{

"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": "i9C8lTkpLGJ\_"

},

"outputs": [],

"source": [

"import cv2\n",

"import numpy as np\n",

"import tensorflow as tf\n",

"import matplotlib.pyplot as plt\n",

"from keras.datasets import mnist, cifar10\n",

"from keras.utils import to\_categorical\n",

"from keras.models import Sequential\n",

"from keras.layers import Conv2D, MaxPooling2D, Flatten, Dense, Dropout\n",

"from sklearn.metrics import classification\_report, confusion\_matrix, roc\_curve, auc\n",

"from PIL import Image\n",

"\n",

"# Task 1: Image Compression\n",

"\n",

"def compress\_image(input\_path, output\_path, quality=50):\n",

" img = Image.open(input\_path)\n",

" img.save(output\_path, \"JPEG\", quality=quality)\n",

" print(f\"Image compressed and saved as {output\_path}\")\n",

"\n",

"def compress\_png(input\_path, output\_path):\n",

" img = Image.open(input\_path)\n",

" img.save(output\_path, \"PNG\")\n",

" print(f\"PNG compressed image saved as {output\_path}\")"

]

},

{

"cell\_type": "code",

"source": [

"# Task 2: CNN Implementation\n",

"def preprocess\_data(dataset):\n",

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

" X\_train, X\_test = X\_train / 255.0, X\_test / 255.0\n",

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

" return (X\_train, y\_train), (X\_test, y\_test)\n",

"\n",

"# Define CNN Model\n",

"def build\_cnn(input\_shape, num\_classes):\n",

" model = Sequential([\n",

" Conv2D(32, (3,3), activation='relu', input\_shape=input\_shape),\n",

" MaxPooling2D((2,2)),\n",

" Conv2D(64, (3,3), activation='relu'),\n",

" MaxPooling2D((2,2)),\n",

" Flatten(),\n",

" Dense(128, activation='relu'),\n",

" Dropout(0.5),\n",

" Dense(num\_classes, activation='softmax')\n",

" ])\n",

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

" return model"

],

"metadata": {

"id": "FhVeQACRLeQw"

},

"execution\_count": null,

"outputs": []

},

{

"cell\_type": "code",

"source": [

"# Train and Evaluate CNN\n",

"def train\_evaluate\_cnn(dataset, input\_shape, num\_classes):\n",

" (X\_train, y\_train), (X\_test, y\_test) = preprocess\_data(dataset)\n",

" model = build\_cnn(input\_shape, num\_classes)\n",

" history = model.fit(X\_train, y\_train, epochs=5, validation\_data=(X\_test, y\_test))\n",

"\n",

" y\_pred = model.predict(X\_test)\n",

" y\_pred\_classes = np.argmax(y\_pred, axis=1)\n",

" y\_true = np.argmax(y\_test, axis=1)\n",

"\n",

" print(\"Classification Report:\")\n",

" print(classification\_report(y\_true, y\_pred\_classes))\n",

" print(\"Confusion Matrix:\")\n",

" print(confusion\_matrix(y\_true, y\_pred\_classes))\n",

"\n",

" fpr, tpr, \_ = roc\_curve(y\_test.ravel(), y\_pred.ravel())\n",

" auc\_score = auc(fpr, tpr)\n",

" plt.plot(fpr, tpr, label=f'AUC = {auc\_score:.2f}')\n",

" plt.xlabel('False Positive Rate')\n",

" plt.ylabel('True Positive Rate')\n",

" plt.legend()\n",

" plt.show()\n"

],

"metadata": {

"id": "NF8RQLODLyX7"

},

"execution\_count": null,

"outputs": []

},

{

"cell\_type": "code",

"source": [

"# Run Image Compression\n",

"def main():\n",

" compress\_image('Flower.jpg', 'compressed.jpg', quality=50)\n",

" compress\_png('Flower.jpg', 'compressed.png')\n",

"\n",

" # CNN Training on MNIST\n",

" print(\"Training CNN on MNIST...\")\n",

" train\_evaluate\_cnn(mnist, (28, 28, 1), 10)\n",

"\n",

" # CNN Training on CIFAR-10\n",

" print(\"Training CNN on CIFAR-10...\")\n",

" train\_evaluate\_cnn(cifar10, (32, 32, 3), 10)\n",

"\n",

"if \_\_name\_\_ == \"\_\_main\_\_\":\n",

" main()"

],

"metadata": {

"colab": {

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

"height": 1000

},

"id": "kBK\_L7zIL1mR",

"outputId": "c0837bc9-ceea-49ed-9e70-410ab02b888b"

},

"execution\_count": null,

"outputs": [

{

"output\_type": "stream",

"name": "stdout",

"text": [

"Image compressed and saved as compressed.jpg\n",

"PNG compressed image saved as compressed.png\n",

"Training CNN on MNIST...\n",

"Epoch 1/5\n",

"\u001b[1m1875/1875\u001b[0m \u001b[32m━━━━━━━━━━━━━━━━━━━━\u001b[0m\u001b[37m\u001b[0m \u001b[1m64s\u001b[0m 33ms/step - accuracy: 0.8704 - loss: 0.4104 - val\_accuracy: 0.9859 - val\_loss: 0.0424\n",

"Epoch 2/5\n",

"\u001b[1m1875/1875\u001b[0m \u001b[32m━━━━━━━━━━━━━━━━━━━━\u001b[0m\u001b[37m\u001b[0m \u001b[1m82s\u001b[0m 33ms/step - accuracy: 0.9770 - loss: 0.0779 - val\_accuracy: 0.9900 - val\_loss: 0.0311\n",

"Epoch 3/5\n",

"\u001b[1m1875/1875\u001b[0m \u001b[32m━━━━━━━━━━━━━━━━━━━━\u001b[0m\u001b[37m\u001b[0m \u001b[1m80s\u001b[0m 32ms/step - accuracy: 0.9840 - loss: 0.0552 - val\_accuracy: 0.9901 - val\_loss: 0.0280\n",

"Epoch 4/5\n",

"\u001b[1m1875/1875\u001b[0m \u001b[32m━━━━━━━━━━━━━━━━━━━━\u001b[0m\u001b[37m\u001b[0m \u001b[1m81s\u001b[0m 31ms/step - accuracy: 0.9864 - loss: 0.0433 - val\_accuracy: 0.9920 - val\_loss: 0.0234\n",

"Epoch 5/5\n",

"\u001b[1m1875/1875\u001b[0m \u001b[32m━━━━━━━━━━━━━━━━━━━━\u001b[0m\u001b[37m\u001b[0m \u001b[1m60s\u001b[0m 32ms/step - accuracy: 0.9885 - loss: 0.0364 - val\_accuracy: 0.9931 - val\_loss: 0.0219\n",

"\u001b[1m313/313\u001b[0m \u001b[32m━━━━━━━━━━━━━━━━━━━━\u001b[0m\u001b[37m\u001b[0m \u001b[1m3s\u001b[0m 9ms/step\n",

"Classification Report:\n",

" precision recall f1-score support\n",

"\n",

" 0 0.99 1.00 1.00 980\n",

" 1 0.99 1.00 1.00 1135\n",

" 2 0.99 1.00 0.99 1032\n",

" 3 1.00 1.00 1.00 1010\n",

" 4 0.99 0.99 0.99 982\n",

" 5 0.99 0.99 0.99 892\n",

" 6 1.00 0.98 0.99 958\n",

" 7 0.99 0.99 0.99 1028\n",

" 8 0.99 0.99 0.99 974\n",

" 9 0.99 0.99 0.99 1009\n",

"\n",

" accuracy 0.99 10000\n",

" macro avg 0.99 0.99 0.99 10000\n",

"weighted avg 0.99 0.99 0.99 10000\n",

"\n",

"Confusion Matrix:\n",

"[[ 978 0 0 0 0 0 0 1 1 0]\n",

" [ 0 1133 1 0 0 0 1 0 0 0]\n",

" [ 0 0 1028 0 0 0 0 3 1 0]\n",

" [ 0 0 1 1005 0 2 0 1 1 0]\n",

" [ 0 0 0 0 974 0 0 1 1 6]\n",

" [ 0 0 0 3 0 885 1 1 1 1]\n",

" [ 5 3 1 0 2 5 941 0 1 0]\n",

" [ 0 3 3 0 0 0 0 1019 0 3]\n",

" [ 1 0 2 0 1 1 0 0 966 3]\n",

" [ 0 1 0 0 2 1 0 2 1 1002]]\n"

]

},

{

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

"data": {

"text/plain": [

"<Figure size 640x480 with 1 Axes>"

],

"image/png": "\n"

},

"metadata": {}

},

{

"output\_type": "stream",

"name": "stdout",

"text": [

"Training CNN on CIFAR-10...\n",

"Downloading data from https://www.cs.toronto.edu/~kriz/cifar-10-python.tar.gz\n",

"\u001b[1m170498071/170498071\u001b[0m \u001b[32m━━━━━━━━━━━━━━━━━━━━\u001b[0m\u001b[37m\u001b[0m \u001b[1m2s\u001b[0m 0us/step\n"

]

},

{

"output\_type": "stream",

"name": "stderr",

"text": [

"/usr/local/lib/python3.11/dist-packages/keras/src/layers/convolutional/base\_conv.py:107: UserWarning: Do not pass an `input\_shape`/`input\_dim` argument to a layer. When using Sequential models, prefer using an `Input(shape)` object as the first layer in the model instead.\n",

" super().\_\_init\_\_(activity\_regularizer=activity\_regularizer, \*\*kwargs)\n"

]

},

{

"output\_type": "stream",

"name": "stdout",

"text": [

"Epoch 1/5\n",

"\u001b[1m1563/1563\u001b[0m \u001b[32m━━━━━━━━━━━━━━━━━━━━\u001b[0m\u001b[37m\u001b[0m \u001b[1m76s\u001b[0m 48ms/step - accuracy: 0.3101 - loss: 1.8525 - val\_accuracy: 0.5222 - val\_loss: 1.3770\n",

"Epoch 2/5\n",

"\u001b[1m1563/1563\u001b[0m \u001b[32m━━━━━━━━━━━━━━━━━━━━\u001b[0m\u001b[37m\u001b[0m \u001b[1m72s\u001b[0m 46ms/step - accuracy: 0.4996 - loss: 1.3870 - val\_accuracy: 0.6006 - val\_loss: 1.1562\n",

"Epoch 3/5\n",

"\u001b[1m1563/1563\u001b[0m \u001b[32m━━━━━━━━━━━━━━━━━━━━\u001b[0m\u001b[37m\u001b[0m \u001b[1m81s\u001b[0m 46ms/step - accuracy: 0.5679 - loss: 1.2292 - val\_accuracy: 0.6365 - val\_loss: 1.0580\n",

"Epoch 4/5\n",

"\u001b[1m1563/1563\u001b[0m \u001b[32m━━━━━━━━━━━━━━━━━━━━\u001b[0m\u001b[37m\u001b[0m \u001b[1m84s\u001b[0m 47ms/step - accuracy: 0.5970 - loss: 1.1443 - val\_accuracy: 0.6449 - val\_loss: 1.0111\n",

"Epoch 5/5\n",

"\u001b[1m1563/1563\u001b[0m \u001b[32m━━━━━━━━━━━━━━━━━━━━\u001b[0m\u001b[37m\u001b[0m \u001b[1m80s\u001b[0m 46ms/step - accuracy: 0.6157 - loss: 1.0798 - val\_accuracy: 0.6681 - val\_loss: 0.9497\n",

"\u001b[1m313/313\u001b[0m \u001b[32m━━━━━━━━━━━━━━━━━━━━\u001b[0m\u001b[37m\u001b[0m \u001b[1m5s\u001b[0m 16ms/step\n",

"Classification Report:\n",

" precision recall f1-score support\n",

"\n",

" 0 0.69 0.72 0.70 1000\n",

" 1 0.84 0.74 0.79 1000\n",

" 2 0.61 0.44 0.51 1000\n",

" 3 0.47 0.49 0.48 1000\n",

" 4 0.65 0.56 0.60 1000\n",

" 5 0.55 0.62 0.59 1000\n",

" 6 0.72 0.76 0.74 1000\n",

" 7 0.71 0.74 0.72 1000\n",

" 8 0.76 0.79 0.77 1000\n",

" 9 0.69 0.81 0.75 1000\n",

"\n",

" accuracy 0.67 10000\n",

" macro avg 0.67 0.67 0.67 10000\n",

"weighted avg 0.67 0.67 0.67 10000\n",

"\n",

"Confusion Matrix:\n",

"[[718 13 42 31 9 9 8 14 111 45]\n",

" [ 29 737 4 18 2 4 8 9 32 157]\n",

" [ 75 7 444 75 117 128 81 41 14 18]\n",

" [ 23 6 48 494 60 204 72 50 17 26]\n",

" [ 36 2 63 99 557 41 75 107 12 8]\n",

" [ 17 2 46 179 28 622 22 57 14 13]\n",

" [ 9 12 34 76 36 30 764 11 9 19]\n",

" [ 20 2 29 41 42 77 11 744 6 28]\n",

" [ 81 33 10 14 4 4 6 8 793 47]\n",

" [ 32 63 5 17 8 6 7 13 41 808]]\n"

]

},

{

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

"data": {

"text/plain": [

"<Figure size 640x480 with 1 Axes>"

],

"image/png": "\n"

},

"metadata": {}

}

]

}

]

}