Assignment no :3

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

<u>Requirements :</u>

Last digit of ID: 0

Data set used: CIFAR100

Second last digit of ID: 8

DCNN used: Dense-Net

<u>Notes :</u> 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.

<u>Explanation:</u>

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.

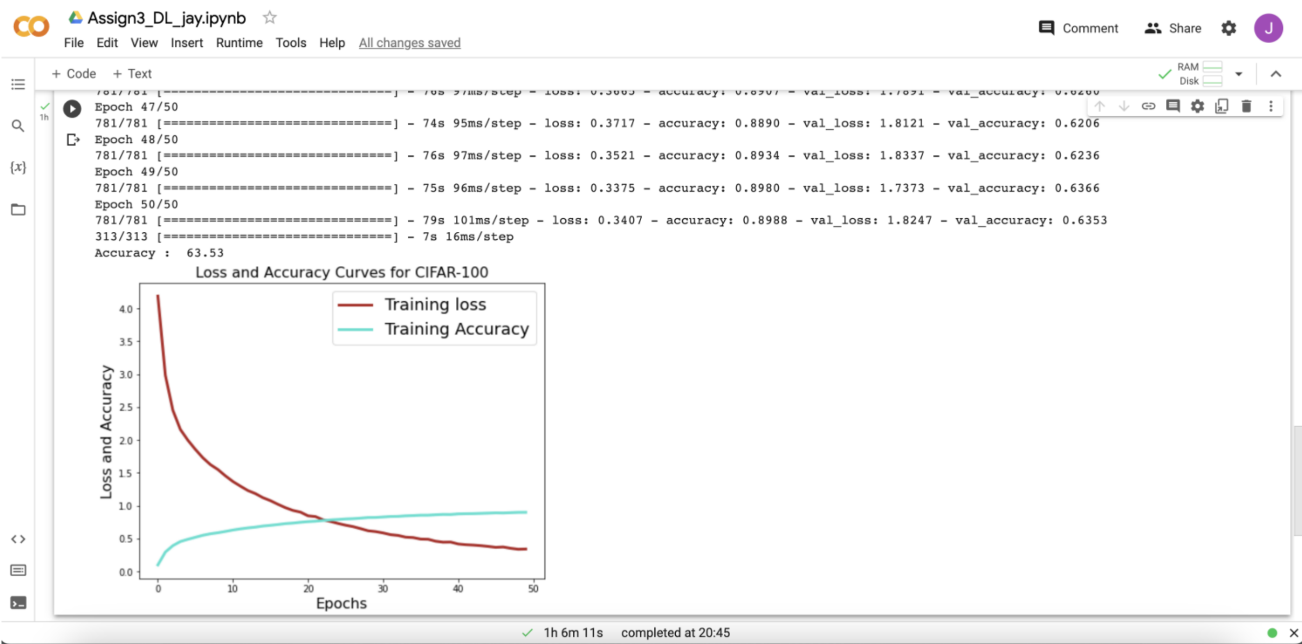
Ans: Obviously, **the performance** is **better** in **transfer learning**. To elaborate it, In transfer learning, we have used the pretrained dense net model on the ImageNet which has 16 million data and directly use it to the cifar100.
So, the **testing time(93s/epoch)** is quite **less** compared 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.

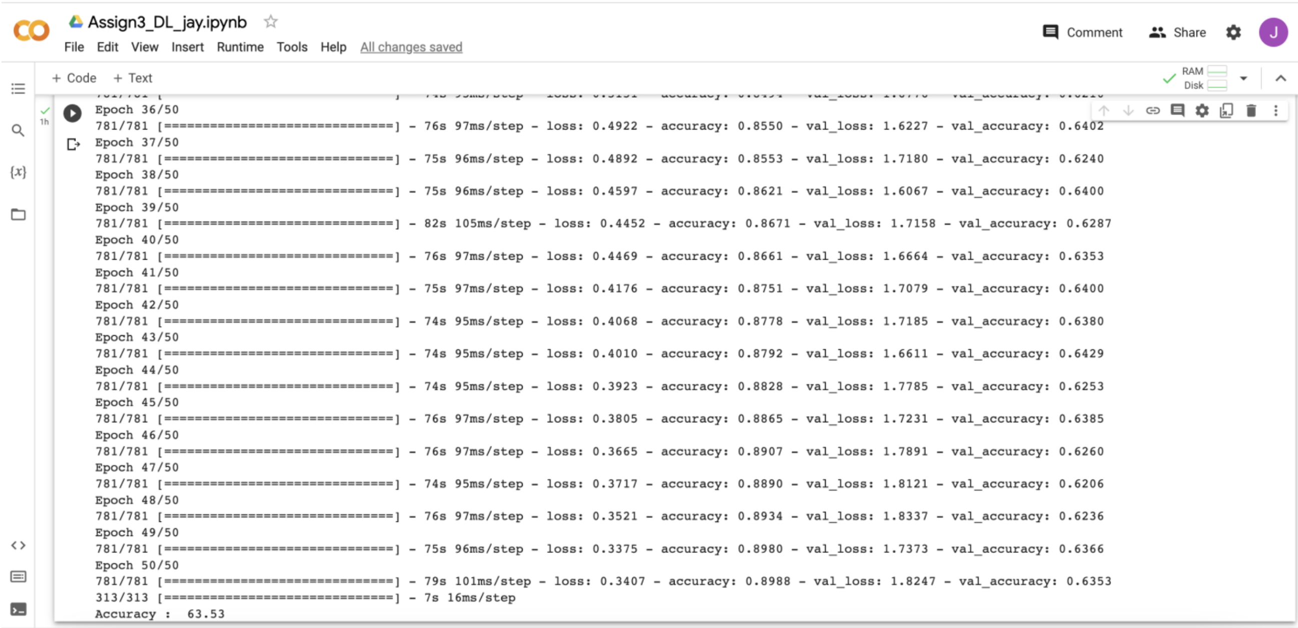
IMPLEMENTATION :

```
# 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')
```

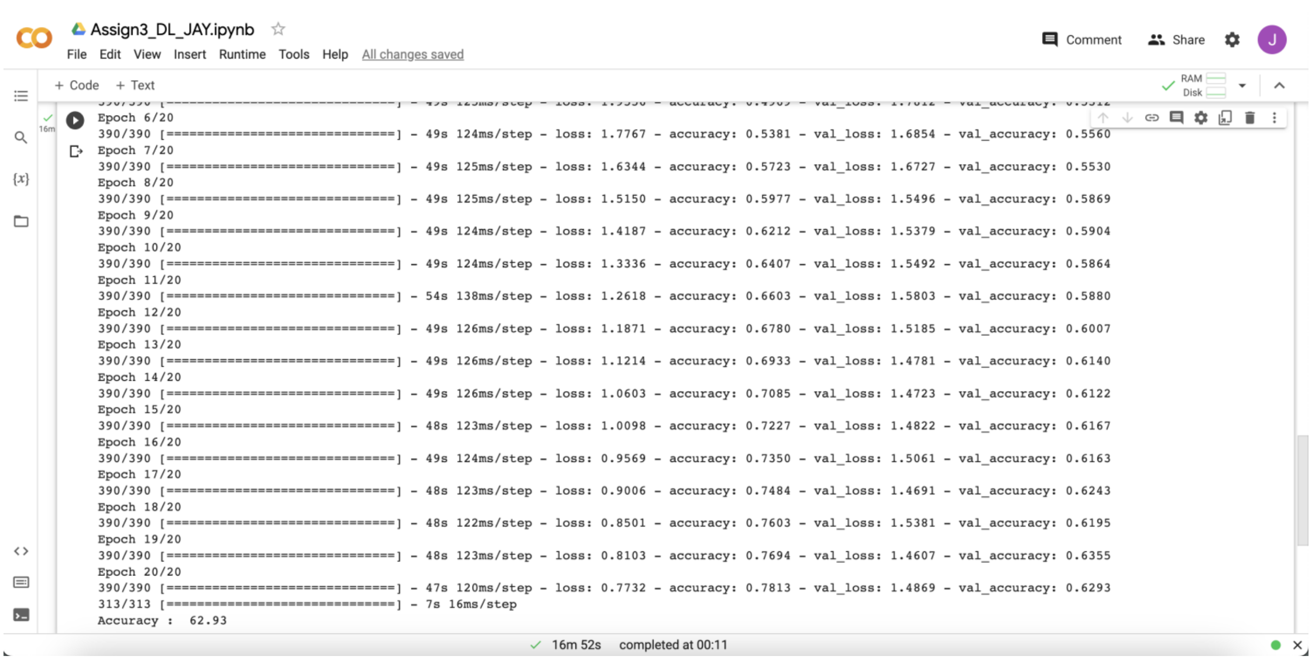
<u>Outputs:1</u> Here i have attached the screenshot of CIFAR100 datasets output which you can see below.





```
Epoch 36/50
781/781 [==============================] - 76s 97ms/step - loss: 0.4922 - accuracy: 0.8550 - val_loss: 1.6227 - val_accuracy: 0.6402
Epoch 37/50
781/781 [==============================] - 75s 96ms/step - loss: 0.4892 - accuracy: 0.8553 - val_loss: 1.7180 - val_accuracy: 0.6240
Epoch 38/50
781/781 [==============================] - 75s 96ms/step - loss: 0.4597 - accuracy: 0.8621 - val_loss: 1.6067 - val_accuracy: 0.6400
Epoch 39/50
781/781 [==============================] - 82s 105ms/step - loss: 0.4452 - accuracy: 0.8671 - val_loss: 1.7158 - val_accuracy: 0.6287
Epoch 40/50
781/781 [==============================] - 76s 97ms/step - loss: 0.4469 - accuracy: 0.8661 - val_loss: 1.6664 - val_accuracy: 0.6353
Epoch 41/50
781/781 [==============================] - 75s 97ms/step - loss: 0.4176 - accuracy: 0.8751 - val_loss: 1.7079 - val_accuracy: 0.6400
Epoch 42/50
781/781 [==============================] - 74s 95ms/step - loss: 0.4068 - accuracy: 0.8778 - val_loss: 1.7185 - val_accuracy: 0.6380
Epoch 43/50
781/781 [==============================] - 74s 95ms/step - loss: 0.4010 - accuracy: 0.8792 - val_loss: 1.6611 - val_accuracy: 0.6429
Epoch 44/50
781/781 [==============================] - 74s 95ms/step - loss: 0.3923 - accuracy: 0.8828 - val_loss: 1.7785 - val_accuracy: 0.6253
Epoch 45/50
781/781 [==============================] - 76s 97ms/step - loss: 0.3805 - accuracy: 0.8865 - val_loss: 1.7231 - val_accuracy: 0.6385
Epoch 46/50
781/781 [==============================] - 76s 97ms/step - loss: 0.3665 - accuracy: 0.8907 - val_loss: 1.7891 - val_accuracy: 0.6260
Epoch 47/50
781/781 [==============================] - 74s 95ms/step - loss: 0.3717 - accuracy: 0.8890 - val_loss: 1.8121 - val_accuracy: 0.6206
Epoch 48/50
781/781 [==============================] - 76s 97ms/step - loss: 0.3521 - accuracy: 0.8934 - val_loss: 1.8337 - val_accuracy: 0.6236
Epoch 49/50
781/781 [==============================] - 75s 96ms/step - loss: 0.3375 - accuracy: 0.8980 - val_loss: 1.7373 - val_accuracy: 0.6366
Epoch 50/50
781/781 [==============================] - 79s 101ms/step - loss: 0.3407 - accuracy: 0.8988 - val_loss: 1.8247 - val_accuracy: 0.6353
313/313 [==============================] - 7s 16ms/step
Accuracy :  63.53
```

Outputs:1.1

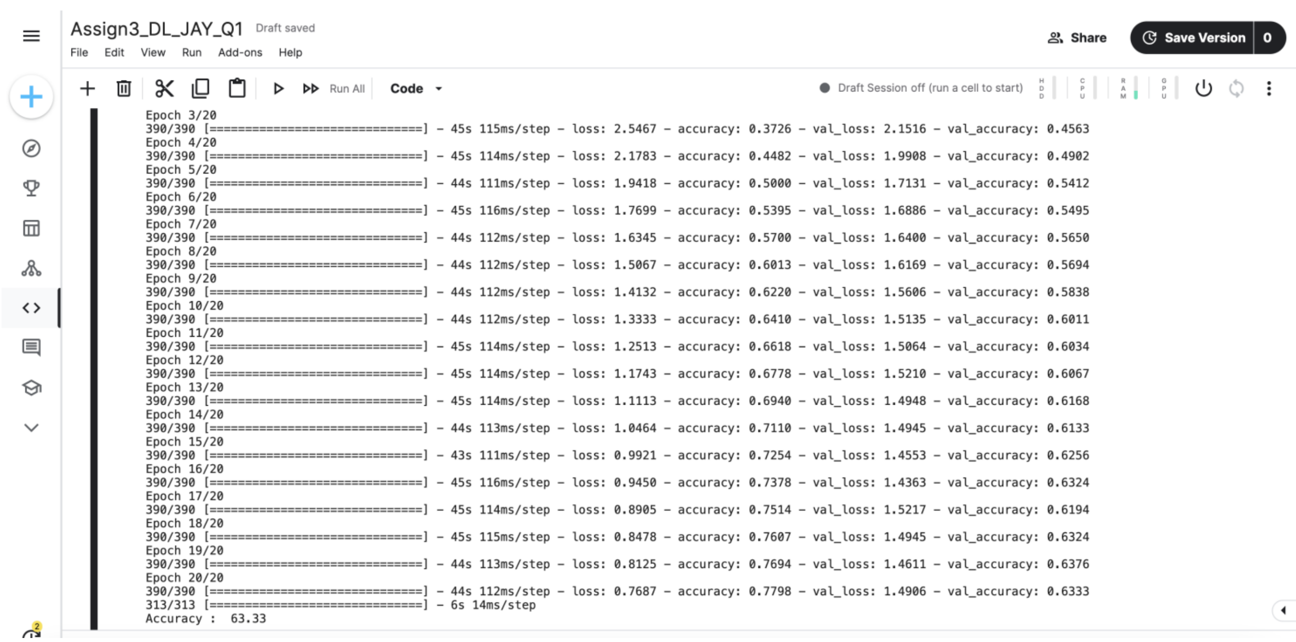


```
Epoch 6/20
390/390 [==============================] - 49s 124ms/step - loss: 1.7767 - accuracy: 0.5381 - val_loss: 1.6854 - val_accuracy: 0.5560
Epoch 7/20
390/390 [==============================] - 49s 125ms/step - loss: 1.6344 - accuracy: 0.5723 - val_loss: 1.6727 - val_accuracy: 0.5530
Epoch 8/20
390/390 [==============================] - 49s 125ms/step - loss: 1.5150 - accuracy: 0.5977 - val_loss: 1.5496 - val_accuracy: 0.5869
Epoch 9/20
390/390 [==============================] - 49s 124ms/step - loss: 1.4187 - accuracy: 0.6212 - val_loss: 1.5379 - val_accuracy: 0.5904
Epoch 10/20
390/390 [==============================] - 49s 124ms/step - loss: 1.3336 - accuracy: 0.6407 - val_loss: 1.5492 - val_accuracy: 0.5864
Epoch 11/20
390/390 [==============================] - 54s 138ms/step - loss: 1.2618 - accuracy: 0.6603 - val_loss: 1.5803 - val_accuracy: 0.5880
Epoch 12/20
390/390 [==============================] - 49s 126ms/step - loss: 1.1871 - accuracy: 0.6780 - val_loss: 1.5185 - val_accuracy: 0.6007
Epoch 13/20
390/390 [==============================] - 49s 126ms/step - loss: 1.1214 - accuracy: 0.6933 - val_loss: 1.4781 - val_accuracy: 0.6140
Epoch 14/20
390/390 [==============================] - 49s 126ms/step - loss: 1.0603 - accuracy: 0.7085 - val_loss: 1.4723 - val_accuracy: 0.6122
Epoch 15/20
390/390 [==============================] - 48s 123ms/step - loss: 1.0098 - accuracy: 0.7227 - val_loss: 1.4822 - val_accuracy: 0.6167
Epoch 16/20
390/390 [==============================] - 49s 124ms/step - loss: 0.9569 - accuracy: 0.7350 - val_loss: 1.5061 - val_accuracy: 0.6163
Epoch 17/20
390/390 [==============================] - 48s 123ms/step - loss: 0.9006 - accuracy: 0.7484 - val_loss: 1.4691 - val_accuracy: 0.6243
Epoch 18/20
390/390 [==============================] - 48s 122ms/step - loss: 0.8501 - accuracy: 0.7603 - val_loss: 1.5381 - val_accuracy: 0.6195
Epoch 19/20
390/390 [==============================] - 48s 123ms/step - loss: 0.8103 - accuracy: 0.7694 - val_loss: 1.4607 - val_accuracy: 0.6355
Epoch 20/20
390/390 [==============================] - 47s 120ms/step - loss: 0.7732 - accuracy: 0.7813 - val_loss: 1.4869 - val_accuracy: 0.6293
313/313 [==============================] - 7s 16ms/step
Accuracy :  62.93
```

Outputs:1.2


Assign3_DL_JAY.ipynb
File Edit View Insert Runtime Tools Help All changes saved
390/390 [==============================] - 49s 124ms/step - loss: 0.9569 - accuracy: 0.7350 - val_loss: 1.5061 - val_accuracy: 0.6163
Epoch 17/20
390/390 [==============================] - 48s 123ms/step - loss: 0.9006 - accuracy: 0.7484 - val_loss: 1.4691 - val_accuracy: 0.6243
Epoch 18/20
390/390 [==============================] - 48s 122ms/step - loss: 0.8501 - accuracy: 0.7603 - val_loss: 1.5381 - val_accuracy: 0.6195
Epoch 19/20
390/390 [==============================] - 48s 123ms/step - loss: 0.8103 - accuracy: 0.7694 - val_loss: 1.4607 - val_accuracy: 0.6355
Epoch 20/20
390/390 [==============================] - 47s 120ms/step - loss: 0.7732 - accuracy: 0.7813 - val_loss: 1.4869 - val_accuracy: 0.6293
313/313 [==============================] - 7s 16ms/step
Accuracy : 62.93
Loss and Accuracy Curves for CIFAR-100
Training loss
Training Accuracy
Loss and Accuracy
Epochs
16m 52s completed at 00:11

Outputs:1.3


Assign3_DL_JAY_Q1 Draft saved
File Edit View Run Add-ons Help
Draft Session off (run a cell to start)
Epoch 3/20
390/390 [==============================] - 45s 115ms/step - loss: 2.5467 - accuracy: 0.3726 - val_loss: 2.1516 - val_accuracy: 0.4563
Epoch 4/20
390/390 [==============================] - 45s 114ms/step - loss: 2.1783 - accuracy: 0.4482 - val_loss: 1.9908 - val_accuracy: 0.4902
Epoch 5/20
390/390 [==============================] - 44s 111ms/step - loss: 1.9418 - accuracy: 0.5000 - val_loss: 1.7131 - val_accuracy: 0.5412
Epoch 6/20
390/390 [==============================] - 45s 116ms/step - loss: 1.7699 - accuracy: 0.5395 - val_loss: 1.6886 - val_accuracy: 0.5495
Epoch 7/20
390/390 [==============================] - 44s 112ms/step - loss: 1.6345 - accuracy: 0.5700 - val_loss: 1.6400 - val_accuracy: 0.5650
Epoch 8/20
390/390 [==============================] - 44s 112ms/step - loss: 1.5067 - accuracy: 0.6013 - val_loss: 1.6169 - val_accuracy: 0.5694
Epoch 9/20
390/390 [==============================] - 44s 112ms/step - loss: 1.4132 - accuracy: 0.6220 - val_loss: 1.5606 - val_accuracy: 0.5838
Epoch 10/20
390/390 [==============================] - 44s 112ms/step - loss: 1.3333 - accuracy: 0.6410 - val_loss: 1.5135 - val_accuracy: 0.6011
Epoch 11/20
390/390 [==============================] - 45s 114ms/step - loss: 1.2513 - accuracy: 0.6618 - val_loss: 1.5064 - val_accuracy: 0.6034
Epoch 12/20
390/390 [==============================] - 45s 114ms/step - loss: 1.1743 - accuracy: 0.6778 - val_loss: 1.5210 - val_accuracy: 0.6067
Epoch 13/20
390/390 [==============================] - 45s 114ms/step - loss: 1.1113 - accuracy: 0.6940 - val_loss: 1.4948 - val_accuracy: 0.6168
Epoch 14/20
390/390 [==============================] - 44s 113ms/step - loss: 1.0464 - accuracy: 0.7110 - val_loss: 1.4945 - val_accuracy: 0.6133
Epoch 15/20
390/390 [==============================] - 43s 111ms/step - loss: 0.9921 - accuracy: 0.7254 - val_loss: 1.4553 - val_accuracy: 0.6256
Epoch 16/20
390/390 [==============================] - 45s 116ms/step - loss: 0.9450 - accuracy: 0.7378 - val_loss: 1.4363 - val_accuracy: 0.6324
Epoch 17/20
390/390 [==============================] - 45s 114ms/step - loss: 0.8905 - accuracy: 0.7514 - val_loss: 1.5217 - val_accuracy: 0.6194
Epoch 18/20
390/390 [==============================] - 45s 115ms/step - loss: 0.8478 - accuracy: 0.7607 - val_loss: 1.4945 - val_accuracy: 0.6324
Epoch 19/20
390/390 [==============================] - 44s 113ms/step - loss: 0.8125 - accuracy: 0.7694 - val_loss: 1.4611 - val_accuracy: 0.6376
Epoch 20/20
390/390 [==============================] - 44s 112ms/step - loss: 0.7687 - accuracy: 0.7798 - val_loss: 1.4906 - val_accuracy: 0.6333
313/313 [==============================] - 6s 14ms/step
Accuracy : 63.33

<u>Notes :</u> 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.

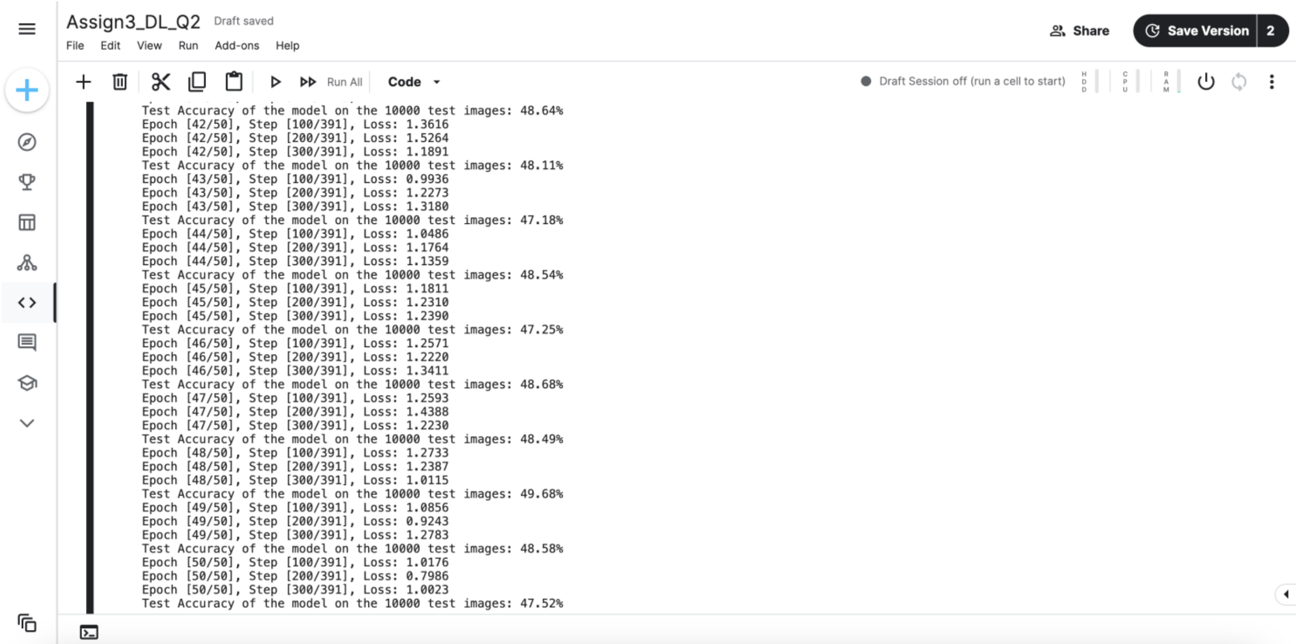
```
#importing library
import torch
import torch.nn as nn
import torchvision
import torchvision.transforms as transforms

# 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.Nor
malize((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))
```

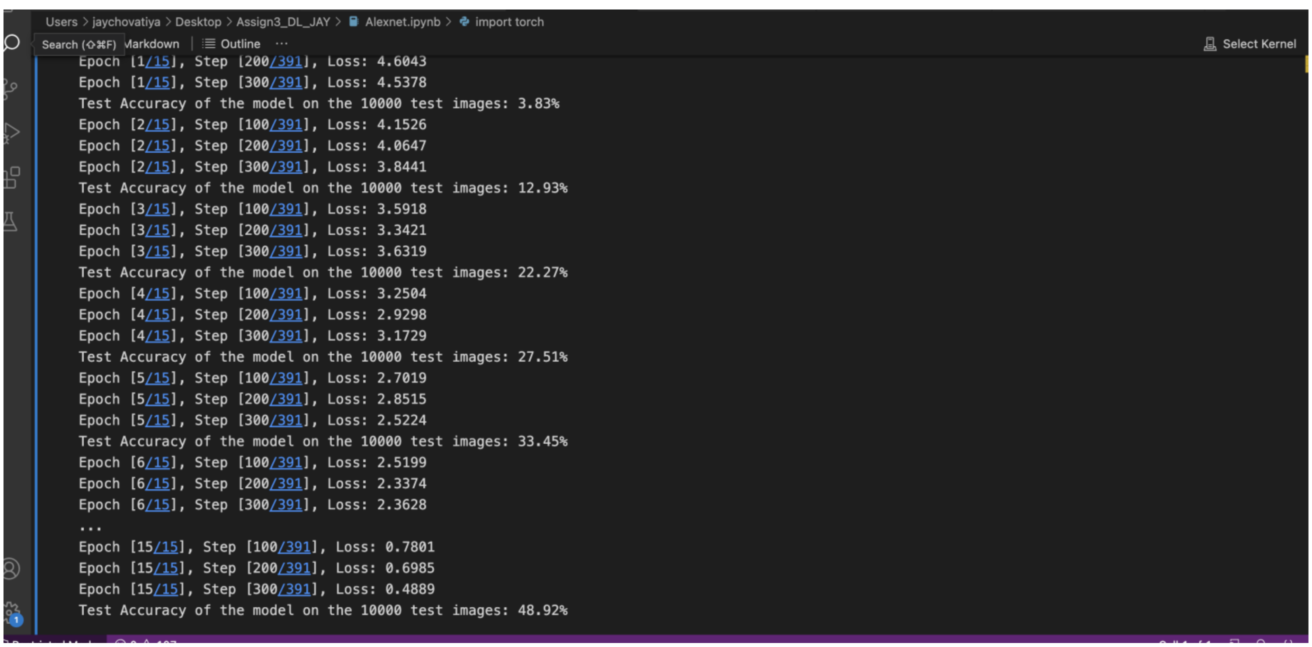
<u>Outputs:</u>2.1 Here i have attached the screenshot of obtain the transfer learning features from a raw dataset output which you can see below.

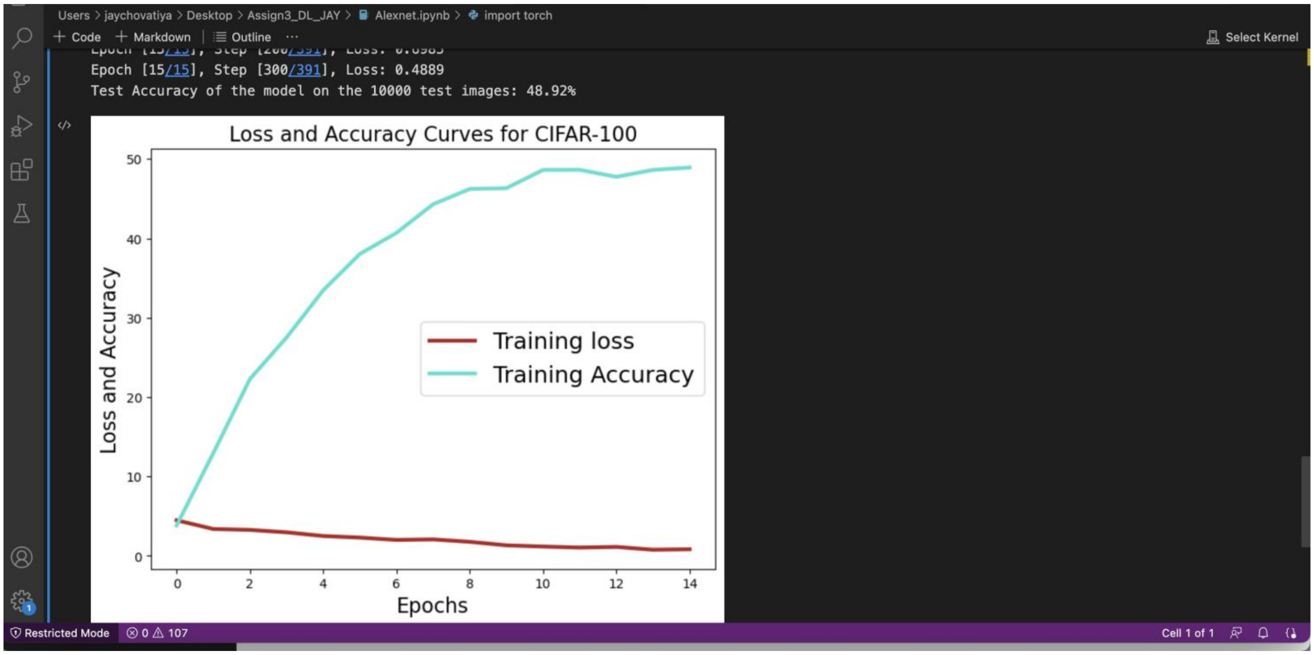
```
Users > jaychovatiya > Desktop > Assign3_DL_JAY > Assign3_DL_Q2.ipynb > import torch
Epoch [1/20], Step [200/391], Loss: 3.9687
Epoch [1/20], Step [300/391], Loss: 3.8321
Test Accuracy of the model on the 10000 test images: 14.52%
Epoch [2/20], Step [100/391], Loss: 3.6442
Epoch [2/20], Step [200/391], Loss: 3.5376
Epoch [2/20], Step [300/391], Loss: 3.3820
Test Accuracy of the model on the 10000 test images: 18.20%
Epoch [3/20], Step [100/391], Loss: 3.3561
Epoch [3/20], Step [200/391], Loss: 3.4391
Epoch [3/20], Step [300/391], Loss: 3.1639
Test Accuracy of the model on the 10000 test images: 22.31%
Epoch [4/20], Step [100/391], Loss: 3.3098
Epoch [4/20], Step [200/391], Loss: 3.0604
Epoch [4/20], Step [300/391], Loss: 2.8664
Test Accuracy of the model on the 10000 test images: 24.24%
Epoch [5/20], Step [100/391], Loss: 2.7170
Epoch [5/20], Step [200/391], Loss: 2.8574
Epoch [5/20], Step [300/391], Loss: 2.8250
Test Accuracy of the model on the 10000 test images: 28.37%
Epoch [6/20], Step [100/391], Loss: 3.0163
Epoch [6/20], Step [200/391], Loss: 2.6568
Epoch [6/20], Step [300/391], Loss: 2.5661
...
Epoch [20/20], Step [100/391], Loss: 1.9101
Epoch [20/20], Step [200/391], Loss: 1.9694
Epoch [20/20], Step [300/391], Loss: 2.0909
Test Accuracy of the model on the 10000 test images: 42.58%
```

<u>Outputs:</u>2.2



<u>Outputs:</u>2.3





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