Deep Learning Image Classification with CNN

Bevanasankalla prem sai aravind

700750996

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

import numpy as np

from keras.datasets import cifar10

from keras.models import Sequential

from keras.layers import Dense, Dropout, Flatten

from keras.constraints import maxnorm

from keras.optimizers import SGD

from keras.layers.convolutional import Conv2D, MaxPooling2D

from keras.utils import np\_utils

​

np.random.seed(7)

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

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

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

y\_train = np\_utils.to\_categorical(y\_train)

y\_test = np\_utils.to\_categorical(y\_test)

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

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), padding='same'))

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'))

​

sgd = SGD(learning\_rate=0.01, momentum=0.9, decay=1e-6)

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

print(model.summary())

Model: "sequential\_4"

\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

Layer (type) Output Shape Param #

=================================================================

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

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

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

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

2D)

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

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

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

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

=================================================================

Total params: 4,210,090

Trainable params: 4,210,090

Non-trainable params: 0

\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

None

epochs = 5

batch\_size = 32

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

​

Epoch 1/5

1563/1563 [==============================] - 90s 56ms/step - loss: 1.6961 - accuracy: 0.3858 - val\_loss: 1.3913 - val\_accuracy: 0.4994

Epoch 2/5

1563/1563 [==============================] - 100s 64ms/step - loss: 1.3462 - accuracy: 0.5179 - val\_loss: 1.1685 - val\_accuracy: 0.5816

Epoch 3/5

831/1563 [==============>...............] - ETA: 42s - loss: 1.1865 - accuracy: 0.5784

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

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

Did the performance change?

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.

import numpy as np

from keras.datasets import cifar10

from keras.models import Sequential

from keras.layers import Dense, Dropout, Flatten

from keras.layers.convolutional import Conv2D, MaxPooling2D

from keras.constraints import maxnorm

from keras.utils import np\_utils

from keras.optimizers import SGD

# Fix random seed for reproducibility

np.random.seed(7)

# 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') / 255.0

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

# One hot encode outputs

y\_train = np\_utils.to\_categorical(y\_train)

y\_test = np\_utils.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), activation='relu', padding='same', 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), activation='relu', padding='same', 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 = 5

learning\_rate = 0.01

decay\_rate = learning\_rate / epochs

sgd = SGD(lr=learning\_rate, momentum=0.9, decay=decay\_rate, 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)

# Evaluate the model

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

1. Visualize Loss and Accuracy using the history object

import matplotlib.pyplot as plt

# Plot the training and validation loss

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

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

plt.title('Model Loss')

plt.ylabel('Loss')

plt.xlabel('Epoch')

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

plt.show()

# Plot the training and validation accuracy

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

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

plt.title('Model Accuracy')

plt.ylabel('Accuracy')

plt.xlabel('Epoch')

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

plt.show()

GIT: <https://github.com/Premsaiaravind/ICP_7>