Meta graph 的官方解释是：一个 Meta Graph 由一个计算图和其相关的元数据构成。其包含了用于继续训练，实施评估和（在已训练好的的图上）做前向推断的信息。

Meta Graph 在具体实现上就是一个 MetaGraphDef （同样是由 Protocol Buffer 来定义的）。其包含了四种主要的信息，根据 Tensorflow 官网，这四种 Protobuf 分别是

1. MetaInfoDef，存一些元信息（比如版本和其他用户信息）
2. GraphDef， MetaGraph 的核心内容
3. SaverDef，图的 Saver 信息（比如最多同时保存的 check-point 数量，需保存的 Tensor 名字等，但并不保存 Tensor 中的实际内容）
4. CollectionDef 任何需要特殊注意的 Python 对象，需要特殊的标注以方便 import_meta_graph 后取回。（比如“train_op”,"prediction" 等等）

Tensorflow 中并没有一个官方的定义说 collection 是什么。简单的理解，它就是为了方别用户对图中的操作和变量进行管理，而创建的一个概念。它可以说是一种“集合”，通过一个 key（string 类型）来对一组 Python 对象进行命名的集合。这个 key 既可以是 tensorflow 在内部定义的一些 key，也可以是用户自己定义的名字（string）。

Tensorflow 内部定义了许多标准 Key，全部定义在了 tf.GraphKeys 这个类中。其中有一些常用的，tf.GraphKeys.TRAINABLE_VARIABLES, tf.GraphKeys.GLOBAL_VARIABLES 等等。tf.trainable_variables() 与 tf.get_collection(tf.GraphKeys.TRAINABLE_VARIABLES) 是等价的；tf.global_variables() 与 tf.get_collection(tf.GraphKeys.GLOBAL_VARIABLES) 是等价的。

```
pred = model_network(X)

loss=tf.reduce_mean(…, pred, …)

train_op=tf.train.AdamOptimizer(lr).minimize(loss)
```

用户希望特别关注 pred, loss train_op 这几个操作，那么就可以使用如下代码，将这几个变量加入到 collection 中去。(假设我们将其命名为 “training_collection”)

```
tf.add_to_collection("training_collection", pred)

tf.add_to_collection("training_collection", loss)

tf.add_to_collection("training_collection", train_op)
```

并且可以通过 Train_collect = tf.get_collection(“training_collection”) 得到一个 python list，其中的内容就是 pred, loss, train_op 的 Tensor。这通常是为了在一个新的 session 中打开这张图时，方便我们获取想要的操作。比如我

们可以直接工通过 get_collection() 得到 train_op，然后通过 sess.run(train_op)来开启一段训练，而无需重新构建 loss 和 optimizer。

通过 export_meta_graph 保存图，并且通过 add_to_collection 将 train_op 加入到 collection 中：

```
with tf.Session() as sess:

  pred = model_network(X)

  loss=tf.reduce_mean(…,pred, …)

  train_op=tf.train.AdamOptimizer(lr).minimize(loss)

  tf.add_to_collection("training_collection", train_op)

  Meta_graph_def =
```

```
      tf.train.export_meta_graph(tf.get_default_graph(),
'my_graph.meta')
```

通过 import_meta_graph 将图恢复（同时初始化为本 Session 的 default 图），并且通过 get_collection 重新获得 train_op，以及通过 train_op 来开始一段训练（ sess.run() ）。

```
with tf.Session() as new_sess:

  tf.train.import_meta_graph('my_graph.meta')

  train_op = tf.get_collection("training_collection")[0]

  new_sess.run(train_op)
```

特殊的说明一下，**Meta Graph 中虽然包含 Variable 的信息，却没有 Variable 的实际值。所以从 Meta Graph 中恢复的图，其训练是从随机初始化的值开始的。训练中 Variable 的实际值都保存在 check-point 中，如果要从之前训练的状态继续恢复训练，就要从 check-point 中 restore。**

export_meta_graph/Import_meta_graph 就是用来进行 Meta Graph 读写的 API。**tf.train.saver.save() 在保存 check-point 的同时也会保存 Meta Graph。但是在恢复图时，tf.train.saver.restore() 只恢复 Variable，如果要从 MetaGraph 恢复图，需要使用 import_meta_graph。**这是其实为了方便用户，有时我们不需要从 MetaGraph 恢复的图，而是需要在 python 中构建神经网络图，并恢复对应的 Variable。

Check-point

Check-point 里全面保存了训练某时间点的的信息，包括参数，超参数，梯度等等。tf.train.Saver()/saver.restore() 则能够完完整整保存和恢复神经网络的训练。Check-point 分为两个文件保存 Variable 的二进制信息。ckpt 文件保存了 Variable 的二进制信息，index 文件用于保存 ckpt 文件中对应 Variable 的偏移量信息