[machine learning code structure]

1. **prepare dataset**
   * load dataset
   * data preprocessing
2. **define model** 
   * model structure
3. **model compile** 
   * loss function
   * metric to supervise during training
   * Optimizer
4. **set callback function**
   * Tensorboard
   * CheckPoint
5. **training/testing**

ex:

# load dataset

(x\_train, y\_train) , (x\_test, y\_test) = tf.keras.datasets.cifar10.load\_data()

# data preprocess

x\_train = x\_train/255

x\_test = x\_test/255

y\_train = tf.keras.utils.to\_categorical(y\_train, 10)

y\_test = tf.keras.utils.to\_categorical(y\_test, 10)

# model structure define

# this model structure is only for training single model program

# for training multiple models, please take a look at next topic.

model = tf.keras.Sequential([

tf.keras.layers.Conv2D(filter=6, kernel\_size=[5,5], padding= “valid”, activation=tf.nn.relu, input\_shape= [32,32,3]), # the first layer can specify the input\_shape which is the training data shape

# only the first layer input\_shape specification valid

tf.keras.layers.Maxpooling2D(pool\_size=[2,2], padding= “valid”),

tf.keras.layers.Conv2D(filter=16, kernek\_size=[5,5], padding= “valid”, activation=tf.nn.relu),

tf.keras.layers.Maxpooling2D(pool\_size=[2,2], padding= “valid”),

tf.keras.layers.Flatten(),

tf.keras.layers.Dense(units=120, activation=tf.nn.relu)

tf.keras.layers.Dense(units=84, activation=tf.nn.relut)

tf.keras.layers.Dense(units=10, activation=tf.nn.softmax)

])

# print out the model summary. it is very useful for model debugging

model.summary()

# model compile

model.compile(loss=tf.keras.losses.categorical\_crossentropy,

optimizer= tf.train.AdamOptimizer(),

metrics=[tf.keras.metrics.categorical\_accuracy])

# callback function(Tensorboard, CheckPoint,…)

tb\_cb= tf.keras.callbacks.Tensorboard(log\_dir= “tensorboard”)

# train\_model

model.fit(x\_train, y\_train, batch\_size=128, epoch=100, callback=[tb\_cb])

[training multiple model program]

Keras can easily mislead people believe that declaration of multiple models objects will generate mulitple tensor “graph”. But the truth is, models you declare will include the graph of previous models.

example:

model\_1 = network() # network is the keras model generating function

model\_2 = network() # model\_2 will include the graph of model\_1

model\_1.fit(…)

model\_2.fit(…) # model\_2.fit still works, but the graph in the tensorboard will include model\_1’s graph.

* better practice

Using tenorflow backend to build the keras model in different graph. This way, the model graph will not tangle with others. But you need to finish dealing with the model (train, evaluate, save…) before new model’s build.

example:

with tf.Graph().as\_default() # generate a new graph and make it default

sess = tf.Session() # create new session

tf.keras.backend.set\_session(sess) # make new session as keras default

<keras\_model\_building>

<keras\_model\_training>

<keras\_model\_blah>