```
1. for iter in range( a r g s . epochs ) :
2.    loss _ locals = [ ] , w_locals = [ ]
3.    m = max( int ( args . frac *args . num_users ) , 1 )
4.    idxs_users = np . random . choice (range( args . num_users ) ,
5.     m, replace=False ) # random activation number of users
6.     for id x in idxs_users :
7.         local = LocalUpdate ( args=args , dataset=dataset_
   train ,
8.           idxs=dict_users [ idx ] )
9.         w, loss = local . train
10.                ( net=copy . deepcopy ( net_glob ) . to ( args
   . device ) )
11.             if args . all_clients :
12.                 w_locals [ idx ] = copy . deepcopy (w)
13.            else :
14.                 w_locals . append ( copy . deepcopy (w) )
15.             loss_locals.append ( copy . deepcopy ( loss ) )
16.     # update global weights
17.     w_glob = FedAvg ( w_locals )
18.     # copy weight to ne t_glob
19.     net_glob . load_ state_ dict ( w_glob )
20.
21.
22.     print( Create Model Training )
23.     print( Waiting for { } clients . . . . format( args
   . num_users ) )
24.     while True :
25.             connection , address = soc . accep t ( )
26.             clients . append ( {
27.                   ” connection ” : connection ,
28.                   ” address ” : address
29.              } )
30.              print( Connected to : + address [0 ] + ’ : ’ +
31.       str ( address [ 1 ] ) )
32.              print( Thread Number + str (len( clients ) ) )
33.              if (len( clients ) >= args . num_users ) :
34.                     print( ” Client sready ” )
35.              else :
36.         continue
37.               sleep ( 3 )
38.
39.     print (FedAVG and Local Update Process )
40.     print( P rocess Completed )
41.     soc . close ( )
42.     print( Socket is closed . )
43.
44.
45.     print( \ nInitializing weights , please wait . . . \ n )
```

```
46.      self . model = ini t_model
47.      self . model_path = " models / server / save /model
   . keras "
48.      self . saveModel ( )
49.
50.      def loadModel ( self , model_path ) :
51.           self . model = keras
   . models . load_model ( model_path )
52.           return self . model
53.
54.      def compile( self , model ) :
55.         model . compile( optimizer= ' adam ' ,
56.             loss=keras . losses . SparseCategoricalCrossentropy
57.               ( f r om_l o gi t s=True ) ,
58.             metrics =[ ' accuracy ' ] )
59.      return model
60.
61.      import tensor flow_model_optimization as tfmot
62.
63.      print( Initializing weights , please wait . . . )
64.      if args . quantize :
65.        init_model , _ = self . train ( init_model )
66.        self . model = tfmot
   . quantization . keras . quantize_model
67.             ( ini t_model )
68.      self . model_path = " models / server / save /model
   . keras "
69.      self . saveModel ( )
70.
71.      def saveModel ( self ) :
72.         if self . args . quantize :
73.           self . model = self . compile( self . model )
74.          self . model . save ( self . model_path )
75.        with zipfile . Zip File ( " { } . zip "
   .format( self . model_path ) ,
76.            'w ' , compression= zip file . ZIP_DEFLATED) as f :
77.              f . write ( self . model_path )
78.          return None
79.
80.      def loadModel ( s e l f , model_path ) :
81.      try :
82.        with tfmot . quantization . keras . quantize_scope ( ) :
83.           self . model = keras
   . models . load_model ( model_path )
84.      return self . model
85.
86.      def compile( self , model ) :
87.         model . compile( optimizer= ' adam ' ,
```

```
88.           loss=keras . losses . SparseCategoricalCrossentropy
89.           ( from_logits=True ) , metrics =[ ' accuracy ' ] )
90.       return model
91.
92.       import tensor flow_model_optimization as tfmot
93.
94.       print( Initializing weights , please wait . . . )
95.       if args . prune :
96.         num_images = self . train_images . shape [0 ]*
97.             (1 - self . args . validation _ split )
98.         end_step = np . ceil ( num_images
   / self . args . bs ) . astype
99.             ( np . int32 )*args . local_ep
100.        pruning_params = {
101.          ' pruning_schedule ' : tfmot
   . sparsity . keras . PolynomialDecay
102.          ( initial _ sparsity =0.50 , final _ sparsity =0.80 ,
103.          begin_step =0, end_step=end_step )
104.          }
105.        Self . model = tfmot
   . sparsity . keras . prune_low_magnitude
106.           ( ini t_model , **pruning_params )
107.        callbacks = [
108.              tfmot . sparsity . keras . UpdatePruningStep ( ) ,
109.              tfmot . sparsity . keras . PruningSummaries
110.                    ( log_ dir= tempfile . mkdtemp ( ) ) ,
111.               ]
112.      self . model , _ = self . train ( self . model , callbacks )
113.      self . model_path = " models / server / save /model
   . keras "
114.      self . saveModel ( )
```