Please install onnx and onnx2keras: <https://github.com/hzhexuan/onnx2keras>

Class ImageClassifier(train\_input, train\_target, test\_input, test\_target, label\_smooth=0, cutout = 16, save="EXP")

#use DARTS (<https://arxiv.org/abs/1806.09055>) to search a neural network for a given dataset

Attributes:

* train\_input: numpy array with shape (num\_samples, H, W, channels)
* train\_target: numpy array with shape (num\_samples, num\_classes) which is supposed to be one-hot encoded
* test\_input: numpy array with shape (num\_samples, H, W, channels)
* test\_target: numpy array with shape (num\_samples, num\_classes) which is supposed to be one-hot encoded
* label\_smooth: parameters for label smooth when calculating cross-entropy loss, default = 0
* cutout: size of cutout windows for image preprocessing, 0 correspond to absence of cutout, default = 16
* save: name of save folder, default = “EXP”
* genotype: genotype of architecture found, obtained after calling ImageClassifier.run()
* finalfit\_init\_channels: number of channels used in final\_fit stage
* finalfit\_layers: number of layers used in final\_fit stage

Methods:

1. fit(seed=0, gpu=0, lr=0.025, weight\_decay=3e-4, momentum=0.9,

batch\_size=64, epochs=50, learning\_rate\_min=0.001, arch\_learning\_rate=3e-4,

arch\_weight\_decay=1e-3, unrolled=True, report\_freq=50, grad\_clip=5,

method="DARTS", num\_reduction=2, init\_channels\_search=16, num\_layers\_search = 8)

#Neural Architecture Search stage

Parameters

* seed: random seed, default = 0
* gpu: gpu to use, only support single gpu mode, default = 0
* lr: initialize learning rate for parameters in each operations, default = 0.025
* weight\_decay: weight decay for parameters in each operations, default = 3e-4
* momentum: momentum for SGD optimizer when training operations, default = 0.9
* batch\_size: default = 64
* epochs: epochs for architecture search stage, default = 50
* learning\_rate\_min: lr is decreased following a cosine annealing schedule, learning\_rate\_min represent the minimum value for lr, default = 0.001
* arch\_learning\_rate: learning rate for architecture parameters, default = 3e-4
* arch\_weight\_decay: weight decay for architecture parameters, default = 1e-3
* unrolled: use one-step unrolled validation loss or not, as described in original paper of DARTS, default = True
* report\_freq: default = 50
* grad\_clip: gradient clipping by L2 norm, default = 5
* method: method to use, 3 choices are provided: DARTS, SF and RAM. Default = “DARTS”
* num\_reduction: number of reduction cells, default = 2
* init\_channels\_search: number of channels used in first convolution during architecture search stage, default = 16
* num\_layers\_search: number of layers used during architecture search stage, default = 8

Suggestions for choosing hyper-parameters:

* DARTS: use default hyper-parameters (e.g. call ImageClassifier.fit() when using DARTS)
* RAM or SF: use batch\_size = 512, epochs = 500 and arch\_learning\_rate = 3e-2 (e.g. call ImageClassifier.fit(method = “RAM” (Resp. “SF”), batch\_size = 512, epochs = 500, arch\_learning\_rate = 3e-2) when using RAM (Resp. SF))

1. finalfit(genotype=None, seed=0, gpu=0, init\_channels=36, layers=20,

lr=0.025, momentum=0.9, weight\_decay=3e-4, batch\_size=96,

epochs=600, drop\_path\_prob=0.2, auxiliary=True,

auxiliary\_weight=0.4, grad\_clip=5, report\_freq=50, num\_reduction=2)

#Final fit the model found in Neural Architecture Search stage

Parameters:

* genotype: genotype of architecture to use in finalfit stage, default = None which correspond to use genotype found during ImageClassifier.fit()
* seed: random seed, default = 0
* gpu: gpu to use, only support single gpu mode, default = 0
* init\_channels: number of channels used in first convolution during finalfit, default = 36
* layers: number of layers used during finalfit, default = 20
* lr: initialize learning rate for parameters in each operations, default = 0.025
* momentum: momentum for SGD optimizer when training operations, default = 0.9
* weight\_decay: weight decay for parameters in each operations, default = 3e-4
* batch\_size: default = 96
* epochs: epochs of finalfit stage, default = 600
* drop\_path\_prob: each path is dropped with a probability equal to drop\_path\_prob during the training, default = 0.2
* auxiliary: use auxiliary tower or not during the training, default = True
* auxiliary\_weight: weight of auxiliary tower, default = True
* grad\_clip: gradient clipping by L2 norm, default = 5
* report\_freq: default = 50
* num\_reduction: number of reduction cells, default = 2

Remark:

* After running ImageClassifier.fit(), one can directly call ImageClassifier.finalfit() to train the architecture found by ImageClassifier.fit().
* However, one can also call ImageClassifier.finalfit(genotype=genotype) without having run ImageClassifier.fit() to train the model with desired genotype. Please refer to Examples for further illustrations.

1. ToKeras(output\_name, path=None, genotype=None, init\_channels=None, layers=None)

#load the torch model from path and convert it to Keras model

Parameters:

* output\_name: name of Keras model
* path: path of torch model, default = None which correspond to use the model obtained in ImageClassifier.finalfit()
* genotype: genotype of the torch model, default = None which correspond to the genotype used in ImageClassifier.finalfit()
* init\_channels: number of channels used in first convolution of the torch model, default = None which correspond to the number used in ImageClassifier.finalfit()
* layers: number of layers of the torch model, default = None which correspond to the number used in ImageClassifier.finalfit()

Remark:

* After running ImageClassifier.finalfit(), one can directely call ImageClassifier.ToKeras() to convert the pytorch model to Keras model
* However, one can also call ImageClassifier.ToKeras(keras\_model\_name, path=path genotype=genotype, init\_channels= init\_channels, layers= layers) without having run ImageClassifier.finalfit() to convert the model which is already saved in path.

1. path()

Return the path of save dir

1. genotype()

Return the genotype found in ImageClassifier.fit()

1. finalfit\_init\_channels():

Return number of channels used in first convolution during final fit stage

1. finalfit\_layers()

Return number of layers used during final fit stage

Examples:

Example 4: Search a Neural Architecture, train the Neural Architecture found and then convert it to Keras model.

from ImageClassifier import \*

import numpy as np

#load train input from train\_input\_path, train\_input is supposed to be shape (num\_samples, H, W, channels) whose values are between 0 and 255

train\_input = np.load(train\_input\_path)

#load train target from train\_target\_path, train\_target is supposed to be shape (num\_samples, num\_classes) with one-hot encodding

train\_target = np.load(train\_target\_path)

test\_input = np.load(test\_input\_path)

test\_target = np.load(test\_target\_path)

#create classifier

classifier = ImageClassifier(train\_input, train\_target, test\_input, test\_target)

#fit the classifier and obtain genotype proposed by classifier

classifier.fit()

#final fit the model with the genotype found before

classifier.finalfit(epochs=800)

#convert the model trained to Keras

classifier.ToKeras("k\_model”)

Example 2: Train directly a Neural Network with a given genotype and then convert it to Keras model

from ImageClassifier import \*

import numpy as np

#load train input from train\_input\_path, train\_input is supposed to be shape (num\_samples, H, W, channels) whose values are between 0 and 255

train\_input = np.load(train\_input\_path)

#load train target from train\_target\_path, train\_target is supposed to be shape (num\_samples, num\_classes) with one-hot encodding

train\_target = np.load(train\_target\_path)

test\_input = np.load(test\_input\_path)

test\_target = np.load(test\_target\_path)

#create classifier

classifier = ImageClassifier(train\_input, train\_target, test\_input, test\_target)

#define a genotype

genotype = Genotype(normal=[('sep\_conv\_3x3', 1), ('sep\_conv\_3x3', 0), ('sep\_conv\_3x3', 1), ('sep\_conv\_5x5', 2), ('sep\_conv\_3x3', 1), ('sep\_conv\_3x3', 0), ('sep\_conv\_3x3', 2), ('sep\_conv\_3x3', 0)], normal\_concat=range(2, 6), reduce=[('avg\_pool\_3x3', 0), ('max\_pool\_3x3', 1), ('dil\_conv\_5x5', 2), ('avg\_pool\_3x3', 0), ('sep\_conv\_3x3', 3), ('avg\_pool\_3x3', 0), ('dil\_conv\_5x5', 3), ('avg\_pool\_3x3', 0)], reduce\_concat=range(2, 6))

#final fit the model with the genotype defined before

classifier.finalfit(genotype=genotype, epochs=800)

#convert the model trained to Keras

classifier.ToKeras("k\_model”)

Example 3: Convert a saved pytorch model to Keras model (suppose pytorch model is saved in My\_path, suppose that the genotype of the pytorch model is My\_genotype, init\_channels = 36 and layers = 20)

from ImageClassifier import \*

#load train input from train\_input\_path, train\_input is supposed to be shape (num\_samples, H, W, channels) whose values are between 0 and 255

train\_input = np.load(train\_input\_path)

#load train target from train\_target\_path, train\_target is supposed to be shape (num\_samples, num\_classes) with one-hot encodding

train\_target = np.load(train\_target\_path)

test\_input = np.load(test\_input\_path)

test\_target = np.load(test\_target\_path)

#create classifier

classifier = ImageClassifier(train\_input, train\_target, test\_input, test\_target)

classifier.ToKeras("k\_model”, path = My\_path, genotype = My\_genotype, init\_channels = 36, layers = 20)

Example 4: predict with converted Keras model. Suppose Keras model is saved in k\_model, mean (resp. std) be the mean values (resp. standard deviation values) along channels in dataset

import numpy as np

import keras

model = keras.models.load\_model(k\_model)

test\_input = np.load(test\_input\_path)

#prepocessing

mean = [51.57/255, 51.94/255, 40.53/255]

std = [54.30/255, 53.61/255, 40.77/255]

test\_input = (test\_input/255.0 - mean)/std

test\_input = np.transpose(test\_input, [0,3,1,2])

predict = model.predict(test\_input)