import numpy as np

from tensorflow.keras.datasets import cifar10

from tensorflow.keras.models import Sequential

from tensorflow.keras.layers import Dense, Dropout, Flatten, Conv2D, MaxPooling2D

from tensorflow.keras.constraints import MaxNorm

from tensorflow.keras.optimizers import SGD

from tensorflow.keras.utils import to\_categorical

# fix random seed for reproducibility

seed = 7

np.random.seed(seed)

# load data

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

# convert from int to float and normalize inputs from 0-255 to 0.0-1.0

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

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

# one hot encode outputs

y\_train = to\_categorical(y\_train)

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

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

# Create the model

model = Sequential()

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

model.add(Dropout(0.2))

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

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

model.add(Flatten())

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

model.add(Dropout(0.5))

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

# Compile model

epochs = 25

lrate = 0.01

sgd = SGD(lr=lrate, momentum=0.9, nesterov=False)

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

print(model.summary())

# Fit the model

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

# Final evaluation of the model

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

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

A screenshot of a computer program

Description automatically generated

A screenshot of a computer

Description automatically generated

import numpy as np

from tensorflow.keras.datasets import cifar10

from tensorflow.keras.models import Sequential

from tensorflow.keras.layers import Dense, Dropout, Flatten, Conv2D, MaxPooling2D

from tensorflow.keras.constraints import MaxNorm

from tensorflow.keras.optimizers import SGD

from tensorflow.keras.utils import to\_categorical

# fix random seed for reproducibility

seed = 7

np.random.seed(seed)

# load data

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

# convert from int to float and normalize inputs from 0-255 to 0.0-1.0

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

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

# one hot encode outputs

y\_train = to\_categorical(y\_train)

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

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

# Create the model

model = Sequential()

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

model.add(Dropout(0.2))

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

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

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

model.add(Dropout(0.2))

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

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

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

model.add(Dropout(0.2))

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

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

model.add(Flatten())

model.add(Dropout(0.2))

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

model.add(Dropout(0.2))

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

model.add(Dropout(0.2))

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

# Compile model

epochs = 25

lrate = 0.01

sgd = SGD(lr=lrate, momentum=0.9, nesterov=False)

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

print(model.summary())

# Fit the model

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

# Final evaluation of the model

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

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

A screenshot of a computer

Description automatically generated

A screenshot of a computer program

Description automatically generated

A close-up of a number

Description automatically generated

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

print(predictions)

print(np.argmax(predictions, axis=1))

print(y\_test[:4])

A screenshot of a computer

Description automatically generated

import matplotlib.pyplot as plt

# Plot training & validation accuracy values

plt.plot(history.history['accuracy'])

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

plt.title('Model accuracy')

plt.ylabel('Accuracy')

plt.xlabel('Epoch')

plt.legend(['Train', 'Validation'], loc='upper left')

plt.show()

# Plot training & validation loss values

plt.plot(history.history['loss'])

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

plt.title('Model loss')

plt.ylabel('Loss')

plt.xlabel('Epoch')

plt.legend(['Train', 'Validation'], loc='upper left')

plt.show()

A graph with blue and orange lines

Description automatically generated

A graph with blue and orange lines

Description automatically generated