{

"nbformat": 4,

"nbformat\_minor": 0,

"metadata": {

"colab": {

"provenance": []

},

"kernelspec": {

"name": "python3",

"display\_name": "Python 3"

},

"language\_info": {

"name": "python"

}

},

"cells": [

{

"cell\_type": "code",

"execution\_count": null,

"metadata": {

"id": "XoVJrne9TkbL"

},

"outputs": [],

"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"

]

},

{

"cell\_type": "code",

"source": [

"(X\_train, y\_train), (X\_test, y\_test) = mnist.load\_data()"

],

"metadata": {

"colab": {

"base\_uri": "https://localhost:8080/"

},

"id": "ldD5h1szUK6I",

"outputId": "9fedc791-bc60-42e8-ac27-0e8e1eaadcbe"

},

"execution\_count": null,

"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": "L8Ak9kITUWhq",

"outputId": "adc691b0-899f-4caa-b8c1-210424624d82"

},

"execution\_count": null,

"outputs": [

{

"output\_type": "display\_data",

"data": {

"text/plain": [

"<Figure size 432x288 with 1 Axes>"

],

"image/png": "\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": "DvJFrCEhVk02",

"outputId": "87b9b496-598e-4b00-da51-514881de14d2"

},

"execution\_count": null,

"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": "markdown",

"source": [

"# RESHAPING SO AS TO CONVERT IMAGES FOR OUR MODEL"

],

"metadata": {

"id": "HexABigsVxOQ"

}

},

{

"cell\_type": "code",

"source": [

"X\_train = X\_train.reshape(60000, 28, 28, 1)\n",

"X\_test = X\_test.reshape(10000, 28, 28, 1)"

],

"metadata": {

"id": "wpgL9Gy1V9J\_"

},

"execution\_count": null,

"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": "1B6JX4a\_Vugq",

"outputId": "886fefde-431b-493e-917f-458e3293b2d6"

},

"execution\_count": null,

"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": "markdown",

"source": [

"# ONE HOT ENCODING"

],

"metadata": {

"id": "2QrfMo7GWSLk"

}

},

{

"cell\_type": "code",

"source": [

"y\_train = to\_categorical(y\_train)\n",

"y\_test = to\_categorical(y\_test)"

],

"metadata": {

"id": "rrt8\_nwlWYk7"

},

"execution\_count": null,

"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": "YVJMmrnqWhKh"

},

"execution\_count": null,

"outputs": []

},

{

"cell\_type": "code",

"source": [

"model.compile(optimizer='adam', loss='categorical\_crossentropy', metrics=['accuracy'])"

],

"metadata": {

"id": "APVrzCu\_Ythp"

},

"execution\_count": 9,

"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": "q6hvgv\_EZcIY",

"outputId": "dc1ac1d3-e9df-4c58-de30-5a37884f4304"

},

"execution\_count": 10,

"outputs": [

{

"output\_type": "stream",

"name": "stdout",

"text": [

"Epoch 1/3\n",

"1875/1875 [==============================] - 88s 46ms/step - loss: 0.9602 - accuracy: 0.7588 - val\_loss: 0.1421 - val\_accuracy: 0.9617\n",

"Epoch 2/3\n",

"1875/1875 [==============================] - 83s 44ms/step - loss: 0.2960 - accuracy: 0.9122 - val\_loss: 0.0714 - val\_accuracy: 0.9779\n",

"Epoch 3/3\n",

"1875/1875 [==============================] - 81s 43ms/step - loss: 0.2110 - accuracy: 0.9369 - val\_loss: 0.0559 - val\_accuracy: 0.9836\n"

]

},

{

"output\_type": "execute\_result",

"data": {

"text/plain": [

"<keras.callbacks.History at 0x7f1a7f4d3d50>"

]

},

"metadata": {},

"execution\_count": 10

}

]

},

{

"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": 577

},

"id": "GBo0\_v2XfO1y",

"outputId": "a158d421-3cfa-4d1a-ee04-b51878cb59f0"

},

"execution\_count": 11,

"outputs": [

{

"output\_type": "stream",

"name": "stdout",

"text": [

"1/1 [==============================] - 0s 114ms/step\n",

"Prediction (Softmax) from the neural network:\n",

"\n",

" [[9.9999928e-01 3.4740781e-13 1.6311596e-08 2.2861069e-11 1.0698914e-12\n",

" 1.6004112e-10 1.7066007e-07 7.1788651e-11 4.5166394e-07 2.4545757e-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": "\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": "mgpSyODvgaua",

"outputId": "5da7fd3c-0bcf-47b8-c80b-ade4a8b8ff6b"

},

"execution\_count": 12,

"outputs": [

{

"output\_type": "stream",

"name": "stdout",

"text": [

"Metrices(test loss and Test Accuracy):\n",

"[0.055905986577272415, 0.9836000204086304]\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": "BPCKgBr1hRP5",

"outputId": "9709874b-953c-4fb2-e385-ff5d63382236"

},

"execution\_count": 13,

"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": "\n"

},

"metadata": {

"needs\_background": "light"

}

}

]

},

{

"cell\_type": "code",

"source": [

"model.save(\"models/mnistCNN.h5\")"

],

"metadata": {

"id": "KaGh8rPyh2Oh"

},

"execution\_count": 14,

"outputs": []

}

]

}