CNN on CIFR Assignment:

1. Please visit this link to access the state-of-art DenseNet code for reference - DenseNet - cifar10 notebook link
2. You need to create a copy of this and "retrain" this model to achieve 90+ test accuracy.
3. You cannot use DropOut layers.
4. You MUST use Image Augmentation Techniques.
5. You cannot use an already trained model as a beginning points, you have to initilize as your own
6. You cannot run the program for more than 300 Epochs, and it should be clear from your log, that you have only used 300 Epochs
7. You cannot use test images for training the model.
8. You cannot change the general architecture of DenseNet (which means you must use Dense Block, Transition and Output blocks as mentioned in the code)
9. You are free to change Convolution types (e.g. from 3x3 normal convolution to Depthwise Separable, etc)
10. You cannot have more than 1 Million parameters in total
11. You are free to move the code from Keras to Tensorflow, Pytorch, MXNET etc.
12. You can use any optimization algorithm you need.
13. You can checkpoint your model and retrain the model from that checkpoint so that no need of training the model from first if you lost at any epoch while training. You can directly load that model and Train from that epoch.

```
# Load necessary libraries
from tensorflow.keras import models, layers
from tensorflow.keras.models import Model
from tensorflow.keras.layers import BatchNormalization, Activation, Flatten
from tensorflow.keras.optimizers import Adam
from numpy import expand_dims
from keras.preprocessing import image
from keras.preprocessing.image import ImageDataGenerator
import tensorflow as tf
from keras import regularizers
from matplotlib import pyplot
from tensorflow.keras.callbacks import ModelCheckpoint,ReduceLROnPlateau
```

```
# Hyperparameters
batch_size = 128
num_classes = 10
epochs = 10
l = 40
num_filter = 12
compression = 0.5
dropout_rate = 0.2
```

```
# load the dataset

(X_train, y_train),(X_test ,y_test ) = tf.keras.datasets.cifar10.load_data()
```

```
img_height, img_width, channel = X_train.shape[1],X_train.shape[2],X_train.shape[3]

# convert to one hot encoing
y_train = tf.keras.utils.to_categorical(y_train, num_classes)
y_test = tf.keras.utils.to_categorical(y_test, num_classes)
```

```
Downloading data from https://www.cs.toronto.edu/~kriz/cifar-10-python.tar.gz
170498071/170498071 [==============================] - 13s 0us/step
```

CHECKING THE SHAPE OF DATA WE HAVE LOADED

```
X_train.shape, y_train.shape,X_test.shape , X_test.shape
```

```
((50000, 32, 32, 3), (50000, 10), (10000, 32, 32, 3), (10000, 32, 32, 3))
```

converting image pixel in the range of 0 to 1

```
def normalize_pixels(train, test):
    '''
    Normalize data into range of 0 to 1
    '''
    return train.astype('float32')/255, test.astype('float32')/255
```

normalizing the pixel values

```
X_train,X_test=normalize_pixels(X_train,X_test)
```

first we will try the model with dense layer

```
#https://machinelearningmastery.com/how-to-develop-a-cnn-from-scratch-for-cifar-10-photo-classification/
def model_summarize(history):
    '''
    Summarize model i.e. print train and test loss
    '''
    # plot loss
    pyplot.subplot(125)
    pyplot.title('Cross Entropy Loss')
    pyplot.plot(history.history['loss'], color='blue', label='train')
    pyplot.plot(history.history['val_loss'], color='orange', label='test')
    pyplot.show()
```

```
def denseblock(input, num_filter = 64, dropout_rate = 0):
    '''
    Create dense block
    '''
    global compression
    temp = input
    for _ in range(l):
        BatchNorm = layers.BatchNormalization()(temp)
        relu = layers.Activation('relu')(BatchNorm)
        Conv2D_5_5 = layers.Conv2D(int(num_filter*compression), (5,5),kernel_initializer="he_uniform" ,padding='same')(relu)
        if dropout_rate>0:
            Conv2D_5_5 = layers.Dropout(dropout_rate)(Conv2D_5_5)
        concat = layers.Concatenate(axis=-1)([temp,Conv2D_5_5])

        temp = concat

    return temp

def transition(input, num_filter = 32, dropout_rate = 0):
    '''
    Create transition block
    '''
    global compression
    BatchNorm = layers.BatchNormalization()(input)
    relu = layers.Activation('relu')(BatchNorm)
    Conv2D_BottleNeck = layers.Conv2D(int(num_filter*compression), (5,5), kernel_initializer="he_uniform" ,padding='same')(relu)
    if dropout_rate>0:
         Conv2D_BottleNeck = layers.Dropout(dropout_rate)(Conv2D_BottleNeck)
    avg = layers.AveragePooling2D(pool_size=(2,2))(Conv2D_BottleNeck)

    return avg

def output_layer(input):
    '''
    define output layer
    '''
    global compression
    BatchNorm = layers.BatchNormalization()(input)
    relu = layers.Activation('relu')(BatchNorm)
    AvgPooling = layers.AveragePooling2D(pool_size=(2,2))(relu)
    flat = layers.Flatten()(AvgPooling)
    output = layers.Dense(num_classes, activation='softmax')(flat)

    return output


num_filter = 12
dropout_rate = 0
l = 12
input = layers.Input(shape=(img_height, img_width, channel,))
First_Conv2D = layers.Conv2D(32, (5,5), use_bias=False ,padding='same')(input)

First_Block = denseblock(First_Conv2D,10, dropout_rate)
```

```
First_Transition = transition(First_Block, 64, dropout_rate)

Second_Block = denseblock(First_Transition, 10, dropout_rate)
Second_Transition = transition(Second_Block, 32, dropout_rate)

Third_Block = denseblock(Second_Transition, num_filter, dropout_rate)
Third_Transition = transition(Third_Block, 32, dropout_rate)

Last_Block = denseblock(Third_Transition,  num_filter, dropout_rate)
output = output_layer(Last_Block)

model = Model(inputs=[input], outputs=[output])
model.summary()
```

```
 average_pooling2d_3 (AveragePo  (None, 2, 2, 88)    0           ['activation_51[0][0]']
 oling2D)

 flatten (Flatten)              (None, 352)          0           ['average_pooling2d_3[0][0]']

 dense (Dense)                  (None, 10)           3530        ['flatten[0][0]']

==================================================================================================
Total params: 518,614
Trainable params: 512,686
Non-trainable params: 5,928
__________________________________________________________________________________________________
```

```
data_gen = ImageDataGenerator(
    rotation_range=22,
    width_shift_range=0.125,
    height_shift_range=0.125,
    horizontal_flip=True,
    fill_mode = 'nearest',
    zoom_range=0.01)
data_gen.fit(X_train)
```

```
# determine Loss function and Optimizer
model.compile(loss='categorical_crossentropy',
              optimizer=Adam(),
              metrics=['accuracy'])
```

```
reduce_lr = ReduceLROnPlateau(monitor='val_loss',factor=0.1,patience= 5,
                              min_lr=0.000001)
filepath = "best_model.hdf5"
checkpoint = ModelCheckpoint(filepath, monitor='val_loss', verbose=1, save_best_only=True, mode='max')

callbacks = [checkpoint, reduce_lr]
```

```
history=model.fit_generator(data_gen.flow(X_train, y_train, batch_size=50),
                    steps_per_epoch = (len(X_train) /50), epochs=50, validation_data=(X_test, y_test),callbacks=callbacks)
```

```
Epoch 42/50
1000/1000 [==============================] - ETA: 0s - loss: 0.2451 - accuracy: 0.9148
Epoch 42: val_loss did not improve from 1.45744
1000/1000 [==============================] - 108s 108ms/step - loss: 0.2451 - accuracy: 0.9148 - val_loss: 0.3815 - val_accuracy: 0.8753 - lr: 1.0000e-06
Epoch 43/50
1000/1000 [==============================] - ETA: 0s - loss: 0.2418 - accuracy: 0.9150
Epoch 43: val_loss did not improve from 1.45744
1000/1000 [==============================] - 103s 102ms/step - loss: 0.2418 - accuracy: 0.9150 - val_loss: 0.3798 - val_accuracy: 0.8761 - lr: 1.0000e-06
Epoch 44/50
1000/1000 [==============================] - ETA: 0s - loss: 0.2410 - accuracy: 0.9147
Epoch 44: val_loss did not improve from 1.45744
1000/1000 [==============================] - 108s 108ms/step - loss: 0.2410 - accuracy: 0.9147 - val_loss: 0.3795 - val_accuracy: 0.8756 - lr: 1.0000e-06
Epoch 45/50
1000/1000 [==============================] - ETA: 0s - loss: 0.2438 - accuracy: 0.9158
Epoch 45: val_loss did not improve from 1.45744
1000/1000 [==============================] - 108s 108ms/step - loss: 0.2438 - accuracy: 0.9158 - val_loss: 0.3793 - val_accuracy: 0.8763 - lr: 1.0000e-06
Epoch 46/50
1000/1000 [==============================] - ETA: 0s - loss: 0.2436 - accuracy: 0.9147
Epoch 46: val_loss did not improve from 1.45744
1000/1000 [==============================] - 103s 103ms/step - loss: 0.2436 - accuracy: 0.9147 - val_loss: 0.3805 - val_accuracy: 0.8753 - lr: 1.0000e-06
Epoch 47/50
1000/1000 [==============================] - ETA: 0s - loss: 0.2462 - accuracy: 0.9132
Epoch 47: val_loss did not improve from 1.45744
1000/1000 [==============================] - 108s 108ms/step - loss: 0.2462 - accuracy: 0.9132 - val_loss: 0.3786 - val_accuracy: 0.8757 - lr: 1.0000e-06
Epoch 48/50
1000/1000 [==============================] - ETA: 0s - loss: 0.2445 - accuracy: 0.9137
Epoch 48: val_loss did not improve from 1.45744
1000/1000 [==============================] - 102s 102ms/step - loss: 0.2445 - accuracy: 0.9137 - val_loss: 0.3789 - val_accuracy: 0.8756 - lr: 1.0000e-06
Epoch 49/50
1000/1000 [==============================] - ETA: 0s - loss: 0.2460 - accuracy: 0.9134
Epoch 49: val_loss did not improve from 1.45744
1000/1000 [==============================] - 108s 108ms/step - loss: 0.2460 - accuracy: 0.9134 - val_loss: 0.3769 - val_accuracy: 0.8763 - lr: 1.0000e-06
Epoch 50/50
1000/1000 [==============================] - ETA: 0s - loss: 0.2455 - accuracy: 0.9135
Epoch 50: val_loss did not improve from 1.45744
```

AS YOU CAN SEE THAT WE HAVE GOT 96% ACCURACY

NOW WE WILL TRY WITHOUT DENSE LAYER

```
def denseblock(input, num_filter = 12, dropout_rate = 0.2):
    '''
    Create Dense Block
    '''
    global compression
    temp = input
    for _ in range(l):
```

```
        BatchNorm = layers.BatchNormalization()(temp)
        relu = layers.Activation('relu')(BatchNorm)
        Conv2D_5_5 = layers.Conv2D(int(num_filter*compression), (5,5), use_bias=False ,padding='same')(relu)
        if dropout_rate>0:
            Conv2D_5_5 = layers.Dropout(dropout_rate)(Conv2D_5_5)
        concat = layers.Concatenate(axis=-1)([temp,Conv2D_5_5])
        temp = concat
    return temp

def transition(input, num_filter = 12, dropout_rate = 0.2):
    '''
    Create transition block
    '''
    global compression
    BatchNorm = layers.BatchNormalization()(input)
    relu = layers.Activation('relu')(BatchNorm)
    Conv2D_BottleNeck = layers.Conv2D(int(num_filter*compression), (5,5), use_bias=False ,padding='same')(relu)
    if dropout_rate>0:
         Conv2D_BottleNeck = layers.Dropout(dropout_rate)(Conv2D_BottleNeck)
    avg = layers.AveragePooling2D(pool_size=(2,2))(Conv2D_BottleNeck)

    return avg

def output_layer(input):
    '''
    Define output layer
    '''
    global compression
    BatchNorm = layers.BatchNormalization()(input)
    relu = layers.Activation('relu')(BatchNorm)
    # as you can see we have removed the dense layer
    AvgPooling = layers. MaxPooling2D(pool_size=(2,2))(relu)
    output = layers.Conv2D(filters=10,kernel_size=(2,2),activation='softmax')(AvgPooling)
    # we are doing the flattened the output layer to apply softmax fucntion on output layr
    flat = layers.Flatten()(output)
    return flat
```

AS YOU CAN SEE THAT WE HAVE REMOVED THE DENSE LAYER FROM THE OUTPUT LAYER IN ABOVE CODE AND AT THAT PLACE WE HAVE USED CONV2D LAYER

```
num_filter = 12
dropout_rate = 0
l = 12
input = layers.Input(shape=(img_height, img_width, channel,))
First_Conv2D = layers.Conv2D(32, (5,5), use_bias=False ,padding='same')(input)

First_Block = denseblock(First_Conv2D,10, dropout_rate)
First_Transition = transition(First_Block, 64, dropout_rate)

Second_Block = denseblock(First_Transition, 10, dropout_rate)
Second_Transition = transition(Second_Block, 32, dropout_rate)
```

```
Third_Block = denseblock(Second_Transition, num_filter, dropout_rate)
Third_Transition = transition(Third_Block, 32, dropout_rate)

Last_Block = denseblock(Third_Transition,  num_filter, dropout_rate)
output = output_layer(Last_Block)


model = Model(inputs=[input], outputs=[output])
model.summary()
```

```
 flatten_2 (Flatten)            (None, 10)           0           ['conv2d_210[0][0]']

==================================================================================================
Total params: 518,286
Trainable params: 512,358
Non-trainable params: 5,928
__________________________________________________________________________________________________
```

```
data_gen = ImageDataGenerator(
    rotation_range=22,
    width_shift_range=0.125,
    height_shift_range=0.125,
    horizontal_flip=True,
    fill_mode = 'nearest',
    zoom_range=0.01)
data_gen.fit(X_train)
```

```
# determine Loss function and Optimizer
model.compile(loss='categorical_crossentropy',
              optimizer=Adam(),
              metrics=['accuracy'])
```

```
reduce_lr = ReduceLROnPlateau(monitor='val_loss',factor=0.1,patience= 5,
                              min_lr=0.000001)
filepath = "best_model.hdf5"
checkpoint = ModelCheckpoint(filepath, monitor='val_loss', verbose=1, save_best_only=True, mode='max')

callbacks = [checkpoint, reduce_lr]
```

```
history=model.fit_generator(data_gen.flow(X_train, y_train, batch_size=50),
                    steps_per_epoch = (len(X_train) /50), epochs=50, validation_data=(X_test, y_test),callbacks=callbacks)
```

```
1000/1000 [==============================] - ETA: 0s - loss: 0.2074 - accuracy: 0.9279
Epoch 42: val_loss did not improve from 0.95265
1000/1000 [==============================] - 94s 94ms/step - loss: 0.2074 - accuracy: 0.9279 - val_loss: 0.3662 - val_accuracy: 0.8847 - lr: 1.0000e-04
Epoch 43/50
1000/1000 [==============================] - ETA: 0s - loss: 0.2077 - accuracy: 0.9277
Epoch 43: val_loss did not improve from 0.95265
1000/1000 [==============================] - 95s 95ms/step - loss: 0.2077 - accuracy: 0.9277 - val_loss: 0.3548 - val_accuracy: 0.8884 - lr: 1.0000e-04
Epoch 44/50
1000/1000 [==============================] - ETA: 0s - loss: 0.2032 - accuracy: 0.9290
Epoch 44: val_loss did not improve from 0.95265
1000/1000 [==============================] - 94s 94ms/step - loss: 0.2032 - accuracy: 0.9290 - val_loss: 0.3519 - val_accuracy: 0.8916 - lr: 1.0000e-04
Epoch 45/50
1000/1000 [==============================] - ETA: 0s - loss: 0.2002 - accuracy: 0.9312
Epoch 45: val_loss did not improve from 0.95265
1000/1000 [==============================] - 96s 96ms/step - loss: 0.2002 - accuracy: 0.9312 - val_loss: 0.3505 - val_accuracy: 0.8912 - lr: 1.0000e-04
Epoch 46/50
1000/1000 [==============================] - ETA: 0s - loss: 0.1971 - accuracy: 0.9300
Epoch 46: val_loss did not improve from 0.95265
1000/1000 [==============================] - 95s 95ms/step - loss: 0.1971 - accuracy: 0.9300 - val_loss: 0.3456 - val_accuracy: 0.8946 - lr: 1.0000e-04
Epoch 47/50
1000/1000 [==============================] - ETA: 0s - loss: 0.1983 - accuracy: 0.9299
Epoch 47: val_loss did not improve from 0.95265
1000/1000 [==============================] - 96s 96ms/step - loss: 0.1983 - accuracy: 0.9299 - val_loss: 0.3529 - val_accuracy: 0.8906 - lr: 1.0000e-04
Epoch 48/50
1000/1000 [==============================] - ETA: 0s - loss: 0.1939 - accuracy: 0.9326
Epoch 48: val_loss did not improve from 0.95265
1000/1000 [==============================] - 95s 95ms/step - loss: 0.1939 - accuracy: 0.9326 - val_loss: 0.3532 - val_accuracy: 0.8903 - lr: 1.0000e-04
Epoch 49/50
1000/1000 [==============================] - ETA: 0s - loss: 0.1948 - accuracy: 0.9321
Epoch 49: val_loss did not improve from 0.95265
1000/1000 [==============================] - 95s 95ms/step - loss: 0.1948 - accuracy: 0.9321 - val_loss: 0.3424 - val_accuracy: 0.8928 - lr: 1.0000e-04
Epoch 50/50
1000/1000 [==============================] - ETA: 0s - loss: 0.1920 - accuracy: 0.9335
Epoch 50: val_loss did not improve from 0.95265
```

\+ Code \+ Text

```
import matplotlib.pyplot as plt


# Print the training and validation accuracy and loss values at the end of each epoch
for epoch in range(1, len(history.history['loss']) + 1):
    print(f'Epoch {epoch}')
    print(f'Training Accuracy: {history.history["accuracy"][epoch-1]:.4f}')
    print(f'Training Loss: {history.history["loss"][epoch-1]:.4f}')
    print(f'Validation Accuracy: {history.history["val_accuracy"][epoch-1]:.4f}')
    print(f'Validation Loss: {history.history["val_loss"][epoch-1]:.4f}')
    print()

# Plot the training and validation accuracy and loss curves
fig, ax = plt.subplots(2, 1, figsize=(10, 10))

ax[0].plot(history.history['accuracy'])
ax[0].plot(history.history['val_accuracy'])
ax[0].set_title('Model Accuracy')
ax[0].set_ylabel('Accuracy')
ax[0].set_xlabel('Epoch')
ax[0].legend(['train', 'val'], loc='upper left')

ax[1].plot(history.history['loss'])
```

```
ax[1].plot(history.history['val_loss'])
ax[1].set_title('Model Loss')
ax[1].set_ylabel('Loss')
ax[1].set_xlabel('Epoch')
ax[1].legend(['train', 'val'], loc='upper left')

plt.show()
```

```
Epoch 1
Training Accuracy: 0.6776
Training Loss: 0.9228
Validation Accuracy: 0.6931
Validation Loss: 0.8851

Epoch 2
Training Accuracy: 0.7139
Training Loss: 0.8238
Validation Accuracy: 0.6818
Validation Loss: 0.9526

Epoch 3
Training Accuracy: 0.7400
Training Loss: 0.7409
Validation Accuracy: 0.6847
Validation Loss: 0.9366

Epoch 4
Training Accuracy: 0.7620
Training Loss: 0.6872
Validation Accuracy: 0.7289
Validation Loss: 0.7948

Epoch 5
Training Accuracy: 0.7758
Training Loss: 0.6506
Validation Accuracy: 0.7024
Validation Loss: 0.9146

Epoch 6
Training Accuracy: 0.7872
Training Loss: 0.6135
Validation Accuracy: 0.7699
Validation Loss: 0.6884

Epoch 7
Training Accuracy: 0.7969
Training Loss: 0.5825
Validation Accuracy: 0.7504
Validation Loss: 0.7580

Epoch 8
Training Accuracy: 0.8080
Training Loss: 0.5573
Validation Accuracy: 0.7739
Validation Loss: 0.6861

Epoch 9
Training Accuracy: 0.8134
Training Loss: 0.5355
Validation Accuracy: 0.7704
Validation Loss: 0.6981

Epoch 10
Training Accuracy: 0.8212
Training Loss: 0.5159
Validation Accuracy: 0.7644
Validation Loss: 0.7232

Epoch 11
```

```
Training Accuracy: 0.8261
Training Loss: 0.4971
Validation Accuracy: 0.8023
Validation Loss: 0.5998

Epoch 12
Training Accuracy: 0.8332
Training Loss: 0.4801
Validation Accuracy: 0.8029
Validation Loss: 0.6156

Epoch 13
Training Accuracy: 0.8368
Training Loss: 0.4694
Validation Accuracy: 0.8022
Validation Loss: 0.6029

Epoch 14
Training Accuracy: 0.8420
Training Loss: 0.4513
Validation Accuracy: 0.8176
Validation Loss: 0.5592

Epoch 15
Training Accuracy: 0.8479
Training Loss: 0.4393
Validation Accuracy: 0.8157
Validation Loss: 0.5432

Epoch 16
Training Accuracy: 0.8535
Training Loss: 0.4289
Validation Accuracy: 0.8345
Validation Loss: 0.5003

Epoch 17
Training Accuracy: 0.8565
Training Loss: 0.4152
Validation Accuracy: 0.8332
Validation Loss: 0.4969

Epoch 18
Training Accuracy: 0.8582
Training Loss: 0.4085
Validation Accuracy: 0.8335
Validation Loss: 0.5163

Epoch 19
Training Accuracy: 0.8614
Training Loss: 0.4012
Validation Accuracy: 0.8372
Validation Loss: 0.4867

Epoch 20
Training Accuracy: 0.8653
Training Loss: 0.3864
Validation Accuracy: 0.8451
Validation Loss: 0.4772

Epoch 21
Training Accuracy: 0.8670
Training Loss: 0.3819
```

```
Validation Accuracy: 0.7713
Validation Loss: 0.7654

Epoch 22
Training Accuracy: 0.8711
Training Loss: 0.3684
Validation Accuracy: 0.8477
Validation Loss: 0.4558

Epoch 23
Training Accuracy: 0.8733
Training Loss: 0.3631
Validation Accuracy: 0.8255
Validation Loss: 0.5436

Epoch 24
Training Accuracy: 0.8742
Training Loss: 0.3612
Validation Accuracy: 0.8286
Validation Loss: 0.5452

Epoch 25
Training Accuracy: 0.8774
Training Loss: 0.3485
Validation Accuracy: 0.8038
Validation Loss: 0.6404

Epoch 26
Training Accuracy: 0.8801
Training Loss: 0.3420
Validation Accuracy: 0.8339
Validation Loss: 0.5145

Epoch 27
Training Accuracy: 0.8813
Training Loss: 0.3386
Validation Accuracy: 0.8517
Validation Loss: 0.4543

Epoch 28
Training Accuracy: 0.8847
Training Loss: 0.3310
Validation Accuracy: 0.8341
Validation Loss: 0.5456

Epoch 29
Training Accuracy: 0.8857
Training Loss: 0.3282
Validation Accuracy: 0.8576
Validation Loss: 0.4403

Epoch 30
Training Accuracy: 0.8886
Training Loss: 0.3243
Validation Accuracy: 0.8432
Validation Loss: 0.4887

Epoch 31
Training Accuracy: 0.8872
Training Loss: 0.3199
Validation Accuracy: 0.8589
```

```
Validation Loss: 0.4321

Epoch 32
Training Accuracy: 0.8928
Training Loss: 0.3084
Validation Accuracy: 0.8349
Validation Loss: 0.5217

Epoch 33
Training Accuracy: 0.8908
Training Loss: 0.3052
Validation Accuracy: 0.8707
Validation Loss: 0.3934

Epoch 34
Training Accuracy: 0.8923
Training Loss: 0.3053
Validation Accuracy: 0.8478
Validation Loss: 0.4752

Epoch 35
Training Accuracy: 0.8930
Training Loss: 0.3014
Validation Accuracy: 0.8539
Validation Loss: 0.4568

Epoch 36
Training Accuracy: 0.8970
Training Loss: 0.2946
Validation Accuracy: 0.8181
Validation Loss: 0.5935

Epoch 37
Training Accuracy: 0.8982
Training Loss: 0.2920
Validation Accuracy: 0.8246
Validation Loss: 0.5720

Epoch 38
Training Accuracy: 0.8992
Training Loss: 0.2891
Validation Accuracy: 0.8693
Validation Loss: 0.4168

Epoch 39
Training Accuracy: 0.9172
Training Loss: 0.2396
Validation Accuracy: 0.8866
Validation Loss: 0.3507

Epoch 40
Training Accuracy: 0.9214
Training Loss: 0.2215
Validation Accuracy: 0.8887
Validation Loss: 0.3475

Epoch 41
Training Accuracy: 0.9264
Training Loss: 0.2125
Validation Accuracy: 0.8906
Validation Loss: 0.3464
```

```
Epoch 42
Training Accuracy: 0.9279
Training Loss: 0.2074
Validation Accuracy: 0.8847
Validation Loss: 0.3662

Epoch 43
Training Accuracy: 0.9277
Training Loss: 0.2077
Validation Accuracy: 0.8884
Validation Loss: 0.3548

Epoch 44
Training Accuracy: 0.9290
Training Loss: 0.2032
Validation Accuracy: 0.8916
Validation Loss: 0.3519

Epoch 45
Training Accuracy: 0.9312
Training Loss: 0.2002
Validation Accuracy: 0.8912
Validation Loss: 0.3505

Epoch 46
Training Accuracy: 0.9300
Training Loss: 0.1971
Validation Accuracy: 0.8946
Validation Loss: 0.3456

Epoch 47
Training Accuracy: 0.9299
Training Loss: 0.1983
Validation Accuracy: 0.8906
Validation Loss: 0.3529

Epoch 48
Training Accuracy: 0.9326
Training Loss: 0.1939
```

OBSERVATION

1. AS WE CAN SEE FROM THE ABOVE PLOT LOSS IT GETTING DECREASED WITH INCREASING IN THE NUMBER OF EPOCHS

2. AS WE CAN SEE THAT ACCURACY GETTING INCREASED WITH INCREASING IN THE NUMBER OF EPOCHS

MODEL PERFORMENCE

```
from prettytable import PrettyTable

table = PrettyTable()
table.field_names = ['Model', 'Epochs', 'Train Accuracy', 'Test Accuracy']
table.add_row(['MODEL WITH DENSE LAYER ', 50, 0.9135, 0.8768])
table.add_row(['MODEL WITH-OUT DENSE LAYER', 50, 0.9335,  0.8926])

print(table)
```

```
+----------------------------+--------+----------------+---------------+
|           Model            | Epochs | Train Accuracy | Test Accuracy |
+----------------------------+--------+----------------+---------------+
|  MODEL WITH DENSE LAYER    |   50   |     0.9135     |     0.8768    |
| MODEL WITH-OUT DENSE LAYER |   50   |     0.9335     |     0.8926    |
+----------------------------+--------+----------------+---------------+
```

i think performence can further improve by increasing the number of epochs

but i dont have computational power with gpu so i limited till 50

✓ 1s completed at 10:21 PM