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?**

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)

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)

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

# Predict the first 4 images of the test data

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

# Convert the predictions to class labels

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

# Convert the actual labels to class labels

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

**Did the performance change?**

The performance of the model is likely to improve with the addition of more layers and higher number of feature maps, but it will also increase the complexity and the training time of the model. The new model architecture provided in the instruction includes several new layers and higher number of feature maps, which may improve the accuracy of the model.

A screenshot of a computer

Description automatically generated

A screenshot of a computer

Description automatically generated

**2. 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.**

# Print the predicted and actual labels for the first 4 images

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

print("Actual labels: ", actual\_label

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()

A screenshot of a computer

Description automatically generated

**3. Visualize Loss and Accuracy using the history object.**

import matplotlib.pyplot as plt

# Plot the training and validation loss

plt.figure(figsize=(12, 6))

plt.subplot(1, 2, 1)

plt.plot(history.history['loss'], label='Training Loss')

plt.plot(history.history['val\_loss'], label='Validation Loss')

plt.xlabel('Epoch')

plt.ylabel('Loss')

plt.title('Training and Validation Loss')

plt.legend()

# Plot the training and validation accuracy

plt.subplot(1, 2, 2)

plt.plot(history.history['accuracy'], label='Training Accuracy')

plt.plot(history.history['val\_accuracy'], label='Validation Accuracy')

plt.xlabel('Epoch')

plt.ylabel('Accuracy')

plt.title('Training and Validation Accuracy')

plt.legend()

plt.tight\_layout()

plt.show()

A screenshot of a computer

Description automatically generated

**GIT HUB LINK :**

<https://github.com/PreethiAdavelli1823/Neural_Assignment3>

**VIDEO LINK:**

<https://drive.google.com/file/d/1hs77sqxDr4YJyO6r3-7BQ7Y1kZoIrxCg/view?usp=drive_link>