{  
 "nbformat": *4*,  
 "nbformat\_minor": *0*,  
 "metadata": {  
 "colab": {  
 "provenance": []  
 },  
 "kernelspec": {  
 "name": "python3",  
 "display\_name": "Python 3"  
 },  
 "language\_info": {  
 "name": "python"  
 },  
 "accelerator": "GPU",  
 "gpuClass": "standard"  
 },  
 "cells": [  
 {  
 "cell\_type": "code",  
 "source": [  
 "import numpy as np*\n*",  
 "from keras.datasets import cifar10*\n*",  
 "from keras.models import Sequential*\n*",  
 "from keras.layers import Dense, Dropout, Flatten*\n*",  
 "from keras.constraints import maxnorm*\n*",  
 "from keras.optimizers import SGD*\n*",  
 "from keras.layers.convolutional import Conv2D, MaxPooling2D*\n*",  
 "from keras.utils import np\_utils"  
 ],  
 "metadata": {  
 "id": "MtZG9dJzw4ld"  
 },  
 "execution\_count": *5*,  
 "outputs": []  
 },  
 {  
 "cell\_type": "code",  
 "source": [  
 "np.random.seed(7)"  
 ],  
 "metadata": {  
 "id": "n4lci1f3w8hC"  
 },  
 "execution\_count": *6*,  
 "outputs": []  
 },  
 {  
 "cell\_type": "code",  
 "source": [  
 "(X\_train, y\_train), (X\_test, y\_test) = cifar10.load\_data()"  
 ],  
 "metadata": {  
 "id": "m4bzJoVOxBX\_"  
 },  
 "execution\_count": *7*,  
 "outputs": []  
 },  
 {  
 "cell\_type": "code",  
 "source": [  
 "X\_train = X\_train.astype('float32') / 255.0*\n*",  
 "X\_test = X\_test.astype('float32') / 255.0"  
 ],  
 "metadata": {  
 "id": "3DvFG935xFI2"  
 },  
 "execution\_count": *8*,  
 "outputs": []  
 },  
 {  
 "cell\_type": "code",  
 "source": [  
 "y\_train = np\_utils.to\_categorical(y\_train)*\n*",  
 "y\_test = np\_utils.to\_categorical(y\_test)*\n*",  
 "num\_classes = y\_test.shape[1]"  
 ],  
 "metadata": {  
 "id": "i\_R7uAN4xIrm"  
 },  
 "execution\_count": *9*,  
 "outputs": []  
 },  
 {  
 "cell\_type": "code",  
 "source": [  
 "model = Sequential()*\n*",  
 "model.add(Conv2D(32, (3, 3), input\_shape=(32, 32, 3), padding='same', activation='relu', kernel\_constraint=maxnorm(3)))*\n*",  
 "model.add(Dropout(0.2))*\n*",  
 "model.add(Conv2D(32, (3, 3), activation='relu', padding='same', kernel\_constraint=maxnorm(3)))*\n*",  
 "model.add(MaxPooling2D(pool\_size=(2, 2), padding='same'))*\n*",  
 "model.add(Flatten())*\n*",  
 "model.add(Dense(512, activation='relu', kernel\_constraint=maxnorm(3)))*\n*",  
 "model.add(Dropout(0.5))*\n*",  
 "model.add(Dense(num\_classes, activation='softmax'))*\n*"  
 ],  
 "metadata": {  
 "id": "z\_NGevkcxMSf"  
 },  
 "execution\_count": *10*,  
 "outputs": []  
 },  
 {  
 "cell\_type": "code",  
 "source": [  
 "sgd = SGD(learning\_rate=0.01, momentum=0.9, decay=1e-6)*\n*",  
 "model.compile(loss='categorical\_crossentropy', optimizer=sgd, metrics=['accuracy'])*\n*",  
 "print(model.summary())"  
 ],  
 "metadata": {  
 "colab": {  
 "base\_uri": "https://localhost:8080/"  
 },  
 "id": "rhzjj8FaxN77",  
 "outputId": "0b033a14-600d-4cf1-8515-ce5515d02091"  
 },  
 "execution\_count": *11*,  
 "outputs": [  
 {  
 "output\_type": "stream",  
 "name": "stdout",  
 "text": [  
 "Model: *\"*sequential\_1*\"\n*",  
 "\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_*\n*",  
 " Layer (type) Output Shape Param # *\n*",  
 "=================================================================*\n*",  
 " conv2d\_2 (Conv2D) (None, 32, 32, 32) 896 *\n*",  
 " *\n*",  
 " dropout\_2 (Dropout) (None, 32, 32, 32) 0 *\n*",  
 " *\n*",  
 " conv2d\_3 (Conv2D) (None, 32, 32, 32) 9248 *\n*",  
 " *\n*",  
 " max\_pooling2d\_1 (MaxPooling (None, 16, 16, 32) 0 *\n*",  
 " 2D) *\n*",  
 " *\n*",  
 " flatten\_1 (Flatten) (None, 8192) 0 *\n*",  
 " *\n*",  
 " dense\_2 (Dense) (None, 512) 4194816 *\n*",  
 " *\n*",  
 " dropout\_3 (Dropout) (None, 512) 0 *\n*",  
 " *\n*",  
 " dense\_3 (Dense) (None, 10) 5130 *\n*",  
 " *\n*",  
 "=================================================================*\n*",  
 "Total params: 4,210,090*\n*",  
 "Trainable params: 4,210,090*\n*",  
 "Non-trainable params: 0*\n*",  
 "\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_*\n*",  
 "None*\n*"  
 ]  
 }  
 ]  
 },  
 {  
 "cell\_type": "code",  
 "source": [  
 "epochs = 5*\n*",  
 "batch\_size = 32*\n*",  
 "model.fit(X\_train, y\_train, validation\_data=(X\_test, y\_test), epochs=epochs, batch\_size=batch\_size)*\n*"  
 ],  
 "metadata": {  
 "colab": {  
 "base\_uri": "https://localhost:8080/"  
 },  
 "id": "COpaIeenxXuH",  
 "outputId": "89ce2eea-a785-4f62-d7bc-567c0b2c98af"  
 },  
 "execution\_count": *12*,  
 "outputs": [  
 {  
 "output\_type": "stream",  
 "name": "stdout",  
 "text": [  
 "Epoch 1/5*\n*",  
 "1563/1563 [==============================] - 19s 7ms/step - loss: 1.7232 - accuracy: 0.3746 - val\_loss: 1.4776 - val\_accuracy: 0.4563*\n*",  
 "Epoch 2/5*\n*",  
 "1563/1563 [==============================] - 10s 6ms/step - loss: 1.3675 - accuracy: 0.5117 - val\_loss: 1.2470 - val\_accuracy: 0.5551*\n*",  
 "Epoch 3/5*\n*",  
 "1563/1563 [==============================] - 10s 6ms/step - loss: 1.2071 - accuracy: 0.5716 - val\_loss: 1.1232 - val\_accuracy: 0.6047*\n*",  
 "Epoch 4/5*\n*",  
 "1563/1563 [==============================] - 10s 7ms/step - loss: 1.0855 - accuracy: 0.6136 - val\_loss: 1.1554 - val\_accuracy: 0.5928*\n*",  
 "Epoch 5/5*\n*",  
 "1563/1563 [==============================] - 10s 7ms/step - loss: 0.9709 - accuracy: 0.6583 - val\_loss: 0.9986 - val\_accuracy: 0.6550*\n*"  
 ]  
 },  
 {  
 "output\_type": "execute\_result",  
 "data": {  
 "text/plain": [  
 "<keras.callbacks.History at 0x7f689d6d65e0>"  
 ]  
 },  
 "metadata": {},  
 "execution\_count": *12* }  
 ]  
 },  
 {  
 "cell\_type": "code",  
 "source": [  
 "scores = model.evaluate(X\_test, y\_test, verbose=0)*\n*",  
 "print(*\"*Accuracy: %.2f%%*\"* % (scores[1]\*100))"  
 ],  
 "metadata": {  
 "colab": {  
 "base\_uri": "https://localhost:8080/"  
 },  
 "id": "gymoyEPsxpd3",  
 "outputId": "ac10174d-d62f-4f23-9766-43fe8b545e9a"  
 },  
 "execution\_count": *13*,  
 "outputs": [  
 {  
 "output\_type": "stream",  
 "name": "stdout",  
 "text": [  
 "Accuracy: 65.50%*\n*"  
 ]  
 }  
 ]  
 },  
 {  
 "cell\_type": "code",  
 "source": [  
 "import numpy as np*\n*",  
 "from keras.datasets import cifar10*\n*",  
 "from keras.models import Sequential*\n*",  
 "from keras.layers import Dense, Dropout, Flatten*\n*",  
 "from keras.layers.convolutional import Conv2D, MaxPooling2D*\n*",  
 "from keras.constraints import maxnorm*\n*",  
 "from keras.utils import np\_utils*\n*",  
 "from keras.optimizers import SGD*\n*",  
 "*\n*",  
 "# Fix random seed for reproducibility*\n*",  
 "np.random.seed(7)*\n*",  
 "*\n*",  
 "# Load data*\n*",  
 "(X\_train, y\_train), (X\_test, y\_test) = cifar10.load\_data()*\n*",  
 "*\n*",  
 "# Normalize inputs from 0-255 to 0.0-1.0*\n*",  
 "X\_train = X\_train.astype('float32') / 255.0*\n*",  
 "X\_test = X\_test.astype('float32') / 255.0*\n*",  
 "*\n*",  
 "# One hot encode outputs*\n*",  
 "y\_train = np\_utils.to\_categorical(y\_train)*\n*",  
 "y\_test = np\_utils.to\_categorical(y\_test)*\n*",  
 "num\_classes = y\_test.shape[1]*\n*",  
 "*\n*",  
 "# Create the model*\n*",  
 "model = Sequential()*\n*",  
 "model.add(Conv2D(32, (3, 3), input\_shape=(32, 32, 3), padding='same', activation='relu', kernel\_constraint=maxnorm(3)))*\n*",  
 "model.add(Dropout(0.2))*\n*",  
 "model.add(Conv2D(32, (3, 3), activation='relu', padding='same', kernel\_constraint=maxnorm(3)))*\n*",  
 "model.add(MaxPooling2D(pool\_size=(2, 2)))*\n*",  
 "model.add(Conv2D(64, (3, 3), activation='relu', padding='same', kernel\_constraint=maxnorm(3)))*\n*",  
 "model.add(Dropout(0.2))*\n*",  
 "model.add(Conv2D(64, (3, 3), activation='relu', padding='same', kernel\_constraint=maxnorm(3)))*\n*",  
 "model.add(MaxPooling2D(pool\_size=(2, 2)))*\n*",  
 "model.add(Conv2D(128, (3, 3), activation='relu', padding='same', kernel\_constraint=maxnorm(3)))*\n*",  
 "model.add(Dropout(0.2))*\n*",  
 "model.add(Conv2D(128, (3, 3), activation='relu', padding='same', kernel\_constraint=maxnorm(3)))*\n*",  
 "model.add(MaxPooling2D(pool\_size=(2, 2)))*\n*",  
 "model.add(Flatten())*\n*",  
 "model.add(Dropout(0.2))*\n*",  
 "model.add(Dense(1024, activation='relu', kernel\_constraint=maxnorm(3)))*\n*",  
 "model.add(Dropout(0.2))*\n*",  
 "model.add(Dense(512, activation='relu', kernel\_constraint=maxnorm(3)))*\n*",  
 "model.add(Dropout(0.2))*\n*",  
 "model.add(Dense(num\_classes, activation='softmax'))*\n*",  
 "*\n*",  
 "# Compile model*\n*",  
 "epochs = 5*\n*",  
 "learning\_rate = 0.01*\n*",  
 "decay\_rate = learning\_rate / epochs*\n*",  
 "sgd = SGD(lr=learning\_rate, momentum=0.9, decay=decay\_rate, nesterov=False)*\n*",  
 "model.compile(loss='categorical\_crossentropy', optimizer=sgd, metrics=['accuracy'])*\n*",  
 "print(model.summary())*\n*",  
 "*\n*",  
 "# Fit the model*\n*",  
 "history = model.fit(X\_train, y\_train, validation\_data=(X\_test, y\_test), epochs=epochs, batch\_size=32)*\n*",  
 "*\n*",  
 "# Evaluate the model*\n*",  
 "scores = model.evaluate(X\_test, y\_test, verbose=0)*\n*",  
 "print(*\"*Accuracy: %.2f%%*\"* % (scores[1] \* 100))*\n*"  
 ],  
 "metadata": {  
 "colab": {  
 "base\_uri": "https://localhost:8080/"  
 },  
 "id": "-Lc36Iq-xsa7",  
 "outputId": "68bb1447-a60a-4f88-de1e-4ee99675a3f8"  
 },  
 "execution\_count": *14*,  
 "outputs": [  
 {  
 "output\_type": "stream",  
 "name": "stdout",  
 "text": [  
 "Model: *\"*sequential\_2*\"\n*",  
 "\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_*\n*",  
 " Layer (type) Output Shape Param # *\n*",  
 "=================================================================*\n*",  
 " conv2d\_4 (Conv2D) (None, 32, 32, 32) 896 *\n*",  
 " *\n*",  
 " dropout\_4 (Dropout) (None, 32, 32, 32) 0 *\n*",  
 " *\n*",  
 " conv2d\_5 (Conv2D) (None, 32, 32, 32) 9248 *\n*",  
 " *\n*",  
 " max\_pooling2d\_2 (MaxPooling (None, 16, 16, 32) 0 *\n*",  
 " 2D) *\n*",  
 " *\n*",  
 " conv2d\_6 (Conv2D) (None, 16, 16, 64) 18496 *\n*",  
 " *\n*",  
 " dropout\_5 (Dropout) (None, 16, 16, 64) 0 *\n*",  
 " *\n*",  
 " conv2d\_7 (Conv2D) (None, 16, 16, 64) 36928 *\n*",  
 " *\n*",  
 " max\_pooling2d\_3 (MaxPooling (None, 8, 8, 64) 0 *\n*",  
 " 2D) *\n*",  
 " *\n*",  
 " conv2d\_8 (Conv2D) (None, 8, 8, 128) 73856 *\n*",  
 " *\n*",  
 " dropout\_6 (Dropout) (None, 8, 8, 128) 0 *\n*",  
 " *\n*",  
 " conv2d\_9 (Conv2D) (None, 8, 8, 128) 147584 *\n*",  
 " *\n*",  
 " max\_pooling2d\_4 (MaxPooling (None, 4, 4, 128) 0 *\n*",  
 " 2D) *\n*",  
 " *\n*",  
 " flatten\_2 (Flatten) (None, 2048) 0 *\n*",  
 " *\n*",  
 " dropout\_7 (Dropout) (None, 2048) 0 *\n*",  
 " *\n*",  
 " dense\_4 (Dense) (None, 1024) 2098176 *\n*",  
 " *\n*",  
 " dropout\_8 (Dropout) (None, 1024) 0 *\n*",  
 " *\n*",  
 " dense\_5 (Dense) (None, 512) 524800 *\n*",  
 " *\n*",  
 " dropout\_9 (Dropout) (None, 512) 0 *\n*",  
 " *\n*",  
 " dense\_6 (Dense) (None, 10) 5130 *\n*",  
 " *\n*",  
 "=================================================================*\n*",  
 "Total params: 2,915,114*\n*",  
 "Trainable params: 2,915,114*\n*",  
 "Non-trainable params: 0*\n*",  
 "\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_*\n*",  
 "None*\n*",  
 "Epoch 1/5*\n*",  
 "1563/1563 [==============================] - 15s 9ms/step - loss: 1.9322 - accuracy: 0.2796 - val\_loss: 1.6108 - val\_accuracy: 0.4168*\n*",  
 "Epoch 2/5*\n*",  
 "1563/1563 [==============================] - 13s 9ms/step - loss: 1.5375 - accuracy: 0.4379 - val\_loss: 1.4261 - val\_accuracy: 0.4795*\n*",  
 "Epoch 3/5*\n*",  
 "1563/1563 [==============================] - 13s 9ms/step - loss: 1.3979 - accuracy: 0.4918 - val\_loss: 1.3406 - val\_accuracy: 0.5164*\n*",  
 "Epoch 4/5*\n*",  
 "1563/1563 [==============================] - 13s 8ms/step - loss: 1.3128 - accuracy: 0.5217 - val\_loss: 1.2901 - val\_accuracy: 0.5367*\n*",  
 "Epoch 5/5*\n*",  
 "1563/1563 [==============================] - 13s 9ms/step - loss: 1.2504 - accuracy: 0.5459 - val\_loss: 1.1804 - val\_accuracy: 0.5735*\n*",  
 "Accuracy: 57.35%*\n*"  
 ]  
 }  
 ]  
 },  
 {  
 "cell\_type": "code",  
 "source": [  
 "# Predict the first 4 images of the test data*\n*",  
 "predictions = model.predict(X\_test[:4])*\n*",  
 "# Convert the predictions to class labels*\n*",  
 "predicted\_labels = numpy.argmax(predictions, axis=1)*\n*",  
 "# Convert the actual labels to class labels*\n*",  
 "actual\_labels = numpy.argmax(y\_test[:4], axis=1)*\n*",  
 "*\n*",  
 "# Print the predicted and actual labels for the first 4 images*\n*",  
 "print(*\"*Predicted labels:*\"*, predicted\_labels)*\n*",  
 "print(*\"*Actual labels: *\"*, actual\_labels)*\n*"  
 ],  
 "metadata": {  
 "colab": {  
 "base\_uri": "https://localhost:8080/"  
 },  
 "id": "vvcyfHmUzJ2n",  
 "outputId": "a39a54f1-43d3-4e60-aa47-abeedd908e1b"  
 },  
 "execution\_count": *18*,  
 "outputs": [  
 {  
 "output\_type": "stream",  
 "name": "stdout",  
 "text": [  
 "1/1 [==============================] - 0s 21ms/step*\n*",  
 "Predicted labels: [3 8 8 8]*\n*",  
 "Actual labels: [3 8 8 0]*\n*"  
 ]  
 }  
 ]  
 },  
 {  
 "cell\_type": "code",  
 "source": [  
 "import matplotlib.pyplot as plt*\n*",  
 "*\n*",  
 "# Plot the training and validation loss*\n*",  
 "plt.plot(history.history['loss'])*\n*",  
 "plt.plot(history.history['val\_loss'])*\n*",  
 "plt.title('Model Loss')*\n*",  
 "plt.ylabel('Loss')*\n*",  
 "plt.xlabel('Epoch')*\n*",  
 "plt.legend(['train', 'val'], loc='upper right')*\n*",  
 "plt.show()*\n*",  
 "*\n*",  
 "# Plot the training and validation accuracy*\n*",  
 "plt.plot(history.history['accuracy'])*\n*",  
 "plt.plot(history.history['val\_accuracy'])*\n*",  
 "plt.title('Model Accuracy')*\n*",  
 "plt.ylabel('Accuracy')*\n*",  
 "plt.xlabel('Epoch')*\n*",  
 "plt.legend(['train', 'val'], loc='lower right')*\n*",  
 "plt.show()*\n*"  
 ],  
 "metadata": {  
 "colab": {  
 "base\_uri": "https://localhost:8080/",  
 "height": *573* },  
 "id": "jLt\_UBB5zTNk",  
 "outputId": "6f238606-4fa3-4d36-8523-65f2ef1cd9a6"  
 },  
 "execution\_count": *19*,  
 "outputs": [  
 {  
 "output\_type": "display\_data",  
 "data": {  
 "text/plain": [  
 "<Figure size 432x288 with 1 Axes>"  
 ],  
 "image/png": "*\n*"  
 },  
 "metadata": {  
 "needs\_background": "light"  
 }  
 },  
 {  
 "output\_type": "display\_data",  
 "data": {  
 "text/plain": [  
 "<Figure size 432x288 with 1 Axes>"  
 ],  
 "image/png": "*\n*"  
 },  
 "metadata": {  
 "needs\_background": "light"  
 }  
 }  
 ]  
 }  
 ]  
}

**OUTPUT:**

{'nbformat': 4,

'nbformat\_minor': 0,

'metadata': {'colab': {'provenance': []},

'kernelspec': {'name': 'python3', 'display\_name': 'Python 3'},

'language\_info': {'name': 'python'},

'accelerator': 'GPU',

'gpuClass': 'standard'},

'cells': [{'cell\_type': 'code',

'source': ['import numpy as np\n',

'from keras.datasets import cifar10\n',

'from keras.models import Sequential\n',

'from keras.layers import Dense, Dropout, Flatten\n',

'from keras.constraints import maxnorm\n',

'from keras.optimizers import SGD\n',

'from keras.layers.convolutional import Conv2D, MaxPooling2D\n',

'from keras.utils import np\_utils'],

'metadata': {'id': 'MtZG9dJzw4ld'},

'execution\_count': 5,

'outputs': []},

{'cell\_type': 'code',

'source': ['np.random.seed(7)'],

'metadata': {'id': 'n4lci1f3w8hC'},

'execution\_count': 6,

'outputs': []},

{'cell\_type': 'code',

'source': ['(X\_train, y\_train), (X\_test, y\_test) = cifar10.load\_data()'],

'metadata': {'id': 'm4bzJoVOxBX\_'},

'execution\_count': 7,

'outputs': []},

{'cell\_type': 'code',

'source': ["X\_train = X\_train.astype('float32') / 255.0\n",

"X\_test = X\_test.astype('float32') / 255.0"],

'metadata': {'id': '3DvFG935xFI2'},

'execution\_count': 8,

'outputs': []},

{'cell\_type': 'code',

'source': ['y\_train = np\_utils.to\_categorical(y\_train)\n',

'y\_test = np\_utils.to\_categorical(y\_test)\n',

'num\_classes = y\_test.shape[1]'],

'metadata': {'id': 'i\_R7uAN4xIrm'},

'execution\_count': 9,

'outputs': []},

{'cell\_type': 'code',

'source': ['model = Sequential()\n',

"model.add(Conv2D(32, (3, 3), input\_shape=(32, 32, 3), padding='same', activation='relu', kernel\_constraint=maxnorm(3)))\n",

'model.add(Dropout(0.2))\n',

"model.add(Conv2D(32, (3, 3), activation='relu', padding='same', kernel\_constraint=maxnorm(3)))\n",

"model.add(MaxPooling2D(pool\_size=(2, 2), padding='same'))\n",

'model.add(Flatten())\n',

"model.add(Dense(512, activation='relu', kernel\_constraint=maxnorm(3)))\n",

'model.add(Dropout(0.5))\n',

"model.add(Dense(num\_classes, activation='softmax'))\n"],

'metadata': {'id': 'z\_NGevkcxMSf'},

'execution\_count': 10,

'outputs': []},

{'cell\_type': 'code',

'source': ['sgd = SGD(learning\_rate=0.01, momentum=0.9, decay=1e-6)\n',

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

'print(model.summary())'],

'metadata': {'colab': {'base\_uri': 'https://localhost:8080/'},

'id': 'rhzjj8FaxN77',

'outputId': '0b033a14-600d-4cf1-8515-ce5515d02091'},

'execution\_count': 11,

'outputs': [{'output\_type': 'stream',

'name': 'stdout',

'text': ['Model: "sequential\_1"\n',

'\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\n',

' Layer (type) Output Shape Param # \n',

'=================================================================\n',

' conv2d\_2 (Conv2D) (None, 32, 32, 32) 896 \n',

' \n',

' dropout\_2 (Dropout) (None, 32, 32, 32) 0 \n',

' \n',

' conv2d\_3 (Conv2D) (None, 32, 32, 32) 9248 \n',

' \n',

' max\_pooling2d\_1 (MaxPooling (None, 16, 16, 32) 0 \n',

' 2D) \n',

' \n',

' flatten\_1 (Flatten) (None, 8192) 0 \n',

' \n',

' dense\_2 (Dense) (None, 512) 4194816 \n',

' \n',

' dropout\_3 (Dropout) (None, 512) 0 \n',

' \n',

' dense\_3 (Dense) (None, 10) 5130 \n',

' \n',

'=================================================================\n',

'Total params: 4,210,090\n',

'Trainable params: 4,210,090\n',

'Non-trainable params: 0\n',

'\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\n',

'None\n']}]},

{'cell\_type': 'code',

'source': ['epochs = 5\n',

'batch\_size = 32\n',

'model.fit(X\_train, y\_train, validation\_data=(X\_test, y\_test), epochs=epochs, batch\_size=batch\_size)\n'],

'metadata': {'colab': {'base\_uri': 'https://localhost:8080/'},

'id': 'COpaIeenxXuH',

'outputId': '89ce2eea-a785-4f62-d7bc-567c0b2c98af'},

'execution\_count': 12,

'outputs': [{'output\_type': 'stream',

'name': 'stdout',

'text': ['Epoch 1/5\n',

'1563/1563 [==============================] - 19s 7ms/step - loss: 1.7232 - accuracy: 0.3746 - val\_loss: 1.4776 - val\_accuracy: 0.4563\n',

'Epoch 2/5\n',

'1563/1563 [==============================] - 10s 6ms/step - loss: 1.3675 - accuracy: 0.5117 - val\_loss: 1.2470 - val\_accuracy: 0.5551\n',

'Epoch 3/5\n',

'1563/1563 [==============================] - 10s 6ms/step - loss: 1.2071 - accuracy: 0.5716 - val\_loss: 1.1232 - val\_accuracy: 0.6047\n',

'Epoch 4/5\n',

'1563/1563 [==============================] - 10s 7ms/step - loss: 1.0855 - accuracy: 0.6136 - val\_loss: 1.1554 - val\_accuracy: 0.5928\n',

'Epoch 5/5\n',

'1563/1563 [==============================] - 10s 7ms/step - loss: 0.9709 - accuracy: 0.6583 - val\_loss: 0.9986 - val\_accuracy: 0.6550\n']},

{'output\_type': 'execute\_result',

'data': {'text/plain': ['<keras.callbacks.History at 0x7f689d6d65e0>']},

'metadata': {},

'execution\_count': 12}]},

{'cell\_type': 'code',

'source': ['scores = model.evaluate(X\_test, y\_test, verbose=0)\n',

'print("Accuracy: %.2f%%" % (scores[1]\*100))'],

'metadata': {'colab': {'base\_uri': 'https://localhost:8080/'},

'id': 'gymoyEPsxpd3',

'outputId': 'ac10174d-d62f-4f23-9766-43fe8b545e9a'},

'execution\_count': 13,

'outputs': [{'output\_type': 'stream',

'name': 'stdout',

'text': ['Accuracy: 65.50%\n']}]},

{'cell\_type': 'code',

'source': ['import numpy as np\n',

'from keras.datasets import cifar10\n',

'from keras.models import Sequential\n',

'from keras.layers import Dense, Dropout, Flatten\n',

'from keras.layers.convolutional import Conv2D, MaxPooling2D\n',

'from keras.constraints import maxnorm\n',

'from keras.utils import np\_utils\n',

'from keras.optimizers import SGD\n',

'\n',

'# Fix random seed for reproducibility\n',

'np.random.seed(7)\n',

'\n',

'# Load data\n',

'(X\_train, y\_train), (X\_test, y\_test) = cifar10.load\_data()\n',

'\n',

'# Normalize inputs from 0-255 to 0.0-1.0\n',

"X\_train = X\_train.astype('float32') / 255.0\n",

"X\_test = X\_test.astype('float32') / 255.0\n",

'\n',

'# One hot encode outputs\n',

'y\_train = np\_utils.to\_categorical(y\_train)\n',

'y\_test = np\_utils.to\_categorical(y\_test)\n',

'num\_classes = y\_test.shape[1]\n',

'\n',

'# Create the model\n',

'model = Sequential()\n',

"model.add(Conv2D(32, (3, 3), input\_shape=(32, 32, 3), padding='same', activation='relu', kernel\_constraint=maxnorm(3)))\n",

'model.add(Dropout(0.2))\n',

"model.add(Conv2D(32, (3, 3), activation='relu', padding='same', kernel\_constraint=maxnorm(3)))\n",

'model.add(MaxPooling2D(pool\_size=(2, 2)))\n',

"model.add(Conv2D(64, (3, 3), activation='relu', padding='same', kernel\_constraint=maxnorm(3)))\n",

'model.add(Dropout(0.2))\n',

"model.add(Conv2D(64, (3, 3), activation='relu', padding='same', kernel\_constraint=maxnorm(3)))\n",

'model.add(MaxPooling2D(pool\_size=(2, 2)))\n',

"model.add(Conv2D(128, (3, 3), activation='relu', padding='same', kernel\_constraint=maxnorm(3)))\n",

'model.add(Dropout(0.2))\n',

"model.add(Conv2D(128, (3, 3), activation='relu', padding='same', kernel\_constraint=maxnorm(3)))\n",

'model.add(MaxPooling2D(pool\_size=(2, 2)))\n',

'model.add(Flatten())\n',

'model.add(Dropout(0.2))\n',

"model.add(Dense(1024, activation='relu', kernel\_constraint=maxnorm(3)))\n",

'model.add(Dropout(0.2))\n',

"model.add(Dense(512, activation='relu', kernel\_constraint=maxnorm(3)))\n",

'model.add(Dropout(0.2))\n',

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

'\n',

'# Compile model\n',

'epochs = 5\n',

'learning\_rate = 0.01\n',

'decay\_rate = learning\_rate / epochs\n',

'sgd = SGD(lr=learning\_rate, momentum=0.9, decay=decay\_rate, nesterov=False)\n',

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

'print(model.summary())\n',

'\n',

'# Fit the model\n',

'history = model.fit(X\_train, y\_train, validation\_data=(X\_test, y\_test), epochs=epochs, batch\_size=32)\n',

'\n',

'# Evaluate the model\n',

'scores = model.evaluate(X\_test, y\_test, verbose=0)\n',

'print("Accuracy: %.2f%%" % (scores[1] \* 100))\n'],

'metadata': {'colab': {'base\_uri': 'https://localhost:8080/'},

'id': '-Lc36Iq-xsa7',

'outputId': '68bb1447-a60a-4f88-de1e-4ee99675a3f8'},

'execution\_count': 14,

'outputs': [{'output\_type': 'stream',

'name': 'stdout',

'text': ['Model: "sequential\_2"\n',

'\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\n',

' Layer (type) Output Shape Param # \n',

'=================================================================\n',

' conv2d\_4 (Conv2D) (None, 32, 32, 32) 896 \n',

' \n',

' dropout\_4 (Dropout) (None, 32, 32, 32) 0 \n',

' \n',

' conv2d\_5 (Conv2D) (None, 32, 32, 32) 9248 \n',

' \n',

' max\_pooling2d\_2 (MaxPooling (None, 16, 16, 32) 0 \n',

' 2D) \n',

' \n',

' conv2d\_6 (Conv2D) (None, 16, 16, 64) 18496 \n',

' \n',

' dropout\_5 (Dropout) (None, 16, 16, 64) 0 \n',

' \n',

' conv2d\_7 (Conv2D) (None, 16, 16, 64) 36928 \n',

' \n',

' max\_pooling2d\_3 (MaxPooling (None, 8, 8, 64) 0 \n',

' 2D) \n',

' \n',

' conv2d\_8 (Conv2D) (None, 8, 8, 128) 73856 \n',

' \n',

' dropout\_6 (Dropout) (None, 8, 8, 128) 0 \n',

' \n',

' conv2d\_9 (Conv2D) (None, 8, 8, 128) 147584 \n',

' \n',

' max\_pooling2d\_4 (MaxPooling (None, 4, 4, 128) 0 \n',

' 2D) \n',

' \n',

' flatten\_2 (Flatten) (None, 2048) 0 \n',

' \n',

' dropout\_7 (Dropout) (None, 2048) 0 \n',

' \n',

' dense\_4 (Dense) (None, 1024) 2098176 \n',

' \n',

' dropout\_8 (Dropout) (None, 1024) 0 \n',

' \n',

' dense\_5 (Dense) (None, 512) 524800 \n',

' \n',

' dropout\_9 (Dropout) (None, 512) 0 \n',

' \n',

' dense\_6 (Dense) (None, 10) 5130 \n',

' \n',

'=================================================================\n',

'Total params: 2,915,114\n',

'Trainable params: 2,915,114\n',

'Non-trainable params: 0\n',

'\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\n',

'None\n',

'Epoch 1/5\n',

'1563/1563 [==============================] - 15s 9ms/step - loss: 1.9322 - accuracy: 0.2796 - val\_loss: 1.6108 - val\_accuracy: 0.4168\n',

'Epoch 2/5\n',

'1563/1563 [==============================] - 13s 9ms/step - loss: 1.5375 - accuracy: 0.4379 - val\_loss: 1.4261 - val\_accuracy: 0.4795\n',

'Epoch 3/5\n',

'1563/1563 [==============================] - 13s 9ms/step - loss: 1.3979 - accuracy: 0.4918 - val\_loss: 1.3406 - val\_accuracy: 0.5164\n',

'Epoch 4/5\n',

'1563/1563 [==============================] - 13s 8ms/step - loss: 1.3128 - accuracy: 0.5217 - val\_loss: 1.2901 - val\_accuracy: 0.5367\n',

'Epoch 5/5\n',

'1563/1563 [==============================] - 13s 9ms/step - loss: 1.2504 - accuracy: 0.5459 - val\_loss: 1.1804 - val\_accuracy: 0.5735\n',

'Accuracy: 57.35%\n']}]},

{'cell\_type': 'code',

'source': ['# Predict the first 4 images of the test data\n',

'predictions = model.predict(X\_test[:4])\n',

'# Convert the predictions to class labels\n',

'predicted\_labels = numpy.argmax(predictions, axis=1)\n',

'# Convert the actual labels to class labels\n',

'actual\_labels = numpy.argmax(y\_test[:4], axis=1)\n',

'\n',

'# Print the predicted and actual labels for the first 4 images\n',

'print("Predicted labels:", predicted\_labels)\n',

'print("Actual labels: ", actual\_labels)\n'],

'metadata': {'colab': {'base\_uri': 'https://localhost:8080/'},

'id': 'vvcyfHmUzJ2n',

'outputId': 'a39a54f1-43d3-4e60-aa47-abeedd908e1b'},

'execution\_count': 18,

'outputs': [{'output\_type': 'stream',

'name': 'stdout',

'text': ['1/1 [==============================] - 0s 21ms/step\n',

'Predicted labels: [3 8 8 8]\n',

'Actual labels: [3 8 8 0]\n']}]},

{'cell\_type': 'code',

'source': ['import matplotlib.pyplot as plt\n',

'\n',

'# Plot the training and validation loss\n',

"plt.plot(history.history['loss'])\n",

"plt.plot(history.history['val\_loss'])\n",

"plt.title('Model Loss')\n",

"plt.ylabel('Loss')\n",

"plt.xlabel('Epoch')\n",

"plt.legend(['train', 'val'], loc='upper right')\n",

'plt.show()\n',

'\n',

'# Plot the training and validation accuracy\n',

"plt.plot(history.history['accuracy'])\n",

"plt.plot(history.history['val\_accuracy'])\n",

"plt.title('Model Accuracy')\n",

"plt.ylabel('Accuracy')\n",

"plt.xlabel('Epoch')\n",

"plt.legend(['train', 'val'], loc='lower right')\n",

'plt.show()\n'],

'metadata': {'colab': {'base\_uri': 'https://localhost:8080/',

'height': 573},

'id': 'jLt\_UBB5zTNk',

'outputId': '6f238606-4fa3-4d36-8523-65f2ef1cd9a6'},

'execution\_count': 19,

'outputs': [{'output\_type': 'display\_data',

'data': {'text/plain': ['<Figure size 432x288 with 1 Axes>'],

'image/png': '\n'},

'metadata': {'needs\_background': 'light'}},

{'output\_type': 'display\_data',

'data': {'text/plain': ['<Figure size 432x288 with 1 Axes>'],

'image/png': '\n'},

'metadata': {'needs\_background': 'light'}}]}]}