Deep Learning Image Classification with CNN

Name: Veda Siva Vasishta Yakkala ID: 700758715

Github link:<https://github.com/VasishtaYakkala/Neural-Network-and-Deep-Learning-icp4>

1. Follow the instruction below and then report how the performance changed. (apply all at once)

* Convolutional input layer, 32 feature maps with a size of 3×3 and a rectifier activation function.
* Dropout layer at 20%.
* Convolutional layer, 32 feature maps with a size of 3×3 and a rectifier activation function.
* Max Pool layer with size 2×2.
* Convolutional layer, 64 feature maps with a size of 3×3 and a rectifier activation function.
* Dropout layer at 20%.
* Convolutional layer, 64 feature maps with a size of 3×3 and a rectifier activation function.
* Max Pool layer with size 2×2.
* Convolutional layer, 128 feature maps with a size of 3×3 and a rectifier activation function.
* Dropout layer at 20%.
* Convolutional layer,128 feature maps with a size of 3×3 and a rectifier activation function.
* Max Pool layer with size 2×2.
* Flatten layer.
* Dropout layer at 20%.
* Fully connected layer with 1024 units and a rectifier activation function.
* Dropout layer at 20%.
* Fully connected layer with 512 units and a rectifier activation function.
* Dropout layer at 20%.
* Fully connected output layer with 10 units and a Softmax activation function.

Did the performance change?

Code:

from re import X

import numpy

import tensorflow as tf

from tensorflow.keras.datasets import cifar10

from tensorflow.keras.models import Sequential

from tensorflow.keras.layers import Dense

from tensorflow.keras.layers import Dropout

from tensorflow.keras.layers import Flatten

from tensorflow.keras.optimizers.legacy import SGD #using legacy optimizer so

#that we can use lr schedular to show more flexibility in how lr changes overtime

from tensorflow.keras.layers import Conv2D

from tensorflow.keras.layers import MaxPooling2D

from tensorflow.keras.utils import to\_categorical

from tensorflow.keras.constraints import MaxNorm

#from keras import backend as K

#K.set\_image\_dim\_ordering('th')

# fix random seed for reproducibility

seed = 7

numpy.random.seed(seed)

# load data

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

# normalize inputs from 0-255 to 0.0-1.0

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

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

X\_train = X\_train / 255.0

X\_test = X\_test / 255.0

# one hot encode outputs

y\_train = to\_categorical(y\_train,10)

y\_test = to\_categorical(y\_test,10)

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 the model

epochs = 5

lrate = 0.01

decay = lrate/epochs

sgd = SGD(learning\_rate=lrate, momentum=0.9, decay=decay, nesterov=False)

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

# fit the model

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

# Evaluate the model

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

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

Output:

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

170498071/170498071 [==============================] - 6s 0us/step

Epoch 1/5

1563/1563 [==============================] - 19s 7ms/step - loss: 1.5360 - accuracy: 0.4478 - val\_loss: 1.2400 - val\_accuracy: 0.5491

Epoch 2/5

1563/1563 [==============================] - 11s 7ms/step - loss: 1.2180 - accuracy: 0.5663 - val\_loss: 1.0865 - val\_accuracy: 0.6200

Epoch 3/5

1563/1563 [==============================] - 11s 7ms/step - loss: 1.0990 - accuracy: 0.6119 - val\_loss: 0.9863 - val\_accuracy: 0.6542

Epoch 4/5

1563/1563 [==============================] - 11s 7ms/step - loss: 1.0298 - accuracy: 0.6364 - val\_loss: 0.9521 - val\_accuracy: 0.6692

Epoch 5/5

1563/1563 [==============================] - 10s 6ms/step - loss: 0.9770 - accuracy: 0.6548 - val\_loss: 0.9475 - val\_accuracy: 0.6705

Accuracy: 67.05%

1. Predict the first 4 images of the test data using the above model. Then, compare with the actual label for those 4 images to check whether or not the model has predicted correctly.

Code:

import numpy as np

# Predict the first 4 images of the test data

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

predicted\_classes = np.argmax(predictions, axis=1)

# Get the actual labels for the first 4 images

actual\_classes = np.argmax(y\_test[:4], axis=1)

# Compare the predicted classes with the actual classes

for i in range(4):

    print(f"Image {i+1}:")

    print(f"Predicted: {predicted\_classes[i]}, Actual: {actual\_classes[i]}")

    print(f"Correct: {predicted\_classes[i] == actual\_classes[i]}")

Output:

1/1 [==============================] - 0s 186ms/step

Image 1:

Predicted: 3, Actual: 3

Correct: True

Image 2:

Predicted: 8, Actual: 8

Correct: True

Image 3:

Predicted: 8, Actual: 8

Correct: True

Image 4:

Predicted: 0, Actual: 0

Correct: True

1. Visualize Loss and Accuracy using the history object

Code:

import matplotlib.pyplot as plt

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

# Extract the loss and accuracy from the history object

train\_loss = history.history['loss']

val\_loss = history.history['val\_loss']

train\_accuracy = history.history['accuracy']

val\_accuracy = history.history['val\_accuracy']

# Define the number of epochs

epochs = range(1, len(train\_loss) + 1)

# Plot training and validation loss

plt.figure(figsize=(14, 5))

plt.subplot(1, 2, 1)

plt.plot(epochs, train\_loss, label='Training Loss')

plt.plot(epochs, val\_loss, label='Validation Loss')

plt.xlabel('Epochs')

plt.ylabel('Loss')

plt.title('Training and Validation Loss')

plt.legend()

# Plot training and validation accuracy

plt.subplot(1, 2, 2)

plt.plot(epochs, train\_accuracy, label='Training Accuracy')

plt.plot(epochs, val\_accuracy, label='Validation Accuracy')

plt.xlabel('Epochs')

plt.ylabel('Accuracy')

plt.title('Training and Validation Accuracy')

plt.legend()

# Display the plots

plt.show()

Output:

Epoch 1/5

1563/1563 [==============================] - 15s 10ms/step - loss: 0.9270 - accuracy: 0.6722 - val\_loss: 0.8954 - val\_accuracy: 0.6881

Epoch 2/5

1563/1563 [==============================] - 11s 7ms/step - loss: 0.8973 - accuracy: 0.6816 - val\_loss: 0.9249 - val\_accuracy: 0.6767

Epoch 3/5

1563/1563 [==============================] - 10s 7ms/step - loss: 0.8671 - accuracy: 0.6954 - val\_loss: 0.8913 - val\_accuracy: 0.6814

Epoch 4/5

1563/1563 [==============================] - 10s 7ms/step - loss: 0.8367 - accuracy: 0.7053 - val\_loss: 0.8704 - val\_accuracy: 0.6959

Epoch 5/5

1563/1563 [==============================] - 11s 7ms/step - loss: 0.8109 - accuracy: 0.7137 - val\_loss: 0.8753 - val\_accuracy: 0.6951

