# Assignment no :3

**Name : CHOVATIYA JAY MAHESHBHAI**

# Student ID : 1161980

**Course: COMP5421 Deep Learning Topic:Datasets: CIFAR100**

**Requirements :**

**Last digit of ID: 0**

**Data set used: CIFAR100**

**Second last digit of ID: 8**

**DCNN used: Dense-Net**

**Notes : In this requirement 2 both convolutional neural network architectures such as Densnet and Alexnet i tried to implement in order to check the accuracy of the model which you can see below in the given output.**

**Explanation:**

**Q-Compare the performance gap between the pretrained DCNN model in the condition of transfer learning, and your customized model in assignment 2 in the condition of training from scratch. Explain why you obtained such results, and give a brief discussion about it.**

**Ans:** Obviously, **the performance** is **better** in **transfer learning**. To elaborate it,

In transfer learning, we have used the pretrained dense net model on the ImageNet which has 16 million data and directly use it to the cifar100.

So, the **testing time(93s/epoch)** is quite **less** compared to **training from scratch(137s/epoch)** and **accuracy(63.54)** is quite **higher** than **training from scratch(52.03).** Hence, we can get good results from pretrained models rather that scratch we had done in the assignment 2 by using the customized model.

## IMPLEMENTATION :

# import all 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.models import Sequential

from keras.layers import Dense,Flatten,Dropout

from keras.preprocessing.image import ImageDataGenerator

import matplotlib.pyplot as plt

# for removing warnings

import warnings

warnings.filterwarnings('ignore')

# define imagenet-pretrained model for densenet(cifar100)

model = Sequential()

model.add(densenet.DenseNet121(weights='imagenet', include\_top=False,

input\_shape=(32,32,3), pooling='max'))

# adding dense layer for flatten

model.add(Dense(256, activation='relu'))

# deactivating 50% nodes

model.add(Dropout(0.5))

model.add(Dense(100, activation='softmax'))

# find 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)

# data augmentation

datagen = ImageDataGenerator(rotation\_range=15, width\_shift\_range=0.1, height\_shift\_range=0.1, horizontal\_flip=True)

datagen.fit(cifarx\_train)

# one-hot encoding

Y\_train = np\_utils.to\_categorical(cifary\_train, 100)

Y\_test = np\_utils.to\_categorical(cifary\_test, 100)

# Using Adam optimizer to speed up training and set learning rate 0.001

optimizer = Adam(lr=1e-4)

# compile the model

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

# train the model

history = model.fit(datagen.flow(cifarx\_train, Y\_train, batch\_size=64),

steps\_per\_epoch=len(cifarx\_train) / 64, epochs=50,

validation\_data=(cifarx\_test, Y\_test))

Preds = model.predict(cifarx\_test)

y\_Pred = np.argmax(Preds, axis=1)

y\_true = cifary\_test.flatten()

# finding accuracy and loss

accuracy = metrics.accuracy\_score(y\_true, y\_Pred) \* 100

# print testing accuracy

print("Accuracy : ", accuracy)

# 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)

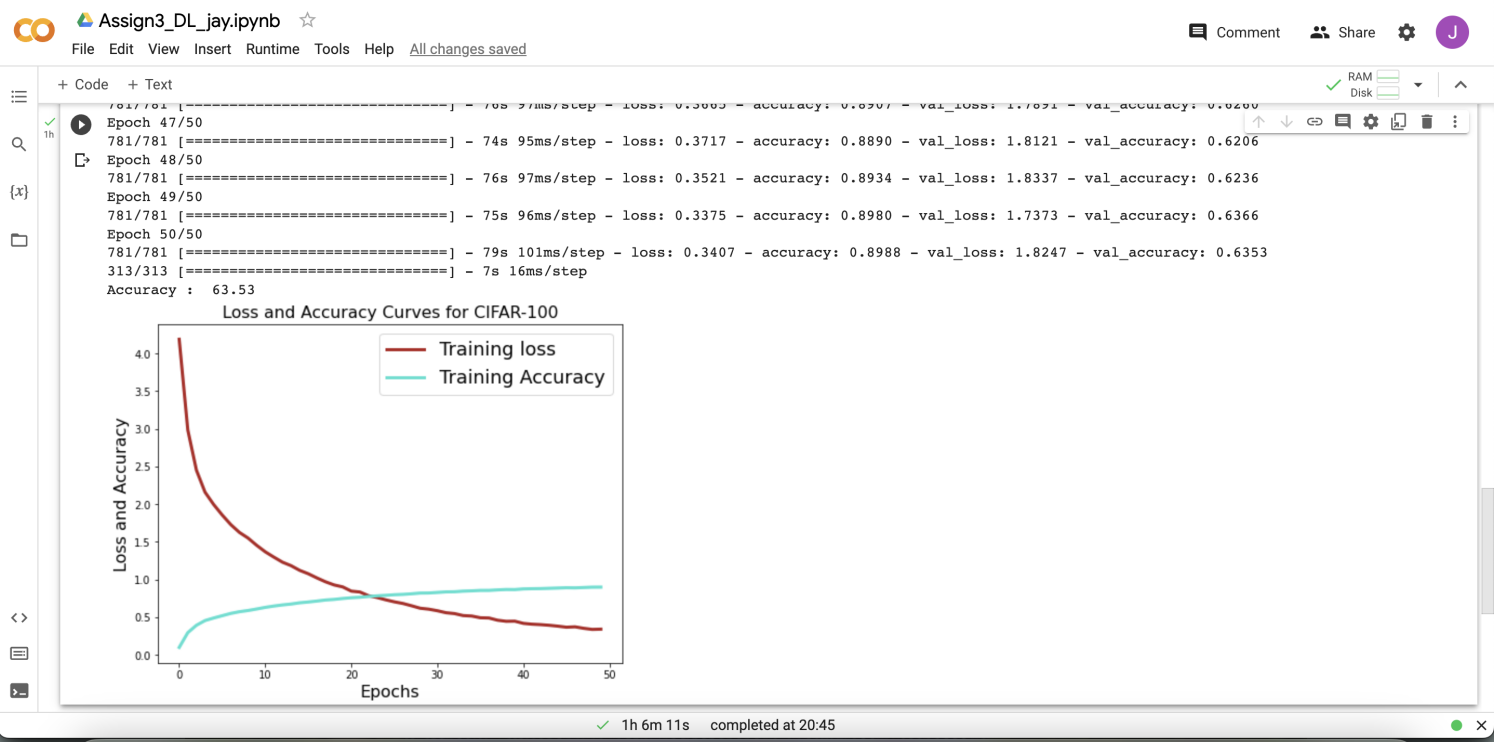
plt.title('Loss and Accuracy Curves for {}'.format(value), fontsize=16)

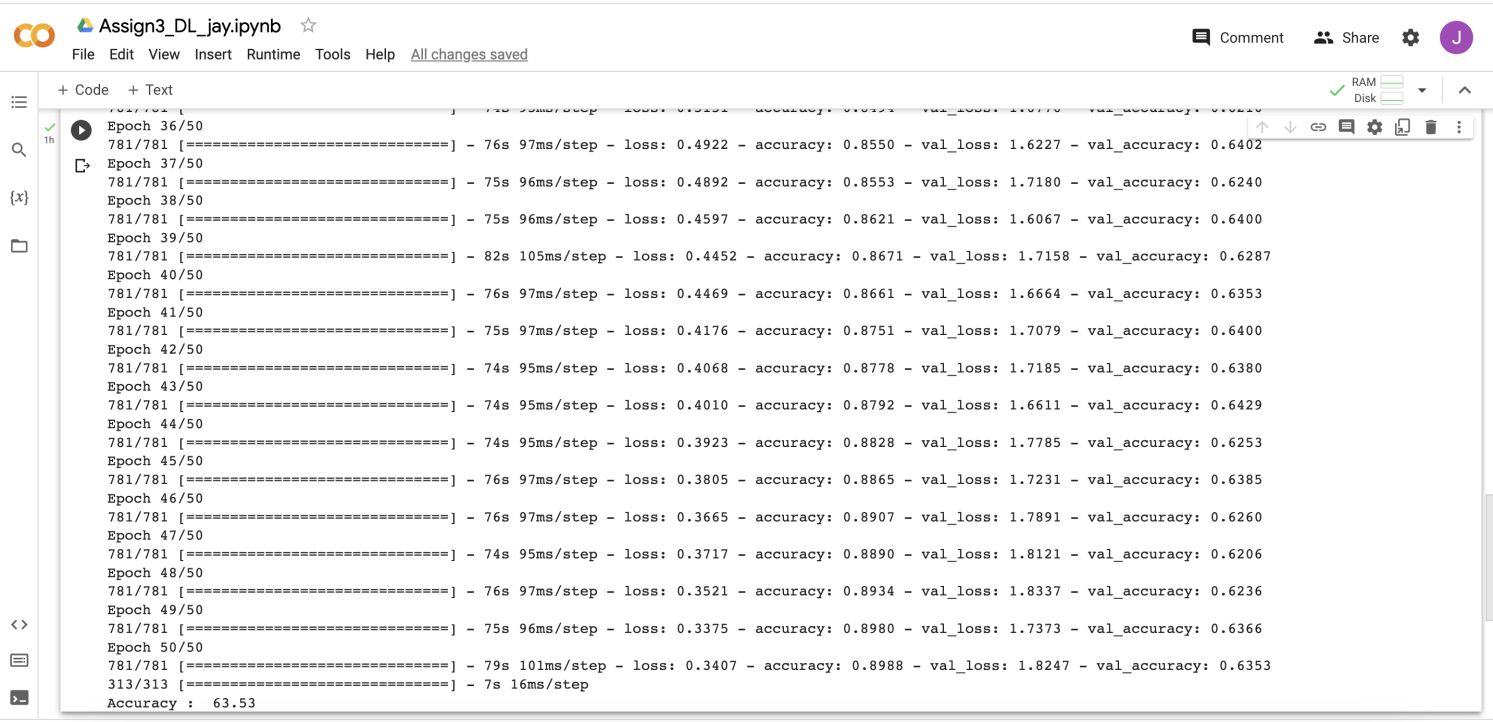
plt.show()

# Plot the training history

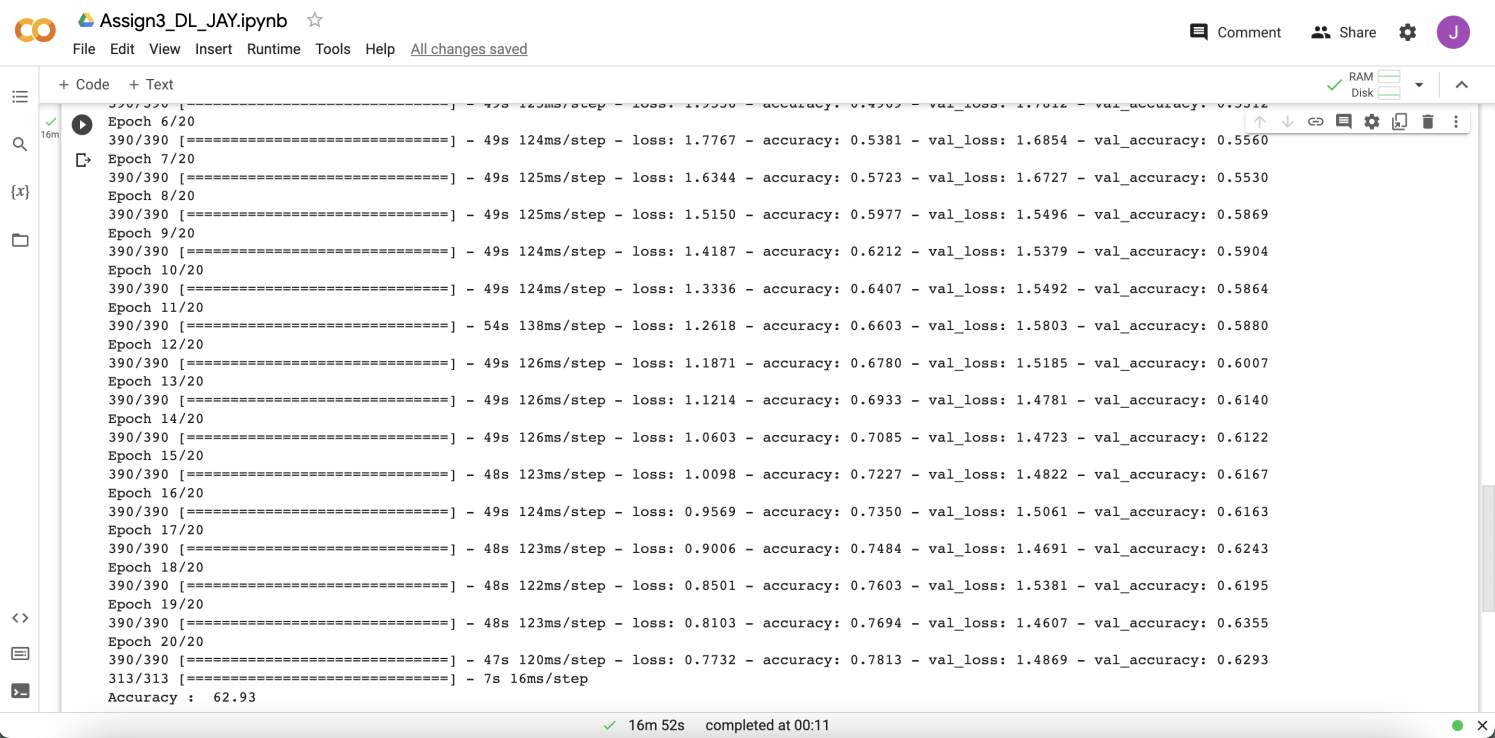
plotchart(history, 'CIFAR-100')

**Outputs:1** Here i have attached the screenshot of CIFAR100 datasets output which you can see below.





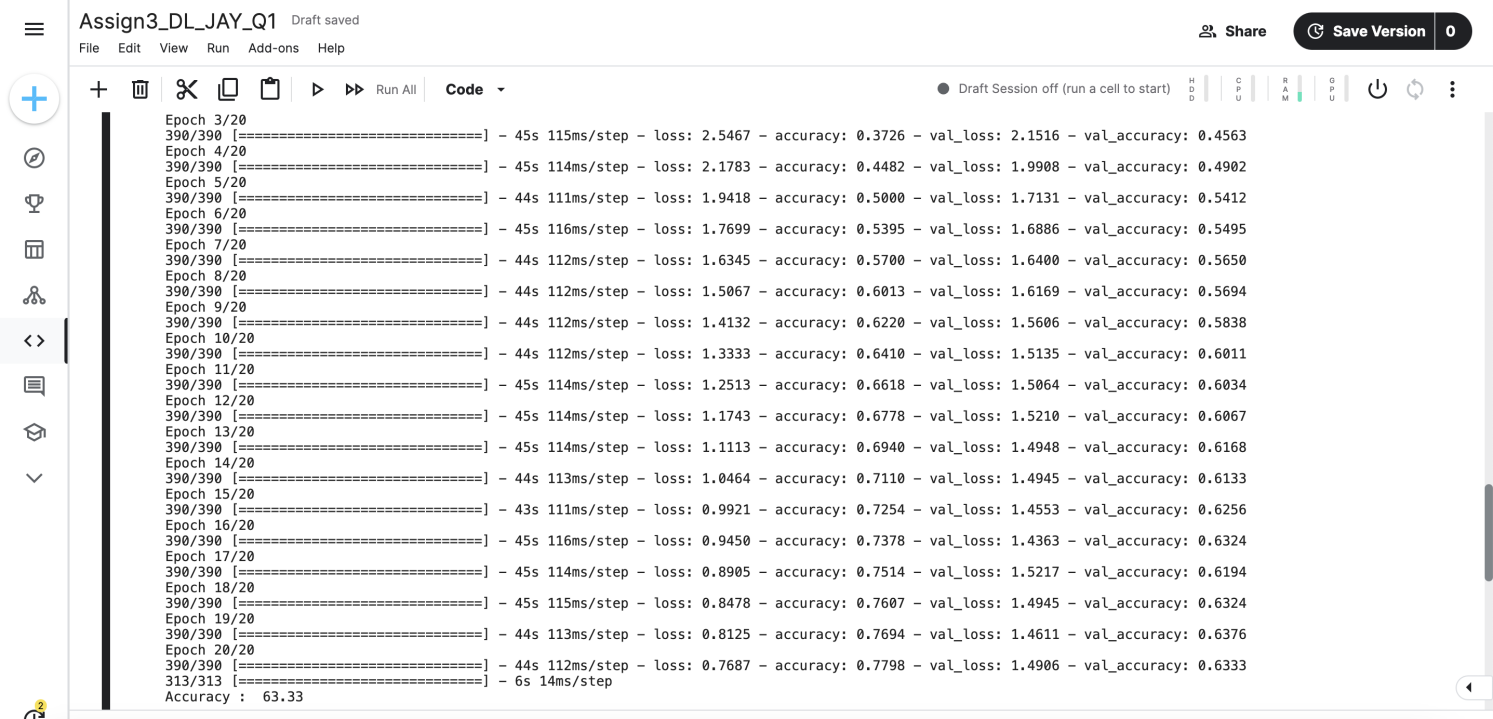
## Outputs:1.1



**Outputs:1.2**



**Outputs:1.3**

****

**Notes : In this question both convolutional neural network architectures such as Densnet and Alexnet i tried to implement in order to check the accuracy of the model which you can see below in the given output.**

**Requirement 2 :Use Freezing layers pretrained DCNN.**

**Q-Obtain the transfer learning features/deep features from a raw dataset.**

#importing library

import torch

import torch.nn as nn

import torchvision

import torchvision.transforms as transforms

# Declaring hyperparameters

epoch\_no = 20 ; no\_classes = 100

batch\_size = 128 ; learning\_rate = 0.1

# Define transformations for training and testing datasets

transform\_train = transforms.Compose([transforms.RandomCrop(32, padding=4),transforms.RandomHorizontalFlip(),transforms.ToTensor(),transforms.Normalize((0.5071, 0.4865, 0.4409), (0.2673, 0.2564, 0.2762))])

transform\_test = transforms.Compose([transforms.ToTensor(),transforms.Normalize((0.5071, 0.4865, 0.4409), (0.2673, 0.2564, 0.2762))])

# Loading CIFAR-100 dataset

cifer\_train = torchvision.datasets.CIFAR100(root='./data', train=True, download=True, transform=transform\_train)

cifer\_test = torchvision.datasets.CIFAR100(root='./data', train=False, download=True, transform=transform\_test)

# Defining data loaders for training and testing datasets

cifer\_loader\_train = torch.utils.data.DataLoader(dataset=cifer\_train, batch\_size=batch\_size, shuffle=True)

cifer\_loader\_test = torch.utils.data.DataLoader(dataset=cifer\_test, batch\_size=batch\_size, shuffle=False)

# Defining DenseNet-121 model and loss function and optimizer

model = torchvision.models.densenet121(pretrained=False, no\_classes=no\_classes)

criterion = nn.CrossEntropyLoss()

optimizer = torch.optim.SGD(model.parameters(), lr=learning\_rate, momentum=0.9, weight\_decay=1e-4)

# Training the model

total\_step = len(cifer\_loader\_train)

for epoch in range(epoch\_no):

for i, (images, labels) in enumerate(cifer\_loader\_train):

outputs = model(images)

loss = criterion(outputs, labels)

optimizer.zero\_grad()

loss.backward()

optimizer.step()

if (i+1) % 100 == 0:

print('Epoch [{}/{}], Step [{}/{}], Loss: {:.4f}'

.format(epoch+1, epoch\_no, i+1, total\_step, loss.item()))

# Testing the model

model.eval()

with torch.no\_grad():

correct = 0

total = 0

for images, labels in cifer\_loader\_test:

outputs = model(images)

\_, predicted = torch.max(outputs.data, 1)

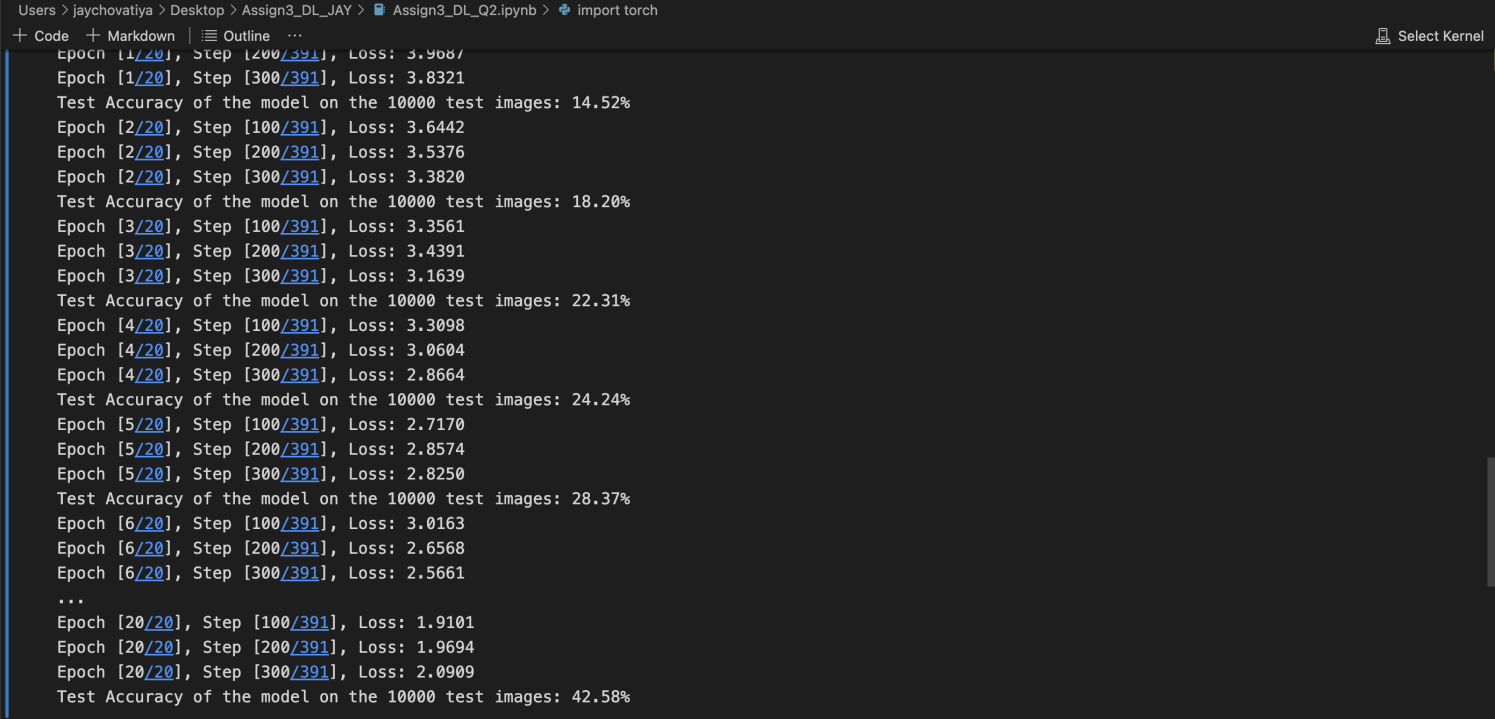
total += labels.size(0)

correct += (predicted == labels).sum().item()

accuracy = 100 \* correct / total

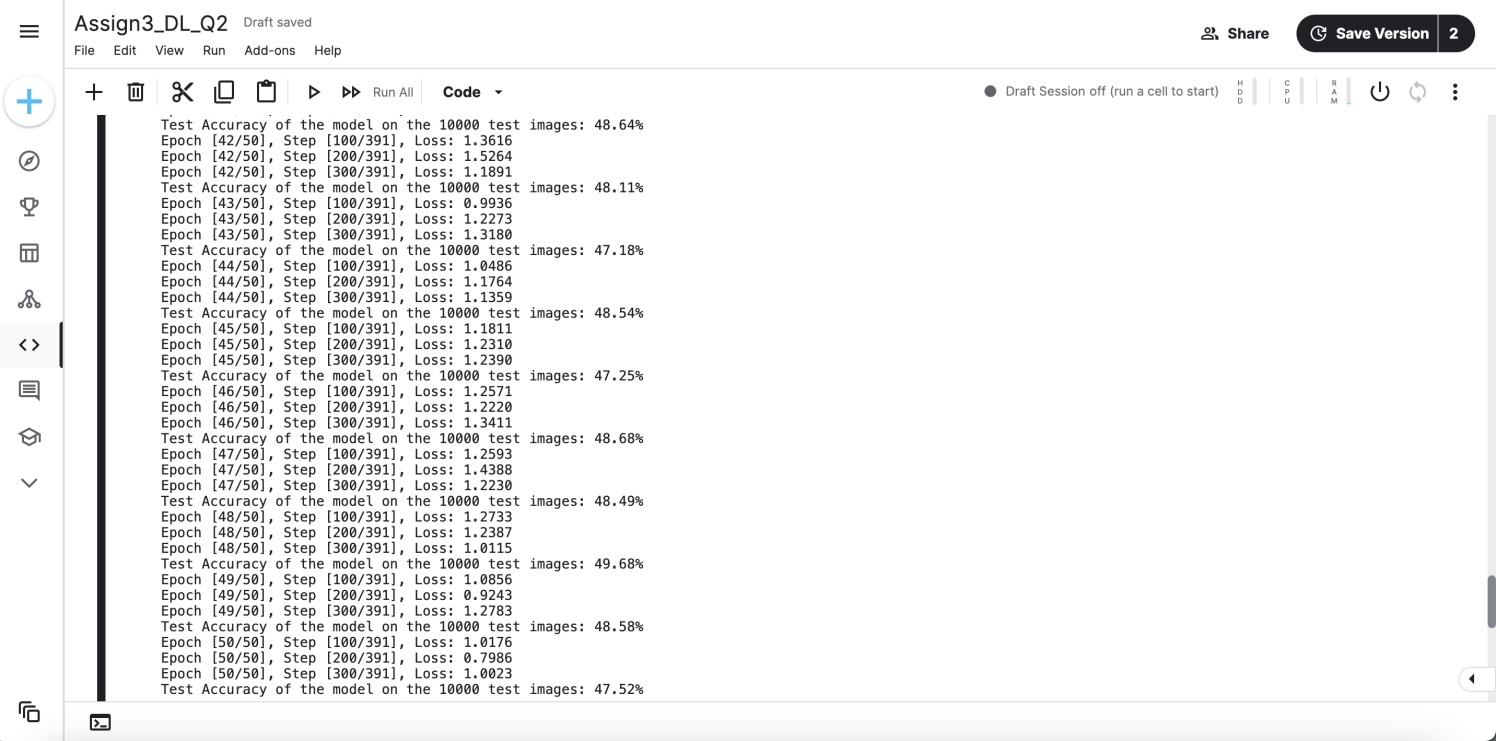
print('Test Accuracy of the model on the {} test images: {:.2f}%'.format(total, accuracy))

**Outputs:2.1** Here i have attached the screenshot of obtain the transfer learning features from a raw dataset output which you can see below.

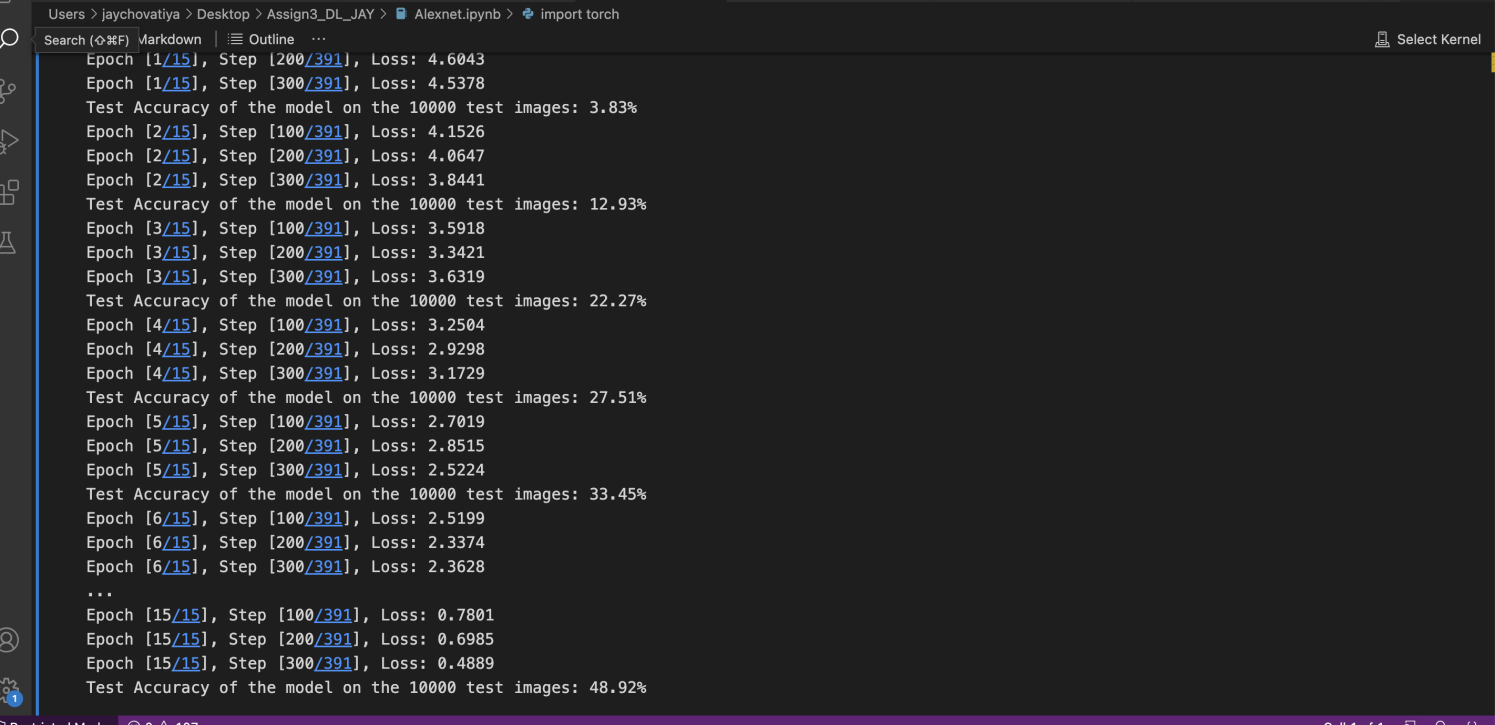


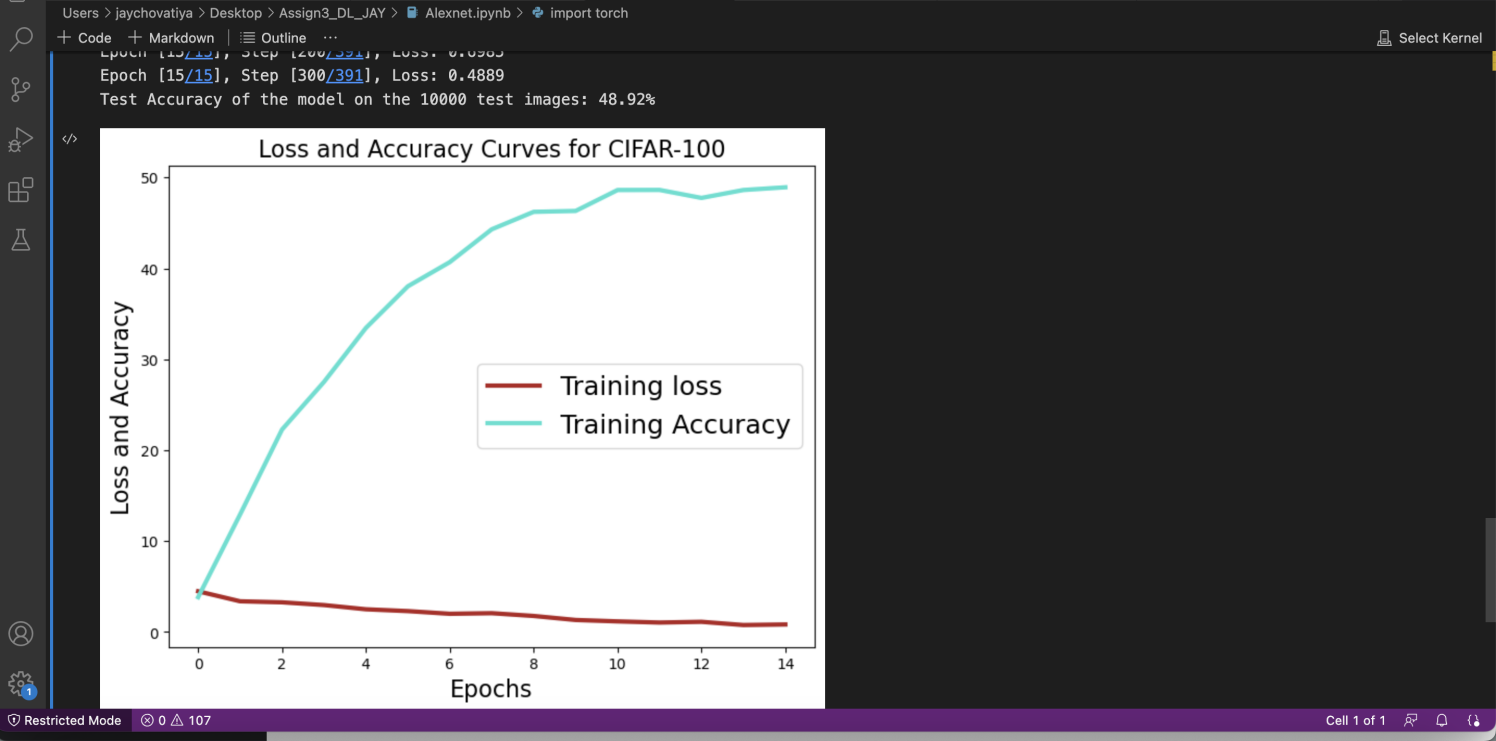
**Outputs:2.2**





**Outputs:2.3**





I have tried to implement using the keras as well in which i received the output like this.

