```
{
 "nbformat": 4,
 "nbformat_minor": 0,
 "metadata": {
  "colab": {
   "provenance": []
  },
  "kernelspec": {
   "name": "python3",
   "display_name": "Python 3"
  },
  "language_info": {
   "name": "python"
  }
 },
 "cells": [
  {
   "cell_type": "code",
   "source": [
    "import cv2\n",
    "import numpy as np\n",
    "from keras.datasets import mnist\n",
    "from keras.layers import Dense, Flatten, MaxPooling2D, Dropout\n",
    "from keras.layers.convolutional import Conv2D\n",
    "from keras.models import Sequential\n",
    "from tensorflow.keras.utils import to_categorical\n",
    "import matplotlib.pyplot as plt"
   ],
   "metadata": {
    "id": "yJs2eLRLOSHM"
   },
   "execution_count": 2,
   "outputs": []
  },
{
 "cell_type": "code",
 "source": [
  "(X_train, y_train), (X_test, y_test) = mnist.load_data()"
 ],
 "metadata": {
  "id": "-NoTriNEPBWl",
  "colab": {
   "base_uri": "https://localhost:8080/"
  },
  "outputId": "1ceca63e-26d0-4a6a-b2c0-0cd952d515f0"
 },
 "execution_count": 3,
 "outputs": [
  {
   "output_type": "stream",
   "name": "stdout",
   "text": [
    "Downloading data from https://storage.googleapis.com/tensorflow/tf-keras-datasets/mnist.npz\n",
    "11490434/11490434 [==============================] - 0s 0us/step\n"
   ]
  }
 ]
},
{
 "cell_type": "code",
 "source": [
  "plt.imshow(X_train[0], cmap=\"gray\")\n",
  "plt.show()\n",
  "print (y_train[0])"
 ],
 "metadata": {
  "colab": {
   "base_uri": "https://localhost:8080/",
       "height": 282
      },
      "id": "s_JYjDk2PGwL",
      "outputId": "d6cc49b8-d286-4bed-97ba-c5889303f296"
     },
     "execution_count": 4,
     "outputs": [
      {
       "output_type": "display_data",
       "data": {
        "text/plain": [
         "<Figure size 432x288 with 1 Axes>"
        ],
        "image/png":
"iVBORw0KGgoAAAANSUhEUgAAAPsAAAD4CAYAAAAq5pAIAAAABHNCSVQICAgIfAhkiAAAAAlwSFlzAAALEgAACxI
B0t1+/AAAADh0RVh0U29mdHdhcmUAbWF0cGxvdGxpYiB2ZXJzaW9uMy4yLjIsIGh0dHA6Ly9tYXRwbG90bGliLm9y
Zy+WH4yJAAAN9klEQVR4nO3df4xV9ZnH8c+zWP6QojBrBrOhKKSyEGg8ZON4gbl6w1hvojGhw1TSexoZE4/YNJaLIhNe
wf1WwwZBU2SzTNTKMWNl1qEzUgaQouoOzGhDgiKo5LdQ2mTEaowZEf/mCHefaPezBTnfu9w7nn3nOZ5/1Kbu69
57nnnicnfDi/7pmvubsATH5/VXYDAJqDsANBEHYgCMIOBEHYgSAuaubCzIxT/0CDubuxIxT/0CDubuNN72uLbuZ3Wpmh8zsPTN7sJ
7vAtBYlvc6u5lNkfRHSUslHZH0qqHZ0qqQudx9IzMOWHWiwRmzZF0t6z93fd/czfd/czkn4raVkd3weggeeJ+2xJfxrz/kg27S5
Z9ZtZfx3LAlCnhp+gc/c/c+SX0Su/FAmerZsg9KmjPm/bezaQBaUD1hf1XlSWb2HTObkkHkrYV0xaAouXejXf3ETPrkbRD0
hRJT7n724V1BqBQuS+95VoYn+VoMwYx+95VoYx+xAwzXkRzUALhyEHQiCsANBEHYgCMIOBEHYgSAIOxAEYQeCIOxAEIQdCIKwA0EQdi
AIwg4EQdiBIAg7EARhB4Ig7EAQhB0IgrEAQhB0IgrEAQhB0IgrADQRB2IAjCDgRB2IEgCDsQBGEHgiDsQBCEHQiCsANBEHYgCMIOBJF7yGZcG
KZMmZKsX3rppQ1dfk9PT83axRdfnJx33rx5yfr999+frD/yyCM1a+vXr0/O+9VXXyXrW7ZsSdYffvjhZL0sdYXdzA5L+lzSGUmj7r64iKYAFK+ILfu/uPunBfweAA3
EZ/1f1VfdKIV0c/1t87/p/o/7t/XRg3P6LdhWGt1S1zuAx5kR1tF9TkPZAlUqWXmYewtxddyv1VcfJUVpEXzO3F6a6Oj81bTuesZAmJlcXGF+r+PmqKxzZOxppozh+eRvOCtCqvU9iBsyexGnsLeU5BGMjYtHY5qSgYWmLUSp2HmRpS8YD5C6zK5cP3dGbQoGS1kZT82PjDjQdhFmIITyN+fxxK
dnuhn7rcuMc5ZlMPn2mRf5XB2fNSq2xTLWXqxsV7XBjRKuc+4ieHwXjyqIs14J20Gln7LVDiSqWpNp2v4cHrG4+yhxcDmjS
SXXyAbVuTrjqAlrO8TvHXo1U0NQJ51m3SfT5jtvBv/U+RZcnYpGjNm4xDnp+dfdjjmS1DgXg1CPVuuiMDHWa3ySSy2FsUnuq
cPuomwH3pQ09bl0Xr4f5fd8Lh4PGqYXVRUSNBVDj9sfsJddm+9H2VV9Q7z1xr/zdX4fq7dr1ZL+8f4GVp2fqnsn2vEn8o/d+zvy7D
9Rzxn4dfbzxD7y6/F9B+w5LIlWw8Whc0VkcaO+NrbTLsfrdeqQUcgwiyL9s6bF1x1wZdOXYrwDc1uCe4gm6TE7e7lcDfX+qgVb
nXfzqy1CN+QfeIXeBRCMXqkd2X7dQkmhMsPVvFsDyPbUW3jFuO/cXDwCZgAYDbp9x0pxZtORZGALIkAYQC1tpPqfY1Ta37Yx2
u8vyxqj8fFc3IjIVS8wuvAZ2cpZSJB4OJUEvAd3IpATEsR7jGPo3tf8+3JAoYSgL/7MDUvxeAQE4WbrxWYS0zkwLFUXBz88+Dbr
tmwz1YgzRtyxyyh+c17UG5SMCyfhNnb07/kNV6x3prCu+P3y0kfv3+HbzhPl+VBaM8IHwD02w+PvE0PsDDJzNSRpGwbrw6e0TOH
c7hmMPh58aLSBNbEAvcfHE4Vgw9bvD9oYjY8ZuoeTq+Utvj86e+Q7NpP79P9j07q+6sPL/DhtmCQLoYpjEu8THyB52lUuz7C4C
LKR0y9BiDlbX8BdGKJ4d/tWD1kqEUvJybNxv67J7b1+KuO5TvT8K1TUQBwAfS4Uu2ulsZbTyi5Jzp36I82bOmyu6x52XJVoh3ch
2CkXXUyCcfzmk4fzuP1OdScZgmomFxUcETVcA/J27Fq3WUYQdzmjIqaoZVx2X+58HbLBmHRJYdS7yu6/fpW89ZJUbzNXnUbcWpqbm
cYQ+h2fULXjP3QFRNI55u4+HSbBkmHRLR2lQFCGK8PMA6WQtjnKuNnU0Tp3YfNgj67TrU2f37+X1S8TW7ebsFXy5eCG+Xd6l8Eq0
hPDxHeJ0vgGJZDUYvqbQd4k3TGMzVOd+Syc1rR9Gzh+d/f8v99qA9BXQy7z2l+Z6a2aHgejbdVmaZrXGcVsHn3cmrWyORqfqEU
Pdl8rP9nUcPoJF5t1TtpUWaQbDhvzPbG9zAbbAFKf5RbRoYqO13A7iyf4KYzEjyPUEgFvRjQgQ84HRdXfYmZ5MUFG4N1Tn3kT2
```

2C0Cta9ldXeMNtFtR6zr63Llz87RUiP7+/mR97dq1yfq2bduKbGfSq7lIN7M5ZrbHzAbM7G0zW5VNbzOzF83s3ex5ZuP
bBZDXRHbjRyT9o7svlPR3klaa2UJJD0ra5e5XStqVvQfQomqG3d2H3H1/9vqkpHckzZa0TNKm7GObJN3VqCYB1O+8jt
nNbK6k70naJ6nd3c/9OPlDSe1V5umW1J2/RQBFmPDZeDP7pqRnJf3M3U+MrXlldMhxB2109z53X+Tui+rqFEBdJhR2
M/uGKkH/jbs/l00+amazsvosScca0yKAItTcjTczk/SkpHfcfcOY0jZJyyWty563NqTDSaC9fdwjnC8tXLgwWX/88ceT9auu
uuq8eyrKvn37kvVHH320am3r1vQ/GW5RLdZEjtn/XtKPJb1lZzgeyaWtUCfnvvzZWfnfQ8DzTHWmOrXn2d+tn7pK1L0b8Q4NyDO9WOdhcgAhrhpEBNvmrvbVfcN/hCi8CtgcXTSLWgWX/88ceT9auu
uuq8eyrKvn37kvVHH320am3r1vQ/GW5RLdZEjtn/XtKPJb1lZgeyaWtUCfnvzGyFpA8k/bAxLQIoQs2wu/t/Sxp3cHdJN
xfbDoBG4eeyQBCEHQiCsANBEHYgCMIOBMEtrhPU1tZWtdbb25uct6OjI1mfN29erp6K8MorryTrH69evT9Z37NiRrH/2
2Wfn3RMagy07EARhB4Ig7EAQhB0IgrADQRB2IAjCDgQR5jr79ddfn6yvWrUqWV+8eHHV2uzZs3P1VJRPP/20aq23tz
JeR955JFk/fTp07l6Quthyw4EQdiBIAg7EARhB4Ig7EAQhB0IgrADQYS5zt7Z2VlXvR6Dg4PJ+vbt25P1kZGRZD11z
HByXsTBlh0IgrADQRB2IAjCDgRB2IEgCDsQBGEHgjB3T3/Abo6kzZLaJbmkPnf/tZmtlfRPkn7/NzN7N7SNIDkv6cfXSNu/++xnelFwagbu4+
7qjLEwn7LEmz3H2/mU2X2/mU2X9Jqku1/2Uuz820SYIO9B41cI+kfHZhyQNZa9Pmtk7kW/NdxcMUsZjAC2mteHbOb2vxJ35O0L5vU
Y2ZvmtlTZjazvZv11dQqgLjV347/8oNk3Jb0saa27P2dm7ZI+UuU4/J/p9V2dW/v8Z3sBsPNFjuY3ZJMrNvSNouaY
e7bxinPlfSdne/psb3EHagwaqFveZuvJmZpCclvTM26NkJu3MWSRqst0kAjTORs/HXS/pvSW9J+iqbvFrSMknzVdmNH5J0b3YyL/U/sWUHGqzaln1C4/OLQtiBxsu9Gw/gwkDYgSAIOxAEYQeCIOxAEIQdCIKwA0EQdiAIwg4EQdiBIAg7EAQhB4Ig7EAQhB0IgrADQRB2IAjCDgRB2IEgCDsQBGEHgiDsQBC
EHQii2UM2fyTpgzHvL8umtaJW7a1V+5LoLa8ie/vbaoWm/3jrOsc3/+tYWb9bv7otIaSGjV3lq1L4ne8mpWb+zGA0EQdiCIssP
eV/LyU1q1t1btS6K3vJrSW6nH7ACap+wtO4AmIexAEKWE3cxuM/MNv8OGZam5m9m9aGbvZs/jjrfmFT/jjrFXUm8pdmdlgtu4OmNntJfU2x8z2mNmAmb1m4v4oaamkI
5JeldTl7gNNbaaRRDqKiYm9s4QA9ox+fbdkpVSDrbvmhFBqftr+FRBW2dzkDeL85ol7SNT/1oHtzRAO17MJmu/
48zzK2LlvlvSeu7v/v7mck/VbSshShL6aHnuvfS8MXmSRNanBVVS6Du5rqEpF4kbV/6ct/Gic5Ir/X33RPRXPpVOK6K3L
48zzK2LIvlvSeu7/v7mck/VbSshL6aHnuvlfS8a9MXiZpU/Z6kyr/WJquSm8twd2H3H1/9vqkpHPDjJe67hJ9NUUZYZ8t6
U9j3h9Ra4337pJ2mtlrZtZddjPjaB8zzNaHktrLbGYcNYfxbqavDDPeMusuz/Dn9eIE3dctcfe/lXSbpJXZ7mpL8soxWCtdO
/2lpPmqjAE4JGl9mc1kw4w/K+ln7n5ibK3MdTdOX01Zb2WEfVDSnDHvv51NawnuPpg9Ppg9H5P0vCqHHa3k6LdRkDdLPnYy
X38yV3P+ruZ919VNKvVOK6y4YZf1bSb9z9uWxy6etuvL6atd7KCPurkq4q40s+++Y2VRJP5K0rYQ+vsbMpmUnTmRm0yJ9hMpmUnTmRm0yT9
QK03FPU2Scuz18slbS2xl7/QKsN4VxtmXCWvu9KHP3f3pj8k3a7KGfn/lfRPZfRQpa95kt7IHm+X3ZukLars1v2fKuc2Vkj
6a0m7JL0r6r6T8ltbVQb/+uytDeb6oSrFkl9bZElV30NyUdyB63l73uEn01Zb3xc1kgCE7QAUEQdiAIwg4EQdiBIAg7EARhB
4Ig7EAQ/w8ie3GmjcGk5QAAAABJRU5ErkJggg==\n"

        },

        "metadata": {

          "needs_background": "light"

        }

       },

       {

        "output_type": "stream",

        "name": "stdout",

        "text": [

          "5\n"

        ]

       }

     ]

    },

    {

     "cell_type": "code",

     "source": [

      "print (\"Shape of X_train: {}\".format(X_train.shape))\n",

```
    "print (\"Shape of y_train: {}\".format(y_train.shape))\n",
    "print (\"Shape of X_test: {}\".format(X_test.shape))\n",
    "print (\"Shape of y_test: {}\".format(y_test.shape))"
   ],
   "metadata": {
    "colab": {
     "base_uri": "https://localhost:8080/"
    },
    "id": "Us6HotvxPPco",
    "outputId": "dddeb261-7494-47d3-f5c1-81f302ed3d69"
   },
   "execution_count": 5,
   "outputs": [
    {
     "output_type": "stream",
     "name": "stdout",
     "text": [
      "Shape of X_train: (60000, 28, 28)\n",
      "Shape of y_train: (60000,)\n",
      "Shape of X_test: (10000, 28, 28)\n",
      "Shape of y_test: (10000,)\n"
     ]
    }
   ]
  },
  {
   "cell_type": "code",
   "source": [
    "# Reshaping so as to convert images for our model\n",
    "X_train = X_train.reshape(60000, 28, 28, 1)\n",
    "X_test = X_test.reshape(10000, 28, 28, 1)"
   ],
   "metadata": {
    "id": "n962FFkFPUzH"
  },
  "execution_count": 6,
  "outputs": []
 },
 {
  "cell_type": "code",
  "source": [
   "print (\"Shape of X_train: {}\".format(X_train.shape))\n",
   "print (\"Shape of y_train: {}\".format(y_train.shape))\n",
   "print (\"Shape of X_test: {}\".format(X_test.shape))\n",
   "print (\"Shape of y_test: {}\".format(y_test.shape))"
  ],
  "metadata": {
   "colab": {
    "base_uri": "https://localhost:8080/"
   },
   "id": "KZEPs_m5PZac",
   "outputId": "36b9a4ca-c313-422d-f84a-28b461b01dcd"
  },
  "execution_count": 7,
  "outputs": [
   {
    "output_type": "stream",
    "name": "stdout",
    "text": [
     "Shape of X_train: (60000, 28, 28, 1)\n",
     "Shape of y_train: (60000,)\n",
     "Shape of X_test: (10000, 28, 28, 1)\n",
     "Shape of y_test: (10000,)\n"
    ]
   }
  ]
 },
 {
   "cell_type": "code",
   "source": [
    "#one hot encoding\n",
    "y_train = to_categorical(y_train)\n",
    "y_test = to_categorical(y_test)"
   ],
   "metadata": {
    "id": "BKtLDY0VQNkU"
   },
   "execution_count": 8,
   "outputs": []
  },
  {
   "cell_type": "code",
   "source": [
    "model = Sequential()\n",
    "\n",
    "## Declare the layers\n",
    "layer_1 = Conv2D(64, kernel_size=3, activation='relu', input_shape=(28, 28, 1))\n",
    "layer_2 = MaxPooling2D(pool_size=2)\n",
    "layer_3 = Conv2D(32, kernel_size=3, activation='relu')\n",
    "layer_4 = MaxPooling2D(pool_size=2)\n",
    "layer_5 = Dropout(0.5)\n",
    "layer_6 = Flatten()\n",
    "layer_7 = Dense(128, activation=\"relu\")\n",
    "layer_8 = Dropout(0.5)\n",
    "layer_9 = Dense(10, activation='softmax')\n",
    "\n",
    "## Add the layers to the model\n",
    "model.add(layer_1)\n",
    "model.add(layer_2)\n",
    "model.add(layer_3)\n",
    "model.add(layer_4)\n",
    "model.add(layer_5)\n",
    "model.add(layer_6)\n",
    "model.add(layer_7)\n",
    "model.add(layer_8)\n",
    "model.add(layer_9)"
   ],
   "metadata": {
    "id": "-afmGNuHCWdh"
   },
   "execution_count": 9,
   "outputs": []
  },
  {
   "cell_type": "code",
   "source": [
    "model.compile(optimizer='adam', loss='categorical_crossentropy', metrics=['accuracy'])"
   ],
   "metadata": {
    "id": "xLJShJAHDLhe"
   },
   "execution_count": 10,
   "outputs": []
  },
  {
   "cell_type": "code",
   "source": [
    "model.fit(X_train, y_train, validation_data=(X_test, y_test), epochs=3)"
   ],
   "metadata": {
    "colab": {
     "base_uri": "https://localhost:8080/"
    },
    "id": "roVOfV9EDRi9",
    "outputId": "22787483-28d2-41ae-9cdf-666f95b41f6a"
   },
   "execution_count": 11,
   "outputs": [
    {
     "output_type": "stream",
     "name": "stdout",
     "text": [
      "Epoch 1/3\n",
      "1875/1875 [==============================] - 58s 31ms/step - loss: 0.8654 - accuracy: 0.7801 -
val_loss: 0.1307 - val_accuracy: 0.9630\n",
      "Epoch 2/3\n",
      "1875/1875 [==============================] - 58s 31ms/step - loss: 0.2703 - accuracy: 0.9201 -
val_loss: 0.0750 - val_accuracy: 0.9757\n",
      "Epoch 3/3\n",
      "1875/1875 [==============================] - 56s 30ms/step - loss: 0.2055 - accuracy: 0.9385 -
val_loss: 0.0746 - val_accuracy: 0.9772\n"
     ]
    },
    {
     "output_type": "execute_result",
     "data": {
      "text/plain": [
       "<keras.callbacks.History at 0x7fc1e21cc510>"
      ]
     },
     "metadata": {},
     "execution_count": 11
    }
   ]
  },
  {
   "cell_type": "code",
   "source": [
    "example = X_train[1]\n",
    "prediction = model.predict(example.reshape(1, 28, 28, 1))\n",
    "print (\"Prediction (Softmax) from the neural network:\\n\\n {}\".format(prediction))\n",
  "hard_maxed_prediction = np.zeros(prediction.shape)\n",
  "hard_maxed_prediction[0][np.argmax(prediction)] = 1\n",
  "print (\"\\n\\nHard-maxed form of the prediction: \\n\\n {}\".format(hard_maxed_prediction))\n",
  "\n",
  "print (\"\\n\\n--------- Prediction --------- \\n\\n\")\n",
  "plt.imshow(example.reshape(28, 28), cmap=\"gray\")\n",
  "plt.show()\n",
  "print(\"\\n\\nFinal Output: {}\".format(np.argmax(prediction)))"
 ],
 "metadata": {
  "colab": {
   "base_uri": "https://localhost:8080/",
   "height": 595
  },
  "id": "styJoT_uDf6D",
  "outputId": "87aecfd4-45cd-441b-c184-0e255b5dc44d"
 },
 "execution_count": 12,
 "outputs": [
  {
   "output_type": "stream",
   "name": "stdout",
   "text": [
    "1/1 [==============================] - 0s 83ms/step\n",
    "Prediction (Softmax) from the neural network:\n",
    "\n",
    " [[9.99999881e-01 7.21094625e-13 7.90088137e-08 3.49195464e-11\n",
    "  1.54954244e-11 5.48896974e-13 1.05098525e-08 1.00683108e-10\n",
    "  7.00186797e-10 1.28125794e-08]]\n",
    "\n",
    "\n",
    "Hard-maxed form of the prediction: \n",
    "\n",
    " [[1. 0. 0. 0. 0. 0. 0. 0. 0. 0.]]\n",
          "\n",
          "\n",
          "--------- Prediction --------- \n",
          "\n",
          "\n"
         ]
        },
        {
         "output_type": "display_data",
         "data": {
          "text/plain": [
           "<Figure size 432x288 with 1 Axes>"
          ],
          "image/png":
"iVBORw0KGgoAAAANSUhEUgAAAPsAAAD4CAYAAAAq5pAIAAAABHNCSVQICAgIfAhkiAAAAAlwSFlzAAALEgAACxI
B0t1+/AAAADh0RVh0U29mdHdhcmUAbWF0cGxvdGxpYiB2ZXJzaW9uMy4yLjIsIGh0dHA6Ly9tYXRwbG90bGliLm9y
Zy+WH4yJAAAOF0lEQVR4nO3dcYxV5ZnH8d8jW4xKIagpTkRr2+AfzUYHQUKyprI2bVw0gcakQozDpk2GxZiZEqpi2VFfqzzQqozDpk2GxZiZEqpi2VFfqzzQqozDpk2GxZiZEqpi2VFfqzzQqozDpk2GxZiZEqpi2VFfqzzQqozDpk2GxZiZEqpi2VFfqzzQqozDpk2GxZiZEqpi2VFfqzzQqozDpk2GxZiZEqpi2VFfqzzQqozDpk2GxZiZEqpi2VFfqzzQqozDpk2GxJJQszGr3VF
IamNjlEZNJE6VFFcqqGjBpi51GaLdmDSOyCpqW1mDFhwZUSNDTKTCs3/cQzPinPcM9557z4Hn+0km997zzLn38Tl/z7
nnPfe85u4CcPI7peoGAHQGYQeCIOxAEIQdCIKwA0H8QydfzMw49A+0mbvbFy2vFdlnZjeb2ftm9qGZ
y5odZzezCZL+Iuk7kvzIelHSSvYX7Sghecd0EathP1csX8d9XhPtuxzzKzXZAbNbLCF
1wLQorYfoHP3fkn9ErvxQJVa2bLvlXTeqMfTs2UAaqiVsL8oaYaZfc3M2JkpaJGlzOW0BKFvTu/Hu/pmZLZO0RdIESvc/
bXSOgNQqqaH3pp6MT6zA23XlpNqAJw4CDsQBGEHiDsQBCEHQiCsANBEHYgCMIOBEHYgSAIOxAEYQeCIOxAEIQd
CIKwA0EQdiAIwg4EQdiBIAg7EARhB4Ig7EAQhB0IoqNTNuPkM2vWrKGR92bJlubWenp7kug8//HCyft999999yXr27dvT9aj
YcsOBEHYgSAIOxAEYQeCIOxAEIQdCIKwA0EwiyuSuru7k/Wmp7ZZSGjJB7W9JmzKff9WqaTLF3ZNtpGzXrr6vqCq8hLkfVDdA7ApEmTkvXnnnn
uWZ83b15u7aKLLkquLqquezfQJ80Tn3mTnn2BS7m4y76aefe6faefrIPYw7/f3kyBvXtbdCmaUubspRRG81m0Nfspr5AsT7u+Qmk7Y4Rpjv2HXjUzZY0KcDb
kbwanG3dJhaRjlzt4AwXZVRzr3tM11eCxUSrfaytLUymbLzGqaWpbgXaCUnmb6PFzK0Q7UBEI1b90YDaNqOZBmg6YDbdmbxt+hbXdhpp1lhqQc
vuuuu3Nq7776bXPflI19O1j/66KNk/Yorrsittfq+nIjYsgNBEHYgCMIOBEHYgSAIOxAEYQeCIOxAEFw3vga6u7uT9Y9YGBg
WR98uTJTb/2M888k6wXfR/+8ssvT9ZT3xt/8MEHk+u+/+//VFBO6WgmcU+8ETTezNWY2bGY7
Ry0708yeNbM3s9uS0x0Y1rLHwcxl/wlHnLTqTqNMUYyBaKwy7uz8v8r6bRCY9P0k+wYoM7wbXYrZDqtnTwXdCW12UfvFqeK3wiF6eg1nYWSMMy0CRzcJ1zuNl1ZpBKpoiBbtOqXqVnrqO6ZmQwGuVGFFAvKZIHI4S6lFavE4sWLKq71CkY+LSqDwWjRK3ZvrK1j5
kJzwVmWazm9YVNLl8KxLXxWjKqlQZi1y6qlnFY6gaxhWSW2oz6SAKBQ2+eyQKkapX+NgdNlrBABCDDHuFm+zUqkqDoZPyaCEC35VvTkXZ8+G2QoW1YR
NPJOt1lhpnL/q737BhQ7J+3XXXNdVTJ+SNsxd+Znf3vLMqvt1SRwA6itNlgSAIOxAEYQeCIOxAEIQdCIJLSZfg1FNPTdZl
1OWpPnz5yfrIyMjyXpPT09ubXdMBLnuaaedlqdXf7551fdQunYsgNBEHYgCMIOBEHYgSAIOxAEIyzl2Dm
zJnJetE4epEFCxYkMG60XTKgMSW3YgDMIOBEHYgSAIOxAEYQeCIOxAEIQdCIJx9hKsWrUrUqWTcb88q+f1c0Ts44enNOOS
V/W3bkyJEOdlIPbmNmBIAg7EARhB4Ig7EAQhB0IgrADQRB2IgrADQRB2IAjG2cfq2cfp6quvzq1q11d3cn1n1y0aMHnjz5rzX
ym6ncoVbdjNbY2bDZrZDZrZz1LKQFA241nN/5Xkq4cY/kv3L07+/lduW0W0b9j1lduW0BKFth2N39eUkfdqAXAG3UygG
6ZWb2SrabPzXvl8ys0MzeMzP0O5sx6y2lZ/Zpd6SrabPzXvl8ys18wGzSw96RiAtmo27KslfUNSl6UtDg7wJr7YhVyZlyQkBH/3933933uftjdj0j6aQ55bYFo
GxNhd3Mu6fLqAZkewCtwmgmeJAHvwNZZ1jzxjYDm0VmjK80Lym6kFhJMMe5S4fFcsK3m1kPN1S8JNHDm5o2Zd0XmZqkxYWLX2G0c3a37nnlpFddcP5p2LjO0+RnPwSh1PgsgB4
GxNhd3MukY9/J6knXm/C6AeCsfZzexRSfMknW1meytStkDTPzLoluaTdkpa2scdaSM1jPnHixyfqGDRuaDRua6ulk
VzTv/cqVK5t+7oGBGoW9lltuaqfq566ow7O6+elzFD7WhFwBtxOmyQBCEHQiCsANBEHYgCMIOBMFXXDg008/TdaHh
oY61Em9FA2t9fX1X1Jes33XnAXl0nKUk6ePBGsn4iYssObsOBdIMHECCsANBEHYgCMIOBEHYgSAIOxAEYQeCIMq4kKwA0Ewzt4BkS8
VnbrMdtE4+bXXXpusb9a9q0KVm/5pprkvVuo2LIDQRB2IAjCDgRB2IEgCDsQBGEHgiDsQBCMs4+TSs5KNsPQgdHTtNA4+TmXkzmTVVk6SFCxcm68uWLUvWu7u7k/XUZZ0fe+yx5Lqtri6vl5eqvlEtJA4+SFvJFcC4+/mf+SrA/+TQbM2+e/rU4x87kvFHVFEzMAVzRWRb5vIPs

```

```
2+qpzq48cYbk/Vbb701tzZlypTkuuvWrUvWe3p6knV8Hlt2IAjCDgRB2IEgCDsQBGEHgiDsQBCEHQiCcfZxcvemapJ0zjn
nJOv33ntvsr5mzZpk/YMPPsitzZ07N7nu9ddfn6xffPHFyfr06dOT9XfeeSe3tmXLluS6999/f7KO41O4ZTez88xsm5m9b
mavmdnybPmZZvasmb2Z3U5tf7sAmjWe3fjPJP2bu39T0lxJPzzb0q7t0q6oq6lj1pXb9/d5H6f8GNIWHWDAzaYGb6fxgcuolt/xk3y
5e797j7b3We31CmAlowr7Gb2JTWCvs7dn8wW3DRsSyzHuQRXF/WZsyYkVy3VS+88EKyvm3bttzabbbfdVnY7SBjPZ/Z/knS9pFfNbEe27CdqhPwxM/
N7AJEyYk6zfccEOyXnRJ5AMHDuTWZsyYkVy3VS+88EKyvm3bttzabbfdVnY7SBjPZ/Z/knS9pFfNbEe27CdqhPwxM/uh
pLclfb89LQIoQ2HY3f1/JOVdneHb5bYDoF04XRYIgrADQRB2IAjCDgRB2IEgrOjrmaW+mFnnXqxkqa9yPv7448l8l1L7300
pZeu+hS1a38G6a+HitJ69evT9ZP5Mtgn6zcfcw/GLbsQBCEHQiCsANBEHYgCMIOBEHYgSAIOxAE4+wl6OrqStaXLl2arP
f19SXrrYyz33PPPcl1V69enazv2rUrWUf9MM4OBEfYgSAIOxAEYQeCIOxAEIQdCIKwA0Ewzg6cZBhnB4Ij7EAQhB0IgrA
DQRB2IAjCDgRB2IEgCsNuZueZ2ZeZ2TYze93MXjOz5dnylWa218x9nylWa218x2ZD/z298ugGYVnlRjZl2Sutx9u5l9WdJLkhaqMR/7QXe/
a9wvxkk1QNvlnVQznvnZhyQNZfdHzOwNSee2W6mOmQjb2x6Adjuuz+xmdZ6mQ2a2WBLn
QJoybjPjTezSZKek/Qzd3/SzKZJ2i/JJf1UjV39HxQ8B7vxvxQJvl7caPK+xm9iVJv5W0xd1XjVG/QNJv3f0fC56HsNt1vQXYa
xxadOHJL0xOujZgbujvidpZ6tNAmif8RyNv0zSHyIlltvgnkhZL6lZjN363pKXZwbzUc7FlB9qspd34shB2oP34PjsQHGEH
EHgiDsQBCEHQiCsANBEHYgCMIOBEHYgSAIOxAEYQeCIOxAEIQdCIKwA0EQdiCIwgtOlmy/pLdHPT47r1ZpbnW1ZHde2trn1J
9NasMnv7al6ho99n/8KLmw26++zKGkioa2917Uuit2Z1qjd244EgCDsQRNVh76/49VPq2lYjM6wHNVnuDiHsQBCVhN3Mrjq
LTuADiHsQBCVhN3MrjSzP5vZLjO7uYoe8pGutL56bI59IbNbjN7NZuGutL56bI59IbNbOeoZWea2bNm9mZ2O+YcexX1VotpvBPTjFf
63lU9/XnHP7Ob2QRJf5H0H0HUl7JL0oabG7v97RRnKY2W5Js9298hMvzOxfJb0vae3w0pXp1FpmdqekeyR/+7/X
LeVOs5pvNvUW9404/+qCt+7Mqc/b0YVW/Y5kna5+1vufkjSekkLK8ij9SekkLKuij9tz9eUkfHrN4gaS12f21avyxdFxOb7Xg7kPuvj2
7PyLp6DTjlb53ib46ooqwnyvpr6Me71G95nt3Sb83s5fjW2zfhGSTN929oJa1W+wG7j3pMa1W+dOTHzr05K01W9P9O06qgOk0Nkt
N0X3SZu18i6V8k/SjbXa0lb3wGq9PY6WpJ31BjDsA4ojGvzgA0Y7E8ZoKrpKv71cDXo2XN4p39S9WSz2Sq8wV5j3s2dHtPu1RCQF1G1EHRn5ffwhtL0QvUzgD1b3
eDue7PbYUlPqfGxo072HZ1BN7sdrrifv3P3fe5+2N2PSPqlKnzvsvnsvsmnGN0pa5+5PZosrsf+/G6qtT71sVYX9R0gwz+5qZTS
0SNLmCvr4AjM7IztwIjM7Q9J3Vb+pqDdLWpbdXybpNxX28gV1mac+bY5xMfcSRNQNOmQa9Cf/bdiMpS
0SNLmCvr4AjM7IztwIjM7Q9J3Vb+pqDdLWpLdXyJpU4W9fE5dpvHOm2ZcFb93lU9/7u4d/5E0X40j8v8n6T+q6CGnr
9L+t/s57Wqe5P0qBq7dX9T49jGDyWdmJ3GrtsDhlcl/bekM2vVU23+qMbX3K2oo6vi3i5Tsf9FUk7sp/5Vb93ib468r5xui
wQBAfogCAIOxAEYQeCIOxAEIQdCIKwA0EQdiCI/wcI826NkY1TiQAAAABJRU5ErkJggg==\n"
        },
        "metadata": {
          "needs_background": "light"
        }
       },
       {
        "output_type": "stream",
        "name": "stdout",
        "text": [
          "\n",
          "\n",
          "Final Output: 0\n"
        ]
       }
      ]
    },
    {
     "cell_type": "code",
     "source": [
      "metrices=model.evaluate(X_test,y_test,verbose=0)\n",
      "print(\"Metrices(test loss and Test Accuracy):\")\n",
      "print(metrices)"
     ],
     "metadata": {
      "colab": {
       "base_uri": "https://localhost:8080/"
      },
      "id": "O05hxMgFFwBf",
      "outputId": "8711908d-bda8-4332-bb23-4a82cbf40acc"
     },
     "execution_count": 13,
     "outputs": [
      {
       "output_type": "stream",
       "name": "stdout",
       "text": [
        "Metrices(test loss and Test Accuracy):\n",
        "[0.07461030036211014, 0.9771999716758728]\n"
       ]
      }
     ]
    },
    {
     "cell_type": "code",
     "source": [
      "image = cv2.imread('test_image.jpg')\n",
      "image = np.full((100,80,3), 12, dtype = np.uint8)\n",
      "grey = cv2.cvtColor(image.copy(), cv2.COLOR_BGR2GRAY)\n",
      "ret, thresh = cv2.threshold(grey.copy(), 75, 255, cv2.THRESH_BINARY_INV)\n",
      "contours,hierarchy = cv2.findContours(thresh.copy(), cv2.RETR_EXTERNAL,
cv2.CHAIN_APPROX_SIMPLE)\n",
      "preprocessed_digits = []\n",
    "\n",
    "for c in contours:\n",
    "    x,y,w,h = cv2.boundingRect(c)\n",
    "    \n",
    "    # Creating a rectangle around the digit in the original image (for displaying the digits fetched via
contours)\n",
    "    cv2.rectangle(image, (x,y), (x+w, y+h), color=(0, 255, 0), thickness=2)\n",
    "    \n",
    "    # Cropping out the digit from the image corresponding to the current contours in the for loop\n",
    "    digit = thresh[y:y+h, x:x+w]\n",
    "    \n",
    "    # Resizing that digit to (18, 18)\n",
    "    resized_digit = cv2.resize(digit, (18,18))\n",
    "    \n",
    "    # Padding the digit with 5 pixels of black color (zeros) in each side to finally produce the image of (28,
28)\n",
    "    padded_digit = np.pad(resized_digit, ((5,5),(5,5)), \"constant\", constant_values=0)\n",
    "    \n",
    "    # Adding the preprocessed digit to the list of preprocessed digits\n",
    "    preprocessed_digits.append(padded_digit)\n",
    "\n",
    "print(\"\\n\\n\\n----------------Contoured Image--------------------\")\n",
    "import os, types\n",
    "import pandas as pd\n",
    "\n",
    "def __iter__(self): return 0\n",
    "\n",
    "print=(\"\\n\\n\\n----------------Contoured Image--------------------\")\n",
    "plt.imshow(image, cmap=\"gray\")\n",
    "plt.show()\n",
    "    \n",
    "inp = np.array(preprocessed_digits)\n"
   ],
   "metadata": {
    "colab": {
        "base_uri": "https://localhost:8080/",
        "height": 337
       },
       "id": "104DE2lDGko0",
       "outputId": "a175187c-999e-4d57-86fc-2d78dbda46f4"
      },
      "execution_count": 16,
      "outputs": [
       {
        "output_type": "stream",
        "name": "stdout",
        "text": [
         "\n",
         "\n",
         "\n",
         "----------------Contoured Image--------------------\n"
        ]
       },
       {
        "output_type": "display_data",
        "data": {
         "text/plain": [
          "<Figure size 432x288 with 1 Axes>"
         ],
         "image/png":
"iVBORw0KGgoAAAANSUhEUgAAAM8AAAD7CAYAAADNasDkAAAABHNCSVQICAgIfAhkiAAAAAlwSFlzAAALEgAACx
IB0t1+/AAAADh0RVh0U29mdHdhcmUAbWF0cGxvdGxpYiB2ZXJzaW9uMy4yLjIsIGh0dHA6Ly9tYXRwbG90bGliLm9
yZy+WH4yJAAAKyklEQVR4nO3dX4ilhXnH8e/PnTVptI1/sizbXVu3KEmXQmoYrGIpRQ1YG6IXqahYqwN2lrmkCi7UV
uK5QYL0pg0YalSKPdSBUJCXZjLlro1jFKUndj3JoaV1ZdQZuSm+5kn17Ma5luZzNnn5kz5536/cAw5/1z5n142e+e95x5Y
VJVSDp35816AGmzMh6pyXikJuORmoxHaJIeqWlN8SS5KckLSY4luXu9hp+Lqz5kD4P2KPwceBp4BNVdWT9xpP
Ga24Nz70aOFZVLwEk+SpwC3DWePKBFJev4YjSLDzDm/Ky+hAfxFeU0+Axr7qKMa0ogrCD0LyXBp1Jnx85YqBjs
HrqHxhU1f6qmq+qef5Pu9LmtZZZXnleBy5Yt7xrWTaaAReCna5hAmpbzgK1Azr7LWuJ5GrgyyW6WorkD+L2Jn70Ic4/
MseWft6xhBGk6Tv/qaU79wSm48Oz7tOOpqsUkfwR8E9gC/HVVPT/xD/gpbPmnLWz9m63dEaSpWbxxkVO3TykegK
r6OvD1tfwMabPyDgOpyXikJuORmoxHaJIeqqcl4pCbjkZqMR2oyHqnJeKQm45GajEdqMh6pyXikJuORmoxHaJIeqcl4pC
bjkZqMR2oyHqnJeKQm45GajEdqMh6pyXikJuORmoxHaJIeqcl4pCbjkZqMR2oyHqnJeKQm45GaVo0nyWVJnkpyJMn
zSe4a1l+S5MkkLw7fLL57+uNJ4TPLKswh8tqr2ANcAn0qyB7gbgbOFRVVwKHHhmXpXWPVeKrqRFuz8n8Bg
2O0AcOu0hpTG6Jze8yS5HHLgKOxniQXkZk4niQXAl8lZk4niQXAl8DPl1VP16+raoKjVeV1X7q2q+q
ubZtqZZpVGZKJ4kW1kk56GenRY/XqSHRY/XqSHcP2HcAb0xlRGJPm0L8CBwtKq+uGzT48Dtw/3bgcdmP5o0fw8
e0meG9b9GfAXCNJ7gReRexfzoejSO0aT1X9I5CEzyvcI2gcERBm6fzojSOK0aT1X9I5CzbL5hfceRNg/vMJCajEd
qMh6pyXikJuORmoxHajIeqcl4pCbjkZqMR2oyHqnJeKQm45GajEdqMh6pyXikJuORmoxHajIeqcl4pCbjkZqM
```

R2oyHqnJeKQm45GajEdqMh6pyXikJuORmoxHajIeqcl4pCbjkZqMR2oyHqnJeKQm45GajEdqMh6pyXikJuORmoxHajI
eqcl4pCbjkZqMR2o6lz8lvyXJs0meGJZ3Jzmc5FiSh5OcP70xpfE5l1eeu4Cjy5bvBe6rqiuAt4A713MwaewmiifJLuB3gQe
G5QDXAweHXQ4At05jQGmsJn3l+RLwOeD0sHwp8HZVLQ7Lx4GdKz0xyb4kC0kWOLmmWaVRWTWeJB8D3qiqZzoH
qKr9VTVfVfNs6/wEaZzmJtjnOuDjSW4G3gv8AnA/cFGSueHVZxfw6vTGlMZn1VeeqrqnqnZV1eXAHcC3quqTwFPAbc
Nue4HHpjalNEJr+T3P54HPJDnG0nugB9dnJGlzmOSy7X9U1beBbw+PXwKuXv+RpM3BOwykJuORmoxHajIeqcl4pCbj
kZqMR2oyHqnJeKQm45GajEdqMh6pyXikJuORmoxHajIeqcl4pCbjkZqMR2oyHqnJeKQm45GajEdqMh6pyXikJuORm
oxHajIeqcl4pCbjkZqMR2oyHqnJeKQm45GajEdqmiieJBclOZjk+0mOJrk2ySVJnkzy4vD94mkPK43JpK889wPfqKoPAR8
GjgJ3A4eq6krg0LAsvWusGk+S9wO/xfCn4qvqv6rqbeAW4MCw2wHg1mkNKY3RJK88u4GTwFeSPJvkgSQXANur6sS
wz2vA9pWenGRfkoUkC5xcn6GlMZgknjngI8CXq+oq4CeccYlWVQXUSk+uqv1VNV9V82xb67jSeEwSz3HgeFUdHpYPs
hTT60l2AAzf35jOiNI4rRpPVb0GvJLkg8OqG4AjwOPA3mHdXuCxqUowjdTchPv9MfBQkvOBl4A/ZCm8R5LcDbwM3D
6dEaVxmiieqnoOmF9h0w3rO460eXiHgdRkPFKT8UhNxiM1GY/UZDxSk/FITcYjNRmP1GQ8UpPxSE3GIzUZj9RkPFKT8
UhNxiM1GY/UZDxSk/FITcYjNRmP1GQ8UpPxSE3GIzUZj9RkPFKT8UhNxiM1GY/UZDxSk/FITcYjNRmP1GQ8UtNE8ST
50yTPJ/nXJH+b5L1Jdic5nORYkoeHv1cqvWusGk+SncCfAPNV9V82xb67jSeEwSz3HgeFUdHpYPs
hTT60l2AAzf35jOiNI4rRpPVb0GvJLkg8OqG4AjwOPA3mHdXuCxqUwojdTchPv9MfBQkvOBl4A/ZCm8R5LcCbwM3D
cBJ4DrgYPD9gPAres/njReq8ZTVa8Cfwn8iKVo/gN4Bni7qhaH3Y4DO1d6fpJ9SRaSLHByfYaWxmCSy7aLgVuA3cAvAhc
AN016gKraX1XzVTXPtvac0uhMctl2I/DDqjpZVaeAR4HrgIuGyziAXcCrU5pRGqVJ4vkRcE2S9yUJcANwBHgKuG3YZy/w
2HRGlMZpkvc8h1n6YUFAa8h4qHtXqhNtQ6FfdNGpoq33PPlBBcCAuuFj0dZ2yQfXXOnGxKK9kFbTTPbskb4f/FTTkG6lK3h
nJrZCNJZbQV+/mfvMrt4wtJwqwwojVWqauMOlpwEfgK8uWEHXZsPsHlmhc0172aa9ZeratuZKzc0HoAkC1U1v6EHb
dpMs8LmmnczzXo2ftomNRmP1DSLePbP4Jhdm2lZzbqZZV7Th73mk/y+8bJOajEdq2rB4ktyU5IUkx5LcvVHHnVS
Sy5I8leRIkueT3D3WsvyLO9Ocng4xw8nOX/WM74jyUVDib5fpKjSa4d87mdxIbEk2QL8
FfA7wB7gE8k2bMRxq4oPcA3wqWHGu4gKPLlu8F7quqK4C3gDtnMtXK7ge+UVf/Aj7M0t
xjPrerq6qpfwHXAt9ctnwPcM9GHHsNMz8GfBR4x4Hi8Bnq2oPcA3wqWHGu4FDVXUlcGhYHou7gKPLlu8F7quqK4C3gDtnMtXK7ge+UVUfAj7M0t
xjPrerq6qpfwHXAt9ctnwPcM9GHHsNMz8GfBR4AdgxrNsBvDDr2YZZdrH0D+564AmWbnh6E5hb6ZzPeNb3Az9k+IB
q2fpRnttJvzbqsm0n8Mqy5ePDulFKcjlwFXAY2F5VJ4ZNrwHbZzTWmb4EfA44PSxfCrxdVe/cpj6mc7wbOAl8ZbjMfCDJ
BYz33E7EDwzOkORC4GvAp6vqx8u31dJ/kTP/bD/Jx4A3quuqZWc8yoTngI8CXq+oqlu5v/F+XaGM5t+dio+J5Fbhs2fKuY
d2oJNnKUjgPVdWjw+rXk+wYtu8A3pjVfMtcB3w8yb8DX2Xp0u1+4KIk79wpP6ZzfBw4XlWHh+WDLMU0xnM7sY2K5
2ngyuHToPOBO4DHN+jYE0kS4EHgaFV9cdmmx4G9w9w+O9LL0XmqmquqeqdlXV5Sydy29V1SeBp4Dbht1GMStAVb0G
vJLkg8OqG4AjjPDcnpMNfNN4M/AD4N+AP5/1m70V5vtNli4Inh8a8A/AJfMetYz5v5t4Inh8a8A/wIcA
/4OeM+s51s2568DC8P5/Xvg4rGf29W+vD1HavIDA6nJeKQm45GajEdqMh6pyXikJuORmv4bWtmd36znqt4AAAAAS
UVORK5CYII=\n"

        },
        "metadata": {
          "needs_background": "light"
        }
      }
    ]
  }
 ]
}