```
{
 "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+zWP6QojBrOhKKSyEGg8ZON4gbl6w1hvojGhw1TSexoZE4/YNJaLIhNe
wf1WwwZBU2SzTNTKMMWNl1qEzUgaQuoOzGhDgiKo5LdQ2mTEaowZEf/mCHefaezBTnfu9w7nn3nOZ5/1Kbu69
57nnnnicnfDi/7pmvubsbZ1ATH5/VXYDAJqDsANBEHYgCMIOBEHYgSAuaububCzIxT/0CDubuNN72uLbuZ3Wpmh8zsPTN7sJ
7vAtBYlvc6u5lNkfRHSUlHZH0qqQudx2uQqQudx9IzMOWHWiwZRmZF0t6kn5z4kn4+w2weggeoJ+2xJfxrz/kg27S+YWbe
Z9ZtZfx3LAlCnhp+gc/c+SX0Su/FAmerZsg9KmHTkbYV0hRD95VoYx+xAwzXkRzUALhyEHQiCsANBEHYgCMIOBEHYgSAIOxAEYQeCIOxAEIQdCIKwA0EQdi
AIwg4EQdiBIAg7EARhB4Ig7EAQhB0IgrADQRB2IAjCDgrB2IEgCDsQBGEHgiDsQBDm3rwftU3WX9B1dHQk67t3707W32WG32baasaHR1N1u
+///5k/dSpU7mXPTQ0lKx//PHHyfqhQ4dyL7vR3N3Gm86WHQiCsANBEHYgSAIOxAEYQeC4Dp7Adra2pL
1ffv2Jevz5s0rsp1C1ep9eHg4Wb/b6Hg4Wb/pppuq1s6cOZMq1eQ1Hg2zpsMGwIFkfenSpcn66dnOnk/dSzdsRKl
ZH316tXJ+h133JGsv/h133JGsv/7668l6rT+pnHLgwIFkfenSpcn66dnOnk/dSzdsRKlVEtiJHYsoNH/Hydz83WNaHKtqrfdDMuiV1
51wOgILUfSOMu3vqxJu790nqkzhBB5Qp76W3o2V2S5Ky52PFtQsMUEgEapuRtvZlskfV/SZWZ2
RNIvJK2T9O2+zuKIFvHu/mJTFCE6Ss4OpnQPShzyi//cpTuo+sqgXFnTqGOTQXbA88y6dOdi6HzQ9+/7ZsfbrCFCgrMGRv4W3X2Y3zC9Y/MXhCgzNNTi71WYhw9i71NBEuFo53UPZFOJDT1eqI+2rvoY93/vr+7t3ADqdn/8t4uGDTuI8wBv4M0y2uRaRmJDCl5SGTtEKbW82xPpbNNDyvGQ7Bbv8E3e2pKF2LN11R8uECNqKZnB5a/3vTWftGA+y0xQZgFnlWBc5gl7MYx
H4zCtZwWPKSGFYBH1aQrtZL/N1b4zcJEE9ubBhzwiDr+DqWFVvVdKS5uZ7Df6z2eeKwEe1oMnpjmQWpLDQoEPNSpIVb9ZFBpB0JLLBwInFzPt
S6Ky5jHuCVBqX+3oIzdrzZ0gkJ+cZ8oXwAdPBLrIOLNbfKZhsWqlNg/LpxIVn3mbaDLsfVfgZm3sP++6tn28CNBbMeDmhzHe0YsEDdc+7mA5
zLLI+fIGJWSxwB2BpKXO88BpxBz/HhW1C5QenFkt1pbp5mNQ3yVwc9Q+BL68k7uDdibEnFuYh1SPlEp0ZRtoSVGdV3NPVczhSg97o3kS2LuUcY0WJ
dGmiOtxfrIpoLHOJGkCyIpi6aPFFNd8xAEVQZq7RWyUnX9FExxKGYzRrLTVKmWPGXrzqvwzs+Iy1cj7FCIgOBWrlgFEwhTGNkKXCFPUjK1TbJ
Lby3N9Q0gFZn1vvZrWmNK8YyBj0bggY7yyhLsQdpKa7AfSqg3K+7ty9DFKr9GwRWGxl4RHCy/hePX6DDMM8yq2dOkfpR7/MVuaRyVuUSGyhJ
9bPtx+GU5PsSbLxxtN/ofXSxnz+1xN5vzVx3lt5LlG2ZESqFg12c25ryWOAoVfnnGa0sl02VrUa4ahpNYUZ17JZPJqfMTs7MfuGY1Y4mQ0XE
mgtmqYzHJt/j+YMS3nxpRetDzuCG6Fb/Fc1xVfsm5e9H8AGc0Se15QZIg3HVjCEdLNiT97v5nPsgCUGiyu4ZK7nQfixtQHp4BW/ovttc
M7H+3pU8ZGJWKVLQtJgOLTxW2t9Fa2SBHSDeTj9o5+4Zv+2D+7PfTQ0Zvs5vXS08B5X2nHcMzLk5trx1YjsMWwL9Qi7FadDBbhHv2Xvw3Jk
z6FP2kY4g9Qe8Zh6xaeWZ4bdq6Mqrp9SyJaXYmqmH2hZTTsIMMQAqXN4dsbTXnM3N3CmOkJeS7TJuDTfTEC7JrDctgJMFfZHvgfC/MjB/g
vi4oTGvC2+ZCdhh2pDOYgN+PHSmwaKkmFPAKOcvBkuYSP8Axc3xOu0QXTYxo5ciTLa1m8bNJ9ZU3BjOuEfcR8LOzzqW9ZrAv2NTCdDl/wc
HcjTL5Qf4yYPpwMqprBVNhfMN3QZmvmjVb3KrHeHCTn0SfOBvcdnEDiMZo3Smom0cQYC3Fj+N6fuOaPYLHtEJ5fbntHrsXFdG8wwMDRPKEhG
bAyZtsv4QAHqctLd6zEqs1bbLNRxe7xEeCkJAibG8YnVbQLLzxl4bLoY3zfePowZLQLkzaLB7ghS84ibDU4rcXw0GEchEntamHEDm6ieHXDfm
ZdeLn6ytIqcEqRLKlXv4jD/b6DzBbf3pN11mP7SlegeVtM0KXy+pjUhBKHiNzkMl7G8EIm2Zb86Fc8J6NBwD4qjSmR4Hj2Fo2dVmF0qmQHvZ
xyBOThd8o6NbE0Dyr0eK5DfqjVsf/nCvTKA0VEjpDKUSMwZEbejemD71zO5GpkqRiKlMn8svnprZhNu07/N4CeFk8c9/XZnWHJZtGc09qf/
NfgFmp98uM5uA+9WobhWiS8PBu8y+ACFYpEv9OQMmRXdvPBqiH4aHajlC0pDZzTzp5wL2m5+mj1e4ewDAAAf+KOxM/QQ3Fod7V7YXuFr1aqn
tH9vh9sW0BKFrNsLv7XknHm9ALgAaq5wRdj5m9me3mz6z2ITPrNrN+M6v+x8gANFzesP9S0Pk7H5XbSo+6E7aYKXXfsKqOmA
3Re6+KOeyABQgV9jd/ai7n3X3UUm/krS42nZz8qNwA8Izq7mW2R9H1l0k6LpfsgdsKG6lc0A8gG0k3NSC9SqvsP6psS
S0LEmMaOrSGeMJsVhPxwJ3Z87yBsA8zXQWX/qxHdhf3vKdQaV98ktFQg6Hn8DRy+qpn2ltUbV4JkdrgXbx1cI3uHQnlrp0TjzDSO0XE3IqwdN
FunvNm4u5zj75zJgxI1m/8847q9Zq7StvNu7l4i/t3r07WV+6dLE1vNy3SQ3CB16MbmeGkp9hAIp1AmHqGdGkGLfCSuFrNIAHGW2QXXbP8AyDBZ0
AEIQdCIKwA0Fwiytk88UUXyTrF12Uvlg9sWf12MrY1n8cCSJaQAtzKrMIUaKCuqKqhuRbStrf3JgAqffbxJaemT7dG2O9lBIzsPj46aTVDgpctLVq3pmlPEf+DVx
dcbYrv22muT9XbvTdZv+6666rW6TrJbIrHq7x/W0ZQcmlLcOnilQVEVVlbwPB9+tJCKgWlAvu0r0+zyi0kB+vvQw4n6w9+b4pO8OsB8tROLdo+mg8hXx8+tcR1TNJ7sGVrbWEwAmyTYfmUbhx2CQIN+G9HTQ3xfRD7ZP5CtwAXX
LAgWe/p6UnW777777mT98mT9+eJurs2bPJ+rPxP8yPvxsvP++eJUrs2bPJ+tBQ+q+Xj46OFtnOThlPoBE9sWxHOKEtOxZHNKtxhVvNfZBY8tOxAEYQeCIOxAEIQdCIKwA0EQdiAIwg4EQdiBIAg7EARhB4IgZ4EwXX

2C0Cta9ldXeMNtFtR6zr63Llz87RUiP7+/mR97dq1yfq2bduKbGfSq7llN7M5ZrbHzAbM7G0zW5VNbzOzF83s3ex5ZuP
bBZDXRHbjRyT9o7svlPR3klaa2UJJD0ra5e5XStqVvQfQomqG3d2H3H1/9vqkpHckzZa0TNKm7GObJN3VqCYB1O+8jt
nNbK6k70naJ6nd3c/9OPlDSe1V5umW1J2/RQBFmPDZeDP7pqRnJf3M3U+MrXlldMhxB2109z53X+Tui+rqFEBdJhR2
M/uGKkH/jbs/l00+amazsvosScca0yKAItTcjTczk/SkpHfcfcOY0jZJYy563NqTDSaC9fdwjnC8tXLgwWX/88ceT9auu
uuq8eyrKvn37kvVHH320am3r1vQ/GW5RLdZEjtn/XtKPPJb1lZgeyaWtUCfnvzGyFpA8k/bAxLQIoQs2wu/t/Sxp3cHdJN
xfbDoBG4eeyQBCEHQiCsANBEHYgCMIOBMEtrhPU1tZWbdb25uct6Oj/I1mfN29erp6K8Mor
2Wfn3RMagy07EARhB4Ig7EAQhB0IAjCDgQR5jr79ddfn6yvXr06WV+8eHHV2zs3P1V2uzZs3P1VJRPP/20am3jxo3
JeR955JFk/fTp07l6Quthyw4EQdiBIAg7EARhB4Ig7EAQhB0IAjCDgQR5jr79ddfn6yvXr06WV+8eHHV2uzZs3P1VJRPP/20am3jxo3
JeR955JFk/fTp07l6Quthyw4EQdiBIAg7EARhB4Ig7EAQhB0IgrADQYS5zt7Z2VlXvXtXvR4DAwP/+vbt25P1P1kZGRZD11z/nw8
HByXsTBlh0IgrADQRB2IAjCDgRB2IEgCDsQBGEHgjB3T3/AbI6kzZLaJLmkXnd/xMweknSvpKGZ+0Pn6kk/NN7SNIDkv6cfXSNu/++xnelFwagbu4+
7qjLEwn7LEmz3H2Mf2/mU2X9Jqku1QZj/2Uuz82O0B41HbOb2vxWSaxHbOb2vxJ35O0L5LjLkSJbr3Zb7s+b5kfy+xmzJ/tlBV+XN6+4+
7qjLEwn7LEmz3H2/mU2X9Jqku1QZj/2Uuz820SYIO9B41cI+kfHZhyQNZa9Pmtk7ksr90ywAztt5HbOb2vxJ35O0L5vU
Y2ZvmtlTZjazyjzdZtZvZv11dQqgLjV347/8oNk3Jb0saa27P2dm7ZI+UuU4/p9V2dW/v8Z3sBsPNFjuY3ZJMrNvNvSNouaY
e7bxinPlfSdne/psb3EHagwaqFveZuvJmZpCclvTM26NmJu3M6JR2st0kAjTORs/FLJP1P0psDZ/jqnCqmdV7nb8NDV9dNbd6tzz/Hs/QPl18Xzt6Dvz2G/4hpGkZQc7uYb+4TNtoGpeOA5hMfhnK7xohm2EY0PS9V+5a5cx+3ZJLOO7kJ9jR
e7bxinPlfSdne/psb3EHagwaqFveZuvJmZpCclvTM26NmJu3M6JR2st0kAjTORs/FLJP2XpLckjWaT10jqktShym78YUk/
zU7mpb6LLTvQYHXtxheFsAONl3s3HsDkQNiBIAg7EARhB4Ig7EAQhB0IgrADQRB2IAjCDgRB2IEgCDsQBGEHgiDsQBC
EHQii2UM2fyTpgzHvL8/2tmUaW7a1V2V+5LoLa8ie/ubaoWm3s/+tYWb9bv7v7otIaSGjV3lq1L4Ne8mpWb+zGA0EQdiCIssP
eV/LyU1q1t1btS6K3vJrSW6nH7ACap+wtO4AmlexAEKWE3syWmtl+MztgZqvMd/kvpPQ+lIHeHrV+x9eq3Y/DHuaWs7kR2UNT8P9g6xzSmcNB29y2V77v8hZ5fQl1ni+Iec2FrX49l/7MmJ1TYcCllWpnlj+gfm2OqEPnpwpTutuZDHsBY11nsYybhvd/lgodUD9nhHJ6ns8ytzn+cJbnJRh4yIkVh2IDQBthN
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+AfzUYHQUKyprI2bVw0gcakQozDpk2GxJJJQsGr3VF
IamNjlEZNJE6VFFcqqqGjBpi51GaLdmDSOyCpKTCs3/cQzDpDk2GxJJJQszGr3VF
IamNjlEZNJE6VFFcqqGjBpi51GaLdmDSOyCpqW1mDFhwZUSNDTKTCs3/cQzPinPcM9557z4Hn+0km997zzLn38Tl/z7
nnPfe85u4CcPI7peoGAHQGYQeCIOxAEIQdCIKwA0EIOxAEIQdCIKwA0H8QydfzMw49A+0mbvbWMtb2rKb2ZVm9mcz22VmN7fyXADa
y5odZzezCZL+Iuk7kvzIelHSHSYnd/PbEOW3agzzdqxZZ8jaZe7v+XuhyStl7SghechecD0EathP1cX8d9d9XhPtuxzzKzXAbNbLCF
1wLQorYfoHP3fkn9ErvxQJVa2bLvlXTeqMfTs2UAaqiVsL8s3JkpaJGlzlTGhfGlzz5pZlZDj6nkdvpaJGlzOW0BKFvvTu/Hu/pmZLZO0RdIESWvc/
bXSOgNQqqaH3pp6M6MrtDHNH6dOW1Ow5c+gXvFC9xjvQa7kEQG4KjKwtv1oq6DgMbOdlGe6vN+ngdX+Lpmn
bXSOgNQqqaH3pp6MT6zA23XlpNqAJw4CDsQBGEHgiDsQBCEHQiCsANBEHYgCMIOBEHYgSAIOxAEYQeCIOxAEIQd
CIKwA0EQdiAIwg4EQdiBIAg7EARhB4Ig7EAQhB0IoqNTNuPkM2vWkrGR92bJlubWenp7kug8/
YcsOBEHYgSAIOxAEYQeCIOxAEIQdCIKwA0EwiyuSuru7k/WWbLG8W15ZkzZ2bX5Zf/xxsn7ff/xxsn7WWWe17bXbXrLG8W15ZOqj
Gz3ZJGJB2W9Jm7z27l+Q0P2zu8+v4XxkUtAtBGf27ZE9S/q/q9e/+8e6FS8XkrGf1vmpAtBEf27YEgWf27S/+q9b8d9/+18yyLT9dtbg8D10s+rSHBc3J/Zl4Tj
1v4XxkUtAtBGf27S/+8v4XkAtBGf2YEgWg27S/q9mb1kZr1j/YKZ9/j1k8X+lmpSFI+1b0xl78VZaZ0L2iuHOd6WVmGWbgF/
2vV94ZZf1Xc/8+6A1jDEW+xJjtRqYGkqGaHksWS2rHy3e+ep1k4PqW7...

zJnJetE4epEFCxYk60XTKgMSW3YgDMIOBEHYgSAIOxAEYQeCIOxAEIQdCIJx9hKsWrUqWTcb88q+f1c0Ts44enNOOS
V/W3bkyJEODdlIPbNmBIAg7EARhB4Ig7EAQhB0IgrADQRB2IAjG2cfp6quvzq5quvzq11d3cn1y2aHnjz5s1N9YS01Fh60b/Jjh07
ym6ncoVbdjNbY2bDZrZz1LKVVZrbXzHZkP61nQAA241nN/5Xkq4cY/kv3L07/lt/1++ldEulOPYtOeGAH3UygG
6ZWb2SrabPzXvl8ys18wGZSw96RiAtmo27KlfUNSt6QEgGVXgQjwUz1uRGQ/bbvVwdpsBQqHfPWkG1BQaQ55bYFo
GxNhd3MukyY9/J6knXXm/rGqQ3S37fsbcgXvo5t6dj4enDR7Y7LRyYlxLYdoLxQFxa7pR1p/fvOrT7utZ+R5Yya6E2QZIzsb5DWZ
VzTv/cqVK5t+7oGBBgWTW9lltuafq566ow7O6+eIzFD7WhFwBtxOmyQBBCEHQiCsANBEHYgCMIOBMFXXDvg008TaR/TdaHH
oY61Em9FA2t9fX1J5es33XRSr5r135570c0KUk6ePBKgs4iYssOBEHYgSAIOxAEYQeCIOxAEIQdCIKwA0EwzAsOBGJcqIpw
VnbrMdtE4+bXXXpusb9q0kVm/5pprkvqVK1fmvwE0CMs4NFGBe1RBqVQPZFEGJyDSOsClUCN+TmVFXnsv0+UrXXIQ5fUL

2+qpzq48cYbk/Vbb701tzZlypTkuuvWrUvWe3p6knV8Hlt2IAjCDgRB2IEgCDsQBGEHgiDsQBCEHQiCcfZxcvemapJ0zjn
nJOv33ntvsr5mzZpk/YMPPsitzZ07N7nu9ddfn6xffPHFyfr06dOT9XfeeSe3tmXLluS6999/f7KO41O4ZTez88xsm5m9b
mavmdnybPmZZvasmb2Z3U5tf7sAmjWe3fjPJP2bu39T0lxJPzKb58haWv2GEBNFYbd3YfcfXt2f0TSG5LO
lbRA0trotrmaW+mFnnXqxkqa9yPv7448l1L7300
pZeu+hS1a38G6a+HitJ69evT9ZP5Mtgn6zcfcw/GLbsQBCEHQiCsANBEHYgCMIOBEHYgSAIOxEEHYgCsNuNuZueZ2Z2
f19SXrrYyz33PPPcl1V69enazv2rUrWUf9MM4OBEfYgSAIOxAEYQeCIOxAEIQdCIKwA0EQdiAIwg4EQdiBIAg7EARhB4Ig7EAQhB0IgrADQRB2IAjCDgRB2IEgCDsQBGEHgiDsQBCEHQiCsANBEHYgCMIOBEH

```
    "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+WH4yJAAAKyklEQVR4nO3dX4ilhXnH8e8eJ/qpg0YalSK1pRQ1YG6IXIqahXYqwN2lrmkCi7uV
uK5QYL0pg0YalSKPdSBUJCXZjLlro1jFKUndj3JoaV1ZdQZuSm+5kn17Ma5luZzNnn5kz5536/cAw5/1z5n+Sw5x5Y
VJVSDp35816AGmzMh6pyXikJuORmoxHajIeqWlN8SS5KckLSY4G2g3R/z5NkC/AD4KPAceBp+NBnBvWT9xpP
Ga24Nz70aOFZVLwEk+SDp+SpC3DWePBGFJev4YjSLDzDm1W17czVa4lnJ/DKsuXjwG+cuVOSfcA+AH+bE
HrqHxhU1f6qmq+qef5Pu9LmtZZXnleBy5Yt7xrWTaaAReCna5hAmpbzgK1Azr7LWuJ5GrgyyW6WorkD+L2Jn70Ic4/
MseWft6xhBGk6Tv/qaU79wSm48Oz7tOOpqsUkfwR8E9gC/HVVPT/xD/gpbPmnLWz9m63dEaSpWbxxkVO3TykegK
r6OvD1tfwMabPyDgOpyXikJuORmoxHajIeqcl4pCbjkZqMR2oyHqnJeKQm45GajEdqMh6pyXikJuORmoxHajIeqcl4pC
bjkZqMR2oyHqnJeKQm45GajEdqMh6pyXikJuORmoxHajIeqcl4pCbjkZqMR2oyHqnJeKQm45GaVo0nyWVJnkpyJMn
zSe4a1l+S5MkkLw7fL57+uNJ4TPLKswh8tqr2ANcAn0qyB7gbOFRVVwKHhmXpXWPVeKrqRFV9Z3j8n8BRYCdwC3Bg
2O0AcOu0hpTG6Jze8yS5HLgKOAxsr6oTw6bXgO1nec6+JAtJFji5hkmlkZk4niQXAl8dPl1VP16+raoKqJWeV1X7q2q+q
ubZtqZZpVGZKJ4kW1kK56GqenRY/XqSHcP2HcAb0xlRGqdJPm0L8CBwtKq+uGzT48De4LH1H08ar7kJ9rkO+H3g
e0meGjZ9DuwBHltVvwEeGpaXbSm6fzojSOK0aT1X9I5CzbL5hfceRNg/vMJCajEdqMh6pyXikJuORmoxHajIeqcl4pCbjZqM
```

R2oyHqnJeKQm45GajEdqMh6pyXikJuORmoxHajIeqcl4pCbjkZqMR2oyHqnJeKQm45GajEdqMh6pyXikJuORmoxHajI
eqcl4pCbjkZqMR2o6lz8lvyXJs0meGJZ3Jzmc5FiSh5OcP70xpfE5l1eeu4Cjy5bvBe6rqiuAt4A713MwaewmiifJLuB3gQe
G5QDXAweHXQ4At05jQGmsJn3l+RLwOeD0sHwp8HZVLQ7Lx4GdKz0xyb4kC0kWOLmmWaVRWTWeJB8D3qiqZzoH
qKr9VTVfVfNs6/wEaZzmJtjnOuDjSW4G3gv8AnA/cFGSueHVZxfw6vTGlMZn1VeeqrqnqnZV1eXAHcC3quqTwFPAbc
Nue4HHpjalNEJr+T3P54HPJDnG0nugB9dnJGlzmOSy7X9U1beBbw+PXwKuXv+RpM3BOwykJuORmoxHajIeqcl4pCbj
kZqMR2oyHqnJeKQm45GajEdqMh6pyXikJuORmoxHajIeqcl4pCbjkZqMR2oyHqnJeKQm45GajEdqMh6pyXikJuORm
oxHajIeqcl4pCbjkZqMR2oyHqnJeKQm45GajEdqmiieJBclOZjk+0mOJrk2ySVJnkzy4vD94mkPK43JpK889wPfqKoPAR8
GjgJ3A4eq6krg0LAsvWusGk+S9wO/xfCn4qvqv6rqbeAW4MCw2wHg1mkNKY3RJK88u4GTwFeSPJvkgSQXANur6sS
wz2vA9pWenGRfkoUkC5xcn6GlMZgknjngI8CXq+oq4+Ceccc+YlWVQXUSk+uqv1VNV9V82xb67jSeEwSz3HgeFUdHpYPs
hTT60l2AAzf35jOiNI4rRpPVb0GvJLkg8OqG4AjwOPA3mHdXuCxqUwojdTchPv9MfBQkvOBl4A/ZCm8R5LcCbwM3D
6dEaVxmiieqnoOmF9h0w3rO460eXiHgdRkPFKT8UhNxiM1GY/UZDxSk/FITcYjNRmP1GQ8UpPxSE3GIzUZj9RkPFKT8
UhNxiM1GY/UZDxSk/FITcYjNRmP1GQ8UpPxSE3GIzUZj9RkPFKT8UhNxiM1GY/UZDxSk/FITcYjNRmP1GQ8UtNE8ST
50yTPJ/nXJH+b5L1Jdic5nORYkoeHv1cqvWusGk+SncCfAPNV9RvAVuAO4D7gC1X1AeCnwJ3THFQam0lP2+aAX0syB7wPOAF8Ani02H7gvyYaWxmCSy7aLgVuA3cAvAhc
AN016gKraX1XzVTXPtvac0uhMctl2I/DDqjpZVaeAR4HrgIuGyziAXcCrU5AU5pRGqVJ4vkRcE2S9yUJcANwBHgKuG3YZy/w
2HRGlMZpkvc8h1n6YOA7wPeG5+wHPgP8WZJjwCXAg1QKY3O3Oq7QFV9AfjCGatfAq5e94mkTcI7DKQm45GajEdq
Mh6pyXikJuORmoxHajIeqcl4pCbjkZqMR2oyHqnJeKQm45GajEdqMh6pyXikJuORmoxHajIeqcl4pCbjkZqMR2oyHqnJ
eKQm45GajEdqMh6pyXikJuORmoxHajIeqcl4pCbjkZom+rOKU3EenN5zmsUbF2c2gnQ2p686DVt/9j6zi2crnNp7ilN3
nJrZCNJZbQV+/mfvMrt4wtJwqwwojVWqauMOlpwEfgK8uWEHXZsPsHlmhc0170ae9a
dpMs8LmmnczzXo2ftomNRmP1DSLePbP4Jhdm2lZW2FzzbqZZV7Th73mk/y+8bJOajEdq2rB4ktyU5IUkx5LcvVHHnVS
Sy5I8leRIkueT3DWsvsyTJk0leHL5fPOtZ35FkSS5JnkzwxLO9Ocng4xw8nOX/WM74jyUVJDib5fpKjSa4d87mdxIbEk2QL8
FfA7wB7gE8k2bMRxz4Hi8Bnq2oPcA3wqWHWHGu4FDVXUlcGhYHou7gKPLlu8F7quqK4C3gDtnMtXK7ge+UVUfAj4M0t
xjPrerq6qpfwHXAt9ctn9ctnwPcM9GHHsNMz8GfBR4Adg+AdgxrNsBvDDDr2mdxSMMz8GfBR4AdgxrNsBvDDr2YZZdrH0D+564AmWbnh6E5hb6ZzPeNb3Az9k+IB
q2fpRnttJvzbqsm0n8Mqy5ePDulFKcjlwFKcjlwFXAY2F5VJ4ZNrwHbZzTWmb4EfA44PSxfCrxdVe/cpj6mc7wbOAl8ZbjMfCDJ
BYz33E7EDwzOkORC4GvAp6vqf5vqx8u8Ap6vqx8u31dJ/kTP/bD/Jx4A3quuZWc8yoTngI8CXq+oqlu5v/F+XaGM5t+dio+J5Fbhs2fKuY
d2oJNnKUjgPVdWjw+rXk+wYtu8A3pjVfMtcB3w8yb8ybDX2Xp0u1+4KIk79wpP6ZzfBw4XlWHh+WDLMU0xnM7sY2K5
2ngyuHToPOBO4DHN+jYE0kS4EHgaFV9cdmmx4Gl9g4xV9cdmmx4G9w+O9LL0XmqmquqeqdlXV5Sydy29V1SeBp4Dbht1GMStAVb0G
vJLkg8OqG4AjjPDcnpMNfNN4M/AD4N+AP5/1m/dLzPx2wBvAP5N/1m70V5vtNli4bvgs8N3A/A/wIcA
/4OeM+s51s2568DC8P5/Xvg4rGf29W+vD1HavIDA6nJeKQm45GajEdqMh6pyXikJuORmv4bWtmd36znqt4AAAAAS
UVORK5CYII=\n"
      },
      "metadata": {
        "needs_background": "light"
      }
    }
   ]
  },
  {
   "cell_type": "code",
   "source": [
    "model.save(\"models/mnistCNN.h5\")"
   ],
   "metadata": {
    "id": "CBPDB-RXH-60"

```
    },
    "execution_count": 22,
    "outputs": []
   }
  ]
}
```