Assignment no :2

Name : CHOVATIYA JAY MAHESHBHAI
Student ID : 1161980
Course: COMP5421 Deep Learning
Topic:Datasets: CIFAR100

Explanation:

Q-Compare the performance differences with the condition of learning from scratch by using different network architecture.

Ans: Obviously, **the performance** is **better** in adding multiple CNN layers along with different learning rate as you can see in the given code that i have made the architecture which is custom convolutional neural network consisting of 5 convolutional layers, 3 max pooling layers, and 2 fully connected layers.
So, the **testing time(nearly80s/epoch)** is quite **less** compared to **training from scratch(100s/epoch)** and **accuracy(64.26%)** is quite **higher** than **training from scratch(55.79%).** Hence, we can get good results from custom network architecture rather than scratch.

IMPLEMENTATION :

```
#importing the libraries
import numpy as np
import sklearn.metrics as metrics
from keras.applications import densenet
from keras.datasets import cifar100
from keras.utils import np_utils
from keras.optimizers import Adam
from keras.preprocessing.image import ImageDataGenerator
import matplotlib.pyplot as plt
# create the model from keras and set weights=None for training from scratch
model = densenet.DenseNet121(weights=None, input_shape=(32,32,3), pooling=None,
classes=100)
# printing the model summary
model.summary()
# Splitting traning and testing set
(cifarx_train, cifary_train), (cifarx_test, cifary_test) = cifar100.load_data()
# Converting to float
```

```
cifarx_train = cifarx_train.astype('float32')
cifarx_test = cifarx_test.astype('float32')
# converting data into normalize form
cifarx_train = densenet.preprocess_input(cifarx_train)
cifarx_test = densenet.preprocess_input(cifarx_test)
# one-hot encoding
cifarY_train = np_utils.to_categorical(cifary_train, 100)
cifarY_test = np_utils.to_categorical(cifary_test, 100)
# Data Augmentation
datagen_train =
ImageDataGenerator(rotation_range=15,width_shift_range=0.1,height_shift_range=0.1,
horizontal_flip=True,)
datagen_train.fit(cifarx_train)
# Using Adam and set learning rate 0.001
optimizer = Adam(lr=0.001)
# compile the model
model.compile(loss='categorical_crossentropy', optimizer=optimizer,
metrics=["accuracy"])
# train the model
history = model.fit(datagen_train.flow(cifarx_train, cifarY_train, batch_size=64,
shuffle=True),
                    steps_per_epoch=len(cifarx_train)/64, epochs=50,
validation_data=(cifarx_test, cifarY_test))
# Evaluate the model
scores = model.evaluate(cifarx_test, cifarY_test, verbose=0)
print(" the test accuracy is : %.2f%%" % (scores[1]*100))
# Define plotchart function
def plotchart(history, value):
    plt.figure(figsize=[8,6])
    plt.plot(history.history['loss'], 'firebrick', linewidth=3.0)
    plt.plot(history.history['accuracy'], 'turquoise', linewidth=3.0)
    plt.legend(['Training loss', 'Training Accuracy'], fontsize=18)
    plt.xlabel('Epochs', fontsize=16)
    plt.ylabel('Loss and Accuracy', fontsize=16)
    plt.title('Loss and Accuracy Curves for {}'.format(value), fontsize=16)
    plt.show()
# Plot the training history
plotchart(history, 'CIFAR-100')
```

<u>Outputs:1</u> Here i have attached the screenshot of CIFAR100 datasets output which you can see below.

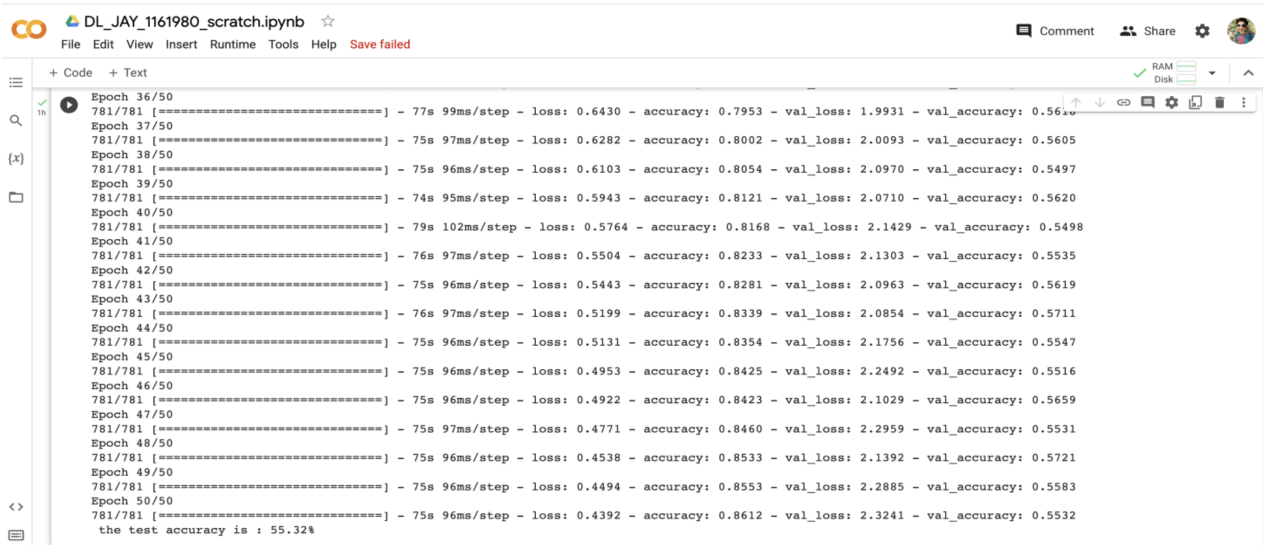
```
Output exceeds the size limit. Open the full output data in a text editor
Model: "densenet121"
________________________________________________________________________________________________
 Layer (type)                   Output Shape         Param #     Connected to
================================================================================================
 input_2 (InputLayer)           [(None, 32, 32, 3)]  0           []

 zero_padding2d_2 (ZeroPadding2  (None, 38, 38, 3)   0           ['input_2[0][0]']
 D)

 conv1/conv (Conv2D)            (None, 16, 16, 64)   9408        ['zero_padding2d_2[0][0]']

 conv1/bn (BatchNormalization)  (None, 16, 16, 64)   256         ['conv1/conv[0][0]']

 conv1/relu (Activation)        (None, 16, 16, 64)   0           ['conv1/bn[0][0]']

 zero_padding2d_3 (ZeroPadding2  (None, 18, 18, 64)  0           ['conv1/relu[0][0]']
 D)

 pool1 (MaxPooling2D)           (None, 8, 8, 64)     0           ['zero_padding2d_3[0][0]']

 conv2_block1_0_bn (BatchNormal  (None, 8, 8, 64)    256         ['pool1[0][0]']
 ization)

 conv2_block1_0_relu (Activatio  (None, 8, 8, 64)    0           ['conv2_block1_0_bn[0][0]']
 n)
...
781/781 [==============================] - 79s 101ms/step - loss: 0.2852 - accuracy: 0.9062 - val_loss: 2.6090 - val_accuracy: 0.5427
Epoch 50/50
781/781 [==============================] - 80s 102ms/step - loss: 0.2803 - accuracy: 0.9078 - val_loss: 2.5623 - val_accuracy: 0.5603
 the test accuracy is : 56.03%
```

<u>Outputs:</u>1.1

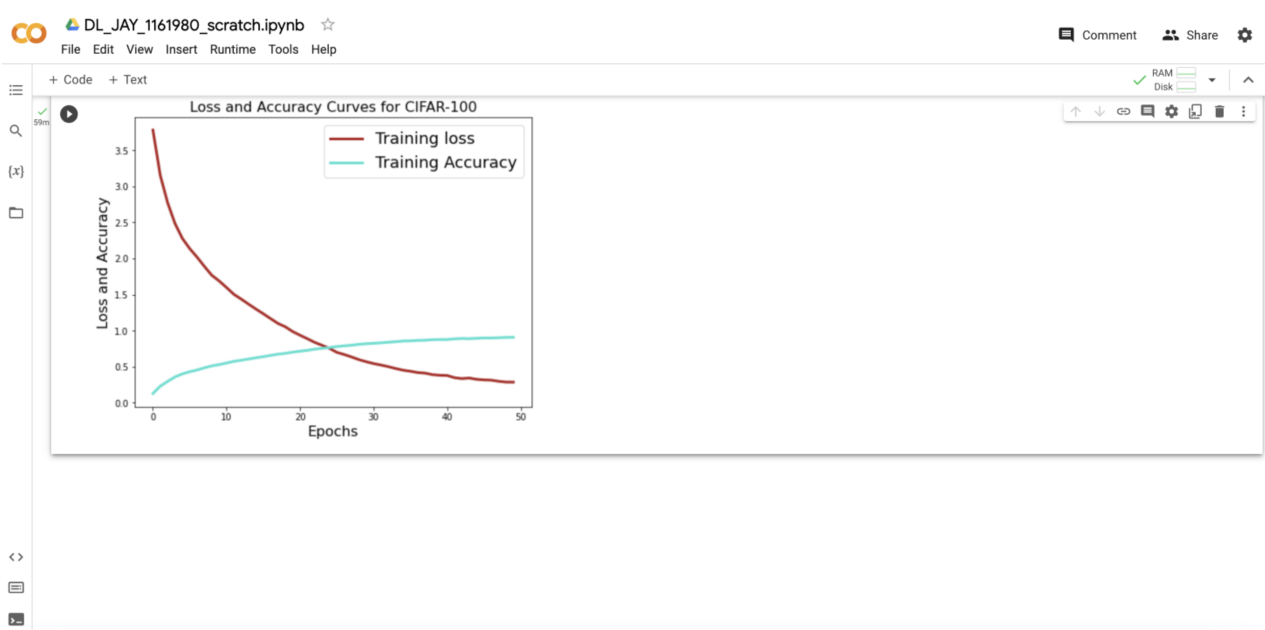
```
Output exceeds the size limit. Open the full output data in a text editor
Epoch 1/50
781/781 [==============================] - 101s 103ms/step - loss: 3.7784 - accuracy: 0.1284 - val_loss: 5.6355 - val_accuracy: 0.1535
Epoch 2/50
781/781 [==============================] - 68s 88ms/step - loss: 3.1504 - accuracy: 0.2309 - val_loss: 3.4178 - val_accuracy: 0.2575
Epoch 3/50
781/781 [==============================] - 69s 88ms/step - loss: 2.7717 - accuracy: 0.2964 - val_loss: 2.6690 - val_accuracy: 0.3324
Epoch 4/50
781/781 [==============================] - 69s 89ms/step - loss: 2.4839 - accuracy: 0.3587 - val_loss: 2.4393 - val_accuracy: 0.3710
Epoch 5/50
781/781 [==============================] - 69s 88ms/step - loss: 2.2778 - accuracy: 0.4002 - val_loss: 2.5138 - val_accuracy: 0.3776
Epoch 6/50
781/781 [==============================] - 69s 88ms/step - loss: 2.1363 - accuracy: 0.4308 - val_loss: 2.6383 - val_accuracy: 0.3973
Epoch 7/50
781/781 [==============================] - 69s 88ms/step - loss: 2.0178 - accuracy: 0.4541 - val_loss: 2.1595 - val_accuracy: 0.4391
Epoch 8/50
781/781 [==============================] - 70s 89ms/step - loss: 1.8891 - accuracy: 0.4849 - val_loss: 2.0608 - val_accuracy: 0.4613
Epoch 9/50
781/781 [==============================] - 68s 87ms/step - loss: 1.7670 - accuracy: 0.5114 - val_loss: 1.9982 - val_accuracy: 0.4743
Epoch 10/50
781/781 [==============================] - 68s 87ms/step - loss: 1.6883 - accuracy: 0.5291 - val_loss: 2.0267 - val_accuracy: 0.4767
Epoch 11/50
781/781 [==============================] - 68s 87ms/step - loss: 1.5974 - accuracy: 0.5510 - val_loss: 1.9293 - val_accuracy: 0.4909
Epoch 12/50
781/781 [==============================] - 68s 87ms/step - loss: 1.5034 - accuracy: 0.5741 - val_loss: 1.9394 - val_accuracy: 0.5000
Epoch 13/50
...
781/781 [==============================] - 67s 86ms/step - loss: 0.2874 - accuracy: 0.9060 - val_loss: 2.5630 - val_accuracy: 0.5509
Epoch 50/50
781/781 [==============================] - 74s 95ms/step - loss: 0.2866 - accuracy: 0.9070 - val_loss: 2.5173 - val_accuracy: 0.5579
 the test accuracy is : 55.79%
```

Loss and Accuracy Curves for CIFAR-100

Outputs:1.2



```
Epoch 36/50
781/781 [==============================] - 77s 99ms/step - loss: 0.6430 - accuracy: 0.7953 - val_loss: 1.9931 - val_accuracy: 0.561
Epoch 37/50
781/781 [==============================] - 75s 97ms/step - loss: 0.6282 - accuracy: 0.8002 - val_loss: 2.0093 - val_accuracy: 0.5605
Epoch 38/50
781/781 [==============================] - 75s 96ms/step - loss: 0.6103 - accuracy: 0.8054 - val_loss: 2.0970 - val_accuracy: 0.5497
Epoch 39/50
781/781 [==============================] - 74s 95ms/step - loss: 0.5943 - accuracy: 0.8121 - val_loss: 2.0710 - val_accuracy: 0.5620
Epoch 40/50
781/781 [==============================] - 79s 102ms/step - loss: 0.5764 - accuracy: 0.8168 - val_loss: 2.1429 - val_accuracy: 0.5498
Epoch 41/50
781/781 [==============================] - 76s 97ms/step - loss: 0.5504 - accuracy: 0.8233 - val_loss: 2.1303 - val_accuracy: 0.5535
Epoch 42/50
781/781 [==============================] - 75s 96ms/step - loss: 0.5443 - accuracy: 0.8281 - val_loss: 2.0963 - val_accuracy: 0.5619
Epoch 43/50
781/781 [==============================] - 76s 97ms/step - loss: 0.5199 - accuracy: 0.8339 - val_loss: 2.0854 - val_accuracy: 0.5711
Epoch 44/50
781/781 [==============================] - 75s 96ms/step - loss: 0.5131 - accuracy: 0.8354 - val_loss: 2.1756 - val_accuracy: 0.5547
Epoch 45/50
781/781 [==============================] - 75s 96ms/step - loss: 0.4953 - accuracy: 0.8425 - val_loss: 2.2492 - val_accuracy: 0.5516
Epoch 46/50
781/781 [==============================] - 75s 96ms/step - loss: 0.4922 - accuracy: 0.8423 - val_loss: 2.1029 - val_accuracy: 0.5659
Epoch 47/50
781/781 [==============================] - 75s 97ms/step - loss: 0.4771 - accuracy: 0.8460 - val_loss: 2.2959 - val_accuracy: 0.5531
Epoch 48/50
781/781 [==============================] - 75s 96ms/step - loss: 0.4538 - accuracy: 0.8533 - val_loss: 2.1392 - val_accuracy: 0.5721
Epoch 49/50
781/781 [==============================] - 75s 96ms/step - loss: 0.4494 - accuracy: 0.8553 - val_loss: 2.2885 - val_accuracy: 0.5583
Epoch 50/50
781/781 [==============================] - 75s 96ms/step - loss: 0.4392 - accuracy: 0.8612 - val_loss: 2.3241 - val_accuracy: 0.5532
 the test accuracy is : 55.32%
```



1) **At least five convolutional layers in your network**
2) **Different learning rates for different convolutional layers**

```
#importing the libraries
import numpy as np
import sklearn.metrics as metrics
from keras.models import Sequential
from keras.layers import Dense, Conv2D, MaxPooling2D, Flatten,
BatchNormalization,Dropout
from keras.datasets import cifar100
from keras.utils import np_utils
from keras.optimizers import Adam
from keras.preprocessing.image import ImageDataGenerator
import matplotlib.pyplot as plt
# Define the model
model = Sequential()
# first convolutional layer
model.add(Conv2D(64, kernel_size=(3,3), padding='same', activation='relu',
input_shape=(32, 32, 3)))
model.add(BatchNormalization())
# second convolutional layer
model.add(Conv2D(64, kernel_size=(3,3), padding='same', activation='relu'))
model.add(BatchNormalization())
model.add(MaxPooling2D(pool_size=(2,2)))
model.add(Dropout(0.25))
# third convolutional layer
model.add(Conv2D(128, kernel_size=(3,3), padding='same', activation='relu'))
model.add(BatchNormalization())
# fourth convolutional layer
model.add(Conv2D(128, kernel_size=(3,3), padding='same', activation='relu'))
model.add(BatchNormalization())
model.add(MaxPooling2D(pool_size=(2,2)))
model.add(Dropout(0.25))
# fifth convolutional layer
model.add(Conv2D(256, kernel_size=(3,3), padding='same', activation='relu'))
model.add(BatchNormalization())
model.add(MaxPooling2D(pool_size=(2,2)))
model.add(Dropout(0.25))
```

```
model.add(Flatten())
#fully connected layer
model.add(Dense(512, activation='relu'))
model.add(BatchNormalization())
model.add(Dropout(0.5))
model.add(Dense(100, activation='softmax'))
# printing the model summary
model.summary()
# loading the cifar100 data
(cifarx_train, cifary_train), (cifarx_test, cifary_test) = cifar100.load_data()
# normalize the pixel values
cifarx_train = cifarx_train.astype('float32') / 255
cifarx_test = cifarx_test.astype('float32') / 255
# convert the labels to one-hot encoded vectors
cifary_train = np_utils.to_categorical(cifary_train, 100)
cifary_test = np_utils.to_categorical(cifary_test, 100)
# define different learning rates for different convolutional layers
lr_schedule = {0: 0.01,1: 0.0002, 2: 0.001,3: 0.0005,4: 0.0001}
# define the adam optimizer
optimizer = Adam(lr=lr_schedule[0])
model.compile(loss='categorical_crossentropy', optimizer=optimizer,
metrics=["accuracy"])
# define data augmentation parameters
datagen = ImageDataGenerator(rotation_range=15, width_shift_range=0.1,
height_shift_range=0.1,horizontal_flip=True)
# Using Adam and set learning rate 0.001
optimizer = Adam(lr=0.001)
# compile the model
model.compile(loss='categorical_crossentropy', optimizer=optimizer,
metrics=["accuracy"])
# train the model
history = model.fit(datagen.flow(cifarx_train,cifary_train, batch_size=128, shuffle=True),
                    steps_per_epoch=len(cifarx_train)/128, epochs=1,
validation_data=(cifarx_test, cifary_test))
# Evaluate the model
scores = model.evaluate(cifarx_test, cifary_test, verbose=0)
print("Test Accuracy: %.2f%%" % (scores[1]*100))
# Define plotchart function
def plotchart(history, value):
    plt.figure(figsize=[8,6])
    plt.plot(history.history['loss'], 'firebrick', linewidth=3.0)
    plt.plot(history.history['accuracy'], 'turquoise', linewidth=3.0)
    plt.legend(['Training loss', 'Training Accuracy'], fontsize=18)
    plt.xlabel('Epochs', fontsize=16)
```

```
    plt.ylabel('Loss and Accuracy', fontsize=16)
    plt.title('Loss and Accuracy Curves for {}'.format(value), fontsize=16)
    plt.show()
# Plot the training history
plotchart(history, 'CIFAR-100 image classification task')
```

<u>Outputs :2</u> Here i have attached the screenshot of custom network architecture output which you can see below.