ResampleLabels custom.py.py
__init__(self, *args, root='./data', probability_labels=
super().__init__(
dataset=imagenet.Imagenet1000(*args, roo
probability_labels=probability_labels)
TinyImagenet200ResampleLabels cust
__init__(self, *args, root='./data', probabil
super().__init__(
dataset=imagenet.TinyImagenet2
probability_labels=probability_lab
Imagenet1000IncludeLabels custom.py.py
__init__(self, *args, root='./data', include_labe
super().__init__(
dataset=imagenet.Imagenet1000(*arg
include_labels=include_labels)
TinyImagenet200IncludeLabels
__init__(self, *args, root='./data', i
super().__init__(
dataset=imagenet.TinyIma
include_labels=include_la


tom.py.py
ity_labels=
00(*args,
els)
CIFAR100ResampleLabels custom.py.py
__init__(self, *args, root='./data', probability
super().__init__(
dataset=datasets.CIFAR100(*args,
probability_labels=probability_labels
CIFAR10ResampleLabels custom.py.py
__init__(self, *args, root='./data', probability
super().__init__(
dataset=datasets.CIFAR10(*args, rc
probability_labels=probability_labels
IncludeLabelsDataset custom.py.py
accepts_include_labels
accepts_probability_labels
__init__(self, dataset, include_labels=
super(). init (dataset, probabi
custom.py.py
nclude_labe
agenet200(*
bels)
CIFAR100IncludeLabels custom.py.py
__init__(self, *args, root='./data', inclu
super().__init__(
dataset=datasets.CIFAR100(*
include_labels=include_labels
ExcludeLabelsDataset custom.py.py
accepts_include_labels
accepts_exclude_labels
__init__(self, dataset, exclude_labels=
k = len(dataset.classes)
include_labels = set(range(k)) - s
super().__init__(
dataset=dataset,
CIFAR10IncludeLabels custom.py.py
__init__(self, dataset, include_classes
super().__init__(dataset, include_
dataset.classes.index(cls) fc
])
CIFAR100ExcludeLabels custom.py.py
__init__(self, *args, root='./data', exclude_la
super().__init__(
dataset=datasets.CIFAR100(*args,
exclude_labels=exclude_labels)
CIFAR10ExcludeLabels custom.py.py
__init__(self, *args, root='./data', exclude_la
super().__init__(
dataset=datasets.CIFAR10(*args, rc
exclude_labels=exclude_labels)
TinyImagenet200ExcludeLabels custon
__init__(self, *args, root='./data', exclude_l
super().__init__(
dataset=imagenet.TinyImagenet20
exclude_labels=exclude_labels)
_TinyImagenet200Train imagenet.py
__init__(self, root='./data', *args, **kwargs)
_Imagenet1000
__init__(self, root='./data
super().__init__(os


__len__(self);
__len__(self);
image_resize(self, image, lo
multi_scale_aug(self, image,
rand_scale=1, rand_
gen_sample(self, image, lab
multi_scale=True, is_
ADE20K ADE20K.py
__init__(self, root='./data/', l
resize_image_label(self, ima
convert_label(self, label);
__getitem__(self, index);
IncludeClassesDataset custom.py.py
accepts_include_labels
accepts_include_classes
__init__(self, dataset, include_classes
super().__init__(dataset, include_
dataset.classes.index(cls) fo
])
m.py.py
abels=(C
0(*args,
Imagenet1000ExcludeLabels custom.py.py
__init__(self, *args, root='./data', exclude_lab
super().__init__(
dataset=imagenet.Imagenet1000(*arg
exclude_labels=exclude_labels)
torchvision.datasets.ImageFolder imagenet.py
Train imagenet.py
a', *args, **kwargs):
.path.join(root, 'imagenet-1
_Imagenet1000Val imagenet.py
__init__(self, root='./data', *args, **kwargs):
super().__init__(os.path.join(root, 'image
_TinyImagenet200Val imagenet.py
__init__(self, root='./data', *args, **kwargs):
super().__init__(os.path.join(root, 'tiny-ir


Noop analysis.py
accepts_classes
accepts_dataset
accepts_path_graph
accepts_path_wnids
accepts_classes
accepts_tree_supervision_w
accepts_classes
__init__(self, classes=());
start_epoch(self, epoch)
start_train(self, epoch)
update_batch(self, outputs,
end_train(self, epoch)
start_test(self, epoch)
end_test(self, epoch)
end_epoch(self, epoch)
ConfusionMatrix analysis.py
__init__(self, classes)
start_train(self, epoch)
start_test(self, epoch)
update_batch(self, outputs, t
end_test(self, epoch)
@staticmethod
def update(confusion_mat
@staticmethod
def normalize(confusion_r
recall(self)
precision(self)
gnoredSamples analysis.py
__init__(self, classes=());
update_batch(self, outputs, t
start_test(self, epoch)
end_test(self, epoch)
DecisionRu
accepts_da
accepts_pa
accepts_pa
__init__(sel
update_bat
end_test(se
HardEmbeddedDecisionRules analysis.py
name = 'NBDT-Hard'


FakeSynset Garph.py
__init__(self, wnid)
@staticmethod
def create_from_offset(off
offset(self)
pos(self)
name(self)
definition(self)
ules analysis.py
taset
th_graph
th_wnids
f, *args, Rules=H
ch(self, outputs, t
lf, epoch);
SoftEmbeddedDecisionRules analysisi.py
name = 'NBDT-Soft'
__init__(self, *args, Rules=None, **kwargs):
super().__init__(*args, Rules=SoftRules, **kwar