```
import torch
import torch . nn . functional as F
from torch.utils. data import DataLoader
import torch.optim as optim
from torch vision import datasets , transforms

%reload_ext autoreload
%autoreload 2
%matplotlib inline

      from fastai import *
      from fastai.vision import *
      from fastai.metrics import accuracy

      path = untar_data(URLs.MNIST)
      path

      path.ls()

      (path/'training').ls()

      filenames = get_image_files(path/'training/5')
      len(filenames), filenames[:10]

      batchsize = 64
      np.random.seed(0)
      transform = get_transforms(do_flip=False)

      databunch = ImageDataBunch.from_folder(path, train='training', valid_pct=0.2, size=28,
                                                 ds_tfms=transform, bs=batchsize, num_workers=8).normalize()

      databunch.show_batch(rows=3, figsize=(10, 10))

      databunch.classes

      learner = cnn_learner(databunch, models.resnet18, metrics=accuracy)
      learner.fit_one_cycle(4)
      learner.unfreeze()
      learner.lr_find()

      learner.fit_one_cycle(3, max_lr=slice(1e-6, 3e-3))

      path_test = Path()/'data'

      path_test.ls()
```

```

test_image = open_image(path_test/'a007.png')
test_image

learner.predict(test_image)

databunch.classes

test_image = open_image(path_test/'a003.png')
test_image

learner.predict(test_image)

test_image = open_image(path_test/'a005.png')
test_image

learner.predict(test_image)

class CNNMnist ( nn . Module ) :
    def __init__ ( self , args ) :
      super(CNNMnist , self ) . __init__ ( )
      self . conv = nn . Conv2d ( args . num_channels , 28 , padding =1 ,
        kernel_ size =3)
       self.pool = nn . MaxPool2d ( 2 )
       self.fc1 = nn . Line ar (28*14*14 , 128)
       self.drop = nn . Dropout ( 0 . 2 )
       self.fc2 = nn . Linear ( 128 , args . num_classes )
       self.act = nn . ReLU ( )
  def forward ( self , x ) :
    x = self . act ( self . conv ( x ) ) # [ batch_size ,28 ,28 ,28 ]
    x = self . pool ( x ) # [ batch_size , 28 , 14 , 14 ]
    x = x . view ( x . size ( 0 ) , -1) # [ batch_size ,28*14*14=5488]
    x = self . act ( self . fc1 ( x ) ) # [ batch_size , 128 ]
    x = self . drop ( x )
    x = self . fc2 ( x ) # [ batch_size , 10 ]
return x

print(Model received )
print( Training model , please wait . . . )
print( Sent updated model to the server )

soc.close ()
print( Socket is closed.)
```